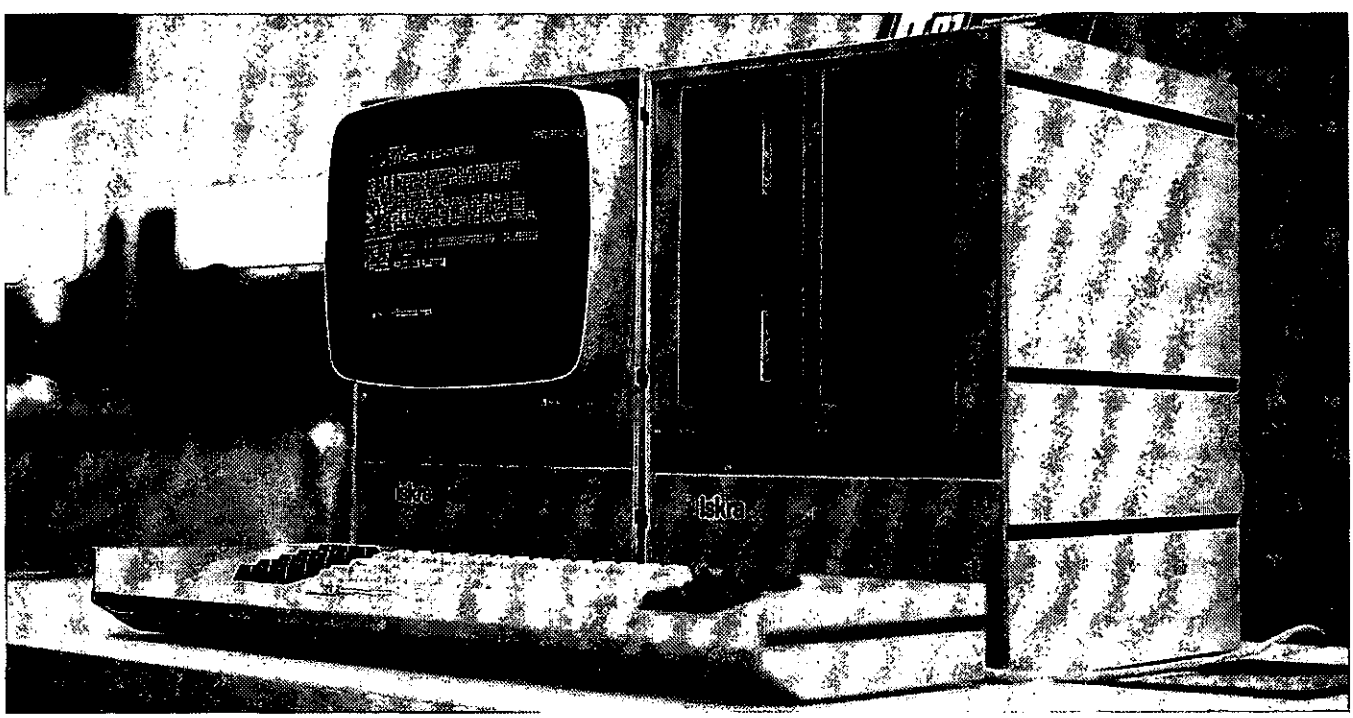
YU ISSN 0350-5596

IskraData

Družina računalniških sistemov za distribuirano obdelavo

ISKRADATA 80-50 Pisalnik
ISKRADATA 80-60 Samostojno delovno mesto
ISKRADATA 80-70 Komunikacijsko delovno mesto
ISKRADATA 80-75 Sinhroni terminal
ISKRADATA 80-80 Samostojni matični sistem
ISKRADATA 80-90 Komunikacijski matični sistem



NAJVAŽNEJŠE ZNAČILNOSTI SISTEMA ISKRADATA

- Modulnost sistema
- Prilagodljivost sistema uporabnikovim zahtevam
- Distribuirana obdelava, ki omogoča popolnoma samostojno delo delovnega mesta in dostop do baze podatkov v matičnem računalniku ali nekem drugem računalniku, ki je vključen v omrežje
- 1—16 inteligenčnih delovnih mest
- Disketna enota omogoča direkten pristop do vsakega podatka
- Za večjo količino podatkov se priključijo na matični računalnik do 4 diskovne enote (40, 80, 160, 200, 300 MB)
- Široka paleta možnosti priključitve perifernih naprav (standardni zaporedni in vzporedni vmesnik RS 232-V 24)
- Sinhroni protokoli ali asinhroni protokoli s kontrolo ali brez nje
- Učinkovita programska jezika BASIC in PASCAL
- Delovno mesto je nezahtevno za okolje
- Večja učinkovitost z manjšimi stroški.

Iskra Elektromehanika Kranj TOZD Tovarna računalnikov

informatica

Časopis izdaja Slovensko društvo INFORMATIKA, 61000 Ljubljana, Parmova 41, Jugoslavija

Uredništvo in uprava: Informatica, Parmova 41, 61000 Ljubljana, telefon (061) 312-988, teleks: 31366 YU DELTA

Letna naročnina za delovne organizacije je 500,00 din, za redne člane 200,00 din, za študente 100,00/50,00 din, posamezne številke 100,00 din
Žiro račun št.: 50101-678-51841
Stališče uredništva se lahko razlikuje od mnenja avtorjev.

Pri financiranju revije sodeluje tudi Raziskovalna skupnost Slovenije.

Na podlagi mnenja Republiškega sekretariata za prosveto in kulturo št. 4210-44/79 z dne 1.2.1979, je časopis oproščen temeljnega davka od prometa proizvodov.

Tisk: Tiskarna KRESIJA, Ljubljana

Grafična oprema: Rasto Kirn

ČASOPIS ZA TEHNOLOGIJO RAČUNALNIŠTVA IN PROBLEME INFORMATIKE
ČASOPIS ZA RAČUNARSKU TEHNOLOGIJU I PROBLEME INFORMATIKE
SPISANIE ZA TEHNOLOGIJA NA SMETANJETO I PROBLEMI OD OBLASTA NA INFORMATIKATA

YU ISSN 0350–5596

LETNIK 5, 1981 – št. 3

V S E B I N A

informatica

Published by INFORMATIKA, Slovene Society for Informatics, 61000 Ljubljana, Parmova 41, Yugoslavia

Headguarters: Informatica, Parmova 41, 61000 Ljubljana, Phone: (061) 312-988, Telex: 31366 Delta

Annual subscription rate for abroad is US $ 22 for companies, and US $ 7,5 for individuals.

Printed by: Tiskarna KRESIJA, Ljubljana

DESIGN: Rasto Kirn

JOURNAL OF COMPUTING AND INFORMATICS

YU ISSN 0350-5596

VOLUME 5, 1981 – No. 3

CONTENTS

TIHA REVOLUCIJA

ANTON P. ŽELEZNIKAR, UREDNIK

UDK: 681.3–181.4

Mikroračunalniška era je pred nami, je v stanju stalnega razvoja od leta 1975 dalje. Takrat so se pojavili prvi mikroračunalniški industrijski izdelki, ko je dalo podjetje Altair iz Albuquerqua v Novi Mehiki (ZDA) na tržišče prvi mikroračunalnik v obliki sestavljenke.

Razvoj mikroračunalnikov je v industriji in tiha revolucija prihaja iz proizvodnih organizacij. Podjetje Intel je proizvajalec neprekosljivih mikroprocesorjev in ima danes praktično najmočnejše raznovrstne mikroprocesorje (8088, iAPX186, iAPX286, iAPX432) oziroma mikroprocesorske družine. Zmogljivosti teh procesorjev presegajo zmogljivosti še do včeraj veljavnih IBMovih standardov oziroma IBMovih sistemov.

Ob t.im. materialni proizvodnji narašča z nezmanjšano močjo tudi informacijska proizvodnja: na tržišče prihajajo programski paketi, od operacijskih do uporabniških sistemov. Usodo mikroračunalniškega tržišča krojita predvsem dva operacijska koncepta in sistema: CP/M in Xenix. Na področju osembitnih in šestnajstbitnih procesorjev se je močno utrdil CP/M, kot novi val pa prihaja iz operacijskega sistema UNIX izpeljani XENIX in še druge operacijske izpeljanke obeh sistemov.

Na osnovnih operacijskih sistemih temelji široko zasnovana in razvejana proizvodnja sistemskih programov, prevajalnikov za visoke programirne jezike in uporabniških programov. Ta programska industrija se razvija predvsem prek malih in srednjevelikih podjetij, ki oblikujejo zares porazdeljeni tip proizvodnje, ki se medsebojno in povratno podpira, pospešuje in napreduje navzgor po strmih finančnih spiralah.

V povezavi s hitrim tehnološkim razvojem procesorjev in programske opreme velja omeniti tudi bistvene tehnološke dopolnitve periferne opreme za mikroračunalniške sisteme. Diski tipa Winchester s pomnilnimi obsegi od 10 do 600 milijonov zlogov ter s svojo veliko zanesljivostjo v delovanju postajajo standardni dodatki tem sistemom ter tako odrivajo včerajšnje upogljive diske na področje prenosa oziroma vstavljanja in in izdajanja podatkov iz winchesterskega diska. Kvaliteta enot z upogljivimi diski pa počasi dosega mere zanesljivosti, ki veljajo za winchesterske diske. Na tržišče prihajajo tudi novi integrirani krmilniki za winchesterske diske (npr. pet integriranih vezij podjetja Western Digital) in vmesniška vezja za upogljive diske (Shugart). Pojavljajo se ceneni iglični tiskalniki, ki omogočajo uporabo znakov nacionalnih abeced in grafičnih znakov ter tiskalniki za kakovostni zapis z nacionalnimi tiskalnimi glavami.

Še pred kratkim se je dozdevalo, da je razvoj in plasma mikroračunalniških sistemov več ali manj določen za naslednje desetletno razdobje in da bodo merila rasti ostala nespremenjena. Tehnologija je namreč daleč presegla današnje stanje uporabe in daje številne možnosti za razvoj novih, zmoglјivejših in cenejših sistemov z uporabo najnovejše, proizvajane tehnologije. In vendar se je brez fanfar in potihoma zgodilo to, kar so strokovnjaki že dalj časa pričakovali: v prostor mikroračunalniških sistemov je vstopil največji in najmočnejši proizvajalec velikih sistemov podjetje IBM.

IBM prihaja na tržišče z osebnim računalnikom, torej z najmanjšim predstavnimom računalniške družine. Ta računalnik naj bi prišel v široko prodajo v ZDA že letos v novembru, prodajali pa ga bodo v trgovinah kot blago za široko potrošnjo. Ta odločitev podjetja IBM je podobna strategiji prodaje IBMovih pisalnih strojev in tudi začetne cene se gibljejo v sprejemljivem intervalu 1565 do 5000 ameriških dolarjev. S to potezo spreminja IBM tudi svojo dosedanjo strategijo tehnološke zaprtosti in vabi k sodelovanju in razvijanju novih modulov za svoj sistem različne proizvajalce materialne in programske opreme. Tudi sestavine najavljenega osebnega računalnika so tuje: Intelov osembitni procesor 8088, operacijski sistem CP/M-86 (Digital Research), UCSD pascal (Softech MIcrosystems) in IBM Personal Computer DOS, ki je ponovitev sistema CP/M (Microsoft).

V čem se kaže odprtost IBMovega koncepta za druge proizvajalce? Operacijski sistem IBM PC DOS bo vseboval dokumentacijo izvirnih programov (s komentarji) za osnovni V/I sistem (BIOS) in za diagnostični program, ki se začne avtomatično izvajati po vklopu računalnika. V priročniku računalnika bodo objavljena logična in tehnična vezja, priključnice in specifikacije pripadajočih standardov. Tako bo moč na IBMov sistem neodvisno in samostojno priključevati poljubno periferijo, dodajati nove sistemske in aplikativne programske pakete, nove plošče tiskanih vezij itn.

Tudi DO Delta, SOZD Elektrotehna je na sejmu Interbiro '81 v Zagrebu prikazala in najavila svoj novi mikroračunalniški sistem Delta 323/M1, ki uporablja procesor Z80, 80k zlogov hitrega pomnilnika, do štiri enote z upogljivimi diski, sinhrone in serijske V/I kanale itd. Tudi ta računalnik uporablja operacijski sistem CP/M in kasneje MP/M in CP/NET. Tudi Deltina proizvodna veriga je odprta za jugoslovanske proizvajalce od tiskanih plošč do operacijske programske opreme. Deltin jugoslovanski koncept se tako uvršča med proizvodne koncepte razvitih, se ne zapira, omejuje in ograjuje proti takoimenovani konkurenci: nasprotno, vabi jo k sodelovanju, k rentabilni in ekonomsko upravičeni proizvodnji, k pokrivanju zahtev domačega in tujega tržišča, k regularni računalniški proizvodnji v SFRJ.

Tiha, mikroračunalniška revolucija je na pohodu tudi pri nas. Razlogi podjetja IBM za prestrukturiranje proizvodnje so znani: prepočasna rast v primerjavi s proizvajalci malih računalniških sistemov. Tudi nerazviti lahko osvajamo računalniško proizvodnjo na njenem spodnjem, lažjem, toda tehnološko in tržno utemeljenem koncu. Nismo še zamudili trenutka, ko se za naslednjih nekaj desetletij lahko vključimo v klub proizvajalcev računalniške tehnologije. Ali bomo administrativno, po merilih nesposobnih sebi preprečili vstop v področje ekonomskega, delovnega in človekovega zamaha navzgor?

USING ADA FOR SPECIFICATION AND DESIGN

I. C. PYLE

UNIVERSITY OF YORK, DEPARTMENT OF COMPUTER SCIENCE

UDK: 681.3.06 ADA

One of the major problems in Information Engineering is that of knowing how to specify a system satisfactorily. The specification occupies a key place in the system life-cycle, concerning the relationship between the client and the system engineer, as well as being the starting point for the design. In this paper I present one major conclusion (that a Specification Language does not solve the problem) and report preliminary work on the use of Ada in the specification and design stages of information engineering.

1. Requirements Specification

It is essential to have a statement of what an information system is supposed to do, both as a basis for the design work at the beginning of a project, and as an independent input in the verification of the delivered product at the end.

The basic dilemma in negotiations for a software contract is that the client wishes to be assured that he is going to get what he needs, without having to do the system design himself. The specification and the design are usually so intimately related that it is virtually impossible to get an adequate specification without having done the design; so where is the contract to start?

The problem lies in the fact that the client has his particular area of expertise which does not (normally) cover specifying information systems. It is for the contractor, rather than the client, to formulate the specification - but it must be to the client's satisfaction.

There have been a number of attempts to overcome this problem by having the specification written in stylised English. The classical example of this is Cobol. Although Cobol was conceived before the problems of requirements specification had been recognised, it was an attempt to define what the program was required to do. A Cobol program was intended to define the job to be done, using a form of English so that the 'user' could write it himself. Clearly that intention has not been achieved.

The designers of Cobol recognised that its users would not like mathematics, so the notation was carefully chosen to permit words rather than mathematical symbols: ADD A TO B GIVING C. I maintain that this is only the tip of the iceberg.

What clients really object to is not mathematics but formality. The cosmetic features of Cobol eliminating the mathematical symbols do not overcome this problem. Equally any other formal language for specifying an information system, such as PSL (Teichroew 1977) SPECIAL (Neumann 1976), HOS (Hamilton 1976), SREP (Alford 1977) or PDL (Caine 1975), meets the same objection: if the system is to be specified in this way at the start of the contract, how is the client to reach that position?

Because of this argument, I reject the common view that systems should be contractually specified using a 'Specification Language' designed for the purpose, necessarily formal, capable of being interpreted by a computer. It inflicts the culture of the System Engineer on the client, in a way that the client will (usually) be unable to tolerate.

2. Requirements Model

The above negative conclusion suggests a re-think of the purpose of the specification. The client wishes to determine as early as possible that a system can be designed to meet his needs subject to his constraints. He will probably not be able to state explicitly what his needs and constraints are, but he will recognise conformance or inadequacy if he is told clearly enough.

How can we give him the information for his decision? The job of the system engineer at the early stages of a contract is to provide the client with enough information (but not too much) to permit this check. It is not a formal check: it is 'the satisfaction of the client'.

There is a place for formalism in requirements analysis, but it is not at the interface between the client and the system engineer: it is within the system engineer's area of concern, where he can interpret it to the client. The formal representation of the requirement may be considered to be a requirements model. The first stage of a contract is therefore to establish a requirements model, to the satisfaction of both the client (so that it expresses his needs and constraints) and the engineer (so that it provides the basis for a design and implementation).

2.1. Client's View

No single view of the requirements model will satisfy the user, any more than a single test of a program demonstrates its correctness. The client's satisfaction and confidence can best be achieved by a multiplicity of views, from different angles, concentrating on different aspects, but all of the same underlying model. This has been called 'animation' of the model.

Thus we may have examples of what the target system might do in various particular circumstances; we may have analyses of cases that have been covered to locate inconsistencies or incompleteness; we may have resource estimates for the target system; we may have resource estimates for the production contract. In some cases the client may be willing to study the formal model itself (obviously advantageous), but this should not be taken for granted.

Any single view can be checked by the client; his confidence is achieved by the variety and multiplicity of the checks.

2.2. Engineer's View

A single model is essential, which must eventually be unambiguous and complete. This must be the basis for all the interpretations made to the client by different kinds of analysis of the model.

During the period when the requirements model is being built, it is necessarily incomplete, and may well contain ambiguities. One of the principal reasons for consolidating the information as one model is to allow the analysis to be made for consistency and completeness. The kind of notation used to build up and modify the model could perhaps be called a Specification Language, but its role would be very different from the usual connotation of that phrase: here we are thinking of something solely for the engineer and not intended to be understandable by the client.

The model should almost certainly be held in a computer-manipulable form, both for ease of modification as facts about it are discovered, and for analysis through the application of various software tools. We discuss later what form a model might take.

2.3. Organisational View

In most cases the client and the engineer are organisations rather than individuals. The requirements are probably not known in total to any one person, although several people throughout the client organisation could check them for satisfaction from some particular points of view. Likewise the engineering organisation will expect to partition the formulation and design work among members of its staff.

For this reason again, we find it necessary to have one model with many views.

It is common when acquiring information about the requirements that the engineer gets inconsistent statements from client's staff. This to be is expected in a human organisation, and should lead to a recognition by the engineer of a hitherto unnoticed subtlety, or a decision by a more senior member in the client organisation on which statement to believe. (Or the decision to adopt several provisionally, intending to reject one after implications have been explored.)

With current technology, it also happens that engineers give different (inconsistent) interpretations of the requirements model. This is clearly undesirable, and since it is in the engineer's area of expertise, should not be tolerated. Derivations from a common model give the best chance of avoiding confusion.

3. Requirements and Design

The requirements are needed to allow design to begin. Requirements state what is to be achieved; design works out how they are to be achieved. Both the requirements and the design have to be specified in some way; as we shall see, the design eventually leads to a program. At many stages during the design, progress consists of identifying subsidiary requirements, and expressing the solution of one part in terms of subsidiaries to be designed subsequently. Thus the above discussion also applies to a considerable extent to the subsequent stages of design, after the requirements have been identified.

The significant difference here is that both 'requirer' and 'designer' are members of the engineer's organisation, and can be expected to have a much closer rapport and common philosophy than between the client and the engineer. In particular the argument rejecting formality no longer holds.

Thus the internal requirements can be expected to be specified formally, forming in this case the interface between two parts of the engineer's organisation, concerning one unit that is designed in terms of other units.

To focus attention on the design activity, we will compare a program specification with the eventual program that meets that specification. What is the relationship between these? To what extent do they contain the same information? What is involved in the progression from requirements specification to eventual program?

3.1. Size and Structure

An important question at this point is whether the requirements specification and eventual program are essentially different in nature, so that a different notation must be used irrespective of the amount of information concerned. If there are differences, we must discover their origins and consider how they can be reconciled.

The specification is of course shorter than the program, so contains less information. The first point to consider is whether there is any information in the specification which is not in the program. In conventional programming technology, the answer is yes: much of the information about overall intention is absent from the program (unless the designer has taken care to incorporate the specification as comments).

Viewed in retrospect, after the program has been designed, it is advantageous (for program maintenance) that the program text should incorporate information about the requirements, namely the functional specification. We can therefore consider the next question: what is the additional information in the program which is not in the functional specification?

Broadly the answer is that the program contains more mechanistic detail. During the design stage, the increase of information is progressive, as the needs for particular program units are recognised, their necessary characteristics are identified, and the techniques to be used to obtain those characteristics are determined.

One significant difference in notation is that between a diagram and text. We have reported (Pyle 1980) some implications of this, relating to MASCOT diagrams to corresponding Ada text. Although a diagram may be a useful primary starting point, it is not a good working medium during the design stage, because of the mechanical difficulty of keeping a set of diagrams up to date. In the absence of a graphics editor, it is better to work with a text representation (which can be easily edited) and derive the diagrams from the text.

4. Design and Programming

It is a primary basis for this work that the design process covers a gradual development of the program, by a series of transformations. At each stage there is a 'current understanding'; there is no clear separation between designing an information system, and programming it. A design can be viewed as an incomplete program – that is a program in which certain entities or concepts are used which have not yet themselves been designed. There is no single sudden transition from "desired effect" to "means of achieving effect", but many such transitions incrementally (through intermediate specifications) throughout the design process.

During the design stage, the program is necessarily incomplete: some parts which are eventually needed have not yet been considered. At the beginning of the design stage we have only the overall specification; at the end we have the whole program.

A programming language which is capable of stating specifications and progressive detail is also appropriate for expressing a design. A compiler for a such a language would reject a design, because it would be an incomplete program. However, the diagnostic messages from the compiler would be important, as they would identify omissions, and draw attention to any inconsistencies.

4.1. Design activities

This view of gradual transition from functional specification to eventual program underlies our search for effective information engineering techniques. Our starting point is that there is an essential similarity between the requirements model, the intermediate functional specifications, and the corresponding program. The crucial difference will emerge in the course of the work.

We can identify two kinds of activity during the design stage which increase the amount of information in the program: creation and discovery. These occur many times over, creation being the recognition and identification of units needed, and discovery the incorporation of facts found to be relevant about the environment of the program. As the new information is obtained, it must be checked for consistency with the design so far, and the relevant specification.

At many times during the design stage, a point is reached when a mistake is made or a prior decision is regretted: some alternative design path is seen to be preferable. No design goes straight from start to finish, and a good design tool is one which permits the designer to locate the relevant mistake or decision point, and recover as much as possible of the work which has been done after the mistake or decision.

Given these points, it is desirable to use a consistent and coherent notation throughout the design process. If there are to be any discontinuities in the design notation, there are sure to be errors (omissions and inconsistencies) which will need further effort to detect and correct.

High level programming languages are obvious tools to use for this purpose. They already include features for separation of concerns and limiting scopes as a means of reducing inter-dependency, and are supported by tools which check for internal consistency.

5. Ada as a Program Design Language

The features of Ada for programming in the large (separate compilation, with a database containing mutually consistent compilation units, which include packages and generic program units) make it a very attractive prospect as a program design language. Within a program, Ada recognises the distinction between specification and body of a program unit; in other words, between the information available externally about that unit (which the rest of the program may use) and the internal information (which determines how that program unit is designed to work). The same kind of distinction applies between the specification of a program as a whole and its realisation.

Ada also recognises that programs are written incrementally, usually by more than one person, and provides for textual separation of specifications and bodies.

The module structure of Ada programs (more powerful than the traditional block structure), together with the visibility rules and the separate compilation features, make Ada very suitable for use as a program design language. Subsequent sections discuss its virtues and some of its defects.

5.1. Examples of Ada as a Program Design Language

To illustrate this point, consider the following program unit for a filtration plant

```
procedure SINGLE_FILTER is
begin
  START_UP;
  loop
    CLEAN_FILTER;
    DELIVER_WATER;
  end loop;
exception
  when others =>
    CLOSE_DOWN;
end SINGLE_FILTER;
```

Written like this, the designer has shown the overall sequence of operations intended, and invented names for the major phases. As it stands, however, this text is incomplete.

We can deduce from the context that the names are intended to denote procedures, but currently they are just undeclared identifiers. Ada allows us to express the essential characteristics of these major phases as a package specification:

```
package MAJOR_PHASES is
  procedure START_UP;
  procedure DELIVER_WATER;
  procedure CLEAN_FILTER;
  procedure CLOSE_DOWN;
end MAJOR_PHASES;
```

This package specification can be separately compiled in Ada. Once this has been done, the SINGLE_FILTER program can be made into a proper compilation unit by giving it a context specification:

```
with MAJOR_PHASES;  use MAJOR_PHASES;
```

The design would then be progressed by writing the package body for MAJOR_PHASES. This is top - down design.

As another illustration, the filtration plant contains a number of valves which control the flows of fluid in the plant; they are all used in similar ways, so we will certainly need facilities to manipulate them. The actual details of the valves will probably not be known to the designer initially (they may not yet have been manufactured or selected), so information about them will be discovered (or negotiated) as the design progresses. However, we can record partial information about them thus :

```
generic
  type VALVE is (<>);
package VALVE_DETAILS is
  type VALVE_STATUS is (OPEN, CLOSED);
  type CHANNEL is new NATURAL;
  type VALVE_PARAMETERS is
    record
      CONDITION : VALVE_STATUS;
      CONTROL : CHANNEL;
    end record;
  CONTROL_VALVE : array (VALVE) of VALVE_PARAMETER
end VALVE_DETAILS;
```

This shows that we know already that there will be a number of valves, although we have not yet decided (or discovered) how many there will be or what they will be called. Each valve might be open or closed, and will need an internal record of its current condition and the channel to which it is connected. Further design work may elaborate this, when we discover for example that each valve has an associated lamp to show its condition and four channels instead of one:

```
type VALVE_PARAMETERS is
  record
    OPERATE : array (VALVE_STATUS) of CHANNEL;
    CHECK   : array (VALVE_STATUS) of CHANNEL;
    LAMP    : CHANNEL;
  end record;
```

Further development will identify procedures to carry out operations on the valves, such as

```
procedure MOVE_VALVE (V : VALVE; TO : VALVE_STATUS);
```

This is bottom-up design.

As a third illustration, consider programs involving numerical calculations. Errors of numerical analysis must be considered, and decisions must be taken about representations (fixed point, floating point; single length, double length). Ada permits the program to be written using numeric types of as yet unspecified precision, by using a generic parameter such as

```
type REAL is digits <>;
```

We can express sensible ranges and formulae thus

```
type PRINCIPAL_ANGLE is new REAL range 0.0 .. 2.0*PI;
function ARC_COS(X : REAL range -1.0 .. 1.0)
      return PRINCIPAL_ANGLE range 0.0 .. PI;
PSI   : PRINCIPAL_ANGLE range 0.0 .. PI;
ALPHA: REAL range 0.0 .. 2.0;
         -- range ABS(1.0 - BETA) .. (1.0 + BETA)
BETA : REAL range 0.0 .. 2.0;
         -- range ABS(ALPHA - 1.0) .. (ALPHA + 1.0)
```

This gives a basis for analytical checking of the algorithms and ranges. In this case, for example, we show the ranges for the argument and result of the ARC_COS function, and introduce variables with much more precisely defined properties than usual. We can also add further facts, such as

```
--  assert ALPHA + BETA >= 1.0;
--  assert ABS(ALPHA - BETA) <= 1.0;
```

and confirm the sense of formulae:

```
PSI := ARC_COS( (ALPHA ** 2 + BETA ** 2 - 1.0) /
                ( 2.0 * ALPHA * BETA ) );
```

The subtypes for PSI and ARC_COS'RETURN are consistent; the argument of ARC_COS has the right range since

```
(ALPHA + BETA) ** 2 >= 1.0;
(ALPHA - BETA) ** 2 <= 1.0

ALPHA ** 2 + BETA ** 2 + 2.0 * ALPHA * BETA >= 1.0
ALPHA ** 2 + BETA ** 2 - 2.0 * ALPHA * BETA <= 1.0

-2.0 * ALPHA * BETA
     <=  ALPHA ** 2 + BETA ** 2 - 1.0
         <= 2.0 * ALPHA * BETA

-1.0 <=
     ( ALPHA ** 2 + BETA ** 2 - 1.0 ) /
        ( 2.0 * ALPHA * BETA )
          <= +1.0
```

The actual precision needed may be investigated as a part of the design activity, separate from establishing the algorithm to be used.

5.2. Support Aids with Ada

Clearly Ada has much to offer in many aspects of program design. The major benefit from using Ada as a design language is that it gives a smooth transition from the requirements to the eventual program. Incidental benefits are that software tools designed for use with Ada programs may be used with the partially - designed program, and are likely to give help to the designer during the design process. With Ada, we already have rudimentary design analysis tools.

Imagine a partially designed Ada program submitted to the compiler. Diagnostic messages are likely to be given; in general, these will be useful check-lists for the designer, identifying items which must be studied further. At some stage a sufficient set of facilities and types can be written to form a program unit which may be compiled without diagnostic messages : this will then identify the separate subunits and module bodies which must be written subsequently. When a module body is written, the compiler will automatically check for consistency with its specification.

Throughout the text written at any stage, the strict typing rules will ensure that entities are named and used in consistent ways, and also give early warning of any usage which appears inconsistent.

This idea has already been used to good effect in a real project, to enforce house standards on design documentation. An Ada library package was written, introducing types for the various kinds of entity involved, with the possible values enumerated, e.g.

```
type FILE_ACCESS is
      (READ, UPDATE, APPEND, CREATE, DELETE);
type FILE_AREA is
      (INPUT, REVIEW, MAIN_OUTPUT, CONSOLE);
```

and procedure specifications for relationships between them. The actual design is written as separate compilation units, in the context of this library unit, so that they all use the same nomenclature, and the relationships are expressed consistently.

Other Ada language tools will also assist the designer - a diagram generator will show relationships pictorially; a formatting printer will keep programs neat while they are undergoing development; a cross-reference generator will allow usage of entities to be connected with their definitions; and an Ada-specific editor would simplify input.

6. Inadequacy of Programming Languages

Having noted the relationship between programming and design, we must beware of going too far. Conventional programming languages do not permit the expression of some crucial design information.

We distinguish between declarations and statements, as in ordinary programming languages. For the purposes of specification, declarations are relevant but statements are not. This is because the specification has to define the effect but not the means; the statements express the means but not (explicitly) the effect. To specify a design, we want the functionality implied by a sequence of statements.

Another way of seeing this distinction is to recognise that the functional relationships are a permanent feature of the system, to be described in its specification and satisfied by the designer. Permanence is the crucial fact: invariance of the functionality during execution which is independent of the particular

design. Statements are necessarily dynamic, since each is executed at a particular time in the program, and it changes the state of the system.

The major deficiency of most programming languages is that they do not allow the effect of a piece of program to be stated, other than by its means - the sequence of statements to achieve that effect.

By focusing attention on this point, we can begin to characterise information systems, and describe the ways in which their functionality may be expressed.

Another way of putting this is to consider the statements and semicolons interspersed in a piece of program. The statements describe transitions from one state to another; the semicolons correspond to states (which are not explicitly described). What we want is the reverse emphasis: explicit descriptions of the states that are relevant, but elision of the detailed description of the transitions.

6.1. Shortcomings of Ada as a PDL

The benefits described above arise from the rules of Ada, ensuring consistency and compatibility of interfaces. But there is nothing in Ada concerning the program semantics, that is the functionality of the program units, defining their desired effects. We now consider how the semantics of a program may be expressed in an Ada-like notation.

An Ada subprogram is specified by its heading, which gives the name and parameters. The intended effect of the subprogram is not specified; it is usually expressed implicitly in the name, e.g.

```
function SQRT (X : FLOAT) return FLOAT;
procedure MOVE_VALVE(V : VALVE; TO : VALVE_STATUS);
```

The reader of such a program is simply left to guess that because the name is SQRT, the function takes a square root, and because the procedure is named MOVE_VALVE, it cause the specified valve to be moved to the status given by the second parameter.

Formally, Ada does not deal with the program semantics, above the level of the elementary statements. The body of a procedure expresses the actions to be carried out when the procedure is called, in terms of the individual statements (whose semantics are defined). But this is too mechanistically detailed to be regarded as the semantics of the procedure itself, that is, of a statement calling that subprogram from elsewhere. This is the crucial difference between requirements specification and eventual program - not so much level of detail as the difference between intent and achievement: between the effect obtained and the way of obtaining it. Conventionally, the effect is conveyed by the choice of identifier, usually supplemented by comments. What we need is a means of expressing the relationship between the states of the system immediately before and immediately after a call of the procedure : what in the program proving literature are known as the preconditions and postconditions of the action.

7. Transitions in State Space

At a sufficiently large scale, and at a sufficiently small scale, the concept of an action is important in information systems. It could be a job in a batch processing system, or a single machine instruction. It carries out a particular effect, in a finite time, by putting the system in a final state which is in some way related to its initial state, taking finite resources to do it. (And it may fail.)

A necessary and sufficient description of the functionality of such an action is the relationship between the initial and final states of the system.

If we imagine the initial and final states as points in a state space, and the functionality specification as a relationship between them, then the nature of the design task is to discover a trajectory in the state space which is feasible and compatible with the resources available.

The trajectory is not discovered entirely in one leap (although a breakthrough in realising feasibility usually implies this); we do not locate a continuous trajectory through state space, but a number of intermediate points, with appropriate relationships between them.

Verifying this design consists of showing that the relationships involving the intermediate states can be combined together to give the required relationship between the initial and final states.

7.1. Subprograms

We can specify a subprogram in this way, subject for the moment to the limitation that the state space can only cover the variables in scope at the time of the action, and no commitment is made as to their representation.

Where the subprogram involves only internal manipulations, we can express the relationship as an assertion, e.g.

```
procedure SWAP
          (A,B : in out DATA
          -- assert A'IN = B'OUT;
          -- assert B'IN = A'OUT;
          );
```

The effect of the procedure is stated by the 'assert' predicates, where the IN and OUT qualifiers denote the initial and final values of the parameters. (These are not necessary for pure in and out parameters.)

The assertion here is a relic of preliminary Ada : revised Ada does not contain assert statements. The role I see for assertions is as declarations rather than as statements. The assertions specify the high-level semantics of the subprogram, and could in principle be checked against the body when it is eventually written.

The description of the effect for procedure SWAP rests solely on the notion of equality. Other subprograms may depend on more powerful notions:

```
function IS_EVEN (I: INTEGER) return BOOLEAN
   -- assert IS_EVEN'RETURN = (I mod 2 = 0);
          ;
```

This depends on the notions of arithmetic and the operations of the type INTEGER; the effect of the function, stated by the assertion, is to calculate a result which bears the given relationship to the parameter(s). The RETURN qualifier on the function name denotes the result.

More notions of arithmetic are needed for:

```
Function SQRT
    (X : REAL range 0.0 .. REAL'LAST) return REAL
         -- assert SQRT'RETURN > = 0.0;
         -- assert (ABS(SQRT'RETURN**2 - X)<
         -- SQRT'RETURN * REAL'EPSILON);
               ;
```

Numerical analysis is required here to state the relationship. Some optimisation is involved - the intention is to get the best possible result, whose square is closest to X. We could perhaps have done better by first defining the notion of 'nearest':

```
generic
  type T is private;
  with function F (Z : T) return REAL;
function NEAREST (X : REAL) return T is
  function CLOSE (Z : T)
    return REAL range 0.0 .. REAL'LAST is
  begin
    return ABS ( F(Z) - X );
  end CLOSE;
begin
  --  for all Z in T
  --    assert CLOSE(NEAREST'RETURN) <= CLOSE(Z);
end NEAREST;
```

and instantiating this in the present case:

```
subtype POSITIVE_REAL is
    REAL range 0.0 .. REAL'LAST;
function SQUARE (X : POSITIVE_REAL)
    return POSITIVE_REAL
     -- assert SQUARE'RETURN = X * X;
    ;
function SQRT
    --       (X : POSITIVE_REAL) return POSITIVE_REAL
    is new NEAREST (T => POSITIVE_REAL;
                    F => SQUARE);
```

Incidentally, this shows that the result of the originally stated function should have been specified as POSITIVE_REAL.

This discussion has deliberately started with simple examples. To express the functionality of more complicated subprograms, we find it necessary to state subsidiary notions in terms of which the desired relationship will be expressed. These typically concern properties or entities of a given type, so occur in the specification of a package introducing the relevant data type.

7.2. Packages

A package introduces a set of facilities that may be used in the program around it. In the state-transition model it corresponds not to an individual transition relationship, but to the concepts underlying a set of transitions. Characteristically, in the abstract specification of a package we find not transitions but invariants.

The types introduced in a package, and the subprograms introduced to manipulate objects of that type, have their own internal requirements. The types express package-specific concepts (whose implementation details are hidden), and the invariants express permanent properties of these concepts which are supported by all the manipulations. Assertions in the package specification can express these invariants. By placing them in the specification, we make them visible with the package, and thus applicable to entities outside the package. For example, where a type is defined in the package, variables of that type may be declared outside. The assertions about the type in the package specification apply to all variables of that type, and can be regarded as both preconditions and postconditions for every manipulation done by the package. In the case of private types, where all manipulations on such variables have to be in the package, they are invariants.

The following example in Ada shows how this might arise with queues.

```
generic
  type OBJECT;
package QUEUE_HANDLING is
type ITEM;
type POINTER is
  access ITEM;

type ITEM is
  record
    CONTENTS : OBJECT;
    NEXT : POINTER;     -- null for last item in queue
    PREVIOUS : POINTER  -- null for first item in queue
  end record;
    -- for all ANY_POINTER in POINTER assert
    --   ( ANY_POINTER.NEXT = null or else
    --     ANY_POINTER.NEXT.PREVIOUS = ANY_POINTER) and
    --   ( ANY_POINTER.PREVIOUS = null or else
    --     ANY_POINTER.PREVIOUS.NEXT = ANY_POINTER);

type QUEUE is
  record
    FIRST : POINTER;  -- null if queue is empty
    LAST : POINTER;   -- null if queue is empty
  end record;
    --  for all ANY_QUEUE in QUEUE assert
    --   ( ANY_QUEUE.FIRST = null or else
    --     ANY_QUEUE.FIRST.PREVIOUS = null) and
    --   ( ANY_QUEUE.LAST = null or else
    --     ANY_QUEUE.LAST.NEXT = null);

procedure STRIP (ONE, ANOTHER : QUEUE);
    -- move ONE.FIRST to become ANOTHER.LAST

end QUEUE_HANDLING;
```

We can use these assertions usefully when writing the bodies:

```
package body QUEUE_HANDLING is

  procedure STRIP (ONE, ANOTHER : QUEUE) is
      -- move ONE.FIRST to become ANOTHER.LAST
    THIS : POINTER;
  begin
    -- assert ONE.FIRST /= null;
      -- something to move
    THIS := ONE.FIRST;
    -- assert THIS.PREVIOUS = null;
      -- since it was the first item
    ONE.FIRST := THIS.NEXT;
    ONE.FIRST.PREVIOUS := null;
      -- mark it as first item in queue
    THIS.PREVIOUS := ANOTHER.LAST;
      -- attach it to other queue
    THIS.NEXT := null;
      -- mark it as last item in queue
    ANOTHER.LAST.NEXT := THIS;
      -- adjust previous last item
    ANOTHER.LAST := THIS;  -- new last item
    -- assert ANOTHER.LAST.NEXT = null;
  end STRIP;
end QUEUE_HANDLING;
```

The assertions give us preconditions about the parameters on entry to the procedure. We can (and must) check that everything we change in the procedure (apart from local variables) conforms to the assertions on exit.

In the context of this package, using the notions it introduces, we can state the semantics of other procedures:

```
procedure ADD_TO_QUEUE_END
          (I:OBJECT;
           Q: in out QUEUE
-- assert  Q'OUT.LAST.CONTENTS = I
-- assert  Q'OUT.LAST.PREVIOUS = Q'IN.LAST
          );
```

8. Continuing Systems

Not all information systems can be specified by a state-space transition, although there may well be inner levels of detail for which that model is valid.

A principal characteristic of embedded computer systems is that they are required to perform some function 'continuously' rather than as a discrete action. In terms of a state model, they have to maintain some relationship over a period of time, while various perturbing influences apply, which must be observed and responded to. Here concurrency can no longer be ignored.

The major omission for the above state-space model is its inability to communicate with other systems.

8.1. Intercommunication

At a higher level than the 'action' level, the crucial functionality is not the state changes, but the information communicated from one system to another. The basic purpose of a communications protocol is to allow two partners to cooperate by each knowing about the internal state of the other: the communication is used to convey the state information.

We can cover the elementary input/output characteristics by treating them as changes of representation. They establish a mapping between internal states and the environment of the information system, thus allowing the scope of the state-space model to be extended to cover actions involving data input and output. Thus we may have for MOVE_VALVE the postcondition that valve V is in the state TO. (The procedure may be developed by adding the possibility that if the required postcondition cannot be reached in a given time, an exception is raised.)

The representation change covered by the elementary input/output is that between internal descriptors of the valve status and the corresponding actual valve state. We normally work with two complementary relationships: desired state(internal) to actual state(external), and actual state(external) to observed state(internal). The model does not yet cover the 'control' aspects of input/output.

9. Conclusion

Functional requirements are best specified as a formal model, interpreted in many different ways. Functional models are useful at many levels within a system. We may categorise information systems by the kind of functional model they require: distinguishing transition-effecting systems, facility-providing systems and relation-maintaining systems. Ada subprograms, packages and tasks correspond to these.

As well as the functional requirements of a system, there are also constraint requirements, concerning performance, cost, reliability etc. The present work does not directly address these problems, although by filtering out the functional aspects, more light may be shed on the constraints.

We have reviewed the prospect of Ada as a Program Design Language, and found that it has adequate notations for the major issues, particularly structural ones. A serious deficiency remains, that the specifications of interfaces do not specify effects; the emphasis in Ada (as in any programming language) is on transitions between states, rather than the states themselves. The notation of Ada can be stretched to cover these points, using declarative assertions.

10. Acknowledgements

The ideas presented here have been stimulated by work on Application Oriented Specifications at EWICS, on program specifications at the Graz International Workshop on Real Time Programming, (Haase 1981), and at the IEE symposium on Software Design Techniques, 1980. I have also appreciated discussion with K.C. Mander and C.J. Tully.

References

Alford, M.: A Requirements Engineering Methodology for Real-Time Processing Requirements. IEEE Transactions on Software Engineering, Vol SE-3, No 1, January 1977.

Caine, S.H. & Gordon E.K.: PDL - A tool for software design. (USA) National Computer Conference, 1975.

Haase, V. H. (ed) Real Time Programming, 1980, Pergamon Press 1981.

Hamilton, M. & Zeldin, S.: Higher Order Software - A Methodology for Defining Software. IEEE Transactions on Software Engineering, Vol SE-2, No 1, March 1976.

Naumann, P.G. et al.: A method of software requirements definition Software development and proofs of multi-level security. Second International Conference on Software Engineering, October 1976; IEEE 76 CH 1125-4C.

Pyle, I.C. : Ada as a Design Tool. IEE Colloquium on Software Design Techniques, December 1980.

Pyle, I.C. Towards specifying an information system; University of York, Computer Science Report YCS.43 (1981).

Teichroew, D., E.A. Hershey III: PSL/PSA: a computer-aided technique for structured documentation and analysis of information processing systems. IEEE Transactions on Software Engineering, Vol SE-3, No 1, January 1977.

INFORMATICA 3/1981

PODELA NA SLOGOVE SRPSKOHRVATSKIH REČI

DUŠKO VITAS

MATEMATIČKI INSTITUT, BEOGRAD

UDK: 681.3.06:808.61

U radu se opisuje jedan postupak automatskog rastavljanja na slogove reči srpskohrvatskog jezika. Problem rastavljanja konsonantskih grupa kao i problemi svojstveni srpskohrvatskoj ortografiji su istaknuti. Prikazani su primeri rastavljanja karakterističnih reči.

SYLLABLE DIVISION OF SERBOCROATIAN WORDS. The paper describes a procedure of automatic syllable division of serbocroatian words. Problem of consonant groups decomposition as well as problems inherent to serbocroatian orthography is outlined. Examples of syllable division of characteristic words are shown.

1. UVOD

Problem rastavljanja reči na kraju retka usko je povezan sa problemom rastavljanje reči na slogove. Kako se slog, obično, shvata kao najmanja artikulaciona jedinica govora, rastavljanje reči na granici sloga omogućuje lak izgovor prenesenog dela reči.

Slog se sastoji iz centralne foneme i marginalnih fonema. Centralne foneme su najčešće vokali ali, u nekim jezicima, to mogu biti i druge foneme (npr. u srpskohrvatskom, r, l, itd.). Marginalne foneme su konsonanti.

Lingvistički gledano, fonematska struktura sloga je odredjena skupom pravila specifičnih za svaki poseban prirodni jezik. Ako sa v označimo vokal a sa k konsonant, onda razlikujemo otvoren slog (sa strukturom kv) i zatvoren slog (sve ostale strukture). Postojanje zatvorenih slogova u jednom jeziku povlači postojanje konsonantskih grupa.

Zadatak odredjivanja granice sloga se, stoga, svodi, u osnovi, na problem identifikacije zatvorenih slogova, odn. na problem odredjivanja skupa pravila za rastavljanje konsonantskih grupa. O nekim drugim teškoćama, specifičnim za odredjivanje granice sloga u srpskohrvatskom jeziku, biće reči kasnije.

Za srpskohrvatski jezik su predložena dva skupa pravila (Belić/34/, Stevanović /64/), koja se razlikuju po načinu rastavljanja nekih konsonantskih grupa. Verovatno kao posledicu živih lingvističkih diskusija početkom pedesetih godina o granici sloga, Pravopis srpskohrvatskoga književnog jezika (Pravopis /60/), umesto pravila o rastavljanju konsonantskih grupa, uvodi intuitivne pojmove "lako" i "teško" izgovorljivih konsonantskih grupa.

Jedan pokušaj prikazivanja statističkih pokazatelja o slogovima odredjene strukture, ali ne i o strukturi konsonantskih grupa je dat u (Koković/79/). U ovom radu nisu eksplicitno navedena pravila upotrebljena za rastavljanje konsonantskih grupa. Prema ovom istraživanju, u uzorku od oko 400.000 slogova, oko 70% je otvorenih slogova (sa strukturom v i kv), dok preostalih 30% čine zatvoreni slogovi, kod kojih se uočava 17 različitih rasporeda marginalnih fonema oko centralne foneme.

Kako problem rastavljanja reči na slogove nije konačno lingvistički rešen, iz do sada izloženih činjenica sledi da svaki pokušaj automatskog rastavljanja može dati samo delimično zadovoljavajuće rešenje u odnosu na Pravopisom postavljene zahteve.

U daljem tekstu ovog rada opisujemo program koji vrši rastavljanje konsonantskih grupa, opciono, prema Belićevim ili Stevanovićim pravilima. Navedena su takodje i rešenja nekih drugih problema u automatskom rastavljanju reči.

2. OPIS PRAVILA

Postupak za odredjivanje granice sloga se zasniva, u opštem slučaju, na odredjivanju položaja susednih centralnih fonema u reči i ispitivanju svojstava konsonanata koji se medju njima, eventualno, nalaze.

Pri tome, kao centralne foneme u srpskohrvatskom jeziku se mogu pojaviti:
(a) vokali (a,e,i,o,u),
(b) glas r, koji se kao vokal pojavljuje u sledećim slučajevima:
-na početku reči, ispred konsonanta (npr. rt)
-u interkonsonantskom položaju (npr. krt)
-na kraju reči, iza konsonanta (npr. masakr)
-ispred o postalog od l (npr. grôce),
(c) glasovi l, m, n na kraju reči, iza konsonanta (npr. bicikl, njutn).

U daljem izlaganju centralne foneme ćemo označavati slovom v a marginalne slovom k, sa ili bez indeksa. Pravopisnim rastavnim znakom (-) označavaćemo položaj granice sloga.

Jednosložne reči, tj. reči u kojima se pojavljuje samo jedna centralna fonema, se ne rastavljaju, pa neće biti predmet daljeg razmatranja.

Kod višesložnih reči, izmedju svake dve centralne foneme može postojati grupa marginalnih fonema, pa se u svakoj višesložnoji reči mogu identifikovati segmenti oblika:

$$v_1 k^n v_2 \quad (n \geq 0) \ldots (x).$$

U odnosu na ovakav raspored centralnih i marginalnih fonema se odredjuju pravila o granici sloga.

Izlažemo dalje pravila koja su zajednička svim u Uvodu pomenutim skupovima pravila:
(O_1) Ako reč počinje konsonantskom grupom, onda ta konsonantska grupa pripada slogu obrazovanom oko prve centralne foneme.
(O_2) Ako u (x) n=0 ili n=1, onda je granica sloga iza v_1:

$$v_1 - k^n v_2$$

Radi jednostavnijeg izlaganja pravila o rastavljanju konsonantskih grupa označimo
skup svih konsonanata sa C
skup strujnih dentala{s,š,z,ž} sa S
skup praskavih {b,d,g,p,t,k} sa P
skup pravih sonanata {v,j,l,lj,r}sa R

Tada, prema pravilima koja potiču iz (Belić/34/) za dvočlanu konsonantsku grupu k_1k_2 važi:
(B_1) Ako i) $k_1 \varepsilon S$ i $k_2 \varepsilon C$, ili
ii) $k_1 \varepsilon C-R$ i $k_2 \varepsilon R$, onda
se grupa k_1k_2 ne rastavlja, tj. $v_1 - k_1k_2 v_2$

(B_2) U svim ostalim slučajevima grupa k_1k_2 se rastavlja, tj. $v_1k_1 - k_2v_2$

(B_3) Za $n \geq 3$ u (x), konsonantska grupa se rastavlja prema (B_1) i (B_2).

Prema pravilima koja potiču iz (Stevanović /64/), za dvočlanu konsonantsku grupu k_1k_2 važi:
(S_1) Ako i) $k_1 \varepsilon P$ i $k_2 \varepsilon C-R$, ili
ii) $k_1 \varepsilon R$ i $k_2 \varepsilon C$, onda se grupa k_1k_2 rastavlja, tj. $v_1k_1 - k_2v_2$

(S_2) U svim ostalim slučajevima grupa k_1k_2 se ne rastavlja, tj. $v_1 - k_1k_2v_2$.

(S_3) Za $n \geq 3$ u (x) konsonantske grupe se rastavljaju prema (S_1) i (S_2).
Pravopisom su dopuštene veće slobode u rastavljanju reči na kraju retka no što ih daju izložena pravila, ali kao što je ranije rečeno, pojmovi "lako" i "teško" izgovorljivih konsonantskih grupa su neodredjeni, te se stoga ne mogu formalno opisati.
Primeri reči rastavljenih na slogove prema pravilima (B) i pravilima (S) su dati na slici 1. Ovi primeri su pravopisni primeri "teško" izgovorljivih konsonantskih grupa. Kako se takve grupe rastavljaju bar jednim od skupova pravila (B) ili (S), može se kao heuristički savet usvojiti da se pri rastavljanju prednost daje onom skupu pravila koji rastavlja konsonantsku grupu.
Na slici 2 su prikazani primeri rastavljanja "lako" izgovorljivih konsonantskih grupa.

Na osnovu opisanih pravila, konstruisan je program koji rastavlja reči na slogove. Osnovni koraci u programu su:
1. identifikacija prefiksa (videti 3(b));
2. identifikacija prve centralne foneme sleva na desno;
3. traži se sledeća centralna fonema;
4. ako ne postoji, stop;
5. ako je pronadjena, ispituje se priroda skupa marginalnih fonema i postavlja se granica sloga. Dalje ispitivanje se nastavlja od poslednje identifikovane centralne foneme. Ide se na korak 3.

3. OGRANIČENJA

Navodimo dalje slučajeve u kojima dolazi do odstupanja od izloženih pravila:
(a) U latiničnom zapisu reči, slovne oznake lj, nj, dj, dž stvaraju neotklonjiv ambigvitet jer se mogu interpretirati i kao oznake glasova i

kao oznake grupe glasova (npr. dj u odjednom i rodjendan). Ako su oznakama lj, nj, dj, dž pretstavljena dva glasa, onda se, prema pravopisnim pravilima, ovi glasovi razdvajaju (tj.od-jednom ali ro-djendan). Prema tablicama bigrama u srpskohrvatskom jeziku (Tomić/78/), frekvencija pojavljivanja pomenutih oznaka kao oznaka za jedan glas iznosi 1,6% ukupnog broja pojavljivanja slova u tekstu a kao oznaka za dva glasa svega 0.05%. Otuda sledi da se redje greši ako oznake dj, lj, nj, dž tretiramo kao oznake za jedan glas.

(b) Prefiks se, prema pravopisnom pravilu, izdvaja u samostalni slog. Sa stanovišta automatskog rastavljanja reči na slogove, problem pričinjavaju oni prefiksi koji se završavaju na konsonant (npr. raz-, od-, ob-, itd.). Ovde se, takodje, radi o neotklonjivom ambigvitetu (npr. imamo raz-vući ali ra-zum). Identifikaciju prefiksa je moguće izvršiti konsultovanjem tablice prefiksa ali to ne daje uvek korektno rešenje. Primeri identifikovanih prefiksa su dati na slici 3. Slični se problemi javljaju i sa sufiksima (npr. konsonant-ski umesto konsonan-tski). Takodje, izvestan broj reči se rastavlja prema semantičkoj podeli (npr. ga-vran umesto gav-ran). Potpuno rešavanje ovog problema bi se sastojalao u ugradjivanju u program osim rečnika prefiksa i sufiksa, i rečnika reči koje se dele prema semantičkoj podeli a koje imaju značajniju frekvenciju u jeziku.

(c) Transkribovane strane reči se rastavljaju prema pravilima iznesenim u tački 2. Primeri su dati na slici 4. Napomenimo da pri izvornom zapisivanju stranih reči, prema Pravopisu, treba poštovati pravila izvornog jezika.

Navedimo na kraju i dva pravopisna ograničenja koja se odnose pre na rastavljanje reči na kraju retka nego na prirodu sloga:

(d) Ako u reči poslednji slog tvori samo jedan vokal ili samo konsonantska grupa, prenošenje se ne vrši.

(e) Pri rastavljanju polusloženica na rastavnom znaku (npr. šar-planina) ovaj znak ima vrednost slova i prenosi se u sledeći red. Dakle, šar--planina.

LITERATURA

Belić,A./34/: Pravopis srpskohrvatskog književnog jezika, Beograd, 1934, 15-18

Pravopis /60/: Pravopis srpskohrvatskoga književnog jezika, Novi Sad-Zagreb, 1960, 130-131

Stevanović,M./64/: Savremeni srpskohrvatski jezik, Beograd, 1964, 152-156

Kokoviċ,M./79/: Slogovna struktura srpskohrvatskog jezika, Seminar za matematičku lingvistiku, Matematički institut, Beograd, 1979.

Tomić, T./78/: Statistička analiza srpskohrvatskog teksta pomoću računara, in: Kompjuterska obrada lingvističkih podataka (ed.M. Šipka), Institut za jezik i književnost u Sarajevu, Sarajevo, 1978, 221-235

```
ZAMKA
BELIC.....ZAM=KA
STEV......ZA=MKA

BORBA
BELIC.....BOR=BA
STEV......BOR=BA

BUMBAR
BELIC.....BUM=BAR
STEV......BU=MBAR

SUNCE
BELIC.....SUN=CE
STEV......SU=NCE

BRATSTVO
BELIC.....BRAT=STVO
STEV......BRAT=STVO

SXKOLSKI
BELIC.....SXKOL=SKI
STEV......SXKOL=SKI

SREDSTVO
BELIC.....SRED=STVO
STEV......SRED=STVO

SUDSKI
BELIC.....SUD=SKI
STEV......SUD=SKI
```

Sl.1. "Teško" izgovorljive konsonantske grupe

```
SEDLO
BELIC.....SE=DLO
STEV......SE=DLO

VEZXBATI
BELIC.....VE=ZXBA=TI
STEV......VE=ZXBA=TI

MISXJI
BELIC.....MI=SXJI
STEV......MI=SXJI

SESTRA
BELIC.....SE=STRA
STEV......SE=STRA

DRUSXTVEN
BELIC.....DRU=SXTVEN
STEV......DRU=SXTVEN

ZEMLXA
BELIC.....ZE=MLXA
STEV......ZE=MLXA

SLAVLXE
BELIC.....SLAV=LXE
STEV......SLAV=LXE
```

Sl.2. "Lako" izgovorljive konsonantske grupe

```
RAZVUCXI
PREFIKS...RAZ
BELIC.....VU-CXI
PREFIKS...RAZ
STEV......VU-CXI

RAZORUZXAN
PREFIKS...RAZ
BELIC.....O-RU-ZXAN
PREFIKS...RAZ
STEV......O-RU-ZXAN

OTPREMITI
PREFIKS...OT
BELIC.....PRE-MI-TI
PREFIKS...OT
STEV......PRE-MI-TI
```

Sl.3. Prepoznavanje prefiksa

```
FRESKO
BELIC.....FRE-SKO
STEV......FRE-SKO

KULOARI
BELIC.....KU-LO A-RI
STEV......KU-LO-A-RI

KOMBAJN
BELIC.....KOM-BAJN
STEV......KO-MBAJN

PARDON
BELIC.....PAR-DON
STEV......PAR-DON

BORDO
BELIC.....BOR-DO
STEV......BOR-DO
```

Sl.4. Rastavljanje stranih reči

JEZIK PL/I IN MIKRORAČUNALNIKI IV

ANTON P. ŽELEZNIKAR

SOZD ELEKTROTEHNA, DO DELTA

UDK: 681.3.06 PL1:181.4

Četrti del članka o uporabi jezika PL/I-80 na mikraračunalniškem sistemu (kot je npr. Delta 323/MI) prinaša uporabo tega jezika v komercialnih obdelavah. Pri tem se najprej raziskujejo decimalne aritmetične operacije in njihov pomen pri komercialnih podatkovnih obdelavah. Dan je pregled pretvorbe med podatkovnima tipoma decimalno s trdno vejico in binarno s plavajočo vejico. Pri izpisih se izdatno uporablja tudi PICTURE format. Članek obravnava v okviru te tematike tri praktične primere, in sicer načrt odplačevanja posojila, navadno amortizacijo in podrobni ter formatirani načrt odplačevanja posojila. V sklepnem delu so nakazane prednosti programiranja v jeziku PL/I pa tudi zahtevnost, ki je prisotna pri takem programiranju. Dani so podatki o učinkovitosti s prevajalniki generiranih kodov za različne mikroračunalniške prevajalnike.

PL/I Language and Microcomputers IV. The fourth part on PL/I usage with microcomputers presents its application in commercial data processing. First, the decimal arithmetic operations and their importance in commercial data processing are examined. An overwiev is given dealing with data types in fixed decimal and floating binary. PICTURE format is applied for several data presentations. Further, the article shows three examples: a simple loan payment schedule, ordinary annuity and formatted loan payment schedule. In the last part of the article some advantages of PL/I programming are pointed out and benchmarks for several compilers are listed.

14. Uporaba jezika PL/I v komercialnih obdelavah

Doslej smo spoznali že vrsto primerov, vendar nismo zapisali ničesar tistega, kar je bistveno za komercialne obdelave. Najprej poudarimo, da moramo spoznati podrobno decimalne aritmetične operacije. Pregledali bomo tudi pretvorbo med podatkovnima tipoma decimalno s trdno vejico in binarno s plavajočo vejico. Pri tem bomo upoštevali uporabo knjižnične funkcije FTC (plavajoče v znakovno). Imeli bomo primere z že obravnavanim PICTURE formatom v okviru decimalne predstavitve operandov.

14.1. Primerjava decimalnih in binarnih operacij

PL/I je jezik, ki omogoča izbiro v predstavitvi operandov glede na potrebe neke specifične aplikacije. COBOL, ki je jezik za komercialne obdelave, uporablja decimalno aritmetiko. Tudi jeziki tipa BASIC, ki so predvideni za komercialne obdelave, morajo imeti decimalno aritmetiko. FORTRAN uporablja vselej binarno aritmetiko s trdno ali plavajočo vejico in podobno velja tudi za PASCAL, ki ima navadno možnost večje natančnosti.

Podatki s trdno decimalno vejico so osnovni pogoj za komercialne obdelave. Vzemimo tale dva programa:

```
dec_primerjava:
   proc options(main);
   dcl
     i fixed,
     t decimal(7,2);
   t = 0;
     do i = 1 to 10000;
        t = t + 3.10;
     end;
   put edit(t) (f(10,2));
end dec_primerjava;
```

in

```
bin_primerjava:
   proc options(main);
   dcl
     i fixed,
     t float(24);
   t = 0;
     do i = 1 to 10000;
        t = t + 3.10;
     end;
   put edit(t) (f(10,2));
end bin primerjava;
```

Oba programa prištevata vrednost 3.10 desettisočkrat. Razlika med njima je v tem, da dec_primerjava izračuna pravilno vrednost 31000, bin_primerjava pa izračuna vrednost 30997,30. Razlika je 2,70 zaradi binarne aproksimacije vrednosti 3,10, ko se napaka tega približka razširi desettisočkrat. Če predvidevamo, da bomo imeli pri komercialnih

obdelavah vselej pretvorbo ulomka 1/10 v vrednost 0,1, je pogoj za natančnost zagotovljen. Pri pretvorbi ulomka 1/3 v približek 0,33... pa bi se napaka približka tudi lahko razširila.

14.2. Decimalni izračuni

Uporaba tipa decimalno s trdno vejico ima prednosti in pomanjkljivosti v primerjavi s formatom pomične vejice. Decimalna aritmetika zagotavlja, da se ne bi pojavila izguba bistvenih številk rezultata. Zaokrožanja na najnižjem mestu ne bo, pojavila pa se bo potreba za obdelavo eksponenta, kar pa bo operacije le pohitrilo. Ker so vsa mesta podatka bistvena, mora programer upoštevati obseg aritmetičnih operandov. PL/I komercialni podatki imajo svojo natančnost in stopnjo. Natančnost je določena s številom decimalnih mest spremenljivke ali konstante, stopnja pa določa število mest ulomljenega dela. Natančnost tipa decimalno s trdno vejico je največ 15, stopnja pa ne sme preseči natančnosti. Tako imamo npr. določilo:

```
dcl x fixed decimal (10,3);
```

Konstanta -123.45 ima natančnost 5 in stopnjo 2. V računalniku se tip decimalno s trdno vejico shranjuje kot več parov elementov, kjer je element binarno kodirana decimalna (BKD) številka. V enem zlogu sta tako dve številki. Najvišje BKD mesto je prihranjeno za predznak, kjer pomeni 0 pozitivno in 9 negativno število. Število 83.62 se shrani kot 08 36 20, ko sta v zlogu po dve številki.

Negativno število je desetiški komplement (glede na aritmetične operacije). Vrednost -3 se shrani kot 97, ko imamo

```
99-3 ---> 96+1 ---> 97.
```

Pri seštevanju in odštevanju dveh števil, katerih (natančnost, stopnja) sta (p,q) in (r,s), se nižja stopnja poravna z višjo (npr. pri q 'večje' s). Pri množenju ni problema poravnave decimalne vejice, povečata pa se vobče natančnost in stopnja rezultata. Pri deljenju se lahko pojavijo še rezalne napake.

14.3. Konverzija tipov trdno decimalno v pomično binarno in obratno

Vzemimo dani program in si na njegovem primeru oglejmo konverzijo:

```
konverzija:
   proc options(main);
   dcl
      FTC entry(float)
          returns(char(17)var);
   dcl
      d fixed decimal(8,2),
      f float binary;
   d = -123456.78;
   f = char(d);
   f = 0.314159265 e1;
   d = FTC(f);
end konverzija;
```

V tem primeru se trdna decimalna vrednost d najprej inicializira na vrednost -123456.78. Nato se uporabi vgrajena funkcija char nad d in se oblikuje znakovna nizna konstanta 'b-123456.78', kjer je b presledek. Prireditveni stavek f = char(d) pretvori trdno decimalno v plavajočo binarno vrednost z možno rezalno napako. Nato shranimo v f vrednost števila pi kot plavajočo binarno vrednost. Prireditev iz f v d povzroči rezalno napako ulomljenega dela. FTC funkcija se uporabi nad f in tako dobimo znakovno nizno konstanto dolžine 17:

```
'b3.14159200000000'
```

kjer stoji b namesto znaka +, pri negativni vrednosti pa imamo na tem mestu minus (-). Prireditvena operacija d = FTC(f) povzroči v d vrednost 3.14, ker ima lahko d skladno z določilom le dve mesti za vejico. Velja si zapomniti, da dopušča pomična binarna predstavitev približno sedem in pol decimalnih številk, pri več številkah pa se pojavi ob konverziji rezalna napaka.

14.4 Primer načrta odplačevanja posojila

Oglejmo si primer komercialne obdelave podatkov na listi 23. Ta program natisne načrt odplačevanja posojila pri dani obrestni meri in mesečnih obrokih. Vhodne vrednosti (s konzole) so: posojilojemalec, posojilo (celotna posojilna vsota), obrestna mera, mesečni obrok ter tekoči mesec in leto. Za vsak mesec se potem izračuna ostanek posojila kot

```
posojilo = (posojilo + obrestna_mera *
             * posojilo) - obrok
```

V programu liste 23 imamo za ta izračun iteracijo med vrsticami 35 in 51, ko se vrednost posojila zniža na nič ter je posojilo v celoti odplačano.

V tem programu je posojilo trdna decimalna spremenljivka (z maksimalno vrednostjo 9999999,99). Vrednost obroka je omejena z 99999,99 , ko imamo določilo (natančnost, stopnja) = (7,2). Obrestna mera je določena s trdno decimalno (4,2), torej z najvišjo mero 99,99 % . Spremenljivki mesec in leto se nanašata na tekoči mesec in tekoče leto.

Na listi 24 imamo izvajanje programa z liste 23. Dodatna pojasnila niso več potrebna, omenimo le, da se posamezne konstante, spremenljivke in izrazi izračunavajo s tole natančnostjo in stopnjo:

```
obrestna_mera                (4,2)
posojilo                     (9,2)
obrestna_mera*posojilo (13,4) (trdno, dec)
1200                         (4,0)
(obrestna_mera*posojilo)/1200        (13,4),
  ker velja pri deljenju  (13,13-13+4-0)
```

Deljenje s 1200 nastane zaradi izražave obrestne mere v procentih (deljenje s 100) prek enoletnega razdobja (deljenje z 12). Vmesni rezultat se zaokroži (round) na drugem decimalnem mestu in prišteje k posojilu.

14.5. Primer navadne amortizacije

Oglejmo si program na listi 25, ki pri dani obrestni (anuitetni) meri izračunava trenutno vrednost (tren_vred) ali obrok ali število obrokov (stev_obr). Ta program prikazuje vrsto komercialnih izračunov z mešano aritmetiko (plavajoče in trdne decimalne vejice).

Amortizacijski program izračuna neznano vrednost z uporabo statičnih formul, torej ne z iteracijo. V teh formulah mora biti amortizacijska obrestna mera večja od nič. Tako imamo

```
tren_vred = obrok(1-(1/(1+i)exp n)/i
```

kjer je i amortizacijska obrestna mera in n število amortizacijskih obrokov. Nadalje imamo

```
obrok = tren_vred(i/(1-(1/(1+i)exp n)))
```

Lista 24. Lista na desni strani kaže izvajanje programa OBROKI (zbirka OBROKI.COM), ko se najprej vstavijo vhodni podatki, za vstavitvijo pa se avtomatično začne izračun, ki upošteva v celoti izplačani znesek posojilojemalca in obresti na prvotno posojilo (totalno obrestno mero, ki se razlikuje od bančne obrestne mere). Inflacija pri odplačevanju ni upoštevana.

```
A>type b:oboki.prn

PL/I-80 V1.3  COMPILATION OF: OBROKI

D: Disk Print
L: List Source Program

   NO ERROR(S) IN PASS 1

   NO ERROR(S) IN PASS 2

PL/I-80 V1.3  COMPILATION OF: OBROKI

   1 a 0000 obroki:
   2 a 0006   proc options (main);
   3 c 0006   dcl
   4 c 0006     mesec fixed binary,
   5 c 0006     leto fixed binary,
   6 c 0006     stev_obr fixed binary,
   7 c 0006     posojilo fixed dec (9,2),
   8 c 0006     banka fixed dec(9,2),
   9 c 0006     pos fixed dec(9,2),
  10 c 0006     obrok fixed dec (7,2),
  11 c 0006     obrestna_mera fixed dec (4,2),
  12 c 0006     bancna_mera fixed dec (5,2),
  13 c 0006     posojilo_jemalec char (25) var;
  14 c 0006
  15 c 0006   do while ('1'b);
  16 c 0006     put skip list ('Posojilojemalec: ');
  17 c 0022     get list (posojilo_jemalec);
  18 c 003C     put skip list ('Posojilo: Din ');
  19 c 0058     get list (posojilo);
  20 c 0077     put skip list ('Obrestna mera (%): ');
  21 c 0093     get list (obrestna_mera);
  22 c 00B2     put skip list ('Mesecni obrok: Din ');
  23 c 00CE     get list (obrok);
  24 c 00ED     put skip list ('Tekoci mesec (stevilka), leto: ');
  25 c 0109     get list (mesec, leto);
  26 c 012D     put skip(2) edit ('Leto', 'Mesec', 'Posojilo',
                  'Obrok', 'Stevilka obroka')
                  (a,col(7),a,col(15),a,col(27),a,col(35),a);
  27 c 017B     put skip list (
                '-------------------------------------------------------------');
  28 c 017B     put skip edit (leto)(col(1),f(4));
  29 c 0197     pos = posojilo;
  30 c 01B9     stev_obr = 0;
  31 c 01C9     banka = 0;
  32 c 01CF     do while (posojilo > 0);
  33 c 01DF     if mod(mesec,12)=0 then
  34 c 01F5       do;
  35 c 0208       mesec = 0;
  36 c 0208       leto = leto + 1;
  37 c 020E       put skip edit (leto)(col(1),f(4));
  38 c 0215       end;
  39 c 0215     mesec = mesec + 1;
  40 c 0237     stev_obr = stev_obr +1;
  41 c 0237     put skip list (mesec, posojilo);
  42 c 023E     posojilo=posojilo+round(obrestna_mera*posojilo/1200,2);
  43 c 0245     if posojilo < obrok then
  44 c 0273       obrok = posojilo;
  45 c 027B     put list (obrok,stev_obr);
  46 c 02B8     posojilo = posojilo - obrok;
  47 c 02CE     banka = banka + obrok;
  48 c 02DE     end;
  49 c 0307   put skip(2) edit(
  50 c 0322     'Banki bom izplacal v celoti Din ',banka,',')
  51 c 0340     (a(31),f(15,2),a(1));
  52 c 0340   bancna_mera=(divide(banka,pos,7,4)-1)*100;
  53 c 037F   put skip edit(
  54 c 037F     'kar znese ',bancna_mera,' % na prvotno posojilo.')
  55 c 037F     (a(10),f(5,2),a(23));
  56 c 03BF   put skip(2) edit ('Posojilojemalec:')(col(23),a(16));
  57 c 03FE   put skip(3) edit ('.....')
  58 c 03FE     (col(23),a(27));
  59 c 03FE   put skip edit ('(',posojilo_jemalec,')')(col(23),
  60 c 0420     a,a,a);
  61 c 0442   end;
  62 c 0442
  63 c 047B
  64 c 047B
  65 a 047B end obroki;

CODE SIZE = 047B
DATA AREA = 01C2
FREE SYMS = 3E4F
END  COMPILATION
```

Lista 23. Lista zbirke OBROKI prikazuje PLI program, katerega učinek je prikazan na listi 24. Ta program izračunava (ko je preveden v zbirko OBROKI.COM) preprost načrt odplačevanja posojila z mesečnimi obroki. Program je nekoliko bolj podrobno opisan v tekstu (podpoglavje 14.4).

```
B:OBROKI

Posojilojemalec: 'ANTON Pavel Zeleznikar

Posojilo: Din 50000

Obrestna mera (%): 32.5

Mesecni obrok: Din 4000

Tekoci mesec (stevilka), leto: 9,1981

Leto  Mesec    Posojilo     Obrok     Stevilka obroka
-------------------------------------------------------
1981
        10     50000.00    4000.00          1
        11     47354.17    4000.00          2
        12     44636.68    4000.00          3
1982
         1     41845.59    4000.00          4
         2     38978.91    4000.00          5
         3     36034.59    4000.00          6
         4     33010.53    4000.00          7
         5     29904.57    4000.00          8
         6     26714.49    4000.00          9
         7     23438.01    4000.00         10
         8     20072.79    4000.00         11
         9     16616.43    4000.00         12
        10     13066.46    4000.00         13
        11      9420.34    4000.00         14
        12      5675.47    4000.00         15
1983
         1      1829.18    1878.72         16

Banki bom izplacal v celoti Din      61878.72,
kar znese 23.75 % na prvotno posojilo.

                        Posojilojemalec:


                        .........................
                        (ANTON Pavel Zeleznikar)

Posojilojemalec: ^C
A>
```

```
A>type b:amortiza.prn

PL/I-80 V1.3  COMPILATION OF: AMORTIZA

D: Disk Print
L: List Source Program

   NO ERROR(S) IN PASS 1

   NO ERROR(S) IN PASS 2

PL/I-80 V1.3  COMPILATION OF: AMORTIZA

 1 a 0000 amortizacija:
 2 a 0006    proc options(main);
 3 c 0006    %replace
 4 c 000D       brisi by '''z',
 5 c 000D       pravilno  by '1'b;
 6 c 000D    dcl
 7 c 000D       obrok fixed decimal(7,2),
 8 c 000D       tren_vred  fixed decimal(9,2),
 9 c 000D       IP  fixed decimal(6,6),
10 c 000D       x   float binary,
11 c 000D       yi  float binary,
12 c 000D       i   float binary,
13 c 000D       stev_obr  fixed;
14 c 000D    dcl
15 c 000D       ftc entry (float binary) returns (char(17) var);
16 c 000D
17 c 000D    put list (brisi,
18 c 002F      '''iN A V A D N A   A M O R T I Z A C I J A');
19 c 002F    put skip (2) list
20 c 0048      ('''iVstavi vrednosti ali 0 za vsako iteracijo');
21 c 0048
22 c 0048    on error
23 d 0052       begin;
24 e 0055       put skip list('''iNeveljaven podatek, ponovi');
25 e 0071       go to ponovi;
26 d 0074       end;
27 d 0074
28 c 0074    ponovi:
29 c 0078       do while (pravilno);
30 c 0078       put skip(3) list
31 c 0097          ('''iTrenutna vrednost ');
32 c 0097       get list(tren_vred);
33 c 00B6       put list('''iAmortizacijski obrok      ');
34 c 00CD       get list(obrok);
35 c 00EC       put list('''iAmortizacijska obrestna mera ');
36 c 0103       get list(yi);
37 c 011E       i = yi / 1200;
38 c 012F       put list('''iStevilo amortizacijskih obrokov   ');
39 c 0146       get list(stev_obr);
40 c 015E
41 c 015E       if tren_vred = 0 | obrok = 0 then
42 c 0190          x = 1 - 1/((1+i)**stev_obr;
43 c 01B3       if tren_vred = 0 then
44 c 01B5          do;
45 c 01C9       /* izracunaj trenutno vrednost */
46 c 01C9       tren_vred = obrok * dec(ftc(x/i),15,6);
47 c 01FD       put edit('''iTrenutna vrednost znasa ',
48 c 0237          tren_vred,'  Din')
49 c 0237             (a,p'sssbsssbss9v,99',a);
50 c 0237       end;
51 c 0237
52 c 0237    if obrok = 0 then
53 c 0237       do;
54 c 024D       /* izracunaj amortizacijski obrok */
55 c 024D       obrok = tren_vred * dec(ftc(i/x),15,8);
56 c 024D       put edit('''iAmortizacijski obrok znasa ',
57 c 0281          obrok,'  Din')
58 c 02B8             (a,p'sssbsssbss9v,99',a);
59 c 02B8       end;
60 c 02B8
61 c 02B8    if stev_obr = 0 then
62 c 02B8       do;
63 c 02C4       /* izracunaj stevilo amortizacijskih obrokov */
64 c 02C4       IP = ftc(i);
65 c 02C4       x = char(tren_vred * IP / obrok);
66 c 02D7       stev_obr = ceil ( - log(1-x)/log(1+i) );
67 c 0305       put edit('''i',stev_obr,
68 c 0342          ' amortizacijskih obrokov')
69 c 0378             (a,p'ZZZ9',a);
70 c 0378       end;
71 c 0378    end;
72 c 0378 end amortizacija;
73 a 0378
```

Lista 25. Program na listi spodaj izračunava navadno amortizacijo ter uporablja pri tem direktne formule za izračun vhodnega parametra pri danih treh vhodnih parametrih (trenutna vrednost, amortizacijski obrok, obrestna mera in število obrokov). Program je preprost in je podrobneje opisan v tekstu (podpoglavje 14.5).

```
A>b:amortiza
        N A V A D N A   A M O R T I Z A C I J A

        Vstavi vrednosti ali 0 za vsako iteracijo


        Trenutna vrednost 50000
        Amortizacijski obrok        5000
        Amortizacijska obrestna mera 32.5
        Stevilo amortizacijskih obrokov   0
          12 amortizacijskih obrokov


        Trenutna vrednost 0
        Amortizacijski obrok        3000
        Amortizacijska obrestna mera 12.5
        Stevilo amortizacijskih obrokov   36
        Trenutna vrednost znasa      +89 675,70  Din


        Trenutna vrednost 50000
        Amortizacijski obrok        0
        Amortizacijska obrestna mera 12.5
        Stevilo amortizacijskih obrokov   36
        Amortizacijski obrok znasa     +1 672,69  Din


        Trenutna vrednost ^C
A>
```

Lista 26. S pozivom AMORTIZA se začne izvajati program (zbirka AMORTIZA.COM), ki je v obliki PLI programa prikazan z listo 25. S tem programom lahko izračunamo enega od vhodnih parametrov pri danih treh vhodnih parametrih. Izjema je le amortizacijska obrestna mera, ki je ni moč izračunavati s tem programom. Za vrednost, ki jo želimo izračunati, vstavimo ničlo, nakar se pri danih ostalih podatkih ta vrednost izračuna vselej v peti vrstici. Vejico lahko vstavimo, če je vrednost glede na prejšnji izračun ostala nespremenjena. Lista tudi kaže, kako iz programa izstopimo.

in končno še

```
n = -(log(1-tren_vred(i/obrok)))/log(1+i)
```

Program ima eno glavno zanko (lista 25, vrstica 29 do 72), v kateri se tren_vred, amortizacijski obrok in letna amortizacijska obrestna mera berejo s konzole. Uporabnik mora vstaviti tri neničte vrednosti in eno ničto vrednost pri vsaki iteraciji. Program izračuna vrednost, ki je bila vstavljena kot ničta vrednost. Vrednosti se ohranijo do naslednje iteracije in vejico (',') uporabimo, če vrednosti glede na prejšnjo iteracijo nismo spremenili.

Vgrajena funkcija CHAR(x) povzroči konverzijo vrednosti x v znakovni (številčni) niz. Vgrajena aritmetična funkcija CEIL(x) vrne najmanjše celo število, ki je večje od x ali enako x.

Glede na prejšnje formule lahko ugotovimo, da imamo izračun izraza

```
1-1/(1+i)**stev_obr
```

tako pri izračunu tren_vred kot obroka. Vrstica 42 izračuna ta izraz in ga shrani kot vrednost x. Pri tem vemo, da je x le približek decimalne vrednosti, določene s tem izrazom. Če vstavi uporabnik ničlo za tren_vred, se izvršijo stavki med vrsticami 46 in 50. Pri tem se računa tren_vred z uporabo zunanje subrutine "ftc", kot je določeno z vrstico 15. Pri tem imamo

```
tren_vred = obrok*dec(ftc(x/i),15,6)
```

(v vrstici 47), kjer je x/i plavajoče, binarno, funkcija ftc pa to preslika iz plavajoče v znakovno obliko. Če je npr. x/i enako vrednosti 1.234567E+01, se z ftc ta vrednost preslika v 12.34567, ki je za preslikavo v decimalno sprejemljiva. Pogoj ERROR(1) se sproži z ftc in označuje konverzijsko napako, ko argumenta s plavajočo vejico ni moč pretvoriti v 15-mestno decimalno število. Funkcija "dec" (glej vrstico 47) se uporabi nad znakovnim nizom za konverzijo z določeno natančnostjo (15) in ulomljenim območjem (6), nakar sledi multiplikacija. Kako smo se odločili za ti posebni vrednosti natančnosti in območja?

Vzemimo preprostejši primer

```
dcl
   obrok fixed decimal(7,2),
   tren_vred fixed decimal(9,2),
   Q fixed decimal(u,v);
tren_vred = obrok*Q;
```

kjer moramo določiti konstantni vrednosti u in v. Spremenljivka tren_vred ima natančnost in območje (9,2) in ima tako sedem mest v celem in dve mesti v ulomljenem delu. Sedem mest v celem delu bo generiranih tedaj, kadar bo imel produkt obrok*Q natančnost in območje

```
(9,2), (10,3), (11,4), (12,5), (13,6),
   (14,7)  ali  (15,8)
```

in bo prireditev k tren_vred odrezala vse ulomkovne številke za drugim decimalnim mestom. Ker ima obrok natančnost in območje (7,2), lahko izberemo (15,6) kot natančnost in območje za Q, tako da dobimo

```
(min(15,7 + 15 + 1),2 + 6) = (15,8)
```

kot posledico pravil, ki veljajo za multiplikacijo. V primeru

```
   a     =     b     *     c
(p,q)       (r,s)       (u,v)
```

lahko velja za c

```
u = 15,  v = 15-p+q-s
```

kar povzroči v našem primeru

```
v = 15-9+2-2 = 8,  torej (u,v) = (15,6)
```

kot (u,v) za Q.

Program v listi 25 uporablja za izpis tren_vred PICTURE format v vrstici 50.

Kadar je uporabnik vstavil neničto vrednost za tren_vred in ničto vrednost za obrok se začne izračun obroka z vrstico 56, ko imamo

```
obrok = tren_vred*dec(ftc(i/x),15,8);
```

Tu veljajo za izračune različnih vrednosti in za prireditev podobne opombe kot smo jih imeli za izračun tren_vred.

15. Primer formatiranja načrta odplačevanja posojila

Program v listi 27 je podoben programu z liste 23, ima pa razširjeno analizo in prikazovalni format. Ta program včita več podatkovnih enot, in sicer:

PV	trenutna vrednost (tudi začetna)
yi	letna obrestna mera
PMV	mesečni obrok
ir	letna inflacijska mera
sm	začetni mesec odplačevanja (1 - 12)
sy	začetno leto odplačevanja (0 - 99)
fm	obračunski (fiskalni) mesec, konec fiskalnega leta (1 - 12)
dl	prikazovalni format (ravnina) (0 - 2)

Začetna (trenutna) vrednost in mesečni obrok sta spremenljivki tipa trdno, decimalno (10,2) in dovoljujeta vrednost do 99999999,99 din (vrstici 16 in 19 liste 27). Letna obrestna mera in inflacijska mera sta izraženi procentualno z najvišjo vrednostjo 99,99, kot je določeno z vrsticama 24 in 29 liste 27. Mesečni spremenljivki sm in fm ter letna spremenljivka sy imajo trdni, binarni format. Spremenljivka dl določa količino prikazanih podatkov v vsaki posebni ponovitvi programa, ko njena vrednost 0 povzroči okrajšani prikaz, vrednost 1 dodatne podatke, 2 pa podrobno zasledovanje poteka izračuna.

Program ima glavno zanko med vrsticama 97 in 126-, ko se začetna vrednost povečuje za mesečne obresti in zmanjšuje z mesečnim obrokom, dokler ne doseže vrednosti nič. Več primerov izvajanja programa je prikazanih v listi 28. Prvi primer te liste kaže minimalni prikaz, ko imamo posojilo 50000 din, pri 14 %-nih obrestih, z obrokom 3000 din in inflacijsko mero 0 %. Odplačevanje se prične v oktobru 1981, fiskalni mesec pa je februar. Za ta primer je prikazan datum letnega obračuna z vrednostjo, obrestmi v februarju, obrokom, plačano (odplačano) vrednostjo in višino obresti, plačanih do zadnjega meseca obračunskega leta.

Drugi primer liste 28 kaže izvršitev programa za enake vrednosti kot v prvem primeru, vendar z uporabo prikazovalnega formata (ravnine) 1. Tudi v tem primeru imamo podatke iz prvega primera, izpostavljene pa so letne obresti.

```
PL/I-80 V1.3  COMPILATION OF: POSOJILO

  1 a 0000 pmt:
  2 a 0006     proc options(main);
  3 c 0006     %replace
  4 c 0006         true   by '1'b,
  5 c 0006         false  by '0'b,
  6 c 0006         clear  by '^z';
  7 c 0006     dcl
  8 c 0006         end bit(1),
  9 c 0006         m   fixed binary,
 10 c 0006         sm  fixed binary,
 11 c 0006         y   fixed binary,
 12 c 0006         sy  fixed binary,
 13 c 0006         fm  fixed binary,
 14 c 0006         dl  fixed binary,
 15 c 0006         P   fixed decimal(10,2),
 16 c 0006         PV  fixed decimal(10,2),
 17 c 0006         PP  fixed decimal(10,2),
 18 c 0006         PL  fixed decimal(10,2),
 19 c 0006         PMT fixed decimal(10,2),
 20 c 0006         PMV fixed decimal(10,2),
 21 c 0006         INT fixed decimal(10,2),
 22 c 0006         YIN fixed decimal(10,2),
 23 c 0006         IP  fixed decimal(10,2),
 24 c 0006         yi  fixed decimal(4,2),
 25 c 0006         i   fixed decimal(4,2),
 26 c 0006         INF fixed decimal(4,3),
 27 c 0006         ci  fixed decimal(15,14),
 28 c 0006         fi  fixed decimal(7,5),
 29 c 0006         ir  fixed decimal(4,2);
 30 c 0006
 31 c 0006     dcl
 32 c 0006         name char(14) var static init('$con'),
 33 c 0006         output file;
 34 c 0006
 35 c 0006     put list(clear,
 36 c 002F    '^iP R E G L E D   O D P L A C E V A N J A   P O S O J I L A');
 37 c 002F
 38 c 002F     on undefinedfile(output)
 39 d 0037         begin;
 40 e 003A         put skip list('^i^ise ne more vpisati v',name);
 41 e 005F         go to open_output;
 42 d 0062         end;
 43 d 0062
 44 c 0062     open_output:
 45 c 0069         put skip(2) list('^i^iIme izhodne zbirke ');
 46 c 0085         get list(name);
 47 c 009F
 48 c 009F     if name = '$con' then
 49 c 00AD         open file(output) title('$con') print pagesize(0);
 50 c 00CC     else
 51 c 00CC         open file(output) title(name) print;
 52 c 00E6
 53 c 00E6     on error
 54 d 00ED         begin;
 55 e 00FD         put skip list('^i^iSlabi vhodni podatki, ponovi');
 56 e 010C         go to retry;
 57 d 010F         end;
 58 d 010F
 59 c 010F     retry:
 60 c 0116         do while(true);
 61 c 0116         put skip(2)
 62 c 0132             list('^i^iVrednost        ');
 63 c 0132         get list(PV);
 64 c 0151         P = PV;
 65 c 0161         put list('^i^iObresti         ');
 66 c 0178         get list(yi);
 67 c 0197         i = yi;
 68 c 01A7         put list('^i^iObrok           ');
 69 c 01BE         get list(PMV);
 70 c 01DD         PMT = PMV;
 71 c 01ED         put list('^i^iInflacija (%)   ');
 72 c 0204         get list(ir);
 73 c 0223         fi = 1 + ir/1200;
 74 c 0253         ci = 1.00;
 75 c 0263         put list('^i^iZacetni mesec   ');
 76 c 027A         get list(sm);
 77 c 029Z         put list('^i^iZacetno leto    ');
 78 c 02A9         get list(sy);
 79 c 02C1         put list('^i^iFiskalni mesec ');
 80 c 02D8         get list(fm);
 81 c 02F0         put edit('^i^iPrikazovalna ravnina  ',
 82 c 032E              '^i^iLetni rezultati : 0 ',
 83 c 032E              '^i^iLetne obresti   : 1 ',
 84 c 032E              '^i^iVse vrednosti   : 2 ')
 85 c 032E             (skip,a);
 86 c 032E         get list(dl);
 87 c 0346         if dl < 0 | dl > 2 then
 88 c 0357             signal error;
 89 c 035E         m = sm;
 90 c 0364         y = sy;
 91 c 036A         IP = 0;
 92 c 037A         PP = 0;
 93 c 038A         YIN = 0;
 94 c 039A         if name ^= '' then
 95 c 03A8             put file(output) page;
 96 c 03BA         call header();
 97 c 03BD             do while (P > 0);
 98 c 03D3             end = false;
 99 c 03D8             INT = round ( i * P / 1200, 2 );
100 c 0408             IP  = IP + INT;
101 c 0423             PL  = P;
102 c 0433             P   = P  + INT;
103 c 044E             if P < PMT then
104 c 0464                 PMT = P;
105 c 0474             P   = P - PMT;
106 c 048F             PP  = PP + (PL - P);
107 c 04B5             INF = ci;
108 c 04CA             ci = ci / fi;
109 c 04EA             if P = 0 | dl > 1 | m = fm then
110 c 0520                 do;
111 c 0520                 put file(output) skip
112 c 0558                     edit('|',100*m+y) (a,p'99/99');
113 c 0558                 call display(PL * INF, INT * INF,
114 c 0601                 PMT * INF, PP * INF, IP * INF);
115 c 0601                 end;
116 c 0601             if m = fm & dl > 0 then
117 c 061E                 call summary();
```

Lista 27. Program na tej listi je zanimiv zaradi tega, ker daje možnost upoštevanja inflacije pri odplačevanju posojila. Hkrati nudi tudi možnost izbire različnih prikazovalnih formatov, od globalnega do podrobnega (načrt mesečnih obrokov). Ta program omogoča tudi listanje na različne periferne naprave (konzola, tiskalnik), pri čemer lahko z enostavno modifikacijo deklaracije v vrstici 6 dosežemo brisanje zaslona za poljuben terminal. Lista se nadaljuje še na naslednji strani. K razumevanju delovanja tega programa nam pomaga lista 28, kjer je prikazano izvajanje zbirke POSOJILO.COM in tam so razvidni tudi vhodni podatki in nastavitev prikazovalnega formata.

Lista 27 (nadaljevanje). Ta del programa POSOJILO kaže med drugim (na svojem koncu) tudi vrsto subrutin za tiskanje posameznih delov tabel v listi 28 (prikazovanje, pregled, glava tabele, vodoravna črte tabele itn.) Za izpis različnih podatkov se uporabljajo stavki PICTURE formata.

```
118 c 0621            m = m + 1;
119 c 0628            if m > 12 then
120 c 0634               do;
121 c 0634               m = 1;
122 c 063A               y = y + 1;
123 c 0641               if y > 99 then
124 c 064D                  y = 0;
125 c 0656               end;
126 c 0656            end;
127 c 0656         if dl = 0 then
128 c 065F            call line();
129 c 0665         else
130 c 0665         if ^end then
131 c 066C            call summary();
132 c 0672         end;
133 c 0672
134 c 0672      display:
135 c 0672         proc(a,b,c,d,e);
136 e 0672         dcl
137 e 067F            (a,b,c,d,e) fixed decimal(10,2);
138 e 067F         put file (output) edit
139 e 0731            ('|',a,'|',b,'|',c,'|',d,'|',e,'|')
140 e 0731            (a,2(2(p'zzbzzzbzz9v,99',a),
141 e 0731                   p'zzzbzz9v,99',a));
142 c 0731         end display;
143 c 0731
144 c 0731      summary:
145 c 0731         proc;
146 e 0731         end = true;
147 e 0736         call current_year(IP-YIN);
148 e 0757         YIN = IP;
149 c 0768         end summary;
150 c 0768
151 c 0768      current_year:
152 c 0768         proc(I);
153 e 0768         dcl
154 e 076F            yp fixed binary,
155 e 076F            I fixed decimal(10,2);
156 e 076F         yp = y;
157 e 0775         if fm < 12 then
158 e 0781            yp = yp - 1;
159 e 0788         call line();
160 e 078B         put skip file(output) edit
161 e 0804         ('|','Obresti placane v letu ''',yp,
162 e 0804            '-''',y,' znasajo ',I,'|')
163 e 0804         (a,x(15),2(a,p'99'),a,p'zzzbzzzbzz9V,99',x(16),a);
164 e 0804         call line();
165 c 0808         end current_year;
166 c 0808
167 c 0808      header:
168 c 0808         proc;
169 e 0808         put file(output) list(clear);
170 e 0822         call line();
171 e 0825         put file(output) skip edit
172 e 0860      ('|','P R E G L E D   O D P L A C E V A N J A   P O S O J I L A',
173 e 0860         '|')
174 e 0860            (a,x(10));
175 e 0860         call line();
176 e 0863         put file(output) skip edit
177 e 08E3         ('|','Obrestna mera',yi,'%',
178 e 08E3            'Inflacijska mera',ir,'%','|')
179 e 08E3            (a,x(14),2(a,p'b99v,99',a,x(6)),x(9),a);
180 e 08E3         call line();
181 e 08E6         put file(output) skip edit
182 e 0942            ('|Datum|',
183 e 0942            '    Vrednost    |',
184 e 0942            'Plus   obresti|',
185 e 0942            '    Obrok    |',
186 e 0942            'Plac. vrednost|',
187 e 0942            'Plac. obresti |') (a);
188 e 0942         call line();
189 c 0946         end header;
190 c 0946
191 c 0946      line:
192 c 0946         proc;
193 e 0946         dcl
194 e 0946            i fixed bin;
195 e 0946         put file(output) skip edit
196 e 099E         ('-------','-----------',
197 e 099E            ('---------------' do i = 1 to 4)) (a);
198 c 099E         end line;
199 a 099E      end pmt;

CODE SIZE = 099E
DATA AREA = 03AA
FREE SYMS = 386A
END  COMPILATION
```

Tretji primer liste 28 uporablja enake začetne vrednosti kot prva dva primera, vendar prikazuje tudi vse vmesne podatke (prikazovalni format 2).

Zadnji primer liste 28 upošteva pri nekoliko spremenjenih vhodnih podatkih (obrok, fiskalni mesec) tudi neničto inflacijsko mero, ki znaša 25 %. Ta mera je za obdobje odplačevanja posojila konstantna. Čeprav v naših praktičnih obračunih inflacije nikoli ne upoštevamo pri odplačevanju posojil, je zadnji primer liste 28 zelo poučen in nam postavlja vprašanje, ali so inflacijska gibanja res stabilizacijsko, gospodarsko in tudi strateško tako nepomembna, da jih v poslovanju z denarnimi sredstvi ni vredno upoštevati. Zaradi inflacije se konstantno zmanjšuje vrednost obroka, ki iz začetne vrednosti 4000 din v oktobru 1981 pade na dejansko vrednost 3120 din v oktobru 1982.

Za razumevanje delovanja programa POSOJILO si oglejmo še pomen spremenljivk med vrsticami 15 do 29 liste 27:

P — ima začetno vrednost PV, toda se med izvajanjem programa spreminja (vrstice 64, 102 in 105)

PP — celotna vplačana vrednost (vrstica 106)

PL — vrednost za trenutno prikazano vrstico, ki hrani P za prikaz (vrstici 109 in 113)

PMT — obrok je v začetku PMV, med izvajanjem programa pa se spreminja (vrstici 70 in 104)

INT — izračunane obresti za tekoči mesec (vr-

```
A>B:POSOJILO
```

P R E G L E D O D P L A C E V A N J A P O S O J I L A

Ime izhodne zbirke ,

Vrednost 50000
Obresti 14
Obrok 3000
Inflacija (%) 0
Zacetni mesec 10
Zacetno leto 81
Fiskalni mesec 2

Prikazovalna ravnina
Letni rezultati : 0
Letne obresti : 1
Vse vrednosti : 2 0

P R E G L E D O D P L A C E V A N J A P O S O J I L A					
Obrestna mera 14,00%			Inflacijska mera 00,00%		
Datum	Vrednost	Plus obresti	Obrok	Plac. vrednost	Plac. obresti
02/82	40 162,85	468,57	3 000,00	12 368,58	2 631,42
02/83	7 758,62	90,52	3 000,00	45 150,86	5 849,14
04/83	1 905,71	22,23	1 927,94	50 000,00	5 927,94

Vrednost ,
Obresti ,
Obrok ,
Inflacija (%) ,
Zacetni mesec ,
Zacetno leto ,
Fiskalni mesec ,

Prikazovalna ravnina
Letni rezultati : 0
Letne obresti : 1
Vse vrednosti : 2 1

P R E G L E D O D P L A C E V A N J A P O S O J I L A					
Obrestna mera 14,00%			Inflacijska mera 00,00%		
Datum	Vrednost	Plus obresti	Obrok	Plac. vrednost	Plac. obresti
02/82	40 162,85	468,57	3 000,00	12 368,58	2 631,42
Obresti placane v letu '81-'82 znasajo			2 631,42		
02/83	7 758,62	90,52	3 000,00	45 150,86	5 849,14
Obresti placane v letu '82-'83 znasajo			3 217,72		
04/83	1 905,71	22,23	1 927,94	50 000,00	5 927,94
Obresti placane v letu '82-'83 znasajo			78,80		

Vrednost ,
Obresti ,
Obrok ,
Inflacija (%) ,
Zacetni mesec ,
Zacetno leto ,
Fiskalni mesec ,

Prikazovalna ravnina
Letni rezultati : 0
Letne obresti : 1
Vse vrednosti : 2 2

P R E G L E D O D P L A C E V A N J A P O S O J I L A					
Obrestna mera 14,00%			Inflacijska mera 00,00%		
Datum	Vrednost	Plus obresti	Obrok	Plac. vrednost	Plac. obresti
10/81	50 000,00	583,33	3 000,00	2 416,67	583,33
11/81	47 583,33	555,14	3 000,00	4 861,53	1 138,47
12/81	45 138,47	526,62	3 000,00	7 334,91	1 665,09
01/82	42 665,09	497,76	3 000,00	9 837,15	2 162,85
02/82	40 162,85	468,57	3 000,00	12 368,58	2 631,42
Obresti placane v letu '81-'82 znasajo			2 631,42		
03/82	37 631,42	439,03	3 000,00	14 929,55	3 070,45
04/82	35 070,45	409,16	3 000,00	17 520,39	3 479,61
05/82	32 479,61	378,93	3 000,00	20 141,46	3 858,54
06/82	29 858,54	348,35	3 000,00	22 793,11	4 206,89
07/82	27 206,89	317,41	3 000,00	25 475,70	4 524,30
08/82	24 524,30	286,12	3 000,00	28 189,58	4 810,42
09/82	21 810,42	254,45	3 000,00	30 935,13	5 064,87
10/82	19 064,87	222,42	3 000,00	33 712,71	5 287,29
11/82	16 287,29	190,02	3 000,00	36 522,69	5 477,31
12/82	13 477,31	157,24	3 000,00	39 365,45	5 634,55
01/83	10 634,55	124,07	3 000,00	42 241,38	5 758,62
02/83	7 758,62	90,52	3 000,00	45 150,86	5 849,14
Obresti placane v letu '82-'83 znasajo			3 217,72		
03/83	4 849,14	56,57	3 000,00	48 094,29	5 905,71
04/83	1 905,71	22,23	1 927,94	50 000,00	5 927,94
Obresti placane v letu '82-'83 znasajo			78,80		

Lista 28. Ta lista prikazuje delovanje programa POSOJILO, ko se izvaja zbirka POSOJILO.COM. Na tej strani liste imamo tri primere z enakimi vhodnimi podatki toda z različnimi prikazovalnimi formati. Inflacija znaša tu 0 %.

```
Vrednost      ?
Obresti       ?
Obrok         4000
Inflacija (%) 25
Zacetni mesec ?
Zacetno leto  ?
Fiskalni mesec 12

Prikazovalna ravnina
Letni rezultati : 0
Letne obresti   : 1
Vse vrednosti   : 2 2
```

```
|-------------------------------------------------------------------------|
|          P R E G L E D   O D P L A C E V A N J A   P O S O J I L A       |
|-------------------------------------------------------------------------|
|          Obrestna mera 14,00%      Inflacijska mera 25,00%              |
|-------------------------------------------------------------------------|
|Datum| Vrednost  |Plus  obresti|  Obrok  |Plac. vrednost|Plac. obresti |
|-----|-----------|-------------|---------|--------------|--------------|
|10/81|  50 000,00|             |  583,33 |  4 000,00    |  3 416,67    |  583,33
|11/81|  45 605,08|             |  532,05 |  3 916,00    |  6 728,86    |1 103,13
|12/81|  41 358,60|             |  482,52 |  3 836,00    |  9 944,87    |1 563,12
|-------------------------------------------------------------------------|
|          Obresti placane v letu '81-'81 znasajo     1 629,95           |
|-------------------------------------------------------------------------|
|01/82|  37 252,15|             |  434,60 |  3 760,00    | 13 073,23    |1 966,76
|02/82|  33 204,91|             |  387,39 |  3 680,00    | 16 087,69    |2 312,30
|03/82|  29 327,06|             |  342,14 |  3 608,00    | 19 038,78    |2 609,21
|04/82|  25 512,25|             |  297,64 |  3 532,00    | 21 872,10    |2 851,89
|05/82|  21 823,75|             |  254,61 |  3 460,00    | 24 631,62    |3 048,37
|06/82|  18 230,93|             |  212,69 |  3 388,00    | 27 294,37    |3 197,62
|07/82|  14 753,44|             |  172,12 |  3 320,00    | 29 894,42    |3 305,57
|08/82|  11 367,87|             |  132,62 |  3 252,00    | 32 401,50    |3 370,49
|09/82|   8 086,16|             |   94,34 |  3 188,00    | 34 857,49    |3 398,50
|10/82|   4 886,01|             |   57,00 |  3 120,00    | 37 176,98    |3 383,01
|11/82|   1 785,62|             |   20,83 |  1 806,45    | 38 200,00    |3 334,45
|-------------------------------------------------------------------------|
|          Obresti placane v letu '82-'82 znasajo     2 734,52           |
|-------------------------------------------------------------------------|
```

```
Vrednost  ?C
A>
```

Lista 28 (nadaljevanje). V tem delu liste imamo primer prikazovalnega formata 2 (podrobno prikazovanje) in v tem primeru je upoštevana kot zanimivost inflacijska mera 25 %. Ta primer naj bi bil zanimiv tudi za naše bančnike, saj že preprosti poskusi z realnimi inflacijskimi merami (z realno devalvacijo denarne enote) kažejo, da se vrednost obroka hitro zmanjšuje in je že odplačevanje na kratka razdobja pri veliki inflaciji vse prej kot ekonomsko upravičeno. Lista kaže, kako so možne hitre ponovitve prejšnjih primerov s spreminjanjem nekaterih vhodnih podatkov, saj je program po opravljenem izračunu takoj spet pripravljen za naslednji poskus oziroma izpis.

stice 99, 102 in 113)

YIN obresti na začetku tekočega leta (vrstice 93, 147 in 148).

IP v celoti plačane obresti (vrstice 91, 100 in 147)

i obrestna mera, ki je v začetku enaka yi, tj. letni obrestni meri (vrstica 67)

INF procentualna devalvacija denarne enote zaradi inflacije (vrstice 107, 113 in 114)

ci trenutna devalvacija zaradi inflacije (vrstice 74, 107 in 108)

fi faktor za izračun trenutne inflacije (vrstici 73 in 108)

Omenimo še, da sta P in PMT delovni spremenljivki za vrednost (glavnico) in obrok, tako da se pravi spremenljivki PV in PMV med izračunom ne pokvarita. Uporabnik lahko za vsak podatek vtipka znak ´,´, če želi ohraniti že preje (v prejšnji iteraciji) vtipkano vrednost tudi v tej iteraciji.

Izvajanje programa se začne z vrstico 35 liste 27, ko se najprej izda znak (ali zaporedje znakov) za brisanje zaslona (clear). Glede na uporabljen zaslonski terminal moramo spremeniti deklaracijo v vrstici 6, ko vstavimo namesto znaka ´CTL z´ predpisano zaporedje krmilnih znakov (npr. za HOME in nato brisanje zaslona).

Ker bomo uporabili OPEN stavek, nastavimo najprej past z ON pogojem za možne OPEN napake (od vrstice 38 do 42). Z vrstico 46 vstavi uporabnik ime zbirke, ki je bilo že prej inicializirano z vrstico 32 kor ´$con´. Če vstavi uporabnik vejico, postane konzola naprava za izhod. Sicer pa se odpre izhodna zbirka kot normalna PRINT naprava ali pa se uporabi fizična naprava za listanje (npr. ´$lst´). Napake se signalizirajo z uporabo vrstice 88.

Glavna zanka programa začenja z vrstico 97 in se ponavlja, dokler vrednost posojila ne pade na nič. Mesečne obresti za trenutno vrednost (glavnico P) se izračunavajo in sumirajo z IP v vrsticah 99 in 100. Ko obrok preseže ostanek vrdnosti v vrstici 103, se obrok zniža na pokritje ostanka. Celotna plačana vrednost je sešteta v vrstici 106, inflacijska mera pa se določi z vrstico 107.

Program ima možnost izbire treh prikazovalnih formatov (ravnin), pri tem se uporabljajo PICTURE stavki in drugi pripomočki. Nekatere subrutine, ki sestavljajo prikazovalni mehanizem, so še posebej zanimive (na koncu programske liste) in jih je vredno prebrati za uporabo v kasnejših programih.

16. Sklep

V zaporedju člankov o jeziku PL/I-80 in njegovi uporabi (glej navedbe 22, 23 in 24 na koncu članka) smo pokazali tudi nekaj značilnih primerov, ki so zanimivi za prakso. Prevajalnik PL/I-80 je realiziran tudi na sistemu Delta 323/M1, ki uporablja operacijski sistem CP/M; ta računalniški sistem je bil javno prikazan v Zagrebu na razstavi Interbiro ´81. Z uporabo jezika PL/I-80 na sistemu Delta 323/M1 je dana možnost tudi našim uporabnikom, da začno intenzivneje uporabljati jezik PL/I pri programiranju sistemskih in poslovnih nalog.

Za konec zapišimo še tole: programirna jezika BASIC in PASCAL sta prirejena in namenjena začetnikom na področju programiranja. Za razliko od teh jezikov pa zahteva jezik PL/I več programirnih izkušenj, nadarjenosti in znanja ter daje bolj učinkovite objektne (računalniške) kode kot prej navedena jezika. Učinkovitost kodov (prevodov), generiranih z različnimi prevajalniki za mikroračunalnike, je bila prikazana v članku (25). Pri izvajalnem času 1,00 za prevedeni PL/I-80 program so bili dobljeni časi za druge mikroračunalniške prevajalnike tile: jezik C čas 1,11, RATFOR 1,18, FORTRAN 1,21, kompilator BASIC 1,32, PASCAL MT 1,35, Intel PL/M 3,43, UCSD PASCAL 3,86, FORTH 6,07, PASCAL/M 32,14, CBASIC2 34,57, Microsoft COBOL 365,30 itd.

Bralce, ki jih je problematika programiranja v jeziku PL/I zanimala in se sami ukvarjajo s programiranjem v tem jeziku, vabimo, da pošljejo v objavo svoje prispevke v okviru rubrike "Uporabni programi". Uredništvo si bo prizadevalo, da objavi v tej rubriki tudi čim več uporabnih PL/I programov.

Dodatna literatura

(22) A.P.Železnikar, Jezik PL/I in mikroračunalniki I, Informatika 4(1980), št.4, 3-10.

(23) A.P.Železnikar, Jezik PL/I in mikroračunalniki II, Informatica 5(1981), št.1, 16-27.

(24) A.P.Železnikar, Jezik PL/I in mikroračunalniki III, Informatica 5(1981), št.2, 31-43.

(25) J.Gilbreath, A High-Level Language Benchmark, Byte 6(1981), No.9, 180-198.

AVTOMATSKO PREPOZNAVANJE PREDMETOV V ROTOTIZIRANEM PROCESU EMAJLIRANJA

JADRAN LENARČIČ

UKD: 681.3:007.52

UNIVERZA EDVARDA KARDELJA,
INSTITUT JOŽEF STEFAN, LJUBLJANA

V našem delu podajamo inženirsko obravnavo prepoznavanja predmetov v industrijskem procesu emajliranja in možnosti aplikacije na mikroračunalniku. Opisujemo računski postopek za avtomatsko določanje števila in položaja svetlobnih senzorjev za podani nabor elementov pri konstrukciji robotskega sistema za prepoznavanje. Določanje šifre posameznim predmetom naslanjamo na minimizaciji kvadratne pogreške med prejeto in nominalno kodo, ki jo preuredimo za aplikacijo na mikroračunalniku z uporabo korelacij. Rezultati naše obravnave se kažejo v enostavnosti in učinkovitosti ter predvsem v cenenosti izvedbe sistema senzorjev za prepoznavanje in v obdelavi prejetih podatkov. Pomembnost našega dela pa je v opisanih možnostih industrijske aplikacije.

AUTOMATIC RECOGNITION OF ELEMENTS IN ROBOTISED PROCESS OF ENAMELLING:
This work deals with the engeneering way of recognition of elements in an industrial process of enamelling and the possibilities of its application on microcomputer. We describe in particular the process of an automatic choice of number and positions of light sensors for the given set of elements in construction of a robot system for recognition. The determination of cipher of single element is based on minimisation of quadratic error between the received and nominal code which we elaborate for application on microcomputer by use of correlations. The results of this research are evident in the semplicity and efficiency, but above all in the inferrior cost of the realization of the system of sensors for recognition and also in the elaboration of the received data. But the importance of this research is above all, as mentioned, in the possibilities of industrial application.

UVOD

Sodobni razvoj robotike je precej uspešno usmerjen predvsem v področje industrijskih manipulatorjev. Da je temu tako, seveda ni naključje, saj so to sistemi, ki v najkrajšem času opravičijo drago investicijo.

Pomemben podsklop nekega industrijskega manipulatorja je sistem senzorjev, od katerega kompleksnosti močno zavisi univerzalnost in uporabnost manipulatorja, oziroma je struktura sistema senzorjev v naprej dana z robotovo namembnostjo. V večini industrijskih aplikacij predstavlja ta sistem le nekaj stikal ali tipkal primerno razporejenih v robotovi okolici ali na samem robotu, s čemer lahko za omejene potrebe povsem dobro modeliramo proces.

Obstajajo pa seveda zahtevnejši procesi, za katere potrebujemo kompleksnejše in obsežnejše strukture senzorjev, ki omogočajo razvoj učinkovitejšega krmilnega sistema. V industrijskih aplikacijah pa je pri gradnji posameznih podsklopov velikega pomena postavka cene, kar na žalost močno omejuje konstruktorja pri razvoju ustreznih podsklopov robota. Lahko trdimo, da je za večino industrijskih manipulatorjev cena

uporabljenih senzorjev napram ceni celotnega sistema zanemarljivo majhna, kar pa je seveda pogojeno predvsem z nezahtevnostjo nalog in opravil, ki jih ti roboti v industriji izvršujejo.

Pri projektiranju industrijskega robota za emajliranje je potrebno vključiti tudi sistem senzorjev, ki omogočajo avtomatsko prepoznavanje obdelovancev v realnem času. Robot je vključen v delovni proces ob emajlirni liniji, po kateri s konstantno hitrostjo potujejo predmeti za emajliranje, ki jih robot prepoznava in pravilno obdeluje. Ponuja se množica načinov kako izvesti prepoznavanje.

Zelo drago, vendar gotovo najbolj učinkovito, če izvedljivo, bi bilo prepoznavanje vzorcev za emajliranje s televizijsko kamero in ustrezno instrumentalno in programsko opremo. Predmeti bi se gibali mimo robota, ki bi z obdelavo tako prejetih informacij določil geometrijo posameznih elementov in hitrost njihovega potovanja. S temi podatki bi oblikoval programe, oziroma bi na osnovi svoje dinamike in danih kriterijev optimalno ali kako drugače definiral nominalne tirnice, katerim bi sledil ob samem emajliranju. Jasno je, da nam današnja računalniška tehnologija še ne omogoča česa takega. Numerično izračunavanje trajektorij posameznih stopenj prostosti robota iz podame trajektorije robotovega vrha je celo za minimalno konfiguracijo toliko kompliciranо in počasno, da ni uporabno za aplikacijo v industrijskem procesu v realnem času, da ne govorimo o težavah, predvsem časovnih, ki bi jih prinašalo že samo detaljno razpoznavanje elementov. Zaradi tega se največkrat odločimo za enostavnejšo varianto, ker nam to omogoča tudi obravnavani industrijski proces. Seznam elementov, ki jih je potrebno emajlirati, je namreč v naprej znan. Zatorej poteka določanje nominalnih trajektorij robota na zalogo in se na tak ali drugačen način shranjuje v ustrezne podatkovne strukture v robotovem pomnilniku, čemur navadno pravimi učenje robota. Potrebno je le konstruirati mehanizem, ki bi omogočal prepoznavanje in ločevanje med danim naborom predmetov, kar pa je razumljivo daleč enostavneje, ceneje in predvsem izvedljivo.

Namen našega dela je opis in prikaz ene izmed teh možnosti, kako bi čim ceneje in dovolj učinkovito, industrijsko in na osnovi največje verjetnosti prepoznavali v dveh dimenzijah enolično definirane vzorce, katerih kode so v naprej podane kot logične funkcije v časovnem prostoru. Sistem pa pri tem vključuje minimalno število točkovnih svetlobnih senzorjev - fotocelic. Bralca želimo seznaniti z metodologijo pravilne izbire tako položaja kot tudi števila fotocelic. V tem je pravzaprav tudi pomembnost našega dela in ne v neki posebne vrste izbrani tehnologiji ali obdelavi prejete informacije, ki bi uporabljala tako imenovane metode umetne inteligence.

Minimizacijski postopek števila fotocelic in izbor njihovega položaja glede na obešene predmete za emajliranje sloni na analizi vseh možnih stanj tako, da startamo z najboljšim stanjem in z izločevanjem nadaljujemo k slabšim, dokler ne pridemo do rešitve. Tako evaluiramo minimalno potrebno število fotocelic za razločevanje danega nabora predmetov, istočasno pa pridemo tudi do podatkov, kje naj te celice ležijo. Obravnava motenj in iskanje ustrezne šifre obdelanih predmetov vsebuje minimizacijo kvadratne pogreške med sprejeto in nominalno kodo, kar je izvedeno z uporabo korelacij.

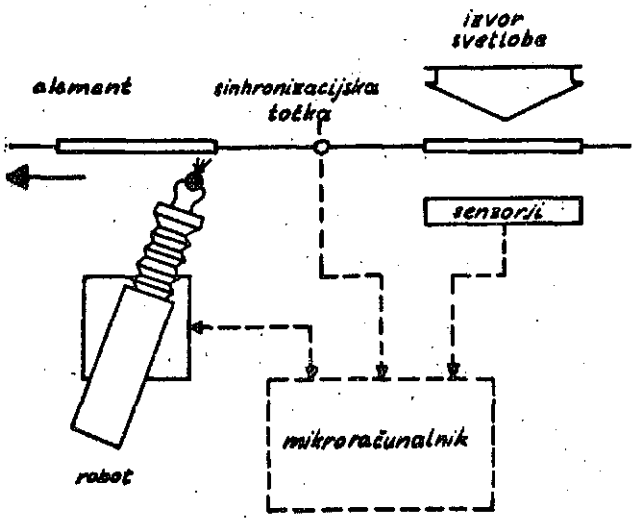
V našem delu najprej poenostavljeno prikažemo proces emajliranja, na katerem apliciramo sistem senzorjev za prepoznavanje predmetov - obdelovancev. Opišemo težave, s katerimi se srečujemo in razložimo smisel in možnosti uporabe fotocelic. Nato podamo avtomatski postopek minimizacije števila fotocelic in sicer v konstrukcijske namene ter na koncu podamo obravnavo prepoznavanja predmetov v realnem delovnem okolju in diskusijo o možnostih aplikacije na mikroračunalniku. Vse podane algoritme podpremo z enostavnimi primeri.

OPIS ROBOTIZIRANEGA PROCESA EMAJLIRANJA

Kot smo že omenili, se bomo tokrat omejili le na obravnavo sistema senzorjev za prepoznavanje obdelovancev apliciranega na realnem robotiziranem procesu emajliranja, ki je del produkcijskega procesa elementov gospodinjske opreme.

Robot je nameščen ob emajlirni liniji, po kateri potujejo obešeni predmeti, ki jih robot najprej prepozna in nato primerno emajlira. V ta namen je na liniji fiksirana točka, ki služi kot časovno in prostorsko izhodišče in je ključnega pomena za sinhronizacijo robota z linijo.

Robot je nameščen tik za sinhronizacijsko točko, senzorni sistem pa tik pred njo glede na gibanje linije. Obdelovanec mora torej najprej prepotovati sistem fotocelic, s katerimi ga robot prepozna ter šele nato prekorači sinhronizacijsko točko in sproži sinhronizacijski impulz, ki opomni robota, da prične z emajliranjem. Postopka prepoznavanja in emajliranja potekata časovno paralelno in sicer tako, da robot med emajliranjem tekočega predmeta prepoznava sledečega, kar mu omogoča ustrezni večprogramski mikroračunalniški paket. Slika 1 prikazuje stiliziran emajlirni proces.



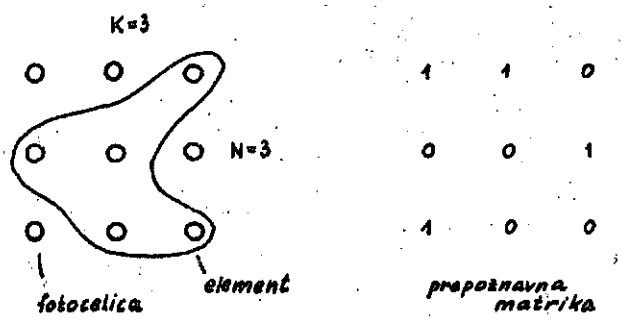
Sl.1: Emajliranje in prepoznavanje elementov s pomočjo sinhronizacijske točke

PROBLEMATIKA PREPOZNAVANJA DVODIMENZIONALNIH ELEMENTOV ZA EMAJLIRANJE

Proces emajliranja, s katerim imamo opraviti, je dinamičen, vendar dovolj enostaven za obravnavo, saj je hitrost linije konstantna. Njegova dinamičnost je celo zaželjena, ker omogoča elegantnejšo in kompleksnejšo proceduro.

Nabor elementov, ki jih je potrebno prepoznati, je v naprej podan, kot je tudi podana hitrost linije. Dimenzije posameznih ploskovnih elementov so definirane in se ne spreminjajo tekom izvajanja procesa, njihovo število pa je lahko praktično poljubno veliko. Tako postavljeni problem prepoznavanja vzorcev je dokaj enostavno rešljiv tudi z nezahtevno tehnologijo, težavam, ki jih prinašajo motnje v samem industrijskem procesu, pa se moremo izogniti s primerno sprotno obdelavo prejetih podatkov.

Seveda zaradi tega postopek prepoznavanja vzorcev, ki ga konstruiramo v našem delu, ne vsebuje tako imenovanih elementov umetne inteligence in ga v takšni luči tudi ne nameravamo prikazati. Naš namen je namreč sestaviti postopek ali celotno proceduro za konstruiranje sistema senzorjev in nato ustrezno prepoznavanje, ki bo dovolj cenen in učinkovit z uporabo najenostavnejše industrijske tehnologije. Posamezni elementi so podani s svojo kodo, ki jo predstavlja neka on - off funkcija. Na sliki 2 je za statični primer prikazana matrična razporeditev fotocelic.

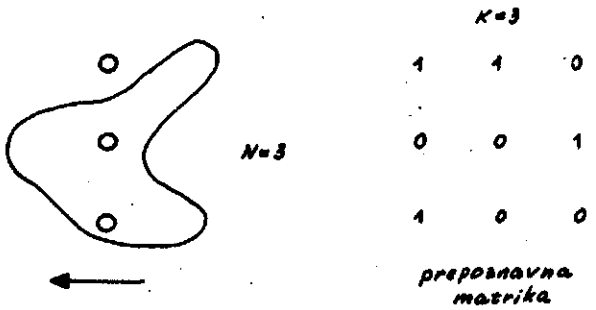


Sl.2: Statična prepoznavna matrika s fotocelicami

Celotna koda h je torej skupek funkcij $f_1, f_2, \ldots, f_N$, funkcija f_i pa je zbirka K on - off komponent, ki popisujejo stanja v fotocelicah. N je število vrstic, K pa število stolpcev prepoznavne matrike, katere elemente tvorijo soležne fotocelice. Kompletno število fotocelic je M=NxK. Vsak element je podan s svojo kodo, ki se zaradi svoje redundantnosti pretransformira v ustrezno šifro, katere dolžina N^+ je pogojena s številom vseh elementov in zaradi česar velja neenačba $N^+ \neq N$. Šifra elementa je ime elementa, s katerim neposredno operira robotov mikroračunalnik.

V drugem primeru, ki je prikazan na sliki 3, opazujemo dinamični proces. Ugotovimo, da je potrebno število fotocelic M=N in jih razporedimo v stolpec enega nad drugim, mimo katerega

potuje predmet, ki ga želimo prepoznati.



Sl.3: Dinamična prepoznavna matrika s fotocelicami

Postopek prepoznavanja je razdeljen v K časovnih trenutkov tako, da za vsak k=1,2,...,K pregledamo stanja fotocelic. Če sta K in N v statičnem in dinamičnem primeru enaka ter če sta procesa pravilno umerjena, je sprejeta koda elementa v obeh primerih enaka. Izvedba je v dinamičnem primeru nekoliko zahtevnejša, vendar dovolj enostavna, je pa zato število uporabljenih fotocelic razumljivo K krat manjše. Prepoznavni matriki sta popolnoma enaki, le da so funkcije $f_1, f_2, \ldots, f_N$ za dinamični primer podane kot funkcije časa, kar pa je s stališča računalniške obdelave popolnoma vseeno.

Prejeti kodi nato dekodiramo in ji priredimo šifro, ki je po dolžini krajša ali kvečjemu enaka. Zal pa zaradi motenj v industrijskih procesih to ne moremo narediti neposredno. Zaradi mehanskih električnih ali tudi drugačnih vplivov prejete kode ne ustrezajo nominalno definiranemu naboru. Prirejanje ustrezne šifre je zato podrejeno dodatnemu algoritmu, ki glede na obliko in podobnost prejetih in nominalnih kod na osnovi največje verjetnosti izbere najprimernejšo šifro.

IZBIRA MINIMALNEGA ŠTEVILA FOTOCELIC IN NJIHOVE LEGE V PREPOZNAVNI MATRIKI

Kot je bilo rečeno, sloni naš postopek prepoznavanja na tem, da imamo že v naprej dan nabor vseh elementov, ki jih je potrebno emajlirati. Videli smo tudi, da je pri dinamičnem procesu število potrebnih fotocelic K krat manjše kot pri statičnem primeru. V tem poglavju pa se bomo nekoliko dlje pomudili pri problemu izbire najmanjšega potrebnega števila fotocelic ter njihovega položaja v prepoznavni matriki, da bi še lahko prepoznali in izločili vsakega izmed celokupne zbirke predhodno definiranih elementov.

Namreč, logična težnja konstruktorja je, da pri gradnji sistema za prepoznavanje uporabi kolikor se da minimalno število fotocelic ter seveda v naprej določi njihov položaj, da bo postopek prepoznavanja dajal zadovoljive rezultate.

Če je število uporabljenih fotocelic M, lahko problem formuliramo takole

minimiziraj M (1)

in

določi lego fotocelic

pri pogoju

$h_i \neq h_j$, za vsak i,j=1,2,...,L, i≠j,

kjer je L število vseh elementov za prepoznavanje, h_i pa kode elementov.

Da bi rešili gornji problem, smo izdelali fortranski programski paket, ki vsebuje naslednji koračni algoritem:

Korak 1

Postavi i=0. Izberi primerno resolucijo N in število časovnih trenutkov K začetne fiktivne prepoznavne matrike. To lahko narediš z upoštevanjem sledečega

$$2^{NK} \geq L, \quad (2)$$

kjer je L število elementov. Predpostavi, da se ti posreči z eno samo geometrijsko pravilno vgrajeno fotocelico prepoznati vse elemente, N=1. Sledi

$$2^{K} \geq L. \quad (3)$$

Začetno število časovnih je tedaj enako

$$K^{0} \geq \log_2 L, \quad (4)$$

smiselna fiktivna začetna resolucija pa je

$$N^o = K^o$$

in

$$M^o = N^o. \qquad (6)$$

Korak 2

Postavi i=i+1. Za dane vrednosti N^i in K^i izvedi minimizacijo (1) in testiraj dobljeni M^i ali je večji ali manjši od M^{i-1} in nadaljuj s korakom 3.

Korak 3

Če velja

a) $M^i \leq M^{i-1}$,

sta M^{i-1} in K^{i-1} rešitvi minimizacijskega problema (1),

b) $M^i \not\leq M^{i-1}$,

postavi $K^i=K^{i-1}$ in se vrni h koraku 2,

c) ni rešitve,

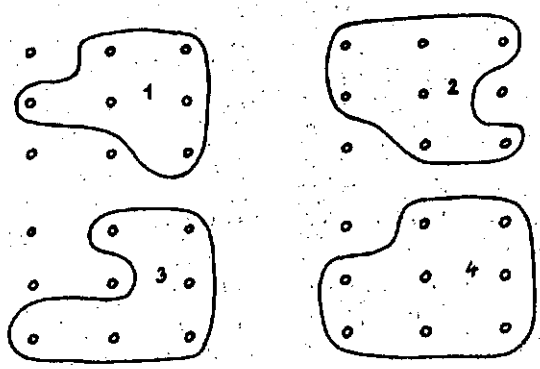
postavi $K^i=K^i+1$ in $N^i=N^i+1$ in se vrni h koraku 2.

REŠEVANJE MINIMIZACIJSKEGA PROBLEMA

Gornji koračni algoritem nas za izbrane začetne vrednosti pripelje do večjega ali manjšega števila na videz ekvivalentnih rešitev. Katero bomo uporabili pri gradnji sistema za prepoznavanje, je predvsem odvisno od našega trenutnega razpoloženja, kar gotovo ni znanstveno, vendar, kot bomo kasneje videli, pa se tudi tu skriva nekaj tehniške logike.

Dolžni smo opisati še proceduro minimizacije števila fotocelic in iskanja njihovih položajev na začetni fiktivni prepoznavni matriki. V ta namen ne uporabljamo nekega optimizacijskega postopka iz neomejene zakladnice znanih metod, ker bi bil le-ta neprimeren in bi prinašal več komplikacij kot ugodnosti. Naša procedura temelji na opazovanju vseh možnih stanj po določenem vrstnem redu in sicer od najboljših po korakih k slabšim do evalvacije rešitev.

Omenjeno minimizacijsko proceduro raje obrazložimo na enostavnem primeru. Imejmo podan nabor predmetov na sliki 4.



Sl.4: Nabor predmetov za prepoznavanje

Dobimo funkcije f_n^k, kjer je n številka vrstice v prepoznavni matriki, k pa številka predmeta

$f_1^1 = 1\ 0\ 0$

$f_1^2 = 0\ 0\ 0$

$f_1^3 = 1\ 0\ 0$

$f_1^4 = 1\ 0\ 0$

$f_2^1 = 0\ 0\ 0$

$f_2^2 = 0\ 0\ 1$

$f_2^3 = 1\ 1\ 0$

$f_2^4 = 0\ 0\ 0$

$f_3^1 = 1\ 1\ 0$

$f_3^2 = 1\ 0\ 0$

$f_3^3 = 0\ 0\ 0$

$f_3^4 = 0\ 0\ 0.$

Postopek nadaljujemo s primerjavo posameznih funkcij pri n=konstanta f_n^k, k=1,2,3,4 ter tvorimo pomožno matriko Y. To naredimo tako: na mesto f_n^1 postavimo 1 ter z njo primerjamo vse ostale f_n^k, k=2,3,4. Če je katera od teh enaka f_n^1, postavimo tudi na njeno mesto 1. Dobimo razred enakih funkcij z imenom 1. Nato nadaljujemo s tvorbo naslednjega razreda po enakem postopku neupoštevajoč že uvrščene funkcije, vendar z imenom 2 ter tako dalje, dokler ne obdelamo vseh funkcij f_n^k, n=1,2,3, k=1,2,3,4. Rezultat našega prizadevanja je pomožna matrika Y z naslednjimi komponentami

Y:

1 1 1
2 2 2
1 3 3
1 1 3.

Primerjava številk v matriki Y nas pripelje do končnega rezultata. Najprej pregledamo, če je v posameznem stolpcu L (4) različnih številk. Ker jih tu ni, pomeni, da s takšno začetno resolucijo N in K ne moremo ločiti vseh štirih predmetov z eno samo fotocelico. Zato združimo po dva in dva stolpca matrike Y in zopet opazujemo, če dobimo po štiri različne številke takole:

1. in 2. stolpec

11
22
13
11

1. in 3. stolpec

11
22
13
13

2. in 3. stoplec

11
22
33
13.

Ugotovimo, da je uspešna samo kombinacija drugega in tretjega stolpca, kar pomeni, da je za gornji primer potrebno uporabiti najmanj dve fotocelici in sicer drugo in tretjo gledano od zgoraj navzdol v začetni fiktivni prepoznavni matriki. V kolikor z združevanjem dveh stolpcev matrike Y še ne bi prišli do rezultata, bi zrduževali 3 in več stolpcev, dokler ne bi obdelali celotne pomožne matrike Y.

Takšna minimizacijska procedura je izredno prikladna za uporabo na digitalnem računalniku in nas v kratkem času zanesljivo pripelje do vseh ekvivalentnih rešitev, če za izbrani začetni resoluciji te obstajajo in nam istočasno določi položaje fotocelic na začetni fiktivni prepoznavni matriki.

PREPOZNAVANJE IN LOCEVANJE ELEMENTOV V [illegible] INDUSTRIJSKEM OKOLJU

Obravnava industrijskega procesa emajliranja na nominalnem nivoju je precej enostavna. Do problemov, ki so včasih težko rešljivi, prihaja, ko obravnavamo proces v realnem okolju. Procedure in algoritmi, ki smo jih razvili na nominalnem nivoju, čestokrat odpovedo, če jih predhodno pravilno ne preuredimo, ali če jim predhodno ne dodamo nekaterih dodatkov.

V prejšnjih poglavjih smo obdelali avtomatski postopek konstrukcije senzornega sistema s fotocelicami za prepoznavanje med podanimi predmeti s fiksirano geometrijo. Postopek temelji na minimizaciji števila uporabljenih enot.

Rezultat procesa dinamičnoga prepoznavanja elementov za emajliranje je časovno podana funkcija h imenovana tudi koda elementa. Razumljivo je, da je zaradi motenj, ki nastopajo v industrijskem procesu, sprejeta koda h_r lahko različna od katerekoli nominalnih kod h_n^i, i=1,2,...,L. Napačno bi bilo ali vsaj nesprejemljivo za proces, da bi zahtevali ponovitev sprejemanja kode. Potrebno je na osnovi nekega kriteriuma iz dane množice nominalnih kod izbrati tisto, ki je sprejeti najbolj podobna.

Matematično moremo problem podobnosti formulirati z najmanjšim kvadratičnim odstopanjem

$$\Phi^i = \frac{1}{T}\int_0^T (h_r(t) - h_n^i(t))^2 dt \rightarrow \min_i,$$

$$i=1,2,\ldots,L, \qquad (6)$$

kjer je T časovni interval, v katerem opazujemo potek funkcije kode, $h_r(t)$ je sprejeta koda, $h_n^i(t)$ pa i-ta funkcija iz nabora nominalnih kod, i=1,2,...,L. Izraz (6) lahko prepišemo v naslednjo obliko

$$\Phi^i = \frac{1}{T}\int_0^T h_r(t)^2 dt + \frac{1}{T}\int_0^T h_n^i(t)\,dt - \frac{2}{T}\int_0^T h_r(t)h_n^i(t)dt,$$

$$i=1,2,\ldots,L. \qquad (7)$$

Po definiciji je korelacijska funkcija podana z izrazom

$$\Phi_{ij}(\tau) = \frac{1}{T}\int_0^T g_i(t)g_j(t+\tau)dt. \qquad (8)$$

S primerjavo (7) in (8) ugotovimo, da so in-

tegrali na desni strani enačbe (7) korelacijske funkcije med $h_r(t)$ in $h_n^i(t)$ za $\tau=0$. Sledi

$$\Phi^i = \Phi_{rr} + \Phi_{nn} - 2\Phi_{rn}^i , \qquad (9)$$

kjer je Φ_{rn}^i križna korelacija, Φ_{rr} in Φ_{nn}^i pa sta avtokorelaciji funkcij $h_r(t)$ in $h_n^i(t)$ pri $\tau=0$. Očitno velja (takrat sta sprejeta in i-ta nominalna koda identični $h_r(t)=h_n^i(t)$)

$$\min_{i=i^*} \Phi^i = 0 \qquad (10)$$

za i^*, ki je rešitev problema (6), kar pomeni z upoštevanjem lastnosti $\Phi^i \geq 0$, da je

$$\Phi^i = \Phi_{rr} + \Phi_{nn}^i - 2\Phi_{rn}^i \geq 0. \qquad (11)$$

Neenačbo (11) preureuredimo v sledečo obliko

$$2\Phi_{rn}^i - \Phi_{nn}^i \leq \Phi_{rr}. \qquad (12)$$

Φ_{rr} je neodvisna od indeksa i, zatorej ne vpliva na rešitev minimizacijskega problema (6). Iz tega sledi, da je z upoštevanjem vsebine neenačbe (12) iskanje podobnosti $h_r(t)$ z naborom $h_n^i(t)$, i=1,2,...,L iskanje sledečega maksimuma

$$2\Phi_{rn}^i - \Phi_{nn}^i \longrightarrow \max_i, \qquad (13)$$

$$i=1,2,\ldots,L.$$

Označimo levo stran izraza (13) z grško črko ψ_{rn}^i in zapišimo detaljneje

$$\psi_{rn}^i = \frac{2}{T}\int_0^T h_r(t)h_n^i(t)dt - \frac{1}{T}\int_0^T h_n^i(t)^2 dt \longrightarrow \max_i, \qquad (14)$$

$$i=1,2,\ldots,L,$$

kar v diskretnem časovnem prostoru po analogiji zapišemo

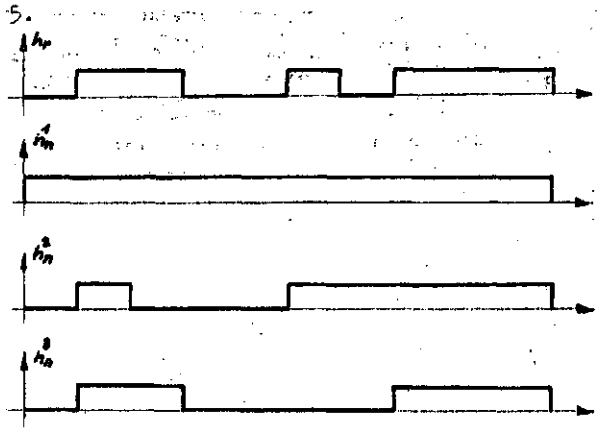
$$\psi_{rn}^i = 2\sum_{k=1}^{N\cdot K} h_r(k)h_n^i(k) - \sum_{k=1}^{N\cdot K} h_n^i(k)^2 \longrightarrow \max_i, \qquad (15)$$

$$i=1,2,\ldots,L.$$

h_r in h_n^i sta podani kot zaporedje ničel in enic in je zato njuno množenje, seštevanje in odštevanje ter na koncu iskanje maksimalne vrednosti zelo enostavno. Drugi del izraza (15), ki je analogen avtokorelaciji, je kar vsota vseh enic v funkciji h_n^i. Hitrost računanja na mikroračunalniku je zanemarljivo majhna, saj pridemo do iskanega maksimuma v končnem številu korakov (L). Določevanje najbolj verjetne šifre predmeta glede na sprejeto kodo torej poteka preko modificiranega korelacijskega filtra, ki neki prejeti kodi pripiše tisto šifro, katere pripadajoča nominalna koda je nojbolj podobna prejeti. Prepoznavanje in nato prirejanje šifre elementom za emajliranje poteka istočasno z emajliranjem predhodnega elementa. Robot sprejme šifro in odredi prirejeno datoteko svojega pomnilnika ter ob sinhronizacijskem impulzu prične z emajliranjem. Vsi računski postopki potekajo v realnem času, tako da ne vplivajo na proces ali kako drugače povzročajo mrtvih časov delovanja robota. Dovolj so enostavni in prikladni za mikroračunalniško obdelavo in dovolj učinkoviti za indústrijsko aplikacijo.

ENOSTAVNI PRIMER

Za ilustracijo naših predhodnih ugotovitev obdelajmo enostavni primer, ki ga podaja slika 5.



Sl.5: Primer prejete in nabora nominalnih kod

Podano imamo torej eno prejeto kodo h_r=0110010111 in tri nominalne kode h_n^1=1111111111, h_n^2=0100011111, h_n^3=0110000111. najprej izračunamo vsote

$$2\sum_{k=1}^{10} h_r(k)h_n^1(k) = 2(0+1+1+0+0+1+0+1+1+1) = 12,$$

$$2\sum_{k=1}^{10} h_r(k)h_n^2(k) = 10,$$

$$2\sum_{k=1}^{10} h_r(k)h_n^3(k) = 10 \qquad (16)$$

in nato

$$\sum_{k=1}^{10} h_n^1(k)^2 = 10,$$

$$\sum_{k=1}^{10} h_n^2(k)^2 = 6,$$

$$\sum_{k=1}^{10} h_n^3(k)^2 = 5, \qquad (17)$$

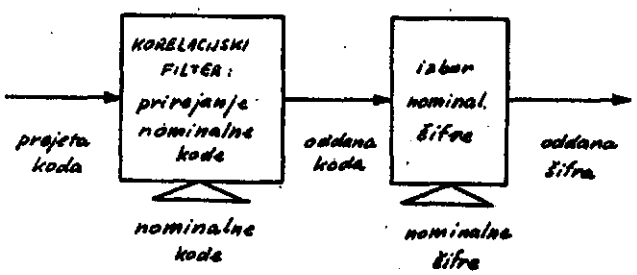
iz česar dobimo, da je

$$v_{rn}^1 = 12-2 = 2,$$

$$v_{rn}^2 = 4,$$

$$v_{rn}^3 = 5. \qquad (18)$$

Očitno je rešitev našega problema podobnosti med prejeto in nominalno kodo koda h_n^3, kar pa smo tudi pričakovali. Tej kodi nato pripišemo ustrezno šifro, ki je primerna za mikroračunalniško obdelavo. Stilizirano prikazujemo postopek na sliki 6.



Sl.6: Prirejanje šifre prejeti kodi

ZAKLJUČEK

V našem delu smo opisali problem prepoznavanja elementov v robotiziranem industrijskem procesu emajliranja. Predstavili smo avtomatski postopek minimizacije števila svetlobnih senzorjev in določanje njihovega položaja glede na elemente, ki je v veliko pomoč konstruktorju pri načrtovanju robotskega sistema za prepoznavanje.

Postopek smo uredili v koračno obliko, ki je zelo primerna za uporabo na digitalnem računalniku. Uporabnost postopka je toliko večja, kolikor večje je število elementov za emajliranje in kolikor bolj so geometrijsko kompliciran, saj je takrat nemogoče na pamet določiti najboljšo varianto razporeditve svetlobnih senzorjev, še manj pa njihovo minimalno potrebno in zadostno število. Seveda pa je postopek primeren za reševanje problema, kjer imamo v naprej dano množico elementov za emajliranje, ki so v dveh dimenzijah enolično določeni.

Opisali smo možnost označevanja posameznih elementov s kodo, ki je definirana s strukturo robotovega sistema za prepoznavanje. Ugotovili smo, da je v realnem industrijskem okolju pričakovati motnje, ki onemogočajo enostavno sprejemanje kode in pretransformiranje v šifro primerno za nadaljno obdelavo. Pokazali smo, kako v zanemarljivo kratkem času izbrati tisto šifro, katere pripadajoča nominalna koda je najbolj podobna prejeti. Enostavnost predlaganega algoritma je tolikšna, da je ustrezna za neposredno aplikacijo na krmilnem mikroračunalniku robota. Z razvojem paralelnega programa je zato omogočeno emajliranje in sprotno prepoznavanje elementov.

Opisana procedura in instrumentalna oprema za prepoznavanje elementov gotovo ni na najvišjem akademskem nivoju, temveč ravno nasprotno, saj je to bil tudi namen avtorjev. Vendar pa je dovolj učinkovita in neobčutljiva na motnje, da je izredno uporabna v industrijskih determiniranih procesih, predvsem pa jo odlikuje izredna cenenost izvedbe. Seveda pa to še ne pomeni, da današnja tehnologija ne omogoča elegantnejših rešitev. V tem smislu naj se inženirsko delo na področju prepoznavanja in robotskega razpoznavanja tudi nadaljuje. Uporabnost in industrijska aplikabilnost metod naj bo merilo njihove uspešnosti.

LITERATURA

P.Bojanić, V.Milačić: Industrijski roboti i veštačka inteligencija - Problemi vizuelnog prepoznavanja, Zbornik referatov 1. jugoslovanskega simpozija o industrijski robotiki in umetni inteligenci, Dubrovnik, 1979

L.Gyergyek: Statistične metode v teoriji sistemov, teorija o informacijah, Fakulteta za elektrotehniko v Ljubljani, 1978

O JEDNOJ METODI PRORAČUNA OPTIMALNOG BALANSA

JOVAN LONČAR

VIŠA ZRAKOPLOVNA ŠKOLA – ZAGREB

UDK: 681.3:51

U radu se promatra problem balansiranja koji se javlja kod konstruiranja zrakoplovnih motora, hidro i parnih turbina i drugdje. U prostor permutacija uvodi se lančana metrika.

DISCUSION OF ONE METHOD FOR THE CALCULATION THE OPTIMAL BALANCE. In a space of permutations the chain metric is introduced. Using this metric a problem of balance is studied.

Vrlo se često susrećemo s problemima gdje treba naći minimum (maksimum) funkcije definirane na prostoru permutacija π. S druge strane postoje takvi problemi koje stroga ograničenja prevode s kontinualnih problema traženja ekstrema na diskretne i to na problem nalaženja ekstrema funkcionala na prostoru permutacija. Jedan od karakterističnih primjera za prvi slučaj je problem balansiranja. Na rotirajući disk stavljaju se lopatice koje zbog nesavršenosti izrade dovode da statički momenti lopatica imaju nekakvu disperziju, pa sumarni debalans zavisi od rasporeda lopatica. Znači da se problem svodi na promatranje slijedećeg funkcionala na skupu permutacija.

$$f(i_1 i_2 \ldots i_n) = \left[\left(\sum_{i=1}^{n} M_{ik} \cos \frac{2\pi k}{n}\right)^2 + \left(\sum_{i=1}^{n} M_{ik} \sin \frac{2\pi k}{n}\right)^2\right]^{\frac{1}{2}}$$

gdje je n-broj lopatica, M_k - statički moment -k-te lopatice.

Jedan od primjera gdje ograničenja prevode kontinualni problem u diskretni je problem odredjivanja minimuma duljine mreže medju pravokutnim radio modulima koji prekrivaju ploču bez rupa i preklapanja.

Kako se radi o praktičnim problemima gdje se n-kreće i do nekoliko stotina to nam diskretnost puno ne pomaže i to iz jednostavnog razloga što bi trebalo ispitati n! vrijednosti funkcionala i onda izabrati najbolju. To praktički znači da već kod n = 10-15 ne dolazi u obzir primjena metode MONTE-KARLO, tj. čisto slučajnog traženja one $p_o \in \pi$ za koju vrijedi

$$f(p_o) \leq f(p_i);\ p_i \in \pi \quad (i = 1,n)$$

gdje p_i - permutacije iz prostora permutacija π.

Kod rješavanja prednjih i mnogih drugih problema pokazala se potreba uvodjenja raznih metrika u prostor permutacija.

Naime, u prostor permutacija možemo uvesti sistem okolina ili kako se to kaže zadati topologiju.

Neke opće definicije [1]

Neka je X - nekakav skup. Topologija u X zove se svaki sistem Z njegovih podskupova, koji zadovoljava slijedeće uvjete:

1. Skup X i prazan skup $\emptyset$ pripadaju Z
2. Unija proizvoljnog broja skupova iz Z pripada Z
3. Presjek konačnog broja skupova iz Z pripada Z.

Skup X sa zadanom na njemu topologijom Z tj. par (X,Z) nazivamo topološkim prostorom.

Skupovi koji pripadaju Z nazivaju se otvoreni skupovi.

Okolinom točke x koja pripada topološkom prostoru zovemo svaki otvoren skup koji sadrži točku x.

Nas će interesirati samo metode uvodjenja topologije u prostor permutacija, a u vezi s metrizacijom tog prostora.

Napomenimo da funkciju d(x,y) zadanu na skupu XxX zovemo udaljenošću, ili metrikom na skupu X ako [2]:

1. za proizvoljne x,y vrijedi
   (1) $d(x,y) \geq 0$
2. za udaljenost od točke x, do nje same vrijedi
   (2) $d(x,x) = 0$
3. ako je udaljenost izmedju dvije točke jednaka nuli onda se te točke podudaraju
   (3) $d(x,y) = 0 \Rightarrow x = y$
4. udaljenost izmedju proizvoljnih točaka x i y jednaka je udaljenosti y i x.
   (4) $d(x,y) = d(y,x)$
5. za tri proizvoljne točke $x,y,z \in X$ vrijedi nejednakost trokuta
   (5) $d(x,z) \leq d(x,y) + d(y,z)$

Skup u kome je zadana udaljenost zove se metrički prostor.

U metričkom prostoru X kuglom radiusa R s centrom u točki x_o zovemo skup točaka $x \in X$ za koje vrijedi

$$d(x_o,x) \leq R$$

LANČANA METRIKA

Promatrajmo sada dvije permutacije p_1, $p_2 \in \pi$ [3]. Za udaljenost $d(p_1,p_2)$ uzmimo minimalni broj rezova koje treba uraditi u jednoj permutaciji da se dobije druga permutacija. Preciznije, vrijedi slijedeća definicija.

DEFINICIJA 1. Neka su zadane dvije permutacije $p_1 = (i_1,i_2 \ldots i_n)$, $p_2=(j_1j_2 \ldots j_n)$. Indeks β elementa j_β za koji je $j_\beta = i_\alpha$ označimo sa $\beta(\alpha)$. Udaljenost $d(p_1,p_2)$ zove se broj takvih parova susjednih elemenata i_α i $i_{\alpha+1}$ za koje je $\beta(\alpha+1) \neq \beta(\alpha)+1$. Tako na primjer $p_1 = (1,2,3,4,5)$; $p_2 = (3\ 4\ 5\ 1\ 2)$ $d(p_1,p_2) = 1$. Iz definicije slijedi da je maksimalna udaljenost u lančanoj metrici izmedju dvije permutacije p_i, p_j od n simbola jednaka n-1.

Pokažimo da metrika uvedena na prednji način zadovoljava sve aksiome metrike.

1. Da je $d(p_1,p_2) \geq 0$ nije potrebno dokazivati.
2. Kad prelazimo od p u nju samu tada susjedi ostaju susjedi pa je $d(p,p) = 0$.
3. Ako je $d(p_1,p_2)=0$ onda svi elementi koji su bili susjedi u p_1, ostaju susjedi i u p_2. Slijedi $p_1=p_2$.
4. Ako je kod prelaza od p_1 u p_2, narušen položaj susjednih elemenata i_α i $i_{\alpha+1}$ onda će kod obratnog prelaza biti narušeno susjedstvo elemenata $j_{\beta(\alpha)}$ i $j_{\beta(\alpha)+1}$. Znači da je $d(p_1,p_2) = d(p_2,p_1)$.
5. Promatrajmo permutacije $p_1 = (i_1i_2 \ldots i_n)$; $p_2 = (j_1j_2 \ldots j_n)$; $p_3 = (k_1k_2 \ldots k_n)$. Neka je $i_\alpha = j_{\beta(\alpha)} = k_{\gamma(\alpha)}$. Ako je $\gamma(\alpha) + 1 \neq \gamma(\alpha+1)$, onda vrijedi bar jedna od slijedećih nejednadžbi:
   $\beta(\alpha)+1 \neq \beta(\alpha+1)$ ili $\gamma[\beta(\alpha+1)] \neq \beta(\alpha)+1$

Drugim riječima, ako je kod prelaza od p_1 na p_3 narušeno susjedstvo elemenata i_α i $i_{\alpha+1}$ onda će njihovo susjedstvo biti narušeno bilo kod prelaza na p_2 bilo kod prelaza od p_2 na p_3. Iz prednjeg slijedi nejednakost trokuta.

Prema tome vidimo da su zadovoljeni svi aksiomi metričkog prostora.

Neka je zadana $p_o \in \pi$ i radius R - zatvorene kugle, a treba modelirati $p \in \pi$ iz intervala $0 < R \leq n-1$.

Prednje se može realizirati pomoću slijedećeg ALGORITMA.

ALGORITAM

1 korak. Generirati R- slučajnih veličina koje s jednolikom vjerojatnošću primaju cjelobrojne vrijednosti iz segmenta (1,n+1).

2 korak. Urediti slučajne veličine da vrijedi

$$0 < j_1 \leq j_2 \leq \ldots \leq j_k < n+1$$

3 korak. Skup elemenata permutacije p_o razbiti na disjunktne podskupove S_α $(\alpha=1,2 \ldots R+1)$ i to tako da element koji stoji na r-tom mjestu u p_o padne u S_α, ako je $j_{\alpha-1} \leq r < j_\alpha$. Stavimo da je $j_o = 0$, a $j_{r+1} = n+1$.

4 korak. Modeliramo permutaciju p^* od R+1 simbola i saglasno njoj smjestimo skupove S_α. Kao rezultat dobije se $p \in \pi$ za koju vrijedi $d(p,p_o) \leq R$ u lančanoj metrici.

Kao što vidimo, algoritam se sastoji u tome da u polaznoj permutaciji p_o slučajno biramo R-rezova, a zatim razrezane dijelove na slučajan način sastavljamo u novu permutaciju.

Na osnovu prednjeg algoritma izradjen je program za elektronski računar koji poziva subrutine: generator slučajnih brojeva i subrutinu koja formira proizvoljnu permutaciju iz n-simbola.

Neka je sada π prostor permutacija iz n- simbola sa bilo kakvom metrikom. Pretpostavimo da funkcional f- ima jedinstven globalni minimum $f(p_o)$.

Označimo sa s_1 - broj permutacija koje leže od p_o - na udaljenosti $\leq R_1$, a sa s_2 broj permutacija koje leže na udaljenosti $\leq R_2$ i izrecimo ovu definiciju.

DEFINICIJA. Kažemo da je funkcional f zadan u π monoton u prosjeku ako ima jedinstven minimum $f(p_o)$ i ako iz $R_1 < R_2$ slijedi

$$\frac{1}{s_1}\sum_{R_1} f(p) < \frac{1}{s_2}\sum_{R_2} f(p)$$

Pretpostavimo sada da su vrijednosti f(p) distribuirane bilo po normalnom zakonu

$$F(z)=P\{f<z\}=\frac{1}{\sqrt{2\pi}\sigma}\int_{-\infty}^{z}\exp\left[\left(-\frac{x-m}{2\sigma}\right)^2\right]dx \qquad (A)$$

ili log- normalnom

$$G(z)=P\{f<z\}=\begin{cases}\dfrac{1}{\sqrt{2\pi}\sigma}\displaystyle\int_{\alpha}^{z}\frac{1}{1-\alpha}\exp\left[-\left(\frac{\ln(x-\alpha)-\mu}{\sqrt{2}\,s}\right)^2\right]dx, & z\geq\alpha\\ 0 \quad - - - - - - - - - - & z<\alpha\end{cases} \qquad (B)$$

Promatrajmo i adekvatne distribucije na užem intervalu (a,b), tj.

$$F^{*}(y)=\begin{cases}0 \quad - - - - - - - - - - & \text{za } y<a\\ \dfrac{1}{\sqrt{2\pi}\sigma}\displaystyle\int_{a}^{y}\exp\left[-\left(\frac{x-m}{2\sigma}\right)^2\right]dx & \text{za } a\leq y\leq b\\ 1 \quad - - - - - - - - - - & \text{za } y>b\end{cases} \qquad (A')$$

gdje je

$$P=\frac{1}{\sqrt{2\pi}\sigma}\int_{a}^{b}\exp\left[-\left(\frac{x-m}{2\sigma}\right)^2\right]dx$$

odnosno

$$G^{*}(z)=\begin{cases}0 \quad - - - - - - - - - - & \text{za } y<a\\ \dfrac{1}{\sqrt{2\pi}\,sD_1}\displaystyle\int_{a}^{y}\frac{1}{x-\alpha}\exp\left[-\left(\frac{\ln(x-\alpha)-\mu}{2\,s}\right)^2\right]dx, & a\leq y\leq b\\ 1 \quad - - - - - - - - - - & \text{za } y>b\end{cases} \qquad (B')$$

gdje je

$$D_1=\frac{1}{\sqrt{2\pi}\,s}\int_{a}^{b}\frac{1}{x-\alpha}\exp\left[-\frac{\ln(x-\alpha)-\mu}{2\,s}\right]^2 dx$$

Neka su:

m_1 - matematičko očekivanje distribucije zadane sa (A)

M_1 - matematičko očekivanje distribucije zadane sa (A'), tada vrijedi slijedeća tvrdnja

Da bi vrijedilo $M_1 < m_1$ nužno je da bude

$$\frac{a+b}{2} < m_1 \qquad (*)$$

Neka su:

m_2 - matematičko očekivanje distribucije zadane sa (B)

M_2 - matematičko očekivanje distribucije zadane sa (B'), tada vrijedi slijedeća tvrdnja

Da bi vrijedilo $M_2 < m_2$, dovoljno je da bude

$$b < \alpha + e^{\mu} \qquad (**)$$

Prednje tvrdnje nas upućuju da treba razvijati takve algoritme koji će odredjenim postupkom smanjivati radijuse iz kojih se promatraju $p \in \pi$, čime u stvari smanjujemo matematičko očekivanje, a to onda vodi do toga da se smanjuje vjerojatnost izbora loših vrijednosti funkcionala f(p), za razliku od čisto slučajnog izbora gdje to nije slučaj.

U tabeli 1 zadani su statički momenti M_1.

Na slici 1. vidimo rezultate uzorka od 100 vrijednosti funkcionala (sumarnog debalansa) kada su birane $p \in \pi$ na slučajan način.

Dobiveni su rezultati:
Minimum funkcionala = 134,75
Maksimum funkcionala = 4714,5
Matematičko očekivanje = 1488,2

Minimum funkcionala dobiven je kod rasporeda danog u tabeli 2., a maksimum kod rasporeda danog u tabeli 3.

Na slici 2. vidimo rezultate od 100 vrijednosti funkcionala kada se permutacije biraju iz okoline minimuma funkcionala dobivenog na slučajan način, a iz radiusa kugle R = 5 koristeći lančanu metriku. Dobiveni su rezultati:

Minimum funkcionala = 39,8
Maksimum funkcionala=3100,9
Matematičko očekivanje = 1045,56
Raspored koji odgovara minimumu funkcionala naveden je u tabeli 4. a maksimum funkcionala u tabeli 5.

TABELA 1

Broj lopatice	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Statički moment	0	109	-309	-51	-411	-179	420	-145	-301	61	181	231	-162	-341	-261	175
Broj lopatice	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Statički moment	49	0	-69	-51	-46	176	-511	-412	-317	470	79	13	-142	35	-152	-88
Broj lopatice	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
Statički moment	15	-625	- 6	21	-259	- 53	17	11	-259	15	91	-121	475	- 9	-191	81
Broj lopatice	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
Statički moment	15	42	3	-11	-249	22	5	-11	- 5	- 2	17	-113	17	-17	-39	488
Broj lopatice	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
Statički moment	5	25	15	41	5	-26	22	-11	27	-23	-58	75	49	31	-457	-45
Broj lopatice	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
Statički moment	-57	- 9	9	0	-11	-25	- 7	31	65	31	44	21	61	21	51	31

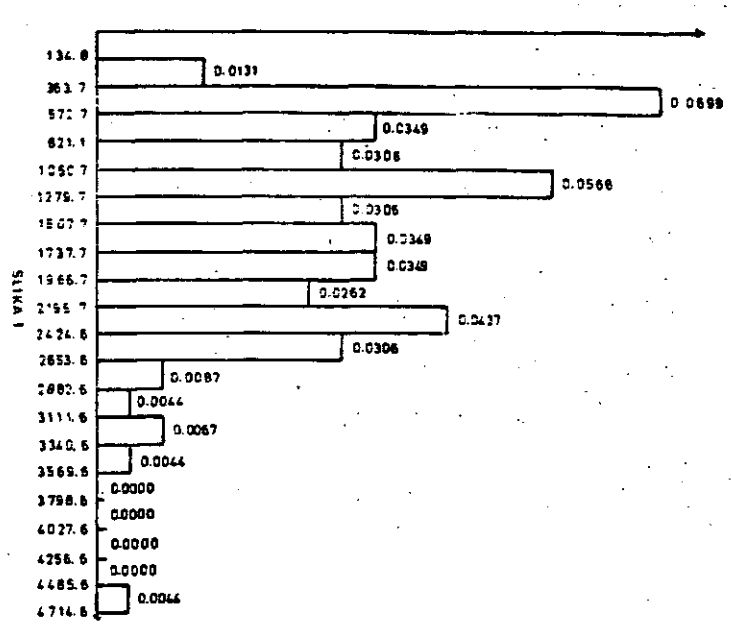


TABELA 2

Broj mjesta	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Broj lopatica	52	73	16	37	13	54	79	60	61	63	86	51	4	87	00	85
Statički moment	-11	27	175	-259	-162	22	-457	-45	-57	9	-25	3	-31	- 7	31	-11
Broj mjesta	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Broj lopatica	89	90	91	92	17	19	23	28	11	53	77	32	12	72	59	75
Statički moment	65	31	44	21	49	-69	-511	13	181	-249	49	-88	231	-11	17	-58
Broj mjesta	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
Broj lopatica	78	64	93	82	56	30	39	60	63	25	74	8	38	94	55	69
Statički moment	31	0	61	- 9	11	35	17	-113	- 39	-317	-23	-145	-53	21	5	5
Broj mjesta	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
Broj lopatice	95	96	47	67	2	35	57	1	15	26	20	29	33	7	5	14
Statički moment	51	31	-191	15	109	- 6	5	0	-281	470	-31	-142	13	420	-411	-341
Broj mjesta	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
Broj lopatice	34	36	22	41	44	46	50	49	64	56	27	40	10	21	31	42
Statički moment	-625	21	176	-259	-121	- 9	42	15	488	-2	79	81	61	-46	-152	15
Broj mjesta	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
Broj lopatice	61	6	9	24	40	45	3	18	62	65	66	88	70	71	43	76
Statički moment	17	-179	-301	-412	11	475	-309	0	-17	5	25	41	-26	22	91	75

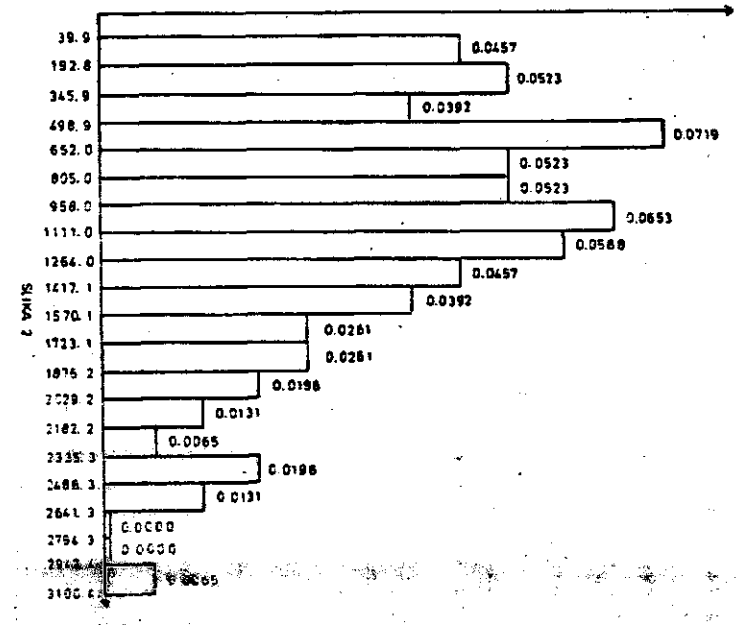


TABELA 4

Broj mjesta	1	2	3	4	5	6	7	8	9	1o	11	12	13	14	15	16
Broj lopatice	39	91	92	17	19	48	1o	21	31	42	61	6	23	28	51	4
Statički moment	17	44	21	49	-69	81	61	-46	-152	15	17	-179	-511	13	3	-31
Broj mjesta	17	18	19	2o	21	22	23	24	25	26	27	28	29	30	31	32
Broj lopatice	87	88	72	59	9	24	4o	65	96	47	45	3	1o	62	66	68
Statički moment	- 7	31	-11	17	-3o1	-412	11	5	31	-191	475	-3o9	0	-17	25	41
Broj mjesta	33	34	35	36	37	3o	39	4o	41	42	43	44	45	46	47	48
Broj lopatice	7o	71	43	76	83	77	32	11	12	75	78	84	93	82	56	37
Statički moment	-26	22	91	75	9	49	-88	181	231	-58	-31	0	61	- 9	-11	-259
Broj mjesta	49	5o	51	52	53	54	55	56	57	58	59	6o	61	62	63	64
Broj lopatice	13	54	79	8o	81	25	74	8	38	94	55	69	95	67	2	35
Statički moment	-162	22	-457	-45	-57	-317	-23	-145	-53	21	5	5	51	15	109	- 6
Broj mjesta	65	66	67	68	69	7o	71	72	73	74	75	76	77	78	79	8o
Broj lopatice	57	1	15	26	2o	29	33	7	5	3o	6o	63	52	73	16	44
Statički moment	5	0	-281	47o	-31	-142	13	42o	-411	35	-113	-39	-11	-27	175	-121
Broj mjesta	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
Broj lopatice	46	53	86	14	34	36	22	41	5o	49	64	58	27	65	89	9o
Statički moment	- 9	-249	- 25	-341	-625	21	176	-259	42	15	488	- 2	79	-11	65	31

TABELA 3

Broj mjesta	1	2	3	4	5	6	7	8	9	1o	11	12	13	14	15	16
Broj lopatice	72	7	22	74	86	26	37	2	12	48	8	5o	64	67	27	8o
Statički moment	-11	42o	176	-23	-25	47o	-259	1o9	231	81	-145	42	488	15	79	-45
Broj mjesta	17	18	19	2o	21	22	23	24	25	26	27	28	29	3o	31	32
Broj lopatice	82	87	91	92	94	18	62	95	1o	69	96	11	16	3o	63	28
Statički moment	- 9	- 7	44	21	21	0	-17	-51	-61	5	31	181	175	35	-39	-13
Broj mjesta	33	34	35	36	37	38	39	4o	41	42	43	44	45	46	47	48
Broj lopatice	29	71	36	51	56	58	39	73	77	49	57	65	81	89	21	47
Statički moment	-142	22	21	3	-11	- 2	17	27	49	15	5	5	-57	65	-46	-191
Broj mjesta	49	5o	51	52	53	54	55	56	57	58	59	6o	61	62	63	64
Broj lopatice	93	32	9	46	15	88	31	3	24	17	54	35	55	6o	53	52
Statički moment	61	-88	-3o1	- 9	-281	31	-152	-3o9	-412	49	22	- 6	5	-113	-249	-11
Broj mjesta	65	66	67	68	69	7o	71	72	73	74	75	76	77	78	79	8o
Broj lopatice	75	76	14	59	78	2o	79	19	44	66	13	23	83	5	4	38
Statički moment	-58	75	-341	17	31	-31	-457	-69	-121	25	-162	-511	9	-411	-31	-53
Broj mjesta	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
Broj lopatice	7o	41	84	85	68	9o	42	4o	43	33	1	6	45	25	34	61
Statički moment	-26	-259	0	-11	41	31	15	11	91	13	0	-179	475	-317	-625	17

TABELA 5

Broj mjesta	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Broj lopatice	63	77	32	11	45	3	18	62	66	68	70	71	43	76	3o	94
Statički moment	9	49	-88	181	475	-3o9	0	-17	25	41	-26	-22	91	75	-55	-21
Broj mjesta	17	18	19	20	21	22	23	24	25	25	26	27	28	29	3o	31
Broj lopatice	55	69	95	67	2	35	57	1	15	26	2o	29	33	7	5	3o
Statički moment	5	5	51	15	109	- 6	5	0	-281	470	-31	-142	13	420	-411	35
Broj mjesta	33	34	35	36	37	38	39	4o	41	42	43	44	45	46	47	48
Broj lopatice	39	6o	63	52	12	75	78	84	93	82	56	37	13	54	79	8o
Statički moment	17	-113	-39	-11	231	-58	31	0	61	-9	-11	-259	-162	22	-457	-45
Broj mjesta	49	5o	51	52	53	54	55	56	57	58	59	6o	61	62	63	64
Broj lopatice	81	25	74	8	91	92	17	19	48	1o	21	31	42	61	6	23
Statički moment	-57	-317	-23	-145	44	21	49	-69	81	61	-46	-152	15	17	-179	-511
Broj mjesta	65	66	67	68	69	7o	71	72	73	74	75	76	77	78	79	8o
Broj lopatice	28	51	4	87	88	72	59	9	24	4o	65	96	47	73	16	44
Statički moment	13	3	-31	- 7	31	-11	17	-3o1	-412	11	5	31	-191	27	175	-121
Broj mjesta	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
Broj lopatice	46	53	86	14	34	36	22	41	5o	49	64	58	27	65	89	9o
Statički moment	- 9	-249	-25	-341	-625	21	176	-259	42	15	488	- 2	79	-11	65	31

LITERATURA:

1 S.Mardešić - Matematička analiza I dio. Školska knjiga, Zagreb 1974.

2 S.Kurepa - Funkcionalna analiza - Elementi teorije operatora. Školska knjiga, Zagreb 1981.

3 D.I.Golenko - Statističeskie modeli v upravlenii proizvodstvom. Statistika 1973.

TURING MACHINES AS A PROGRAMMING LANGUAGE

ŽARKO MIJAJLOVIĆ,
ALEKSANDAR JOVANOVIĆ,

UDK: 681.3.06:519.682

FACULTY OF SCIENCES, DEPT. OF MATHEMATICS, BELGRADE
MATHEMATICAL INSTITUTE, BELGRADE

Abstract

The first motivation for this paper was educational, to supply students with a system in the higher programming language form which supports Turing machine programming. As the beginning elements of a complexity hierarchy, languages TURING 0 and TURING 1 are proposed, which might be of a theoretical interest. Several kinds of semantics (operational, linguistic, denotational, algebraic etc.), were recognized ilustrating the way in which these semantics could be applied in other programming languages.

TURINGOVE MAŠINE KAO PROGRAMSKI JEZIK

Apstrakt

Osnovna motivacija za ovaj rad je edukativnog karaktera, da se studentima da jedan sistem u formi viših programskih jezika, a koji bi služio za podršku programiranja na Turingovim mašinama. Takodje, konstruisani su jezici TURING 0 i TURING 1 kao početni elementi jedne hijerarhije složenosti, što može imati nezavisan teorijski interes. Razmatra se, takodje, nekoliko vrsta semantika (operacijska, lingvistička, denotacijska, algebarska itd.) kao i načini mogućih primena u drugim programskim jezicima.

TURING 0.

Every Turing machine represents some syntactic object, a finite sequence of instructions. Every instruction is a quintuple of the form

(I) $q\ x\ y\ o\ p$

where q, p are states (natural numbers)
$x, y \in V \cup \{\#\}$, V is an alphabet
o is an operation symbol, one of L, R, S,
The informal meaning of the instruction I is: "If the Turing machine is in state q, and reads the cell with the content $x \in V$, then it writes y in the cell, executes the operation o and shifts to state p."

At least three types of semantics can be defined for Turing machines.

Operational semantics. Turing machines are interpreted using "real machines". In this approach the following codes correspond to the basic operations:

LEFT - L
RIGHT - R
STOP - S
SKIP - #

Denotational semantics. If t is a Turing machine over an alphabet V, then the following mapping is correspondent to the machine t

$\underset{\sim}{t} : F \times Z \times \omega \to (F \times Z \times \omega)^{\omega}$ where
$\omega = \{0, 1, 2, \ldots\}$ (the set of states)
$Z = \{\ldots, -2, -1, 0, 1, \ldots\}$ (the set of cells)
$F = Z \to V = \{ f \mid f : Z \to V \}$ (the set of tape descriptions).

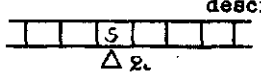


Figure 1.

If $\underset{\sim}{t}(w) = (v_0, v_1, \ldots)$ then v_i represents the result of application of the machine t on input w in the i-th step. Also the following partial function is corresponded

$t^{\wedge} : F \times Z \times \omega \xrightarrow{p} F \times Z \times \omega$

$t^{\wedge}(w) = v$ iff for some i and all $j \geq i$ $v_i = v$.

Linguistic semantics. Every Turing machine over an alphabet V can be represented by some Semi-Thue process (ie. the set of productions over the alphabet V).

This approach is useful in the theory of formal languages. It is used in solving some decision problems in this theory.

TURING 1.

Language TURING 1 is more complex than the language TURING 0, though it is designet having Turing machines as basis as well.

The basic characteristics of TURING 1 programme are:
Every TURING 1 programme is a finite sequence of instructions and commands. The language contains composition and branching of Turing machines. Instructions of the language make a cumulative hierarchy; at the initial stage of the hierarchy there are a few basic instructions. TURING 1 admits direct implementation of new instructions by simple naming of programmes which are built up by use of earlier defined programmes or by use of basic instructions and commands.

1. Hierarchy of TURING 1 programmes.

Let S be some (infinite) set of syntactic objects, for example
$S=\{xv \mid x\in A, v\in V^*\}$ [1], $A=\{A,B,C,\dots,Z\}$, $V=A\cup\{0,1,\dots,9\}$
Hierarchy of TURING 1 programmes is any system
$(T_0, \bar\mu_0, \bar\mu_1, \dots, N_0, N_1, \dots)$ where
$T_0 \subseteq S$, T_0 is called the set of basic (atomic) programmes. $\bar\mu_n$ are partial functions satisfying
$\bar\mu_0 = id_{T_0}$
for all $i\in\omega$ $\bar\mu_i$ is 1-1
$\bar\mu_0 \subseteq \bar\mu_1 \subseteq \dots$
Also, $\bar\mu_i$ and N_i satisfy the following recursive conditions: $N_0 = T_0$;
For $i \geq 1$, T_i is the set of all finite sequences of elements of N_{i-1} and commands
IF x THEN n; GO TO n ($x\in V$, $n\in\omega$).
For all $i\in\omega$ $\bar\mu_i: T_i \to S$ [2], $N_i = N_{i-1} \cup \bar\mu_i(T_i)$
Following the above definition we have the situation:

$$\begin{array}{ccccc} T_0 & \subseteq & T_1 \subseteq \dots & T & T=\bigcup_{n\in\omega} T_n \\ \bar\mu_0 \downarrow & & \bar\mu_1 \downarrow & \dots \bar\mu \downarrow & \bar\mu = \bigcup_{n\in\omega} \bar\mu_n \\ N_0 & \subseteq & N_1 \subseteq \dots & N & N=\bigcup_{n\in\omega} N_n \end{array}$$

1) V^* is the set of all words over V.
2) Index p means that $\bar\mu_i$ is partial, i.e. domain $(\bar\mu_i) \subseteq T_i$

Elements of the set T are called the TURING 1 programmes. If $\tau\in N$ and $\bar\mu(t)=\tau$, then τ is the name of t. N is the set of names and $\bar\mu^{-1}(N)$ is the set of named TURING 1 programmes.

Remark 1: T_0 is the set of basic (atomic) TURING 1 programmes and {LEFT, RIGHT, STOP, DELETE} $\cup$ {PRINT x | $x\in V$} $\subseteq T_0$.

Remark 2. Particularly interesting are the following types of hierarchies
1. (ideal case) Every function $\bar\mu_i$ is total. In this case, clearly, for all $i\in\omega$, $S - N_i$ is infinite.
2. For all $i\in\omega$, N_i is finite. Sometimes it is assumed that every Turing machine has name, e.g. for every Turing machine m there is $t\in T$, such that m is a translation of t and $\bar\mu(t)$ exists.
3. (real case) For all $i\in\omega$, N_i is finite and there is $k\in\omega$ so that for all $i \geq k$, $N_i = \emptyset$.
From now on we assume the first case.

Remark 3. It can be assumed that N_i is recursive for all $i\in\omega$ e.g. for any coding function $\gamma: S \xrightarrow{1-1} \omega$, the set of all codes $N_i' = \{\gamma n \mid n\in N_i\}$ is recursive. The same is assumed for N. Functions $\bar\mu_i$ are effective too, e.g. for some coding functions $\alpha: T \to \omega$, $\beta: N \to \omega$ we have

$$\begin{array}{ccc} T & \xrightarrow{\bar\mu} & N \\ \alpha\downarrow & & \downarrow\beta \\ \omega & \xrightarrow{f} & \omega \end{array} \qquad \bar\mu = \beta^{-1}\circ f\circ\alpha', \quad \alpha' = \alpha\restriction\bar\mu^{-1}(N)$$

f is recursive (moreover it may be assumed that f is primitive recursive).

The above conditions of recursiveness are clearly satisfied in the last two examples of TURING 1 hierarchies.

Definition. $t\in T$ has rank k iff $t\in T_k - T_{k-1}$ ($k\geq 1$). t has rank 0 if $t\in T_0$. Rank definition enables inductive definitions and proofs of properties of TURING 1 programmes.

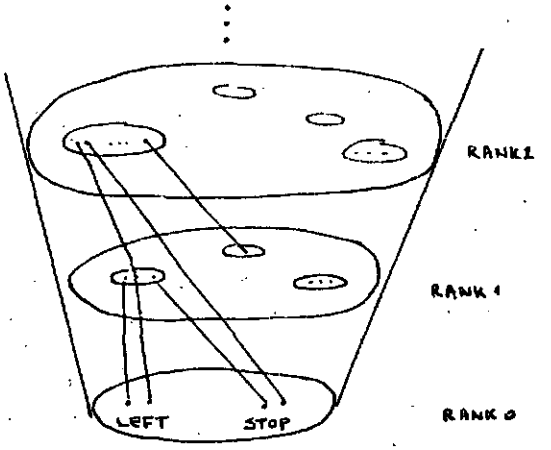


Figure 2: Cumulative hierarchu of TURING 1 programmes

2. Operational semantics of TURING 1 programmes.

To every TURING 1 programme $t \in T$ naturally corresponds some Turing machine. As usual, some mechanical device with tape infinite in both directions, and head which writes and erases signs, and which can be moved for one cell left or right is assumed. Hence, we have the following meaning of earlier mentioned instructions.

LEFT	meaning	move the head to the next left cell and pass to the next instruction; otherwise stop.
RIGHT	"	as above right instead of left
STOP	"	stop
DELETE	"	write blank
PRINT x	"	write sign x

Commands have the following meaning.

IF x THEN n	means	if the current cell content is x then the action transfers to the n-th instruction; otherwise pass to the next instruction.
GO TO n	"	transfer the action to the n-th instruction.

Hence, IF.. THEN.. corresponds to the branching of Turing machines. If $t \in T$, $t = \tau_1 \dots \tau_k$, $\tau_i \in N$, then $\tau_i \tau_{i+1}$ $(i < k)$ designates the compostiton of Turing machines corresponding to the names τ_i, τ_{i+1}

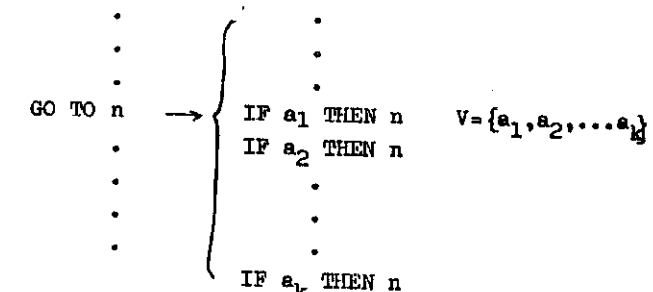
Theorem 1. For every Turing machine m there is a TURING 1 programme equivalent to m.

Theorem 2. For every TURING 1 programme $t \in T_1$ there is a Turing machine equivalent to t.

Theorem 3. For every TURING 1 programme $t \in T$ there is a TURING 1 programme $t \in T_1$, equivalent to t.

Theorem 3 has a special interest, because its proof is used to construct a "compiler" which translate TURING 1 programmes in equivalent Turing machines.

Remark. Every $t \in T$ is equivalent to some $t' \in T_1$ which is built up without use of command GO TO.. The command GO TO n is eliminated as follows



Necessary change of labels in IF..THEN.. are made.

The use of command IF x THEN n is essential in writing TURING 1 programmes as the next theorem shows.

Theorem 4. Let T_0 be any finite subset of S. Let $(T_0, \bar{\mu}_0, \bar{\mu}_1, \dots, N_0, N_1, \dots)$ be any system in which the commands IF.. THEN.. and GO TO .. are not used, i.e. every T_i is a set of finite sequences of elements of N_{i-1}. Then there is a Turing machine m which has no equivalent TURING 1 programme.

Proof of Theorem 1.

We may assume the following for a Turing machine m:

(1) m is represented by a sequence of quadruples of the form $q\ x \begin{Bmatrix} o \\ y \end{Bmatrix} p$, as the quintuple $q\ x\ y\ o\ p$ is equivalent to the pair

$q\ x\ y\ r$

$r\ y\ o\ p$

(2) If a state p occures in m, then for each $a \in V$ there is an instruction in m starting with p a. In this case, the use of skip symbol $\not\#$ is not necessary, so we may assume that $\not\#$ does not occur in m.

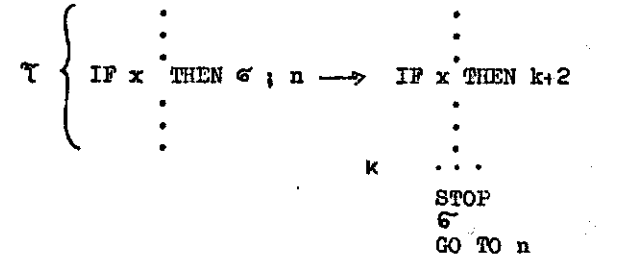
(3) Every permutation of instructions of m gives the same Turing machine, hence we may assume that all instructions of m starting with a same state p are grouped in a block of m.

Also, we introduce the following convention for TURING 1 programmes: Let τ be a sequence of names of TURING 1 programmes, and possibly of lines of the form

$$\text{IF } x \text{ THEN } \sigma\ ;\ n$$

where $x \in V$, $\sigma \in N$, $n \in \omega$.

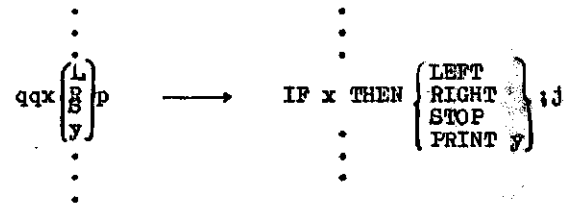
This line in τ is eliminated by the transformation



where k=length(τ).

Thus, τ represents a TURING 1 programme.

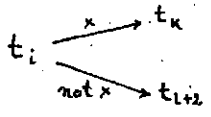
Now, for a Turing machine m which satisfies (1) - (3), the correspondent TURING 1 programme is built up by use of the following transformations:



The labes j is the least index of an instruction in m starting with p, otherwise j=length(m)+1.

Proof of Theorem 2.

Let $t=t_1\ t_2 \dots t_n$ and $i<n$. If $t_i \in T_o$, then $t_i\ t_{i+1}$ denotes composition of Turing machines correspondent to t_i, t_{i+1} if t_{i+1} is an instruction. If t_{i+1} is IF x THEN k, then $t_i\ t_{i+1}$ denotes branching of Turing machines:



Proof of Theorem 3.

Even if this theorem is obvious we supply a proof as it is a typical application of rank function. So, the proof is by induction on rank. Let $t \in T$. Then $t=t_1 t_2 \dots t_n$ or $t \in T_o$. If t_i denote instructions, then rank $(t_i) <$ rank (t), and by induction hypothesis there exist $n^i \in T_1$ equivalent to t_i. Thus, n^i is a programme $n^i_1 \dots n^i_{m_i}$, where $n^i_j \in T_o$. Then t is equivalent to the programme obtained by writing names n^i of programmes n^i in the given order, infered by the order of instructions t_i in t. Also, labels of commands are changed in the following way:

If t_i is of the form IF x THEN s, then the label s is replaced by

$$s' = 1+p+\sum_{\substack{i<s \\ t_i \in N}} \text{length}\,(n^i),$$

p is the number of commands in the sequence $t_1\ t_2 \dots t_s$.

Proof of the Theorem 4.

The class of Turing - calculable functions is representable in Peano's arithmetics. On the other hand the set of functions definable in Peano's arithmetics is not finitely generated (with respect to the function composition).

However, it will be shown (Approximation Theorem) that in a weaker sense the command IF x THEN n can be eliminated.

3. Tree semantics for TURING 1 programmes.

There are at least as many semantics for TURING 1 programmes as there are for Turing machines. However all these semantics are indirect, hence it makes sense to look for some others.

Computation trees.

A binary tree B_t is assigned to every $t \in T$. That tree can be understood as a schematic representation of application of an idealised computer to the computation of a given functional, in our case a TURING 1 programme.

Notation:

$\bar{V} = \{\bar{x} \mid x \in V\}$

V, $\bar{V}$, N are pairwise disjoint.

If length (t) = n then for $i > n$, t_i is defined taking $t_i = s$, $s = \bar{\mu}$ (STOP), $\bar{\mu}$(GO TO ..)$= g$, $g \notin N$.

Definition of the tree $\mathbb{B}_\tau$, $\tau \in N$.

$\mathbb{B}_\tau = (B_\tau, \subseteq)$

B_τ is the set of restrictions to the initial segments of ω, of the below defined functions f.

Hence, we define the branches of the tree $\mathbb{B}_\tau$.

1. dom(f)$=\omega$, or for some $k \in \omega$, dom(f)$=k$[2]. range(f) $\subseteq \omega \times (N \cup V \cup \bar{V} \cup \{g\})$.
2. $f(o) = (o, \tau)$.
3. f satisfies the following recursive conditions: Assume $f(i-1) = (j, \alpha)$.

3.1. Case $\alpha \in N$. Let $\tau_{j+1} = \beta$.
If $\beta \in N \cup \{g\}$ and $\alpha \neq s$, then $f(i)=(j+1,\beta)$
If $\alpha = s$ then dom(f)=i.
If β is the command IF x THEN p, $x \in V$, $p \in \omega$, then $f(i)=(j+1,x)$ or $f(i)=(j+1,x)$.

3.2. Case $\alpha = g$. Then τ_j is the command GO TO p, for some p.
Let $\tau_p = \beta$.
Suppose $p \leq n$. Then:
If $\beta \in N \cup \{g\}$ then $f(i) = (p, \beta)$.
If β is the command IF x THEN q, $x \in V$, $q \in \omega$, then $f(i)=(p,x)$ or $f(i)=(p,x)$.
Suppose $p > n$. Then $f(i)=(n+1,s)$.

3.3. Case $\alpha \in V \cup \bar{V}$. Then τ_j is of the form IF x THEN p, where $\alpha = x$ or $\alpha = \bar{x}$.
Let $\beta = \tau_{j+1}$, $\mu = \tau_p$.

3.3.1. subcase $\alpha = x$.
Suppose $p \leq n$.
If $\mu \in N \cup \{g\}$ then $f(i)=(p,\mu)$.
If μ is the command IF y THEN q, for some y, q, then $f(i)=(p,y)$ or $f(i) = (p,\bar{y})$, $y \in V$.
Let $p > n$. Then $f(i)=(n+1,s)$.

3.3.2. Subcase $\alpha = \bar{x}$.
If $\beta \in N \cup \{g\}$, then $f(i)=(j+1,\beta)$.
If μ is the command IF y THEN q, then $f(i)=(j+1,y)$ or $f(i)=(j+1,y)$, $y \in V$.

Every node of the tree $\mathbb{B}_\tau$ has a designation. If $f \in B_\tau$, dom(f)$=\{0,1,\dots,k\}$ and $f(k)=(j,\alpha)$, then α is the designation of the node f.

Example. The following TURING 1 programme changes 0 to 1 and 1 to 0, to the right from the initial head position.

---

[2] $k = \{0,1,2,\dots,k-1\}$

```
COMPL:    IF  0  THEN 4
          IF  1  THEN 7
          STOP
       4  PRINT 1
          RIGHT
          GO TO  1
       7  PRINT 0
          RIGHT
          GO TO  1
```

p_1	designates	PRINT 1
p_0	"	PRINT 0
s	"	STOP
r	"	RIGHT
g	"	GO TO ...

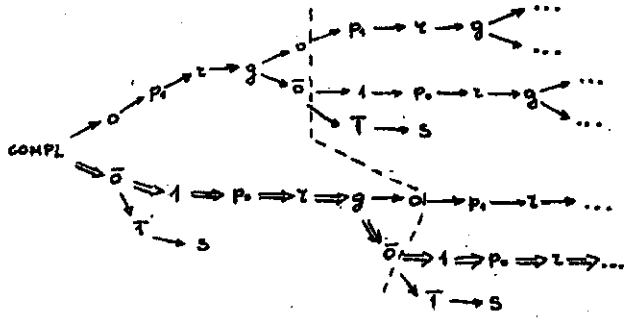


Figure 3

Approximation Theorem.

Let $\tau \in N$ and $v: Z \to V$ be a tape description. If $\tau(v)$ exists then there is $\tau' \in N$, built up of STOP, LEFT, RIGHT, DELETE, PRINT x only ($x \in V$). Also, $\tau(v)$ exists, and $\tau'(v) = \tau(v)$.

Proof.

Induction on rank (τ). Erasing all commands from the finite branch of the tree $\mathfrak{B}_\tau$ which computes v, we obtain τ'.

Basis of the tree $\mathfrak{B}_\tau$.

(Finite representation of the computation tree) The tree $\mathfrak{B}_\tau$ is constructed by a small tree C_τ, by an iterative process. C_τ is defined similarly as the tree B_τ, with the following difference. In the definition of $\mathfrak{B}_\tau$, the clause 1. is substituited with:

for some k dom (f) = k.

The clause 2. is extended with:

if dom (f) = {0,1,...,k} then
f ↾ k is 1 - 1.

Definition.

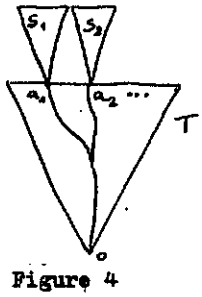


Figure 4

Let T, $S_1,\dots,S_k$ be trees and $a_1,\dots,a_k$ be leaves of the tree T. The tree $T(a_1,\dots,a_k,S_1,\dots,S_k)$ is made by putting the trees $S_1,\dots,S_k$ to the top of the tree T, over the leaves $a_1,\dots,a_k$.

More precisely

$\underset{\sim}{T}(\vec{x},\vec{s}) = \underset{\sim}{T} \cup [0,a_1] \times \underset{\sim}{S}_1 \cup \dots$

It is assumed that S_i are pairwise disjoint.

If T is a tree, then $\underset{\sim}{T}$ is the ordering relation of the tree T.

Further, we define a sequence B^n_τ of trees:

$B^0_\tau = C_\tau$

$B^{n+1}_\tau = B^n_\tau(a_1,\dots,a_k,S_1,\dots,S_k)$

where $a_1,\dots,a_k$ are the leaves of the tree B^n_τ, and $S_1,\dots,S_k$ are (some) subtrees of C_τ.

Then, $\langle B^n_\tau, h_{nm} \mid n \in \omega \rangle$ is one direct system with respect to the natural embeddings

$h_{nm} : B^n_\tau \longrightarrow B^m_\tau \qquad (n \in m)$.

Theorem 6.

$$B_\tau = \varinjlim B^n_\tau .$$

4. Topological semantics.

The domain B_τ of the tree $\mathfrak{B}_\tau = (B_\tau, \subseteq)$ is the subset of Baire space ω^ω (it is the subset of Cantor space 2^ω as well). Writing down the definition of the computation tree $\mathfrak{B}_\tau$, §3, it is easy to see that $\mathfrak{B}_\tau$ is at most Π^0_4 in the arithmetical hierarchy. Hence, the tree semantics for TURING 1 programmes is determined with a functional

$\Phi : N \longrightarrow S(\omega^\omega)$, $S(\omega^\omega) = \{x \mid x \subseteq \omega^\omega\}$

where for $\tau \in N$

$\mathfrak{B}_\tau = (\underset{\sim}{\Phi}(\tau), \subseteq, (0,\tau))$, $\underset{\sim}{\Phi}(\tau) = \{k \restriction \kappa \mid \kappa \in \omega \cup \{\omega\}, k \in \Phi(\tau)\}$.

5. Semi-Thue processes.

Semi-Thue processes can be taken as an alternative semantics for TURING 1 programmes. In one approach, the tape description is almost empty, i.e. if $v: Z \to V$ is a tape description and e stands for blank symbol, then there is $n \in \omega$ such that for all $k \in Z, |k| \geq n$ we have $v(k) = e$.

Definition. The input is any pair (μ, a), where $\mu \in V^+$[3] $a \in Z$.

In the above definition μ is a situation word, a is the head position; if $0 \leq a \leq n$ and $\mu = \mu_0 \mu_1 \dots \mu_n$, then the head is over the a-th cell whose content is μ_{a-1}.

Semi-Thue process $\mathcal{S}_\tau$ assignet to the TURING 1 programme τ is defined by:

Let $\tau = \tau_1 \tau_2 \dots \tau_n$ and input $w = (\mu, a)$.

Productions for $\mathcal{S}_\tau$ are

$w/\tau \longrightarrow w/\tau_1$

for all i, $1 \leq i \leq n$,

if $\tau_i \in N - \{s\}$, then productions

$w/i \longrightarrow iw/j$, $j = i+1$

$iw \longrightarrow \tau_i w$ all belong to $\mathcal{S}_\tau$.

If $\tau_i = s$ then the corresponding production for τ_i is $w/i \longrightarrow w/s$.

---

[3] $V^+ = V^* - \{e\}$, V^* is the set of all words over V.

If $\tau_i = s$, then no production corresponds to τ_i.
If τ_i is IF x THEN k, then the corresponding productions are

$w/i \longrightarrow w/k$ if $\mu_a = x$.
$w/i \longrightarrow w/i+1$ if $\mu_a \neq x$.

If $\tau_k = S$ (or $\tau_{i+1} = S$) then the corresponding production is $w/i \longrightarrow w/s$.
If τ_i is GO TO k, then the corresponding production is
$w/i \longrightarrow w/k$ if $\tau_k \neq S$, otherwise $w/i \longrightarrow w/s$.
We observe the following:

$w/i \longrightarrow w/j$	is read "the result of application of the i-th instruction is the result of application of the j-th instruction"
$w/i \longrightarrow iw/j$	is read "after the application of i-th instruction, the j-th instruction is applied"
$iw \longrightarrow$ RIGHT w	"the result of application of i-th instruction is the result of application of the instruction RIGHT to w"
w/s	is read "The STOP instruction is applied to w"

Example.
The TURING 1 programme COMPL has the following set of productions:

$w = (\mu, a)$, $a \in Z$, $\mu \in V^+$

w/COMPL	$\longrightarrow$	w/1		
w/1	$\longrightarrow$	w/4	if	$\mu_a = 0$
w/1	$\longrightarrow$	w/2	if	$\mu_a \neq 0$
w/2	$\longrightarrow$	w/7	if	$\mu_a = 1$
w/2	$\longrightarrow$	w/3	if	$\mu_a \neq 1$
w/3	$\longrightarrow$	w/s		
w/4	$\longrightarrow$	4w/5		
w/5	$\longrightarrow$	5w/6		
w/6	$\longrightarrow$	w/1		
w/7	$\longrightarrow$	7w/8		
w/8	$\longrightarrow$	8w/9		
w/99	$\longrightarrow$	w/1		

4w $\rightarrow$ PRINT 1w
5w $\rightarrow$ RIGHT w
7w $\rightarrow$ PRINT 0w
8w $\rightarrow$ RIGHT w

The productions for basic TURING 1 programmes:

LEFT — $LEFT(\mu, a) \rightarrow (\mu, a-1)$.

RIGHT — $RIGHT(\mu, a) \rightarrow (\mu, a+1)$.

DELETE — $DELETE(\alpha x \beta, a) \rightarrow (\alpha e \beta, a)$ if $l(\alpha) \overset{4)}{=} a-1$, $1 \leq a \leq l(\alpha x \beta)$
$DELETE(\mu, a) \rightarrow (\mu, a)$ if $l(\mu) < a$ or $0 \geq a$ (is the blank).

PRINT x — $PRINT\ x\ (\alpha y \beta, a) \rightarrow (\alpha x \beta, a)$ if $l(\alpha) = a-1$, $1 \leq a \leq l(\alpha x \beta)$
$PRINT\ x\ (\mu, a) \rightarrow (\mu e^k x, a)$ if $a = 1 + k + l(\mu)$
$PRINT\ x\ (\mu, a) \rightarrow (x e^k \mu, a)$ if $a = -k$

Example.
We ilustrate the application of Semi-Thue process $\mathcal{P}_T$ for τ = COMPL and the input w=(0110,2)

(0110,2)/COMPL $\rightarrow$ (0110,2)/1 $\overset{5}{\rightarrow}$ (0110,2)/2 $\overset{1}{\rightarrow}$
(0110,2)/7 $\rightarrow$ 7(0110,2)/8 $\rightarrow$ PRINT 0(0110,2)/8 $\rightarrow$
(0010,2)/8 $\rightarrow$ 8(0010,2)/9 $\rightarrow$ RIGHT(0010,2)/9 $\rightarrow$
(0010,3)/9 $\rightarrow$ (0010,3)/1 $\overset{5}{\rightarrow}$ (0010,3)/2
(0010,3)/7 $\rightarrow$ 7(0010,3)/8 $\rightarrow$ PRINT 0(0010,3)/8 $\rightarrow$
(0000,3)/8 $\rightarrow$ 8(0000,3)/9 $\rightarrow$ RIGHT(0000,3)/9 $\rightarrow$
(0000,4)/1 $\overset{0}{\rightarrow}$ (0000,4)/4 $\rightarrow$ 4(0000,4)/5 $\rightarrow$ PRINT
1(0000,4)/5 $\rightarrow$ (0001,4)/5 $\rightarrow$ 5(0001,4)/6 $\rightarrow$
RIGHT (0001,4)/6 $\rightarrow$ (0001e,5)/6 $\rightarrow$ (0001e,5)/1 $\overset{0}{\rightarrow}$
(0001e,5)/2 $\overset{1}{\rightarrow}$ (0001e,5)/3 $\rightarrow$ (0001e,5)/s.

The above interpretation allows an extended notion of TURING 1 programmes, that is, programmes which are not ranked. To be more precise, it is possible to build up recursive structures, i.e. such programmes which are already called by its subprogrammes. It is easiest to see this idea in the following (celebrated) example of factorial function.

```
FACTORIAL: COPY
           MINUS 1
           IF 0 THEN 7
           FACTORIAL
           MULTIPLY
         6 STOP
         7 LEFT
           PRINT 1
           LEFT
           GO TO 6
```

To explain this programme, we remind to the standard notation for Turing machines: a^x denotes a block of a's in x consecutive cells, $a \in V$, and a^x means that the head of the machine is on the first cell of the block. A positive integer x is represented by 1^{x+1}, and empty cell by 0.

Then COPY acts as follows $1^x \longrightarrow 1^x 0 1^x$
MINUS 1 acts as follows $1^{x+1} \longrightarrow 1^x 0$
MULTIPLY acts as $1^{x+1} 0 1^{y+1} \longrightarrow 1^{xy+1}$

It is easy to write down TURING 1 programmes which have the intended action of the above instructions. An example of computation under FACTORIAL function:

$1^4 \rightarrow 1^4 0 1^4 \rightarrow 1^4 0 1^3 0 \rightarrow 1^4 0 1^3 0 1^2 0 \rightarrow 1^4 0 1^3 0 1^2 0 \rightarrow$
$1^4 0 1^3 0 1^2 0 1^2 0 \rightarrow 1^4 0 1^3 0 1^2 0 1 0 \rightarrow 1^4 0 1^3 0 1^2 0 1 0 1 \rightarrow 1^4 0 1^3 0 1^2 0 1 0 0 \rightarrow$
$1^4 0 1^3 0 1^2 0 1^2 0 \rightarrow 1^4 0 1^3 0 1^2 0 0 2 0 \rightarrow 1^4 0 1^3 0 0^6 \rightarrow 1^7 0 0^7$.

We see the following:

- When FACTORIAL is used as an instruction, i.e. if it is called during an execution of

4) $l(\alpha)$ is the length of the word α.

a programme, then STOP acts as RETURN in some higher programming language. The star above an arrow in the example denotes call of the instruction named FACTORIAL. Thus, the depth of the use of this instruction is 3.

As an illustration of the Approximation Theorem, for the input w=(0110,2) and τ = = COMPL we have the following TURING 1 programme without commands :

```
PRINT 0
RIGHT
PRINT 0
RIGHT
PRINT 1
RIGHT
STOP
```

$\tau(w) = \tau'(w)$.

In fact this is a destinguished branch of the tree $\mathbb{B}_\tau$.

6. Denotational (Set theoretical) semantics.

This semantics is based on denotational semantics of D. Scott and C. Strachey. The domains are defined as inputs and outputs of TURING 1 programmes. Inputs and outputs are tape descriptions and head positions. We define the domain D with

$D = F \times Z$ where F is a set of functions from Z to V.

Every $f \in F$ is called a tape description; every $z \in Z$ denote a head position. Instead of tape description we also say valuation or an assignement. The following cases could be distinguished:

1. $F = Z \to V$.
2. $F = \{f \mid f:Z \to V$ and for all $z \in Z$, except finitely many, $f(z)=e\}$
3. $F = V^+$.

Obviously 2. and 3. are equivalent. Let us suppose that we have any of the above cases. Three kinds of interpretations could be noticed for $\tau \in N$:

$\sigma : N \longrightarrow (D \to D^\omega)$

$\mu : N \longrightarrow (\omega \times D \to \omega)$

$\Pi : N \longrightarrow (D \rightharpoonup D)$.

Informal meaning for σ, μ, Π:

Let $\tau \in N$, τ is the name for $t = \tau_1 \tau_2 \ldots \tau_n$; for $w \in D$, $\sigma(\tau)(w)=(v_1 v_2 \ldots)$ represents the computation process for the input w, i.e.

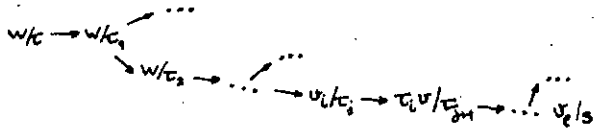


Figure 5.

which represents the "value" of τ for the input w, through the correspondent branch of the computation tree $\mathbb{B}_\tau$.

$\mu(\tau)$ (i,w) is the label of the instruction in τ applied in i-th step, i.e.

$\mu(\tau)(i,w)=j$, where τ_j is applied in i-th computation step.

$\Pi(\tau)$ is a partial function $\Pi(\tau): D \rightharpoonup D$ corresponding to the computation, i.e. $\Pi(\tau)(w)=v_1$ on the preceding picture (s means STOP). Clearly, it is partial function, because there are infinite branches in $\mathbb{B}_\tau$; for inputs computed through these branches $\Pi(\tau)$ is not defined or more precisely, $\Pi(\tau)$ diverges for these inputs.

Note. By König's lemma, for all $\tau \in N$, if $\mathbb{B}_\tau$ is infinite then $\mathbb{B}_\tau$ has an infinite branch. However, this does not mean anything, or at least, it does depend on the type of the domain as the following example shows.

Example.

```
IF 0 THEN 4          0 designates blank
RIGHT
GO TO 1
STOP
```

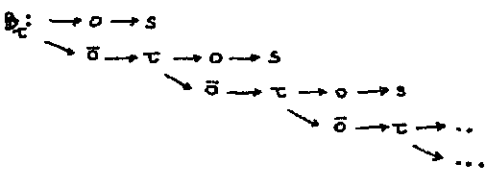


Figure 6.

The machine corresponding to τ searches for the first blank to the right.

Hence:

1. If $F = Z \to V$ then there is an input for which τ diverges, e.g.
   f given by $\forall x \in Z \quad f(x) = 1$.
2. If $F \subseteq V^Z$ is a set of almost constant functions, then τ converges for every input (though there are no upper bound for the computations);

Hence König's lemma "does not" hold in the case 2.

Question. If $F = V^Z$ and if D_τ is infinite, is there an input w for which τ diverges?

Using the computation tree $\mathbb{B}_\tau$, $\sigma(\tau)$ is easily defined.

Now we give formal definitions of the interpretations σ, μ, Π.

Let: $f = \sigma(\tau)$, $c = \mu(\tau)$, $w \in D$, $w=(f,a)$,

$c = \langle c_1 c_2, \ldots \rangle$, $\mu(\tau)(i,w)=c_i$, $f(w)=(v_1 v_2 \ldots)$.

We define f(w) inductively. It should be noticed that the definition is on the rank of the complexity of elements of N.

Let us suppose that v_i, c_i are defined.

Let $k=c_i$. We have the following cases:

1. $\tau_k \in N - \{STOP\}$.

1.1. $\tau_{k+1} \in N$, then $v_{i+1} = \sigma(\tau_{k+1})(v_i)$, $c_{i+1} = c_i+1$.

1.2. if τ_{k+1} is a command, then $v_{i+1} = v_i$, $c_{i+1}=c_i+1$.

if τ_k is STOP, then $v_{i+1}=v_i$, $c_{i+1}=c_i$.

2. if τ_k is GO TO j, then $v_{i+1}=v_i$, $c_{i+1}=j$.

3. if τ_k is IF x THEN j, then $v_{i+1}=v_i$,

$$c_{i+1} = \begin{cases} c_i + 1 & \text{if } \mu_a \neq x \\ j & \text{otherwise.} \end{cases}$$

We say that τ converges to v if for input w, in $\sigma(\tau)(w)=(v_1,v_2,\ldots)$, there is i such that τ_{c_i} is STOP and $v=v_i$. In that case we define $\Pi(\tau)(w)=v$. If there is no i for which τ_{c_i} is STOP, we say that τ diverges in w and write $\tau(w)\uparrow$.

Note that IF .. THEN .. and GO TO .. could be defined as functionals as follows.

$$I : N \times \omega \longrightarrow V \times \omega \longrightarrow D \longrightarrow D$$

$$\sigma(\text{IF x THEN j}) \overset{df}{=} \text{i-th instruction of } \tau$$

$$I(\tau,i)(x,j)(w) = \begin{cases} \sigma(\tau_j)(w) & \text{if } \mu_a = x \\ \sigma(\tau_{i+1})(w) & \text{otherwise} \end{cases}$$

$$G : N \times \omega \longrightarrow \omega \longrightarrow D \longrightarrow D$$

$$\sigma(\text{GO TO j}) = \sigma(\tau,i)(j)(w) = \sigma(\tau_j)(w)$$

(when GO TO j is i-th instruction of τ)

§7. Programming system.

Software environement which supports programming in TURING 0 and TURING 1, is developed and implemented on a DEC 11/70 computer of Faculty of sciences, Dept. of Mathematics, Belgrade. The package is written in BASIC and is used for training by students.

The system consists of eight functional segments, as shown on Figure 7 and uses the following files.

T0S	- "source" TURING 0 programmes (i.e. with faults)
T0SD	- T0S directory
T0	- corect TURING 0 programmes
T0N	- names of programmes in T0, i.e. T0 directory
T1S	- "source" TURING 1 programmes (i.e. with faults)
T1SD	- file T1S directory
T1	- corect TURING 1 programmes
T1N	- file T1 directory
T0R	- system description and user's guide

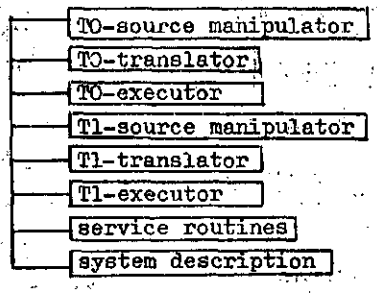


Figure 7

T0-manipulator provides usual editing facilities available in terminal operation, as well as the maintenance of the "dirty TURING 0 programmes" file - T0S.

T0-translator checks syntactic correctness of the entered TURING 0 programme and maintaines the files T0N, T0, T1N, T1.

The main duty of T0-executor is to supervise the TURING 0 programme execution, entering and checking the data as well. The programme execution can be either direct or can use Semi-Thue processes, in which case TURING 0 programme is first transformed in Semi-Thue productions, followed by the word transformation (i.e. the corresponding Semi-Thue system execution).

T1-source manipulator performes similar tasks as T0-source manipulator, but with TURING 1 "source" text.

T1-translator checks TURING-1 programme for correctness, writes the correct TURING 1 programme tree in file T1, translates TURING 1 programme in TURING 0 programme (optional) and saves the TURING 0 translation, thus updating the files T0N, T0, T1N, T1. Three ways of execution of TURING 1 programme are supported (T1-executor): TURING 0 translation execution, direct execution (using TURING 1 programme structure tree), and execution via corresponding Semi-Thue process.

1 name of T1 programme
2 number of input arguments
3 number of output arguments
4 address of name in T1
5 address of name in T0 (if exists)
6 comment
7 final state

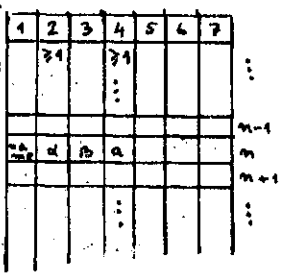


Figure 8: file T1N structure

n is the addres of the name of the programme in the file T1N

n_i is the address of the i-th instruction of the programme in the file T1

s 0 means that this programme has translation in the file T0

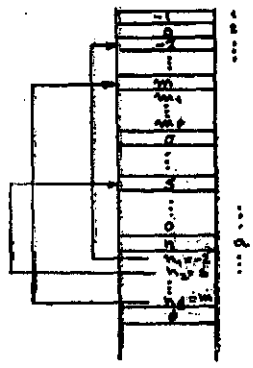


Figure 9: file T1 structure

t_i are instructions of TURING 0 programme t.

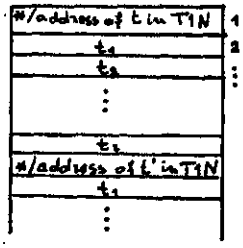
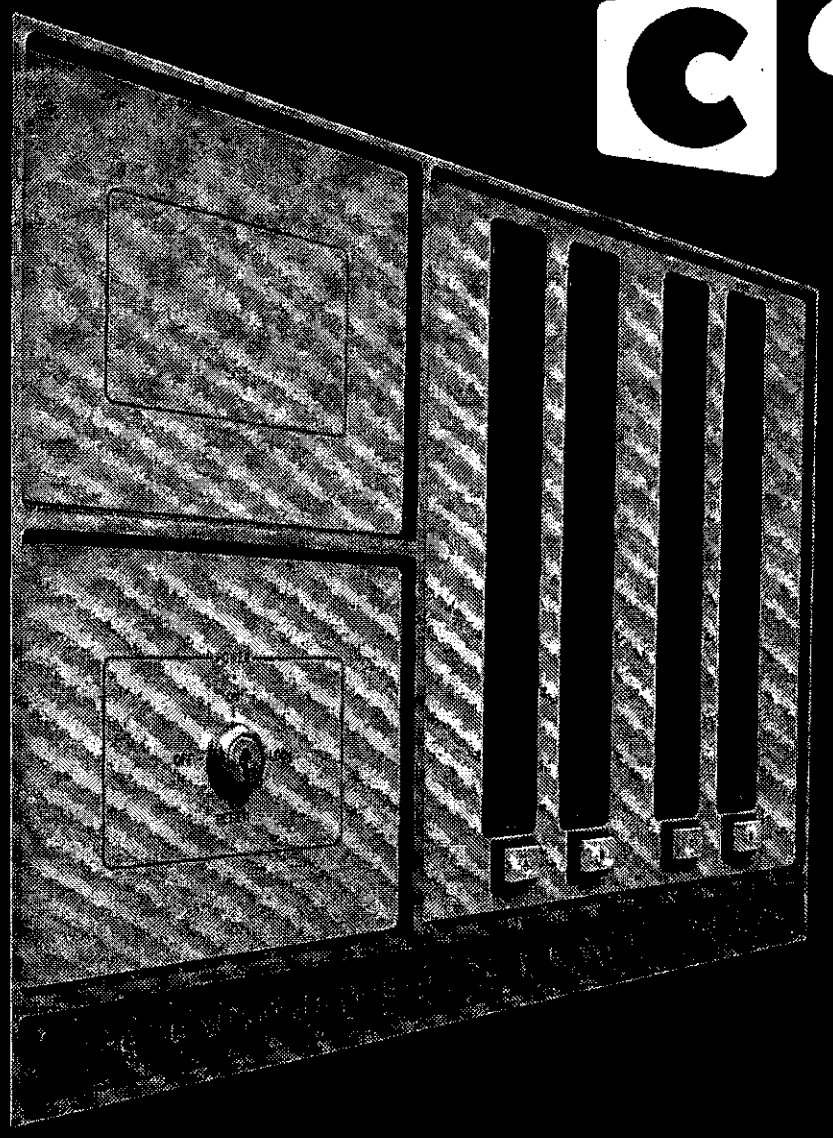


Figure 10: file T0 layout

BIBLIOGRAPHY.

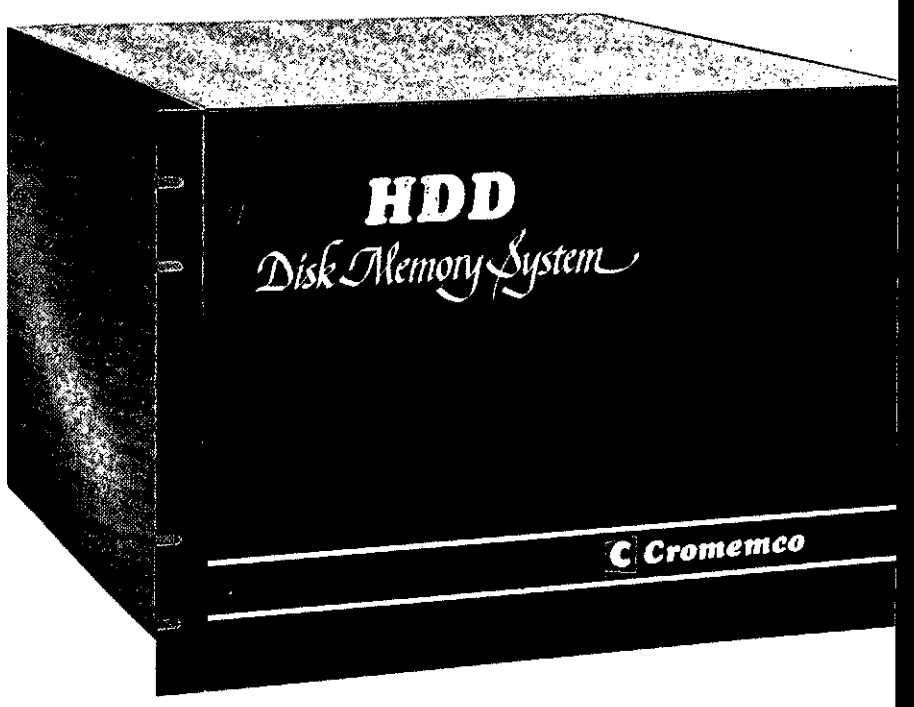
1. S.Alagić, Mathematical Foundation of Programming Languages, Informatika 74, Bled
2. J.E. Donahue, Complementary Definitions of Programming Language Semantics, Lecture Notes in Computer Science, (eds) G.Goos, I. Hartmans, 42, Springer - Verlag (1976)
3. M.J.C. Gordon, The Denotational Description of Programming Languages, Springer - - Verlag, New York (1976).
4. C.A.R. Hoare, P.E. Laner, Consistent and Complementary Formal Theories of the Semantics of Programming Languages, Acta Informatica, 3, (1974).
5. D.Monk, Introduction to Mathematical Logic, Springer Verlag, Berlin 1976.
6. D. Scott, Data Types as Lattices, SIAM Journal of Computing, (1976) Vol 5.
7. D. Scott. Logic and Programming Languages, Communication of the ACM, Vol 20, No 9 (1977)
8. R.D. Tonnent, The Denotational Semantics of Programming Languages, Communications of the ACM, Vol 16, No 8 (1976)



Cromemco
System Three Computer

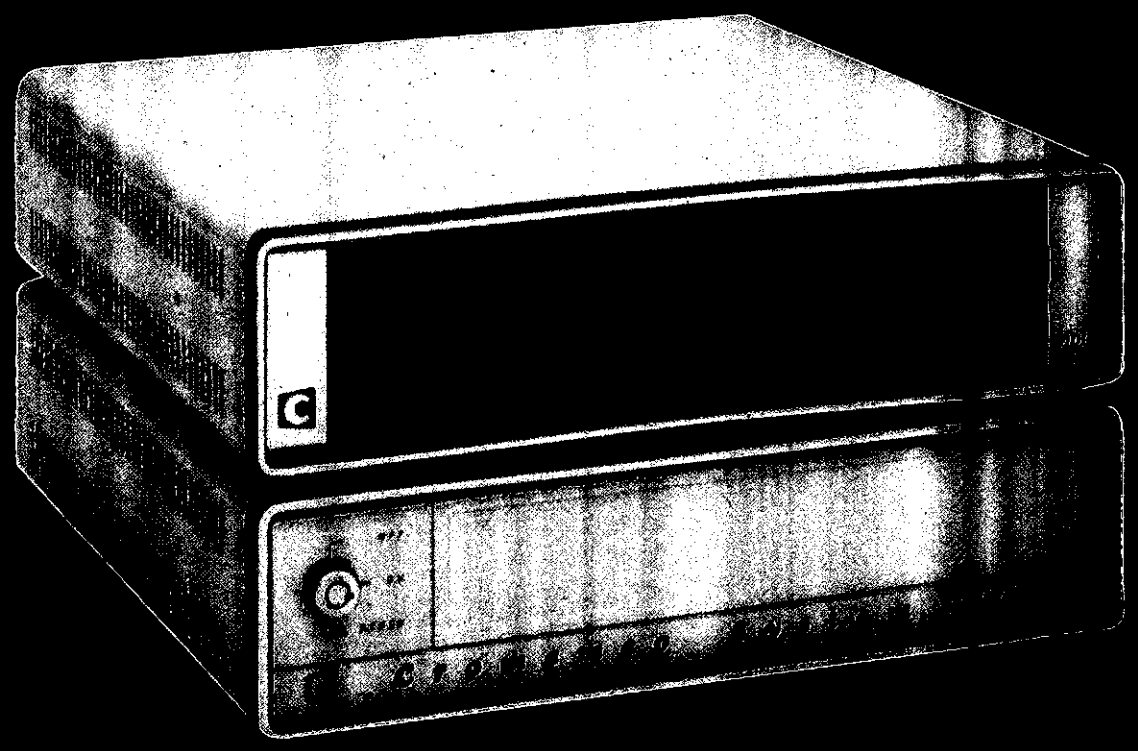
Mikro-kompjutor za profesionalnu upotrebu

11 ili 22 Megabyte Winchester Hard disk



Kolor grafički sistem visoke rezolucije

System zero – namijenjen za specijalne aplikacije



Ekskluzivni zastupnik za SFR Jugoslaviju:

agromarketing

41000 Zagreb, B. Adžije 7/1, P.P. 5
Telefon: (041) 417 662, telex: 21741

POSTUPCI ZA SPREČAVANJE POTPUNOG ZASTOJA I PERMANENTNOG BLOKIRANJA RAČUNARSKOG SISTEMA NA MODELU GRAFA SISTEMA

NIKOLA HADJINA

SVEUČILIŠNI RAČUNSKI CENTAR – ZAGREB

UDK: 681.326:519.17

U radu su, na osnovu nužnih i dovoljnih uvjeta za potpuni zastoj i permanentno blokiranje, izvedene i dokazane osnovne tvrdnje za očuvanje sigurnog stanja sistema. Na osnovu tih tvrdnji uvedeni su postupci za sprečavanje pojave potpunog zastoja i permanentnog blokiranja.

ALGORITHMS FOR PREVENTION OF DEADLOCKS AND PERMANENT BLOCKING IN A COMPUTER SYSTEM USING SYSTEM GRAPH. Based on necessary and sufficient conditions for deadlock and permanent blocking, defined by directed graphs, assertions for safety state are introduced and proved. By means of these assertions algorithms for prevention of deadlocks and permanent blocking are introduced.

1. UVOD

Sprečavanje potpunog zastoja [1] predstavlja poseban problem u izgradnji računarskih sistema, tj. kod usvajanja strategije pridjeljivanja sredstava. Sve metode i postupci vezani su na jedan zajednički osnovni princip da je u postupcima pridjeljivanja potrebno održati sigurnost stanja sistema. Zahtjev za sredstvom bit će zadovoljen ukoliko je novo nastalo stanje pridjeljivanja sigurno. Uz takvu pretpostavku nije moguće doći u stanje potpunog zastoja. Dakle, opće je rješenje za sprečavanje zastoja da su sva stanja pridjeljivanja sigurna. Za višestruko korištenje sredstava trivijalno rješenje za sprečavanje zastoja i permanentnog blokiranja je da se samo po jednom procesu dozvoli držanje sredstava, što dovodi u biti do jednoprogramskih operacija umjesto višeprogramskih operacija, te se izbacuje iz upotrebe kao nepraktično.

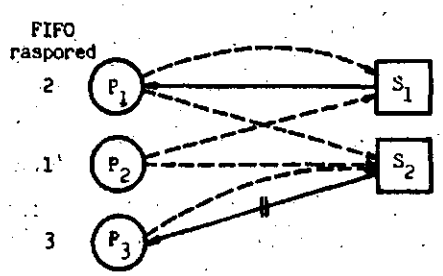
Mnogo praktičnije je rješenje da svaki proces zahtijeva i dobije sva sredstva koja mu trebaju. Proces ne može biti blokiran, jer u toku izvođenja ne traži dodatna sredstva, nego ih samo oslobađa, tj. proces može imati samo pridjeljena sredstva, ili samo zahtjeve za sredstvima, ali ne oboje. Na taj način su sva stanja sigurna i sistem ne može doći u stanje potpunog zastoja. Loša strana ovog rješenja je da su sredstva zauzeta puno prije nego se stvarno koriste. To dovodi do niskog koeficijenta iskorištenja sredstava, odnosno do niskog stupnja propusnosti računarskog sistema.

2. PERMANENTNO BLOKIRANJE

To je situacija gdje se neki zahtjevi procesa nikada ne dopuštaju. Permanentno blokiranje se događa u potpunom zastoju, ali se isto tako događa kada se nekim procesima nikada ne dopuštaju zahtjevi, iako bi oni bili mogući, tj. ne bi izazvali potpuni zastoj [2]. Takav uvjetni zastoj može nastupiti iako ne postoji potpuni zastoj, a prouzrokovan je rasporedivanjem procesa u izvođenje po nekom pravilu (raspoređivač procesa).

Primjer

Neka se sistem sastoji od (a) dva tipa sredstava S_1 i S_2, svaki sa po jednom jedinicom i (b) od tri procesa P_1, P_2 i P_3. Potpuni zastoj će se dogoditi u ovom sistemu ako proces P_1 drži S_1, a zahtijeva S_2, dok proces P_2 drži S_2 a zahtijeva S_1. Neka raspoređivač procesa poslužuje po redoslijedu prijave (engl. first-in-first-out; FIFO) i neka u početnoj situaciji proces P_1 drži sredstvo S_1, a P_3 drži sredstvo S_2. Neka P_1 i P_2 zahtijevaju sredstvo S_1 i S_2 dok P_3 zahtijeva samo S_2. Neka prvo P_2 zahtijeva S_2, nakon toga P_1 zahtijeva S_2 i konačno P_3 oslobađa S_2. Navedena situacija prikazana je na slici 1.



Slika 1. Uvjetni zastoj

Pune strelice označavaju pridijeljena sredstva, a iscrtkane zahtijevana sredstva. Presječena puna strelica znači oslobađanje sredstva.

Budući da se S_2 zahtijeva prvo od P_2 po FIFO pravilu S_2 mora biti pridijeljen prvo procesu P_2 a onda P_1. Međutim, kad bi se S_2 pridijelio procesu P_2, sistem ne bi više bio u sigurnom stanju. Prema tome, raspoređivač neće pridijeliti sredstvo S_2 dok dopuštanje zahtjeva ne rezultira u sigurnom stanju. No, to se neće dogoditi dok P_1 ne oslobodi sredstvo S_1, a P_1 ne može osloboditi sredstvo S_1 dok se ne dozvoli zahtjev procesa P_1 za sredstvom S_2. Dakle, iako je sistem bio uvijek u sigurnom stanju, bilo je nemoguće dopustiti zahtjeve procesa P_1 i P_2. Ta situacija se naziva uvjetni zastoj, budući je raspoređivač procesa zbog FIFO pravila uveo logičke uvjete, pored onih za pridjeljivanje sredstava, te je na taj način blokiran siguran zahtjev procesa P_1 za sredstvo S_2.

Kad god postoji mogućnost raspoređivača procesa da blokira sigurno stanje, potrebno je verificirati da li je sistem ušao u uvjetni zastoj. Ovaj tip zastoja se može izbjeći kršenjem pravila raspoređivanja. Raspoređivaču procesa se nameće uvjet ekspeditivnosti koji zahtijeva da raspoređivač procesa dozvoljava zahtjeve tako dugo, dok su oni sigurni. No i pored ovog uvjeta, može nastupiti situacija permanentnog blokiranja u nekim slučajevima. Npr., neka sistem posjeduje (a) jedno sredstvo S sa po dvije jedinice i (b) tri procesa P_1, P_2 i P_3. Procesi P_1 i P_2 zahtijevaju po jednu jedinicu sredstva S, a proces P_3 zahtijeva dvije jedinice. U početnom stanju jedna jedinica sredstva je pridijeljena procesu P_1. Za takav sistem se lako pokazuje da postoji beskonačna sekvenca pridjeljivanja, tako da proces P_3 ne može nikada zadovoljiti svoj zahtjev. Ako sistem sadrži konačan skup procesa i svaki proces završava nakon konačnog vremena, tada se beskonačna sekvenca ne može dogoditi.

Za gore navedeni sistem procesa i sredstava postoji slijedeća situacija. Svaki puta kad proces P_1 ne drži sredstvo prije nego P_2 oslobađa svoja sredstva, i svaki puta kada P_2 ne drži sredstva, P_2 zahtijeva sredstvo prije nego P_1 oslobađa svoja sredstva. Ako je prioritet procesa P_3 niži od prioriteta P_1 i P_2, tada se procesu P_3 neće nikada dozvoliti zahtjev. Ako proces P_3 zahtijeva obje jedinice, i ako je primjenjen uvjet ekspeditivnosti, tada P_3 neće nikada zadovoljiti svoj zahtjev, bez obzira na njegov prioritet. Prema tome je P_3 permanentno blokiran, a raspoređivač će zauvijek posluživati zahtjeve procesa P_1 i P_2. Dakle, iako je potpuni zastoj spriječen, moguće je permanentno blokiranje.

Uz uvjet ekspeditivnosti na raspoređivač procesa se može primijeniti uvjet eventualnosti koji dozvoljava da raspoređivač blokira sigurne zahtjeve, ali samo na konačno vrijeme.

3. SPREČAVANJE POTPUNOG ZASTOJA NA MODELU GRAFA SISTEMA

Za opći postupak sprečavanja od zastoja potrebno je unaprijed poznavati maksimalne zahtjeve procesa za sredstva. Maksimalni zahtjevi procesa predstavljaju najveći broj jedinica nekog sredstva koja proces treba u nekom trenutku. Ti zahtjevi se mogu spremiti u matricu zahtjeva $[c_{ij}]$; $i=1,\ldots,n$; $j=1,\ldots,m$, gdje c_{ij} predstavlja maksimalni zahtjev procesa P_i za sredstvom S_j.

Za graf sistema postoji, kako je već rečeno, dodatno ograničenje za svaki proces P_i i sredstvo S_j

$$|(P_i,S_j)| + |(S_j,P_i)| \leq c_{ij} \leq t_j$$

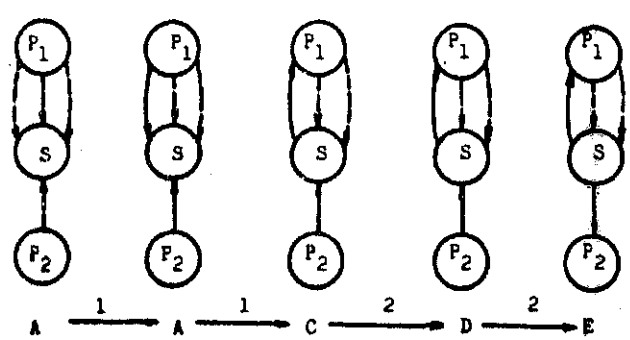
tj. suma pridijeljenih i zahtijevanih sredstava za svaki proces ne smije prekoračiti maksimalni zahtjev, a taj mora biti manji ili jednak broju raspoloživih sredstava.

Stoga se uvodi graf maksimalnih zahtjeva stanja S, kao graf sistema stanja S sa n_{ij} brojem dodatnih strelica zahtjeva (P_i,S_j) za svaki proces P_i i sredstvo S_j, gdje je

$$n_{ij} = c_{ij} - |(P_i,S_j)| - |(S_j,P_i)| .$$

Tako graf, dobiven iz stanja S, predstavlja najlošiji slučaj budućeg stanja $T(S \stackrel{\bullet}{-} T)$ koje je dobiveno uzastopnim zahtjevima za sredstvima ograničenim maksimalnim zahtjevom.

Slika 2. predstavlja graf maksimalnih zahtjeva. Dodatne strelice zahtjeva su iscrtkane, uz pretpostavku da je t=3.



Slika 2. Graf maksimalnih zahtjeva

Zastoj se u ovakvom grafu može spriječiti ukoliko se ne dozvoli pridjeljivanje ako ono ne dovodi do potpuno reducibilnog grafa [1]. Potpuna reducibilnost ovakovog grafa znači da i najlošija situacija stanja pridjeljivanja može biti uspješno obrađena. Valjanost ovog postupka dokazana je u [1].

Postupak redukcije, koji određuje da li je graf potpuno reducibilan, potrebno je da zahtijeva redukciju prvo s procesom P_i. Ako je graf sistema bio potpuno reduciran prije operacije pridjeljivanja, tada on ostaje takav i nakon pridjeljivanja sredstava procesu P_i. Ovaj rezultat je analogan postupku kontinuirane detekcije koja je utvrđena u [1].

Kada sredstva sadrže samo po jednu jedinicu, tada postoje efikasniji postupci testiranja nego redukcijama grafa maksimalnih zahtjeva. Pridjeljivanje proizvodi potpuno reducibilni graf onda i samo onda ako proces za koji se vrši pridjeljivanje ne leži na ciklusu u grafu maksimalnih zahtjeva. U tom slučaju se koristi slijedeći način detekcije.

Iz grafa maksimalnih zahtjeva se napravi neusmjereni *graf sistema. Neusmjereni ciklus se definira kao sekvencija* neusmjerenih lukova (a,b), (b,c)...(r,a) u kojoj se svaki luk pojavljuje samo jednom. Luk (a,b) u neusmjerenom grafu je isto što i (b,a). Sada se sigurnost stanja sistema dokazuje ispitivanjem postojanja ciklusa.

Teorem 1.

Stanje u grafu maksimalnih zahtjeva na sredstvima sa po jednom jedinicom je sigurno onda i samo onda ako neusmjereni graf maksimalnih zahtjeva ne sadrži neusmjereni ciklus.

Dokaz

(a) Neka usmjereni graf maksimalnih zahtjeva ne sadrži ciklus. Tada graf maksimalnih zahtjeva (usmjereni) sigurno ne sadrži ciklus, a to povlači da buduće stanje nije stanje potpunog zastoja.

(b) Neka neusmjereni graf sadrži ciklus. Lukovi neusmjerenog grafa mogu se promijeniti u strelice i neusmjereni ciklus postaje usmjereni. Dakle, graf sistema sadrži ciklus, dovoljan uvjet za zastoj za slučaj sredstava sa po jednom jedinicom.

Budući da su sva stanja neusmjerenog grafa identična, dokazali smo i tvrdnju da su sva stanja u grafu sistema sa maksimalnim zahtjevima sigurna onda i samo onda ako neusmjereni graf sistema ne sadrži neusmjerene cikluse.

Sprečavanje zastoja bazirano na maksimalnim zahtjevima još uvijek dovodi do neiskorištenja sredstava, jer procesi ponekad ne troše maksimalan broj deklariranih sredstava odjednom. Iz tog razloga za slučaj sredstava sa po jednom jedinicom postupak se može realizirati preko matrice povezanosti grafa sistema [1]. Proces P_i koji ulazi u graf sistema s određenim zahtjevima, ulazi i u matricu povezanosti koja definira njegovu povezanost u grafu sistema. Prema tome, na matrici povezanosti potrebno je provesti operacije koje su izomorfne na grafu sistema.

Teorem 2.

Potpuna reducibilnost grafa sistema G odgovara nul-matrici povezanosti A.

Dokaz

Slijedi direktno iz definicije matrice povezanosti i potpuno reducibilnog grafa [1]. Potpuno reducibilni graf sistema sastoji se od izoliranih čvorova, a to odgovara samim nulama u matrici povezanosti grafa, tj. nul-matrici.

Teorem 3.

Redukcija grafa sistema s procesom P_i odgovara postavljanju i-tog retka i j-tog stupca u matrici povezanosti A u nul-redak, odnosno nul-stupac.

Dokaz

Pošto redukcija grafa s procesom P_i proizvodi izolirani čvor, čiji je ulazni i izlazni stupanj jednak nuli, tada po definiciji, da je suma ulaznih i izlaznih stupnjeva čvorova grafa jednaka ukupnom broju lukova u grafu, proizlazi ispravnost gornje tvrdnje.

Prema tome se, na osnovi gornja dva teorema provodi postupak sprečavanja zastoja na matrici povezanosti i to na slijedeći način.

Kod ulaska procesa P_i u sistem potrebno je ispitati da li je redak matrice, koji se formira ulaskom procesa P_i u graf sistema, nul-redak. Ako jest, tada prema teoremu 3 u [1] proces P_i ne utječe na izmjenu sigurnog stanja. Ako redak nije nul-redak, tada se matrica povezanosti proširuje novim retkom i novim stupcem. Daljnje ispitivanje sastoji se u pretraživanju matrice povezanosti za postojanje nul-retka. Ako takav postoji, tada on proizvodi odgovarajući stupac u nul-stupac, a taj može proizvesti novi nul-redak. Tako, ovim transformacijama matrica A može postati nul-matrica, što je dovoljan uvjet za ne postojanje zastoja. Pojava i-tog nul-retka je moguća zbog dinamike sistema

(oslobadanje sredstava), a ona znači da proces P_i nema više nezadovoljenih zahtjeva, te se može isključiti iz razmatranja potpunog zastoja postavljanjem i-tog stupca matrice A u nul-stupac. Poredak redukcija matrice A je nevažan [1].

4. SPREČAVANJE POJAVE PERMANENTNOG BLOKIRANJA

Za sprečavanje ove pojave mora biti zadovoljen opći uvjet da sistem ne smije nikada dostići stanje u kojem, bez obzira što procesi rade, neki zahtjevi ne mogu biti zadovoljeni. U mnogim sistemima, posebno tamo gdje je nizak stupanj iskorištenja sredstava, statistički dolazak zahtjeva i oslobadanje procesa može spriječiti pojavu permanentnog blokiranja. U mnogim slučajevima potrebno je provesti zaštitu na slijedeći način.

Sistem se opisuje sa dva vektora t i u sa po n elemenata. Kada proces P_i postavlja zahtjev za sredstvima, element t(i) se postavlja na vrijeme traženog zahtjeva. Kada se dopušta zahtjev procesu P_i, element t(i) se postavlja na specijalnu vrijednost, npr. -1. Vrijeme čekanja (w) koje je proces P_i čekao da mu bude zadovoljen zahtjev može se računati od trenutnog vremena (t_o) na slijedeći način: Ako je t(i) = -1, tada je w=0, inače je $w = t_o - t(i)$.

Element u(i) daje maksimalno vrijeme koje proces P_i mora čekati prije nego raspoređivač procesa aktivira posebnu strategiju pridjeljivanja koja omogućava zadovoljavanje zahtjeva procesa P_i. Raspoređivač procesa zadovoljava sigurne zahtjeve prema svom pravilu, ali periodički ispituje vektore t i u nebi li pronašao neki proces, npr. P_i koji je čekao na zadovoljavanje svog zahtjeva preko maksimalno dozvoljenog vremena čekanja. Kada se takav proces pronađe, aktivira se slijedeći postupak:

(a) Pronalazi se sigurna sekvencija izvodenja koja se sastoji od procesa P_i i svih procesa koji imaju pridjeljeno barem jedno sredstvo. Nastoji se pronaći sekvencija u kojoj je P_i što bliže početku.

(b) Zahtjevi se dopuštaju samo za prvi proces u sekvenciji sve dok taj proces ne oslobodi sva sredstva koja su mu bila pridijeljena. Nakon toga se dopuštaju zahtjevi slijedećem procesu u sekvenciji sve dok on ne oslobodi sva pridijeljena sredstva, itd. To se nastavlja tako dugo dok se ne oslobodi dovoljno sredstvo da i zahtjev procesa P_i bude siguran. Procesu P_i se tada dopušta zahtjev.

(c) Svaki proces različit od P_i se ispituje da li je njegovo maksimalno vrijeme prekoračeno. Ako jest, tada se proces koji je najviše prekoračio vrijeme proglašava procesom P_i, te se cijeli postupak ponavlja od (a).

Ovaj postupak ne narušava uvjet eventualnosti ali može u koraku (b) narušiti ekspeditivnost formiranjem zahtjeva procesa P_i.

5. ZAKLJUČAK

Kako su postupci detekcije stanja potpunog zastoja često vrlo složeni i zahtjevaju znatan utrošak sistemskog vremena [1], potrebno je koliko god je to moguće prvenstveno spriječiti pojavu potpunog zastoja i permanentnog blokiranja navedenim postupcima, prvenstveno onih koji dovode do visokog koeficijenta iskorištenja sredstava.

R E F E R E N C E :

[1] Hadjina, N. Postupci detekcije stanja potpunog zastoja računarskog sistema na modelu grafa sistema, INFORMATICA 2/1979, 14-17.

[2] Haberman, A. N., Prevention of System Deadlock, Comm. ACM (12)7, 1969, 373-377, 385.

[3] Hadjina, N., Postupci pridjeljivanja u višestruko korištenim računarskim sistemima, Disertacija, 1979.

ON SOLUTIONS FOR SOME OPEN PROBLEMS IN THE DESIGN OF MULTIPROCESSORS OPERATING SYSTEMS

ASIM SMAILAGIĆ

ELEKTROTEHNIČKI FAKULTET, SARAJEVO, JUGOSLAVIJA

UDK: 681.3.065

In this paper we are concerned with some open problems in the design and construction of multiprocessors operating systems. We quantify some design trade-offs in the space where a operating system interacts with the system architecture and resource placement strategies. We also investigate scheduling algorithms that should be used in these systems. It has been shown how a desired level of system performanse may be achieved using the chosen strategies.

INTRODUCTION

Experimental studies reported in (Batson, 1976; Denning, 1980; Ferrary, 1976) show that locality is an inherent property of program behavior. Measurements have revealed that a normal instruction mix on a PDP-11 has about 70% code references and 30% data references (Marathe, 1977). In a multiprocessor system with hierarchical structure which maps into different levels of memory references (Swan, 1977; Ornstein, 1975; Hansen, 1981), or any other multiprocessor with large number of processors, locality of code and data is very important to efficiency. In such cases the viability of architecture which can effectively support a large number of processors depends primarily on two factors. Firstly, the ability to implement cheaply and quickly references to non-local memory, and secondly, the existence of applications which suitable decompose for a multiprocessor and achieve a sufficiently high locality of references. The range of applications which can be decomposed in this way is an open research question. There are some evidence and promising results which indicate that the appropriate areas include: large artifical inteligence systems, real time control, digital signal processing, transaction processing with the implementation of large data base systems, numerical problems and applications of asynchronous iterative methods, computational physics and some other sciences where the effectiveness of new paralel algorithms for the solution of important computational problems can be tested etc.

The basic structure of a multiprocessor, like that presented by a multiprogramming operating system, maps processes consecutively onto a single physical machine, rather than simultaneously onto several machines. Using the multiprocessor structure in an multiprogramming environment has different aspects and demand specific treatment. Such multiprogramming system has some more jobs in main memory and these jobs will share all processors. When a processor becomes available it will take another job from the processor queue to execute it, as each processor might take care of more than one job at a time instant. The system of this configuration is generally expected to have a much larger computational capacity, higher processor and memory utilization than a single processor system. In this environment we will encounter some overheads, caused primarily by requirement for a more complex operating system design and memory interference. The degree of memory interference is expected to be proportional to the extent to which the memory requirements of different processors overlap and to the rate at which each processor issues requests to the memory.

The next part of the investigation relates to the choice of scheduling algorithm that should be used in the multiprocessor operating system. It is dependent on the type of system we are concerned with and on the performanse measure that we want to get improved. An interactive system in multiprocessor environment was modeled in (Sauer, 1978) and compared with a single powerful processor system. The level of multiprogramming was fixed to be equal to the number of processors. The authors make the optimistic assumption that all future service times can be predicted with total accuracy and that the shortest remaining processing time first (SRPTF) discipline is used to schedule processors. This comparison study which consider the percentage improvement in going to the SRPTF discipline from the first come first serve(FCFS) discipline indicates that this change has more impact on the throughput of a single processor system than a dual processor system. In our study we will investigate six scheduling algorithms.

The difficulties in design of multiprocessor operating system suggest that a traditional operating system built without spacial consideration of the problems of a multiprocessor will not be very efficient using the resources of a multiprocessor. In order to overcome these problems they should be taken into consideration at the system design time.

In the next section of this paper we describe the basic strategies that we have chosen, structure of the system and modeling decisions

we have made.

PROBLEM ANALYSIS AND SOLUTION METHOD

The basic structure of the multiprocessor system is shown schematically in Figure 1.
A number of analytical models for multiprocessor systems have been developed, as (Bhandarkar, 1975; Basket, 1976; Hoogendorn, 1977; Sethi, 1979). However, these models are all restricted in various ways, such as requiring identical or even uniform memory reference patterns for all processors, being unable to handle non-zero processing times, there is no direct way to model storage, service disciplines and service time distributions, these models are not detailed enough to enable the evaluation of different parameter changes on system performances, inability to evaluate the structure for the cases where there are more than one clear bottleneck in the system, or the bottleneck is not known a priory.
All these limitations led us to the investigation and development of a better and more realistic model where we use event-driven simulation. The appropriate open queueing network model for our system consists of p processors, m memory modules and k I/O devices, and has three stages of service. In order to reduce average queue length in the waiting queue just before jobs enter the memory we investigate different scheduling algorithms. The main performance measure regarding scheduling algorithms that we use is the average turnaround time T of jobs. T of a job is the sum of the average queueing time and the average service time. The description of the model can be found in (Smailagić, 1981a).

In the Multics time-sharing system foreground-background (FB) algorithm is used, which discriminates against long jobs, and FCFS was studied for dual processor configuration. In (Saltzer, 1981) the Multic's designers explore how to effectively support traditionally expensive operating system functions by interconnection mechanisms.

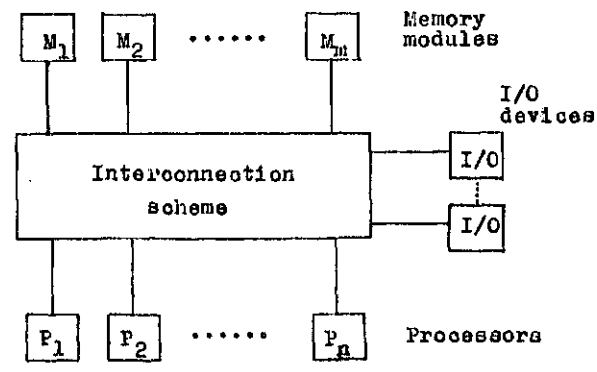
The next part of our investigation relates to the issues put on by introducing local memories. The rate at which processors send requests to shared memory is now reducing and therefore the degree of memory interference is getting decreased. We have examined here some of the design trade-offs between system performances and the memory size, number of processors, impact of I/O trafic, and taking into consideration the computation that is being done. In order to determine the optimal size of local and common memory in terms of given computations we have defined and developed the concept of working set for multiprocessors. The working set of information is the most cost-effective amount of data and code to have in local memory. By 'code' we relate to all the primary memory access resulting from fetching instructions from memory.
For a multiprocessor system with hierarchical structure which maps into different levels of memory references the overal performance depends on the relative frequency of local local references and the relative cost of non-local references. At the implementation level this scheme here resembles that of a classical look-ahead processor (Keller, 1975), with the master processor acting as the controller and the slave as a multiple functional unit. By duplicating, in each processor, the code required for the master and slave functions, the code for schedulers, and distributing the user program, its data area and the tables required by the schedulers across all the processors, and since most code memory accesses are generated inside the run-time system functions, a very good locality can be achieved. In the next section we indeed illustrate the importance of locality factor and also show appropriate system's memory requirement.

As an extension of the PDP-11/70 cache simulation studies (Strecker, 1976), in (Arulpragasam, 1980) a simulation model was used to characterize the performance of the common cache. It was reported that a cache of M words shared among N processors with no code or data sharing would show the same miss ratio as a uniprocessor cache of M/N words- the cache adaptively divides itself among the available running processors.

RESULTS AND SUMMARY

Our investigation has produced some more results for open problems in the design and construction of multiprocessor systems. Due to space limitation we present here just some of them. Figure 2. shows the results for all six investigated scheduling algorithms. The numbers on the curves from 1 to 6 correspond to the following algorithms: shortest job first, shortest processing time first, largest memory first, first come first served, smallest job first, longest job first. It can be seen that shortest job first SJF algorithm gives the best performance measure as its value for T is the smallest. Another conclusion that we can reach here is that for six and more processors all these algorithms perform almost the same. As for the job mix that was used here the memory contains for the most of time six jobs, it follows that when we have enough processors the scheduling algorithm does not make too much influence on the performance (T is in sec). The results on Fig.2 are received under monoprogramming.

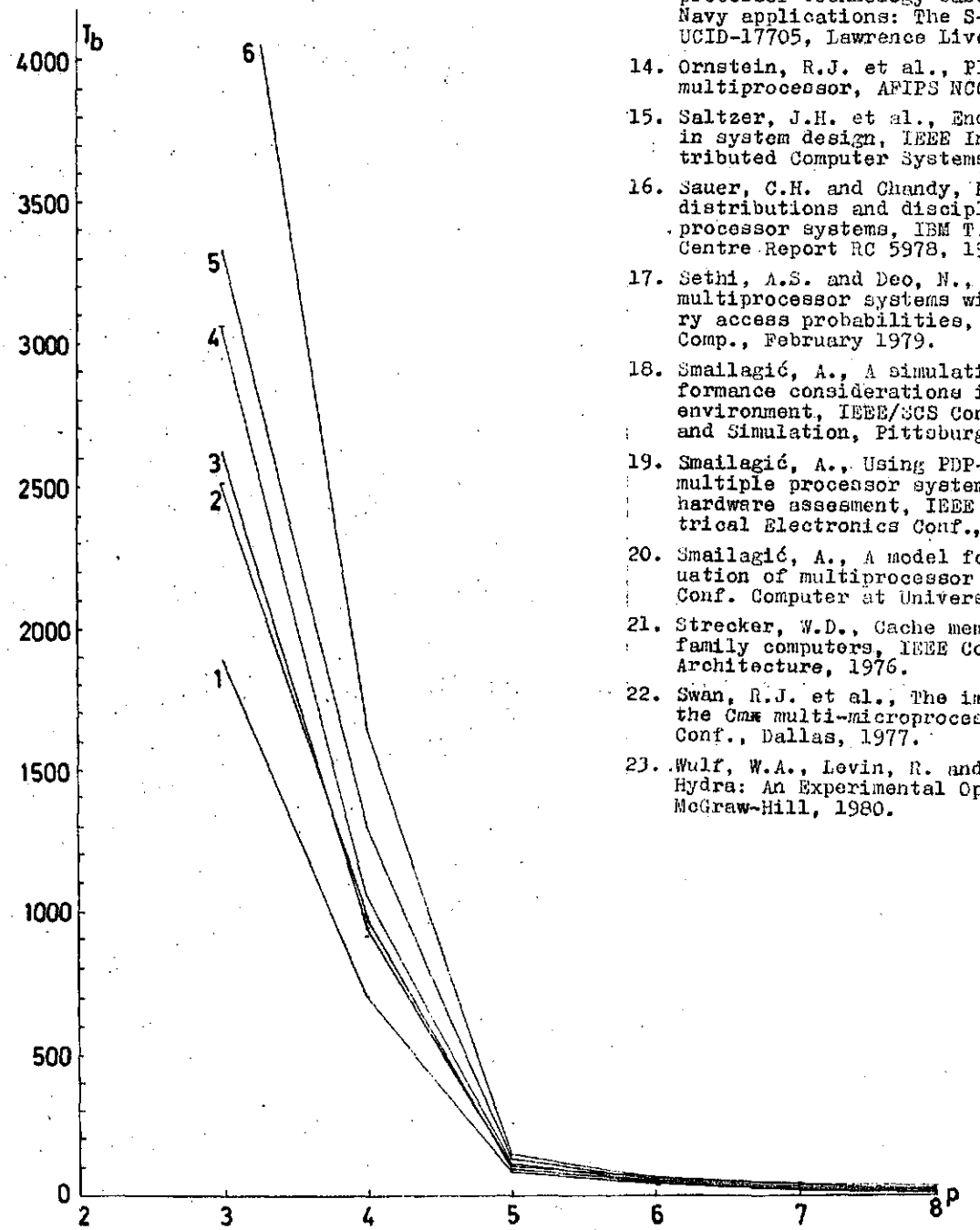
Figure 3 shows the relationship between the number of equivalent processors, as relative speedup indication, and the size of the memory. Performance improvement that we have got after the consecutive markers on the curves comes from the copying of local data, then frequently used part of the code, all code copied and all data copied, respectively. It can be seen that a very significant performance increase is achieved by duplicating frequently accessed code (second marker) and then all code copied (third marker). The number of processors is denoted by P_1 - P_6., and it takes the values 2,6,10,14, 18,22respectivaly.



The basic multiprocessor system structure
Fig. 1.

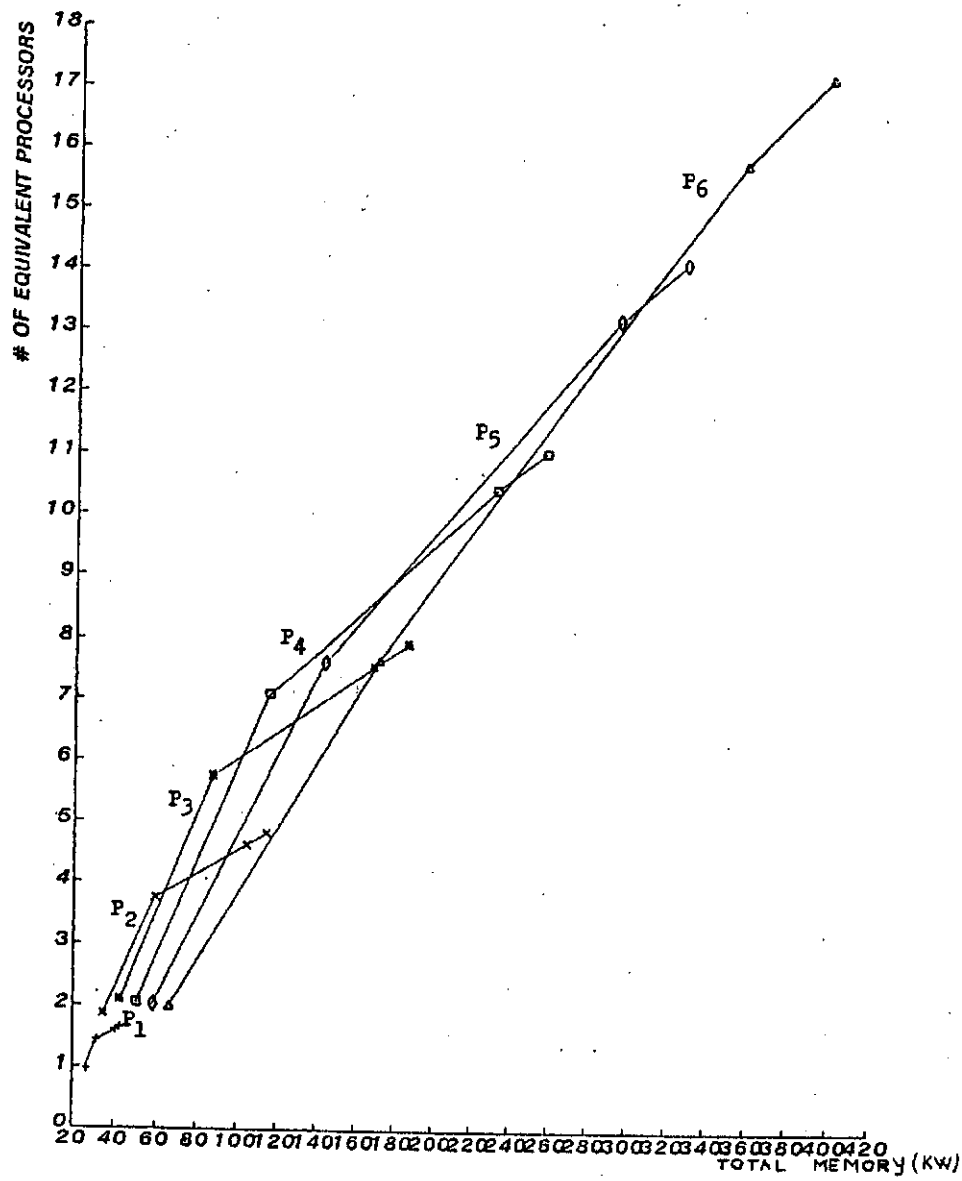
REFERENCES

1. Arulpragasam, J.A. et al., Modular minicomputers using microprocessors, IEEE Trans. on Comp., February 1980.
2. Baskett, F. and Smith, A.J., Interference in multiprocessor computer systems with interleaved memory, CACM, June 1976.
3. Batson, A.P., Program behaviour at the symbolic level, Computer, November 1976.
4. Bhandarkar, D.P., Analysis of memory interference in multiprocessors, IEEE Trans. on Comp., September 1975.
5. Denning, P.J., Working sets past and present, IEEE Trans. on Software Eng., January 1980.
6. Ferrary, D., The improvement of program behaviour, Computer, November 1976.
7. Hansen, P.B., The design of Edison, ACM SIGPLAN Notices, April 1981.
8. Hansen, P.B., Multiprocessor architectures for concurent programs, ACM SIGARCH News, January 1979.
9. Hoogendorn, C.H., A general model for memory interference in multiprocessors, IEEE Trans. on Comp., October 1977.
10. Keller, R.M., Look-ahead processors, ACM Computing Surveys, December 1975.
11. Kleinrock, L., Queueing Systems, Volume II: Computer Applications, John Wiley, 1976.
12. Marathe, M. and Fuller, S.H., A study of multiprocessor contention for shared data in C.mmp, ACM SIGMETRICS Conf., Washington, December 1977.
13. McWilliams, T.M. et al., Advanced digital processor technology base development for Navy applications: The S-1 Project, Report UCID-17705, Lawrence Livermore Lab, USA, 1977.
14. Ornstein, R.J. et al., Pluribus: A reliable multiprocessor, AFIPS NCC Conf., 1975.
15. Saltzer, J.H. et al., End-to-end arguments in system design, IEEE Int. Conf. on Distributed Computer Systems, Paris, April 1981.
16. Sauer, C.H. and Chandy, K.M., The impact of distributions and disciplines on multiple processor systems, IBM T.J.Watson Research Centre Report RC 5978, 1978.
17. Sethi, A.S. and Deo, N., Interference in multiprocessor systems with localized memory access probabilities, IEEE Trans. on Comp., February 1979.
18. Smailagić, A., A simulation model for performance considerations in multiprocessor environment, IEEE/SCS Conf. on Modeling and Simulation, Pittsburgh, April 1981.
19. Smailagić, A., Using PDP-11 in building multiple processor systems- a critical hardware assesment, IEEE International Electrical Electronics Conf., Toronto, Oct. 1981.
20. Smailagić, A., A model for performance evaluation of multiprocessor systems, Int'l Conf. Computer at University, Cavtat, May 1981.
21. Strecker, W.D., Cache memories for PDP-11 family computers, IEEE Conf. on Computer Architecture, 1976.
22. Swan, R.J. et al., The implementation of the Cm* multi-microprocessor, AFIPS NCC Conf., Dallas, 1977.
23. Wulf, W.A., Levin, R. and Harbison, S.P., Hydra: An Experimental Operating System, McGraw-Hill, 1980.



Performances of investigated scheduling algorithms

- Figure 2 -



- Number or equivalent processors vs. total memory -

Figure 3

PRIMER KOMUNIKACIJSKEGA PROTOKOLA

M. ŠUBELJ,
R. TROBEC,
J. KORENINI

UDK: 681.324

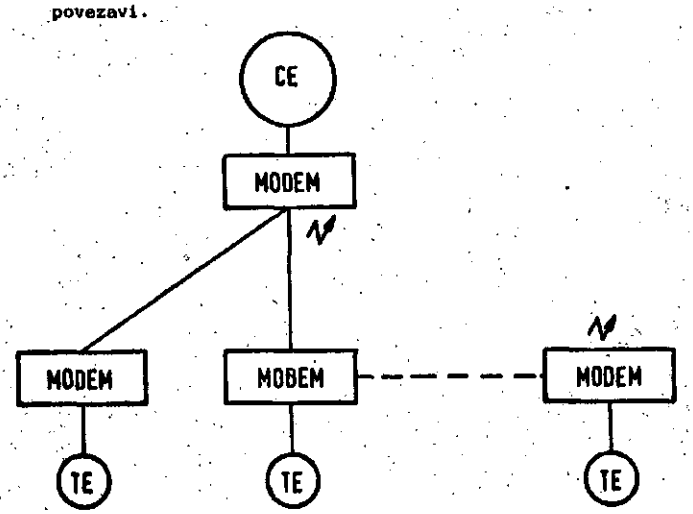
INŠTITUT JOŽEF STEFAN, LJUBLJANA, JUGOSLAVIJA
ODSEK ZA RAČUNALNIŠTVO IN INFORMATIKO

Prispevek opisuje komunikacijski protokol mikroračunalniškega sistema za komunikacijo, komande in kontrolo procesov. Sistem je namenjen kontroli relativno počasnih procesov, zato je informacijska mreža realizirana v multipoint povezavi. Opisan je postopek vzpostavljanja zveze, organizacija sporočila in uporabljeni principi povečanja zanesljivosti prenosa. Programska oprema je implementirana na mikroračunalniku F-8.

THE EXAMPLE OF COMMUNICATION PROTOCOL: Communication protocol for the communication, command and control microcomputer system is described. The system is used to control relatively slow process therefore communication is implemented in multipoint network. Procedure for establising connection message, organization and techniques for achieving high reliability in communication is described. The software package is implemented on the F-8 microcomputer.

1.UVOD

Programska paketa komunikacije center (COMCE) in komunikacije terminal (COMTE) omogočata prenos informacij med centralno enoto (CE) in terminalom (TE) v multipoint povezavi.



Slika 1. Multipoint povezava

Pri tej povezavi ima CE en sam komunikacijski vhodno izhodni kanal. Izbira kanala TE je časovno multipleksirana. Informacijo je možno prenesti iz CE v TE in iz TE v CE. Pri takšni povezavi morajo biti v stanju, ko ni prenosa vsi TE priključeni na sprejem. Zvezo med CE in izbranim TE je možno vzpostaviti samo iz CE. Zveza se vzpostavi tako, da CE odda določeno kodo identifikator (ID), ki vsebuje informacijo o tem s katerim TE želi vzpostaviti zvezo in način priključitve na linijo. CE poleg tega odda tudi podatkovne informacijske bite o režimu delovanja. Režim delovanja določuje tip in s tem število podatkov, ki jih želi CE prenesti v TE ali iz TE. Namen vpeljave te zahteve je razvrstiti podatke po prioriteti prenosa in s tem povečati frekvenco obnove stanj v CE podatkom, ki to zahtevajo. Istočasno CE lahko vzpostavlja zvezo in komunicira samo z enim TE.

2. PROCES VZPOSTAVLJANJA ZVEZE

2.1. Centralna enota

CE vzpostavi zvezo z izbranim terminalom tako, da odpošlje identifikator. ID vsebuje adreso izbranega TE in kodo načina priključitve na linijo. CE se po tem preklopi na sprejem in čaka na odziv TE, ki se mora odzvati v predpisanem času s tem, da vrne sprejeti ID. CE vrnjeni ID primerja z odposlanim in če sta enaka, je proces vzpostavljanja zveze uspešno končan. V primerih, ko se TE ne odzove v predpisanem času ali, da vrnjeni ID ni enak odposlanemu, se proces vzpostavljanja zveze ponovi. Število ponovitev zavisi od zahtevanega režima delovanja.

2.2. Terminal

Vsi terminali so priklopljeni na sprejem in sprejemajo. Vsak TE ima lastno adreso različno ostalim. Če TE razpozna svojo adreso in razpozna sprejete znake kot ID, se priklopi na linijo in ID vrne CE. S tem postopkom je CE uspešno vzpostavil zvezo s TE.

3. NAČINI PRIKLJUČITVE NA LINIJO

V procesu vzpostavljanja zveze CE zahteva od TE enega od naslednjih dveh načinov priključitve na linijo:

- imenovanje (polling)
- selekcioniranje (selecting).

Izbira zavisi od zahteve poziva CE glede smeri prenašanja informacije, iz TE v CE ali iz CE v TE.

3.1. Imenovanje

Zahteva CE po imenovanju pomeni poziv TE, naj pošlje podatke, izbrane z režimom delovanja.

3.2. Selekcioniranje

Zahteva CE po selekcioniranju pomeni TE, naj sprejme podatke in ukrepa z njimi, kot zahteva režim delovanja.

Vzpostavljanje zveze, priključitev na linijo in prenos informacije je zaključen proces, katerega delovanje ni mogoče prekiniti med samim izvrševanjem. Proces se mora vedno izvršiti do konca, predno se začne izvrševati naslednji zahtevani proces prenosa.

4. ORGANIZACIJA SPOROČILA

Spročilo oblikuje pošiljatelj glede na režim delovanja. Sporočilo sestavljajo bloki. Programsko se sporočilo oblikuje samo na nivoju blokov. Blok se prenaša ali ne prenaša, vendar je pri prenosu njegova dolžina vedno enaka. Kontrola prenosa je zato realizirana glede na bloke.

Enota, ki sprejema celoten blok, se odloči o pravilnosti prenosa. Če prenos ni bil pravilen vrne pošiljatelju znak o nepravilnem prenosu (NACK) in čaka na ponoven prenos bloka. Število neuspelih prenosov enega bloka je omejeno. V primerih, ko prenos bloka ni bil mogoč, se TE odklopi od linije.CE v takem primeru odloči, da uspešnost prenosa ni mogoča, zato javi napako o neuspeli vzpostavitvi zveze. V primeru, ko je bil prenos pravilen, sprejemnik blok shrani in pošlje pošiljatelju znak o potrditvi uspešnosti prenosa (ACK).

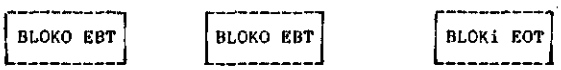
Zadnji znak bloka je lahko EBT (end of block transmission) ali EOT (end of transmission). Znak EOT določuje konec sporočila. To pomeni, da je prenos končan in TE se odklopi od linije.

Sporočilo sestavljajo:

- bloki
- posebni znaki.

Organizacijsko in časnovno prenos bloka izgleda takole:

Pošiljatelj:



Sprejemnik:

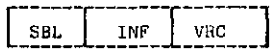
Pri bloku 0 je bil prenos neuspešen.

4.1. Blok

Dolžina bloka je določena s programsko opremo. Med samim delovanjem sistema je tako blok kot enota informacijsko nespremenljiva celota. Uporabljen je globalni način adresiranja. Vsak blok ima v CE ali v TE v pomnilniku naprej predpisan pomnilniški prostor. Adrese polja so v tabelah v obeh enotah, blok pa najde svojo adreso preko številke bloka.

Blok sestavljajo naslednji znaki:

- številka bloka (SBL)
- informacijski biti bloka (INF)
- bitna kontrola parnosti (VRC)



SBL : številka bloka
INF : INF je od 1 do 16 zlogov koristne informacije
VRC : VRC je bitna ali vertikalna kontrola parnosti

4.2. Posebni znaki

- EBT : Namen EBT je sinhron prenos podatkov in razpoznavanje identifikatorja (ID) od zadnjih dveh znakov v bloku sporočila.
- EOT : Nemen EOT je sinhron prenos podatkov in razpoznavanje identikiratorja (ID) od zadnjih dveh znakov v zadnjem bloku sporočila.

5. OPIS PROTOKOLA V PROSTORU STANJ

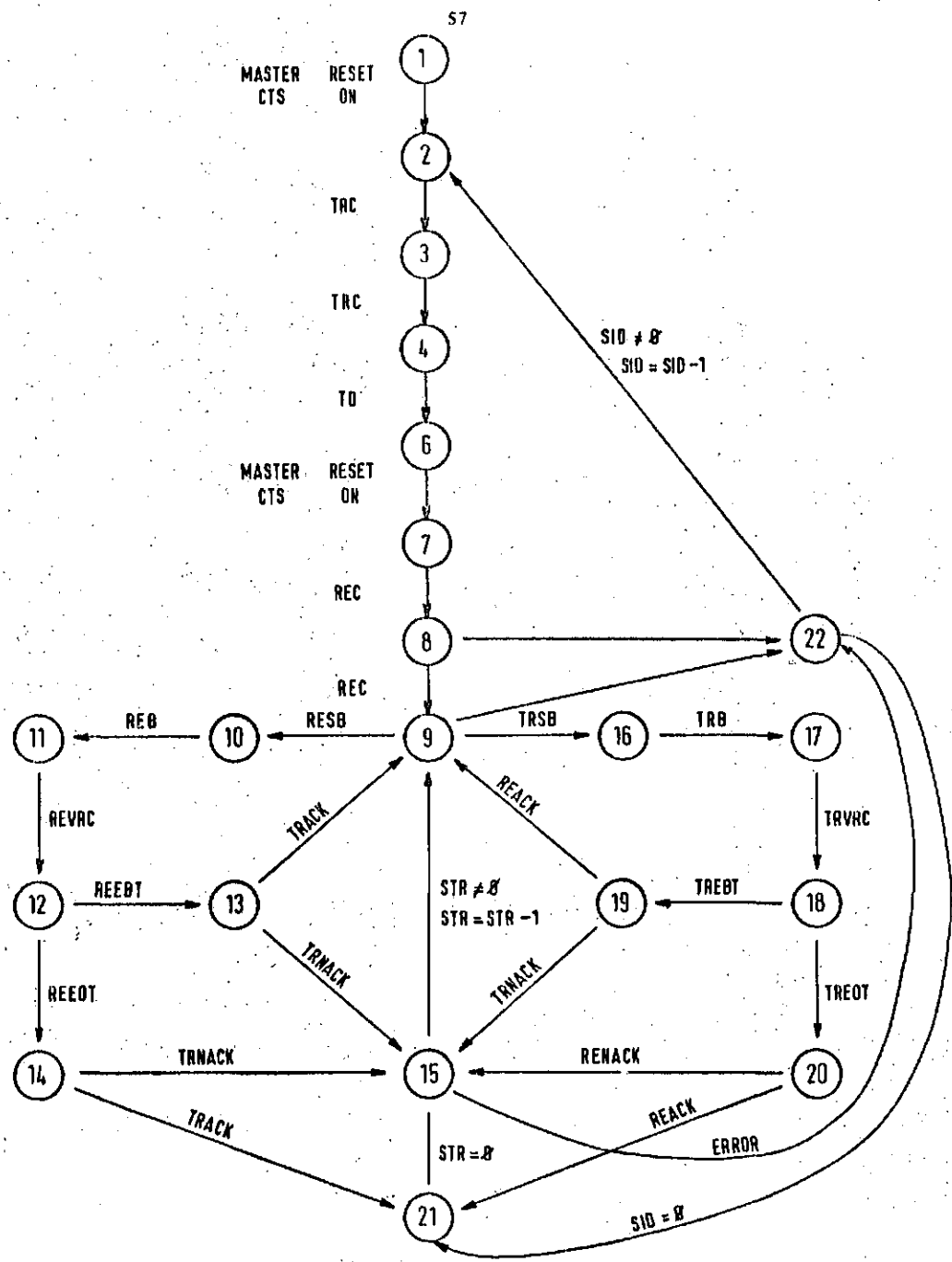
Oglejmo si predstavitev protokola v prostoru stanj.

5.1. Legenda oznak:

- TRC — oddaja znaka.
- REC — sprejem znaka.
- TRB — oddaja bloka.
- REB — sprejem bloka.
- TRSB — oddaja številke bloka.
- RESB — sprejem številke bloka.
- TR VRC — oddaja bitne kontrole parnosti.
- RE VRC — sprejem bitne kontrole parnosti.
- TR EBT — oddaja znaka konec prenosa bloka.
- RE EBT — sprejem znaka konec prenosa bloka.
- TR EOT — oddaja znaka konec prenosa sporočila.
- RE EOT — sprejem znaka konec prenosa sporočila.
- TR ACK — oddaja pritrditve.
- RE ACK — sprejem pritrditve.
- TR NACK — oddaja ni pritrditve.
- RE NACK — sprejem ni pritrditve.
- SID — števec ponovitve pozivov CE TE.
- STR — števec ponovitve prenosov bloka.
- ERROR — detekcija napake pri kontroli linije
- TD — časovna zakasnitev.
- MASTER RESET — inicializacija in vzpostavitev začetnih stanj - ACIA.
- CTS ON — priklopitev nosilca na linijo.
- CTS OFF — odklopitev nosilca od linije.

5.2. Centralna enota

- pozivanje TE.
1 -- 2 inicializacija ACIA in priključitev nosilca na linijo.
2 -- 3 oddaja adrese naslovljenega TE.
3 -- 4 oddaja načina priključitve in režima delovanja.
4 -- 6 čas v katerem TE sprejme celotni ID, se odloči ali je ID ali ne in se priklopi na linijo.
6 -- 7 master reset je potreben, ker zaradi priključitve TE na linijo, linija zaniha in ACIA v CE sprejme znake, ki niso bili odposlani.
7 -- 8 čakanje na sprejem odposlane adrese, čas je omejen.
8 -- 22 adresa ni bila vrnjena v predpisanem času.
8 -- 9 adresa je bila vrnjena v predpisanem času. Čakanje na sprejem načina priključitve in režima delovanja.
9 -- 22 način priključitve na linijo in režim delovanja nista bila vrnjena v predpisanem času.
22 -- 2 SID / = 0 ponoven poziv.
22 -- 21 SID = 0 konec pozivanja. CE javi, da z izbranim TE ni možno vzpostaviti zveze.



Slika 2. Protokol v prostoru stanj za centralno enoto

- pooling priključitev na linijo

9 -- 1o sprejem številke bloka.
1o -- 11 sprejem bloka.
11 -- 12 sprejem kontrole parnosti.
12 -- 13 sprejem znaka konec prenosa bloka.
13 -- 9 ker je bil blok pravilno prenešen, oddaja ACK in sprejem naslednjega bloka.
13 -- 15 ker blok ni bil pravilno prenešen, oddaja NACK.
12 -- 14 sprejem znaka konec prenosa sporočila.
14 -- 15 isto kot 13 -- 15.
14 -- 21 ker je bil zadnji blok pravilno prenešen, oddaja ACK in izhod iz modula.

- selecting priključitev na linijo

9 -- 16 oddaja številke bloka.
16 -- 17 oddaja bloka.
17 -- 18 oddaja kontrole parnosti.
18 -- 19 oddaja znaka konec prenosa bloka.
19 -- 9 ker je bil blok pravilno prenešen in potrjen, sprejem ACK in oddaja naslednjega bloka.
19 -- 15 ker blok ni bil pravilno prenešen, sprejem NACK.
18 -- 2o oddaja znaka konec prenosa sporočila.
2o -- 15 isto kot 19 -- 15.
2o -- 21 ker je bil zadnji blok pravilno prenešen in potrjen sprejem ACK in izhod iz modula.

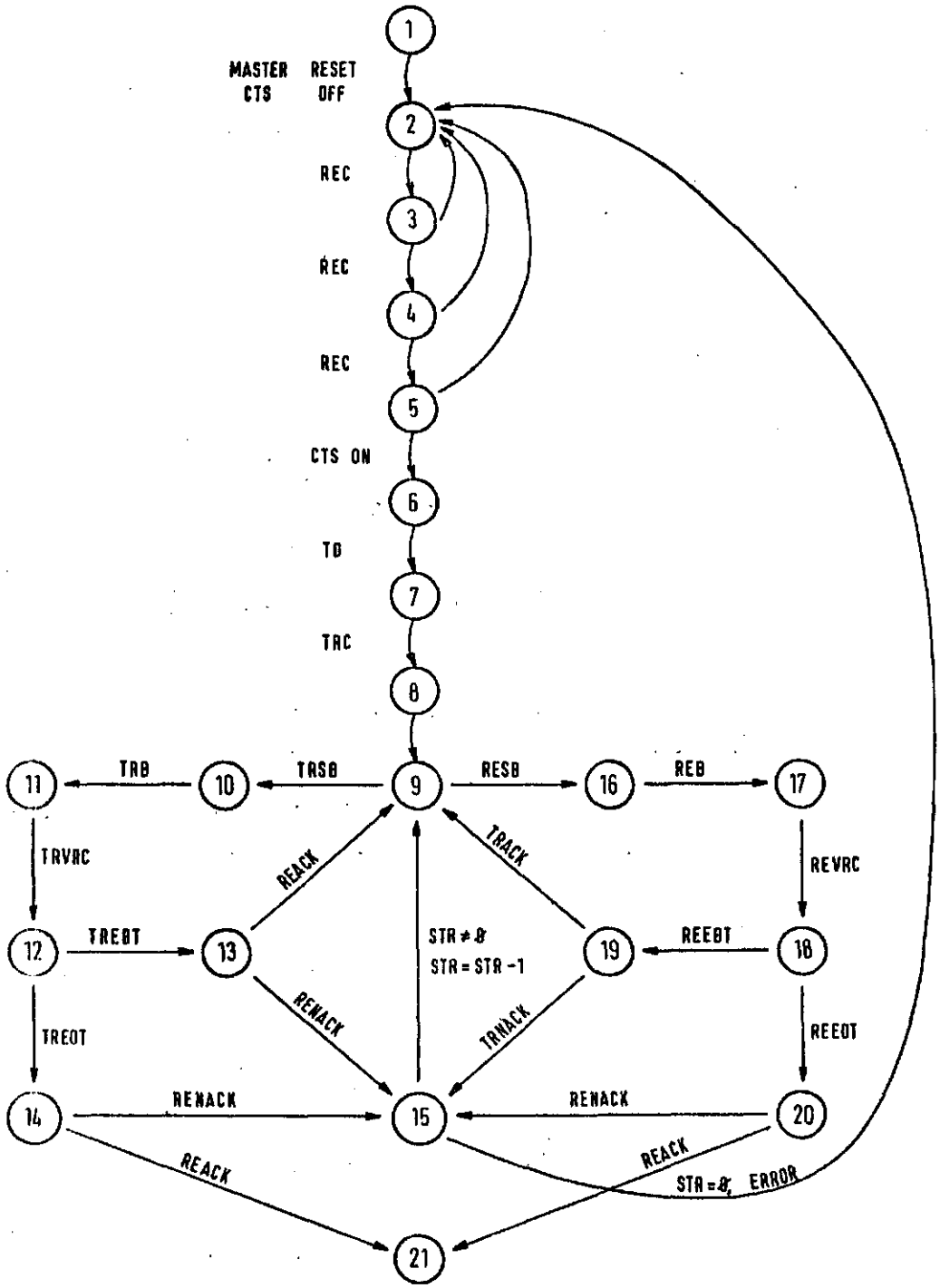
- test napak

15 -- 9 STR / = 0, STR = STR-1 in ponovno procesiranje bloka.
15 -- 22 napaka detektirana pri kontroli linije.
15 -- 21 STR = 0 bloka ni možno prenesti, CE javi, da z izbranim TE ni možno vzpostaviti zveze.

5.3. Terminal

- priklopitev TE

1 -- 2 inicializacija ACIA in odklopitev nosilca od linije.
2 -- 3 sprejem znaka.
3 -- 2 sprejeti znak ni bila adresa TE.



Slika 3. Predstavitev protokola v prostoru stanj za terminal

3 -- 4 med prehodoma iz stanj 2 -- 3 je bila sprejeta adresa TE.

4 -- 2 test ali je možno, da je sprejeti znak, znak načina priključitve na linijo. Znak načina priključitve na linijo se razlikuje od znakov EBT in EOT v dveh bitih. Sprejeti znak ni bil znak načina priključitve na linijo.

4 -- 5 sprejeti znak ni ne EBT ali EOT, zato je sprejeti znak lahko znak načina priključitve na linijo. Testiranje ali bo za njim še kakšen znak.

5 -- 2 sprejet je bil znak. Sprejeta znaka sta bila del bloka sporočila.

5 -- 6 v predpisanem času ni bil sprejet znak. Sprejeti znaki so bili ID. Priključitev nosilca na linijo.

6 -- 7 časovna zakasnitev, da se linija izniha.

7 -- 8 oddaja adrese.

8 -- 9 oddaja znaka načina priključitve na linijo.

- polling priključitev na linijo.

- polling priključitev na linijo je v TE podobna selecting priključitvi v CE.

- selecting priključitev na linijo.

- selecting priključitev na linijo v TE je podobna kot polling priključitev v CE.

- test napak

15 --2 STR = 0 bloka ni možno prenesti ali napaka detektirana pri kontroli linije.

6. ZANESLJIVOST PRENOSA

Zanesljivost prenosa podatkov je zaščitena z informacijsko (identifikator) in decizijsko (podatki) povratno zanko.

6.1. Informacijska povratna zanka

Zveza med naslovljenim TE in CE je vzpostavljena šele takrat, ko CE sprejme odziv TE na identifikator. Oddani identifikator mora biti enak vrnjenemu. Če identifikatorja nista enaka, se naslavljanje ponovi, kar zavisi od režima delovanja.

6.2. Decizijska povratna zanka

Blok sporočila je zaščiten z decizijsko povratno zanko, zato se prenos kontrolira in ponavlja po blokih. Element, ki podatke sprejema se odloči o pravilnosti prenosa. V primeru,ko se odloči, da je bil prenos pravilen, vrne pošiljatelju znak o uspešnosti prenosa. Če prenos ni bil uspešen odda znak o nepravilnosti prenosa in prenos bloka se ponovi. Število ponovitev prenosov je omejeno.

7. KODIRANJE

7.1. Korekturni kod

Vsak znak informacije je kodiran s korekturnim kodom Hammingove razdalje 4. Znak se odda kot dva znaka,ki vsebujeta 4 informacijske bite in 4 kontrolne bite. Kod Hammingove razdalje popravlja vse enojne in odkriva dvojne napake. S korekturnim kodom, ki ne samo odkriva napake, temveč tudi popravlja, je izboljšana kvaliteta kanala.

7.2. Kontrola parnosti

Podatki so zaščiteni še s kontrolo parnosti (zaradi decizijske povratne zanke). Blok; SBL in 1 - 16 zlogov informacije sta zaščitena z bitno (VRC) kontrolo parnsoti. Oddajnik najprej odda blok podatkov, izračuna njihovo parnost in jo odda. Sprejemnik sprejme blok, jim določi parnost, sprejme odposlano parnost in ju primerja. Sprejemnik se na podlagi korekturnega koda (odkrije dvojne napake) in kontrole parnosti odloči o pravilnosti prenosa.

8. SKLEP

Opisani sistem je namenjen procesnemu vodenju relativno počasnih procesov v realnem času. Uporabljena komunikacijska sredstva so lahko telegraf, telefon ali brezžična zveza. Takšno raznovrstno uporabo komunikacijskih sredstev omogoča multipoint povezava. Pri procesnem vodenju relativno hitrih procesov, bi morala biti že mrežna povezava med centralno enoto in terminali drugačna.

9. LITERATURA

1. L.Gyergyek: Statistične metode v teoriji sistemov, teorija o informacijah, Fakulteta za elektrotehniko 1971.
2. M.Šubelj,J.Korenini, F.Novak, R.Trobec: Kodiranje in dekodiranje korekturnega koda z mikroračunalnikom, Informatika 3 / 1979.
3. Electronics Book Series: Basics of data communications, Mc Graw-Hill 1976.

INFORMATICA 3/1981

PRIMJENA INVERZIONE METRIKE KOD OPTIMALNOG BALANSIRANJA

JOVAN LONČAR

VIŠA ZRAKOPLOVNA ŠKOLA – ZAGREB

UDK: 681.3:512.643

U prostor permutacija uvodi se inverziona metrika koja se zatim koristi za proračun optimalnog balansa zrakoplovnih hidro i parnih turbina i drugdje.

APLICATION OF THE INVERSION METRIC FOR CALKULATION THE OPTIMAL BALANCE:
In a space of permutation the inversion metric is introduced and used for the calculation of optimal balance.

Označimo sa $\mathcal{P}$ - prostor permutacija i promatrajmo $p_1, p_2 \in \mathcal{P}$. Neka je

$$p_1 = (i_1 i_2 \ldots i_n); \quad p_2 = (j_1 j_2 \ldots j_n)$$

Neka su $i_k, i_e (k \neq e)$ proizvoljni elementi iz p_1. Ako se u permutaciji p_2 nađe par elemenata j_r, j_s takav da je $i_k = j_r$, $i_e = j_s$, onda kažemo da par j_r, j_s čini inverziju u odnosu na p_1.

Udaljenost $d(p_1, p_2)$ se definira kao broj svih inverzija u p_2 u odnosu na p_1. Pokazuje se da je maksimalno mogući broj inverzija jednak $n(n-1)/2$.

Pokažimo da uvedena udaljenost zadovoljava sve aksiome metrike [1] i [3].

1. Po definiciji je $d(p_1, p_2) \geq 0$
2. Permutacija p u odnosu na samu sebe ne čini inverziju pa je $d(p,p) = 0$
3. Odsutnost inverzije izmedju dvije permutacije, znači da se te permutacije podudaraju.
4. Ako je $i_\alpha = j_\beta$ i $i_\gamma = j_\delta$, onda elementi i_α i i_γ čine inverziju u odnosu na p_2, tada i samo tada kada j_β i j_δ čine inverziju u odnosu p_1.
5. Pogledajmo tri permutacije $p_1 = (i_1 i_2 \ldots i_n)$, $p_2 = (j_1 j_2 \ldots j_n)$, $p_3 = (k_1 k_2 \ldots k_n)$. Neka je $i_\alpha = j_\beta$ i $i_\gamma = j_\delta$. Ako par i_α, i_γ čini inverziju u odnosu na p_3, onda taj par čini inverziju u odnosu na p_2 ili j_β, j_δ čini inverziju u odnosu na p_3.

Znači, da svakoj inverziji p_1 u odnosu na p_3 odgovara inverzija tih istih elemenata u odnosu na p_2 ili (i) njihova inverzija u p_2, u odnosu na p_3. Znači vrijedi relacija trokuta.

Da bi obrazložili algoritam pomoću koga nalazimo $p \in \mathcal{P}$ koja se nalazi u okolini radiusa R potrebno je uvesti neke pojmove.

Uzmemo li sve permutacije od n- simbola $p = (i_1 i_2 \ldots i_n)$ i cjelobrojne vektore oblika $\{b_1 b_2 \ldots b_{n-1}\}$, gdje $0 \leq b_\alpha \leq n-\alpha$; $\alpha = (1, n-1)$. Među njima uspostavimo uzajamno jednoznačnu korespodenciju. Kod toga je b_α broj inverzija koji simbol i_α čini sa simbolima u p, a u odnosu na $p^M = (1,2 \ldots n)$. Obratno pridruživanje, formiranje iz indeksa permutaciji vrši se pomoću slijedećeg algoritma:

ALGORITAM 1

1. korak . Stavimo $i_1 = b_1 + 1$. U nizu brojeva $B_0 = (1,2,\ldots,n)$ precrtamo broj i_1.

Dobiveni niz $(1,2...i_1-1\ i_1+1\ ...n)$ označimo sa B_1.

2. korak. Stavimo da je i_2 jednak (b_2+1) po redosljedu elementu u nizu B_1. U B_1 -precrtamo broj i_2, a dobiveni niz označimo sa B_2.

i-ti korak. Neka i_α kao $(b_\alpha+1)$ po redosljedu broj niza B_α -1. U nizu B_α -1 precrtati broj i_α i dobiveni niz označiti B_α.

n-ti korak. Staviti $i_n = B_{n-1}$

Tako na primjer indeksu I = (2,5,0,3,1,0,0) odgovara permutacija p = (3,7,1,6,4,2,5,8). Permutaciju $p^M = (1,2,3...n)$ zovemo nulom prostora $\mathcal{T}$, jer njoj odgovara indeks I = (0,0...0). Udaljenost izmedju p^M i proizvoljne permutacije $p \in \mathcal{T}$ kojom odgovara indeks $I = (b_1 b_2 ... b_{n-1})$ dobije se pomoću izraza

$$d(p^M,p) = \sum_{\alpha=1}^{n-1} b_\alpha$$

Zbog nejednakosti trokuta $d(p,p_0) \leq d(p,p^M)+d(p_0,p^M)$, ako je $d(p_0,p^M)$ poznata i jednaka d_0, onda za formiranje permutacija p- koje leže od p_0-ne dalje od R- dovoljno je formirati permutacije koje su udaljene od nulte p^M ne više od $R-d_0$. Na upravo rečenom temelji se algoritam za nalaženje $p \in \mathcal{T}$ da vrijedi $d(p,p_0) \leq R$.

ALGORITAM 2

1. korak. Odrediti indeks $I_0 = (b_1 b_2 ... b_{n-1})$ permutacije po.

2. korak. Izračunati $d_0 = d(p_0,p^M)$ pomoću

$$d_0 = \sum_{\alpha=1}^{n-1} b_\alpha$$

3. korak. Pomoću generatora slučajnih brojeva formirati niz slučajnih cijelih brojeva $(C_1,C_2...C_{n-1})$ takvih da imaju slijedeća dva svojstva:

a) $\sum C_\alpha \leq R - d_0$

b) $0 \leq C \leq n-\alpha$

4. korak. Interpretirati niz brojeva $(C_1,C_2,...C_{n-1})$ kao indeks permutacije $p \in \mathcal{T}$ i po tom indeksu formirati permutaciju.

Na osnovu prednjeg algoritma sastavljen je program za računar. Uzeti su podaci iz tabele 1. Izračunate su vrijednosti funkcionala - debalansa:

$$f(i_1 i_2 ... i_n) = \left[\left(\sum_{i=1}^{n} M_{ik} \cos \frac{2\pi k}{n} \right)^2 + \left(\sum_{i=1}^{n} M_{ik} \sin \frac{2\pi k}{n} \right)^2 \right]^{\frac{1}{2}}$$

Na slici 1 vidimo rezultate iz 100 pokusa metodom MONTE-KARLO a na slici 2 rezultate od 100 vrijednosti kada se biraju permutacije iz okoline min f(p) koji je dobiven u [2] na slučajan način, sa radiusom kugle R = 5 koristeći inverzionu metriku.

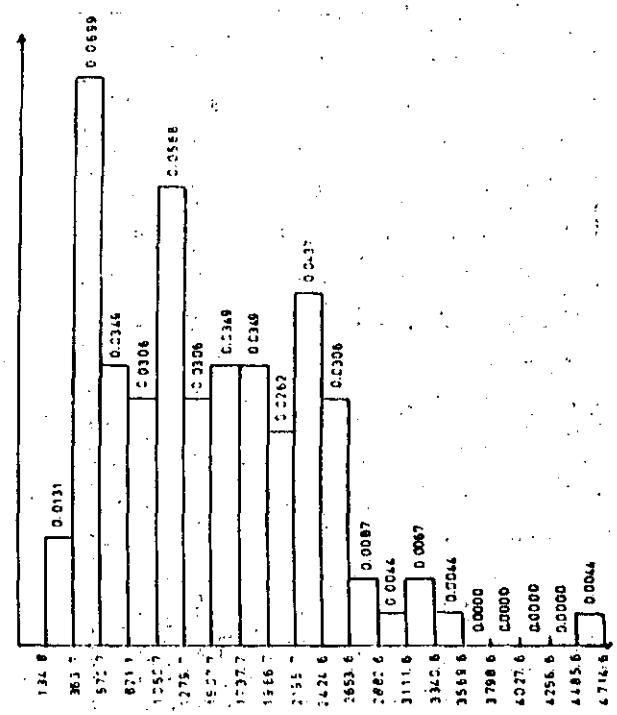
Dobiveno je:

Minimum funkcionala = 148,36

Maksimum funkcionala = 3979,25

Matematičko očekivanje M [f(p)] = 1696,97

Raspored lopatica koji odgovara min f(p) vidimo u tabeli 2., a raspored koji odgovara max f(p) u tabeli 3. Vidimo da su u ovoj metrici dobiveni rezultati različiti od onih u [2] pomoću lančane metrike.



SLIKA 1

TABELA 3

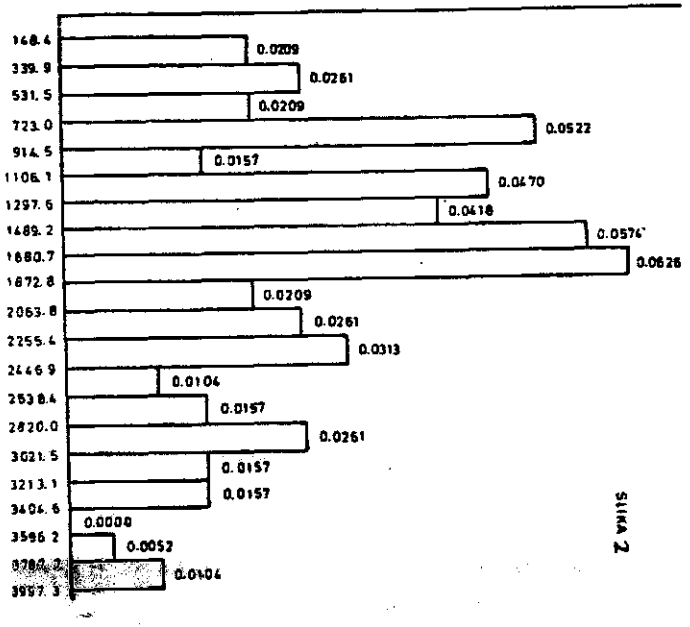
Broj mjesta	1	2	3	4	5	6	7	8	9	1o	11	12	13	14	15	16
Broj lopatica	22	7	45	63	26	46	38	23	4	61	48	12	88	64	54	16
Statički moment	176	420	475	-39	470	- 9	-53	-511	-31	17	81	231	31	468	22	175
Broj mjesta	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Broj lopatica	21	84	4o	11	74	43	95	24	81	20	19	49	37	71	52	59
Statički moment	-46	0	11	181	-23	91	51	-412	-57	-31	-69	15	-259	22	-11	17
Broj mjesta	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
Broj lopatica	82	29	41	78	31	56	91	93	86	89	39	3	79	67	94	1
Statički moment	- 9	-142	-259	31	-152	-11	44	61	-25	65	17	-3o9	-457	15	21	0
Broj mjesta	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
Broj lopatica	2	17	75	34	76	35	90	62	77	7o	8	92	15	14	53	60
Statički moment	109	49	-56	-625	75	- 6	31	-17	49	-26	-145	21	-281	-341	-249	-113
Broj mjesta	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
Broj lopatica	25	10	68	47	27	13	8o	5	58	66	83	18	72	73	30	33
Statički moment	-317	61	41	-191	79	-162	-45	-411	- 2	25	9	0	-11	27	35	13
Broj mjesta	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
Broj lopatica	32	65	9	5o	96	69	87	55	42	36	85	6	51	57	28	44
Statički moment	-88	5	-301	42	31	5	-7	5	15	21	-11	-179	3	5	13	-121

TABELA 2

Broj mjesta	1	2	3	4	5	6	7	8	9	1o	11	12	13	14	15	16
Broj lopatica	89	26	34	79	15	75	2o	9o	91	12	13	78	52	1	9	1o
Statički moment	65	470	-625	-457	-281	-58	-31	51	44	231	-162	31	-11	0	-301	61
Broj mjesta	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Broj lopatica	72	83	71	49	93	82	85	24	62	22	45	16	5	6o	40	56
Statički moment	-11	9	22	15	61	- 9	-11	-412	-17	176	475	175	-411	-113	11	-11
Broj mjesta	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
Broj lopatica	8	69	37	27	8o	42	14	64	33	50	41	70	51	39	95	47
Statički moment	-145	5	-259	79	-45	15	-341	488	13	42	-259	-26	3	17	51	-191
Broj mjesta	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
Broj lopatica	32	11	29	67	61	63	73	31	19	4	84	25	3o	65	43	66
Statički moment	-88	181	-142	15	17	-39	27	-152	-69	-31	0	-317	35	5	91	25
Broj mjesta	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
Broj lopatica	88	36	86	94	3	58	68	81	23	76	28	54	57	21	77	7
Statički moment	31	21	-25	21	-309	-2	41	-57	-511	75	13	22	5	-46	49	420
Broj mjesta	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96
Broj lopatica	17	92	96	6	46	48	74	44	38	59	35	55	18	2	53	87
Statički moment	49	21	31	-179	- 9	81	-25	-121	-53	17	- 6	5	0	109	-209	- 7

TABELA 1

Broj lopatica	1	2	3	4	5	6	7	8	9	1o	11	12	13	14	15	16
Statički moment	0	109	-309	-31	-411	-179	420	-145	-301	61	181	231	-162	-341	-281	175
Broj lopatica	17	18	19	2o	21	22	23	24	25	26	27	28	29	30	31	32
Statički moment	49	0	-69	-31	-46	176	-511	-412	-317	470	79	13	-142	35	-152	-88
Broj lopatica	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
Statički moment	13	-625	- 6	21	-259	- 53	17	11	-259	15	91	-121	475	- 9	-191	81
Broj lopatica	49	5o	51	52	53	54	55	56	57	58	59	6o	61	62	63	64
Statički moment	15	42	3	-11	-249	22	5	-11	- 5	- 2	17	-113	17	-17	-39	488
Broj lopatica	65	66	67	68	69	7o	71	72	73	74	75	76	77	78	79	80
Statički moment	5	25	15	41	5	-26	22	-11	27	-23	-58	75	49	31	-457	-45
Broj lopatica	81	82	83	84	85	86	87	88	89	9o	91	92	93	94	95	96
Statički moment	-57	- 9	9	0	-11	-25	- 7	31	65	31	44	21	61	21	51	31



SLIKA 2

LITERATURA:

1 S.Kurepa - Funkcionalna analiza - Elementi teorije operatora. Školska knjiga, Zagreb 1981.

2 J.Lončar - O jednoj metodi proračuna optimalnog balansa. Biti će publicirano u časopisu INFORMATIKA-LJUBLJANA

3 D.I.Golenko-Statističeskie modeli v upravlenii proizvodstvom. Statistika 1973. 6

U V O D V CP/M* I

ANTON P. ŽELEZNIKAR

SOZD ELEKTROTEHNA, DO DELTA

UDK: 681.3.06 CP/M:181.4

Prvi del uvoda v operacijski sistem CP/M 2.2 obravnava aktualnost, razširjenost in primernost tega sistema, natanko razčlenjuje in pojasnjuje njegove t.im. vgrajene ukaze, krmilne znake, pravila za CP/M zbirke in naposled še tiste prehodne ukaze, ki sodijo neposredno k CP/M sistemu (STAT, PIP, ED, DUMP, SYSGEN, MOVCPM, SUBMIT in XSUB). V nadaljevanju bodo opisani še nadaljnji koncepti, pripomočki in programski paketi, ki sodijo v značilni okvir operacijskega sistema CP/M.

An Introduction to CP/M Operating System I. The first part of the introduction to the CP/M Version 2.2 Operating System deals with importance, spreading, and suitability of this system and describes in detail its built-in commands, control characters (line editing commands), and CP/M file conventions. At the end of the first part CP/M transient commands are presented, also by examples.

1. Uvod

Mikroračunalnik (moR) je povezava medsebojno odvisnih, sodelujočih naprav in programov. Te sestavine moR morajo delovati usklajeno, skladno z uporabniškim programom. CP/M (okrajšava za Control Program/Monitor) je operacijski sistem za moR, ki opravlja nalogo vodenja, usklajevanja in delovanja mikroračunalniškega sistema kot celote, tako da aktivira naprave, ureja podatkovne zbirke, skrbi za ustrezno zaporedje izvajanja programov, komunicira prek konzole z uporabnikom itn.

CP/M lahko uporabljajo tudi domači moR, in sicer ID-80 (Iskra) ter družina Delta 323 (modela /M1 in /M3). Čeprav je CP/M zapleten računalniški program ,je namenjen široki uporabi ter se ga je mogoče priučiti v kratkem času.

CP/M ima vrsto nadaljevalnih produktov, sistemskih in uporabniških. Večuporabniški operacijski sistem ima oznako MP/M (Multiuser CP/M), mreža sistemov s CP/M in MP/M pa se realizira z operacijskim sistemom CP/NET. Sistemska programska oprema za CP/M vsebuje vrsto prevajalnikov za visoke programirne jezike (PL/I, COBOL, FORTRAN, PASCAL, BASIC, ALGOL, LISP, FORTH itd.), uslužnostne sistemske programe (zaščita podatkov, diagnostika), pakete za upravljanje podatkovnih zbirk (DBMS, HDBS, dBASE II, CONDOR itd.) in še bi lahko naštevali. Med značilne uporabniške programe sistema CP/M prištevamo procesorje teksta, finančne in pisarniške programske pakete, programe za odpravljanje poštnih pošiljk itd. itd.

CP/M je bil razvit v letu 1973 (G. Kildall), v letu 1975 pa so ga privzeli tedanji proizvajalci mikrosistemov. Tako je CP/M postal standard na področju moR operacijskih sistemov. CP/M se je lahko priredil za uporabo v poljubnih konfiguracijah procesorjev 8080 in Z80 z različnimi diskovnimi sistemi, nastala pa je tudi močna CP/M uporabniška skupina. Na CP/M so se nadgradili prevajalniki za praktično vse bistvene programirne jezike.

V zaporedju člankov o CP/M bomo obravnavali njegove vgrajene in predhodne ukaze, zbirne in popravljalne pripomočke, uslužnostne, prevajalne in uporabniške programe, sistema MP/M in CP/NET ter zgradbo in uporabo CP/M. Temu bomo dodali še pregledno tabelo ukazov s pojasnili, in sicer za sam CP/M in njegove najbolj značilne nadaljevalne programske pakete.

2. Vgrajeni ukazi CP/M sistema

CP/M je doživel več svojih izdaj, ki so imele označitve 1.3 (originalna izdaja), 1.4 (prečiščena izdaja 1.3), 2.0 (nova izdaja), 2.1 (popravljena izdaja 2.0) in 2.2 (zadnja veljavna izdaja). Mi se bomo omejili na obravnavo izdaje CP/M 2.2. Da bi čim bolj natančno obravnavali sporočila v in iz CP/M, bomo vsa sporočila pisali (in vstavili v naš tekst) na osnovi neposredne interakcije teleprinterja (terminala) in sistema CP/M. Tako se bomo izognili dvoumnostim, ki se pojavljajo v učbenikih o CP/M (glej spisek slovstva) in v realnem komuniciranju uporabnika s sitemom.

CP/M pozna dve bistveni skupini ukazov: vgrajene in prehodne. Vgrajeni ukazi so deli operacijskega sistema, ki se izvaja. Prehodni

---

* CP/M je avtorsko zaščiten izdelek podjetja Digital Research, Pacific Grove, California.

ukazi so imena posebnih zbirk na disku, in če jih aktiviramo, se morajo najprej naložiti iz diska v hitri pomnilnik. Prehodni ukazi so nadalje sistemski (npr. prevajalniki, zbirniki) in uporaabniški (npr. poslovna knjiga, procesiranje teksta).

CP/M sistem sprejema v svojih ukaznih vrsticah velike in male črke, toda notranje jih vselej prepiše v velike črke. Ta dogovor je veljaven tudi za poimenovanje zbirk na disku. Zbirka je poljubna informacija, ki je na disku shranjena kot celota in njena dolžina je omejena s podatkovnim obsegom diska. Zbirke so tako programi, ki se bodo izvajali na moR sistemu, so drugi programi, podatki za določen program, samostojni (informativni) podatki, navodila, poglavje neke knjige, celotna knjiga, strojni kod za vstavitev v ROM itd. Nobeno pravilo ne omejuje informacije, ki je shranjena v zbirki.

Zbirka je lahko strukturirana, ko je sestavljena iz zapisov, polj ali kakšnih drugih enot in podenot.

CP/M 2.X razpozna sedem vgrajenih ukazov, in sicer

DIR TYPE ERA REN SAVE D: USER

Pomen teh ukazov je tale:

DIR D: ime_zbirke.tip

prikaže na zaslonu imenik ali določene zbirke imenika na disku D.

TYPE D: ime_zbirke.tip

prikaže na zaslonu vsebino zbirke z diska D.

ERA D: ime_zbirke.tip

zbriše z diska D eno ali več zbirk.

REN D: novo_ime.tip = staro_ime.tip

preimenuje zbirko na disku D.

SAVE nn D: ime_zbirke.tip

shrani vsebino iz računalniškega pomnilnika v obliki zbirke na disku D v obsegu nn zapisov dolžine 256 zlogov.

D:

aktivira diskovni pogon D.

USER nn

spremeni trenutno veljavno številko uporabnika v nn.

K vgrajenim ukazom moramo dodati še krmilne ASCII znake, ki delujejo prek konzole kot posebni (tudi vgrajeni) ukazi :

´CTL c´ ali ETX

Če je to prvi znak v ukazni vrstici, se izvede ponoven začetek (naložitev) CP/M sistema, to je t. im. ponovni (topli) zagon (začetek).

´CTL e´ ali ENQ

Izvede se pomik na naslednjo ukazno vrstico pri daljših ukaznih zaporedjih.

´CTL h´ ali BS

zbriše nazadnje vtipkani znak in pomakne kurzor v levo.

´CTL j´ ali LF

ima učinek znaka CR (pomaknitev valja v levo).

´CTL m´ ali CR

je pomik kurzorja (valja) na začetek nove vrstice, ko se začne izvajati prejšnja ukazna vrstica.

´CTL p´ ali DLE

vključi in izključi napravo za listanje (tiskalnik).

´CTL r´ ali DC2

ponovi trenutno ukazno vrstico s popravljenimi znaki.

´CTL s´ ali DC3

ustavi podatkovno prikazovanje na zaslonu, ki se lahko nadaljuje s poljubnim znakom (tudi z ´CTL s´)

´CTL u´ ali NAK in ´CTL x´ ali CAN

zbriše (anulira) trenutno ukazno vrstico.

RUB ali DEL ali ´CTL _´

zbriše en znak s t.im. odmevom zbrisanega znaka.

Znaka

? *

Uporaba vprašaja (znak ´?´) in zvezdice (znak ´*´) je v imenih zbirk in njihovih tipov tale :znaka ? in * lahko nadomestita poljubne posamezne znake ali skupine znakov. Če pišemo namesto ime_zbirke.tip izraz *.*, iščemo ali izvajamo ukaz nad vsemi zbirkami določenega diska.

Niz

.

pomeni vse zbirke (ki jih iščemo v imeniku, brišemo z diska) za ukaza DIR in ERA. Ta izraz pa ni veljaven za ukaze TYPE, REN in $AVE. Podobno oblikujemo še kombinaciji *.tip in ime_zbirke.*.

Z vprašajem nadomestimo poljuben znak v imenu zbirke in v njenem tipu.

Oglejmo si nekaj primerov uporabe vgrajenih CP/M ukazov in krmilnih znakov. Za znak CR, ki se na zaslonu ali papirju ne zapiše, bomo vpisali znak ´cr´. Del vrstice, ki ga vnesemo prek tastature, bomo podčrtali, nepodčrtani del bo predstavljal odgovor CP/M sistema.

****
* Primeri: *
****

2.1. Ukazi,povezani z imenikom

DIR in DIR C:

Ta dva ukaza pomenita izpis celotnega imenika na trenutno izbranem diskovnem pogonu in na pogonu C. Imamo primera:

```
A>DIR 'cr'
A: TEX      COM : TEX      PRL : REFM     COM : PLI      TXT
A: CPMINF   TXT : XSSSSSSS LIB : CPMINF   BAK : ED       COM
A: CPMINF1  TXT : CPMINF1  BAK : CPMINF1  PRN : PLI      COM
A: PLI0     OVL : PLI1     OVL : PLI2     OVL : PLI      LIB
A: PLIT     COM : PLIT     DOC : PLI      DOC
A>DIR C:'cr'
C: PIP      COM : SYSGEN   COM : REFM     COM : PLI      TXT
C: CPMINF   TXT : NESSPO   TXT : ED       COM : CBIOS    ASM
C: IKE      COM : IKE1     COM : DELTA    COM
A>
```

Nadalje imejmo

DIR A: PLI?.*

Izpišejo se imena vseh zbirk PLI?, ki imajo namesto znaka '?' kakšen znak (lahko tudi prazen znak) in so poljubnega tipa (npr. COM, OVL itd). Tako dobimo podspisek iz prvega primera:

```
C>DIR A:PLI?.*'cr'
A: PLI      TXT : PLI      COM : PLI0     OVL : PLI1     OVL
A: PLI2     OVL : PLI      LIB : PLIT     COM : PLIT     DOC
A: PLI      DOC
C>A:'cr'
```

DIR C: *.COM

Izpišejo se vse zbirke tipa COM, ko dobimo podspisek iz prvega primera.

```
A>'cr'
A>DIR C:*.COM'cr'
C: PIP      COM : SYSGEN   COM : REFM     COM : ED       COM
C: IKE      COM : IKE1     COM : DELTA    COM
A>C:'cr'
```

DIR PLI.COM

Izpiše se imenovana zbirka :

```
A>DIR PLI.COM'cr'
A: PLI      COM
A>
```

DIR PLI.ASM

Te zbirke ni na disku in imamo:

```
A>DIR PLI.ASM'cr'
NO FILE
A>
```

DIR D:

Na disku D ni nobene zbirke:

```
A>DIR D:'cr'
NO FILE
A>
```

Če vtipkamo namesto DIR

DIW

in na disku ni zbirke DIW.COM, se pojavi sporočilo o napaki, in sicer:

```
A>DIW'cr'
DIW?

A>
```

V primeru, ko CP/M ne more najti diska v izbranem diskovnem pogonu D, ko npr. disk ni vstavljen v pogonsko enoto, ko ni pravilno formatiran (inicializiran), ko ni napetosti v diskovni enoti, ali če vrata enote niso zaprta, se pojavi sporočilo

BDOS ERR ON D:

2.2. Ukaza za prikazovanje ASCII zbirk

Imamo samo dve vrsti ukazov, in sicer

```
TYPE ime_zbirke.tip
TYPE D: ime_zbirke.tip
```

V konkretnem primeru imamo:

```
A>TYPE PESMICA.POE'cr'
OJ, RDECA KAPICA,
 V GOZD JE SLA PO VOLKECA
...
```

Znak

'CTL s'

uporabljamo za ustavljanje (in nadaljevanje) prikazovanja na zaslonu, z znakom

'CTL c'

pa ukinjamo prikazovanje (ne listamo do konca).

Kadar uporabimo ime zbirke ali tipa, ki ne obstaja (ga ni na disku), če uporabimo ? ali *, se pojavi sporočilo

```
A>GREMO.GOR'cr'
GREMO.GOR?

A>GREMO.???'cr'
GREMO.????

A>GREM*.*'cr'
GREM*.*?

A>
```

Pri izvajanju ukaza TYPE se na zaslonu lahko pojavijo le nekateri ASCII znaki (črke, številke, ločila), drugi pa lahko povzročijo izvajanje zaslonskih oziroma kurzorskih funkcij. Zaradi tega z ukazom TYPE ne prikazujemo zbirk, ki so tipa COM, REL, OBJ, INT itd.

Ukaz TYPE se uporablja za prikazovanje zbirk tipov ASM, PLI, BAS, BAK, DAT, HEX, DOC, PAS, FOR, COB oziroma za vse tiste zbirke, ki vsebujejo ASCII tekst oziroma ASCII podatke (npr. STREAM zbirke v PLI programih).

2.3. Ukazi za brisanje zbirk

ERA *.* ali ERA D:*.*

Ta dva ukaza zbrišeta vse zbirke iz diska na trenutno izbrani diskovni enoti ali na enoti D. Imamo tole interakcijo:

```
A>DIR D:'cr'
D: IME      COM : IME      PRN : IME      ASM : IME      SYM
D: PRIIMEK  COM : PRIIMEK  REL : PRIIMEK  PRN : PRIIMEK  PLI
D: ULICA    OBJ : ULUCA    INT : ULICA    BAS
A>ERA D:*.*'cr'
ALL (Y/N)?Y
A>DIR D:'cr'
NO FILE
A>
```

Po izbrisu lahko preverimo izbris z ukazom DIR. Seveda smo pred izbrisom vseh zbirk iz diska z ukazom DIR preverili, ali res lahko izbrišemo vse zbirke.

Z ukazom ERA lahko brišemo tudi skupine zbirk ali eno samo zbirko, pri čemer je skupina na določen način tipizirana (npr. vse zbirke tipa COM ali ASM itd.). Imejmo disk z več zbirkami, ko dobimo z ukazom DIR tole:

```
D>DIR'cr'
D: IME      ASM : IME      COM : PRIIMEK  ASM : PRIIMEK  COM
D: ULICA    ASM : ULICA    COM : KRAJ     ASM : KRAJ     COM
D: DRZAVA   ASM : DRZAVA   PRN : KONTINEN ASM : KONTINEN PRN
D>
```

Nad tem diskom uporabimo ukaz

ERA *.ASM

Po izbrisu dobimo z ukazom DIR tole:

```
D>ERA *.ASM'cr'
D>DIR'cr'
D: IME      COM : PRIIMEK  COM : ULICA    COM : KRAJ     COM
D: DRZAVA   PRN : KONTINEN PRN
D>
```

Podobno lahko uporabimo tudi ukaza

ERA IME.*
ERA IME1.COM

ko se v prvem primeru zbrišejo zbirke IME.COM, IME.ASM, IME.PRN in IME.SYM, v drugem primeru pa le zbirka IME1.COM.

Ukaz ERA ne dopušča uporabe vprašajev!

Če vtipkamo s konzole

ERA BRIS

in zbirka BRIS ni na disku, se pojavi sporočilo, in sicer:

```
D>ERA BRIS'cr'
NO FILE
D>
```

Za sporočilo BDOS ERROR ON D: veljajo podobna pojasnila, kot smo jih imeli za tovrstne napake pri ukazu DIR.

Pred uporabo ukaza ERA vedno preverimo disk z ukazom DIR in po izbrisu zopet uporabimo ukaz DIR.

2.4. Ukazi za preimenovanje zbirk

Ukaz za preimenovanje ima le dve obliki, in sicer

REN novo_ime.tip = staro_ime.tip
REN D:novo_ime.tip = staro_ime.tip

Vzemimo tale primer, ko imamo na disku:

```
D>DIR'cr'
D: PESMICA  LEP : PESMICA  GRD : PESMICA  UH
D>
```

Po izvedbi ukaza REN dobimo:

```
D>REN LIRIKA.AH=PESMICA.GRD'cr'
D>DIR'cr'
D: PESMICA  LEP : LIRIKA   AH  : PESMICA  UH
D>
```

V primeru napačnega imena imamo:

```
D>REN LIRIKA.OH=LIR?IKA.AH'cr'
LIR?IKA.AH?

D>
```

V primeru, ko je novo_ime že ime na disku, se pojavi sporočilo:

```
D>'cr'
D>REN PESMICA.LEP=PESMICA.UH'cr'
FILE EXISTS
D>
```

Ukaz REN ne dovoljuje uporabe znakov '?' in '*' v imenih in tipih zbirk.

2.5. Shranitev pomnilniške vsebine v diskovno zbirko

Pomnilniška vsebina, ki se nahaja v t.im. prehodnem pomnilniškem območju (od naslova 100H navzgor), se shrani kot diskovna zbirka. Oblika ukaza SAVE je ena sama, in sicer:

SAVE nnn x:ime_zbirke.tip

Tu je nnn decimalno število, ki predstavlja število strani pomnilnika, pri čemer je stran sestavljena iz 256 zlogov. x: lahko tudi izpustimo, če je ime zbirke že na trenutno aktivirani diskovni enoti.

Vzemimo primer, ko zbirko tipa HEX pretvorimo v zbirko tipa COM. Imamo tole:

```
A>DDT IKE.HEX'cr'
DDT VERS 2.2
NEXT  PC
1100 0100
-^C
A>SAVE 16 IKE.COM'cr'
A>
```

Tu sta A> znaka sistema CP/M, kjer označuje A trenutno aktivirano diskovno enoto in znak >, da je sistem pripravljen za sprejetje ukaza s konzole. Nato sledi ukaz

```
DDT IKE.HEX
```

s katerim se naloži zbirka IKE.HEX v pomnilnik, in sicer od naslova 100H navzgor. Tretja vrstica kaže stanje programskega števnika PC (naslov 100H) in naslednje proste lokacije NEXT (naslov 1100H). Zbirka IKE.TXT se tako v pomnilniku razprostira od lokacije 100H do 1100H. V peti vrstici se nahaja znak 'CTL-c', ko se doseže ponovna naložitev sistema CP/M. V šesti vrstici uporabimo nato ukaz SAVE. Na disku imamo tako zbirki IKE.HEX in IKE.COM, pri čemer je IKE.COM izvršljiva zbirka (se lahko uporabi kot ukaz IKE).

2.6. Izbira diskovne enote

Diskovno enoto izberemo z ukazom

```
d:
```

na začetku vrstice, ko je d = A,B,...,P. Tako pomeni A: izbiro enote A, ko imamo:

```
C>A:'cr'
A>B:'cr'
B>C:'cr'
C>D:'cr'
D>
```

2.7. Spremembe uporabniške številke

Ukaz USER je en sam, in sicer

```
USER n
```

kjer je n = 0,1,...,15. Če vtipkamo

```
A>USER 16'cr'
NO FILE
A>
```

se pojavi sporočilo o napaki. Po vključitvi oz. z mrzlim začetkom imamo uporabniško številko nič. Vsaka novo oblikovana zbirka se zapiše kot uporabniška zbirka s številko nič. Ko pa vtipkamo npr. USER 2 pred ukazom SAVE, se bo ime te nove zbirke pojavilo samo v imeniku uporabnika s številko 2. Ukaz ERA se bo v tem primeru nanašal samo na imenik uporabnika 2.

3. Zbirke CP/M sistema

Vsaka CP/M zbirka se identificira s svojim imenom in pripono (tipom). CP/M zbirka ima enolično ime, ki je sestavljeno iz enega do osem znakov, z velikimi in/ali malimi črkami. CP/M pretvori male črke v velike še pred identifikacijo.

V zbirčnih imenih se ne smejo pojavljati tile znaki:

```
. , ; : = * < > [ ] ?
```

Prav tako ni dopustna uporaba krmilnih ASCII znakov.

Zbirke so lahko različnih tipov, kar pomeni, da se obravnavajo na različne načine. S pripono je v bistvu določena funkcija (pomen) zbirke. Pripona (za piko) je vselej sestavljena iz 0 do 3 znakov (podobno kot ime). Ime in pripona sta ločena s piko. Nekatere znane pripone so:

.ALG	zbirka ALGOL programa
.ASM	zbirka v zbirnem jeziku
.BAK	prejšnja zbirka
.BAS	zbirka BASIC programa
.COB	zbirka COBOL programa
.COM	izvršljiva zbirka
.DAT	podatkovna zbirka
.DOC	dokumentacijska zbirka
.FOR	zbirka FORTRAN programa
.HEX	heksadecimalna zbirka
.INT	zbirka v vmesnem kodu (BASIC)
.LIB	knjižnična zbirka
.MAC	makrojska zbirka
.OBJ	objektna zbirka
.PAS	zbirka PASCAL programa
.PCO	pascalski izvajalni modul
.PLI	zbirka PL/I programa
.PRL	po straneh premestljiva zbirka
.REL	premestljivi strojni program
.SRC	izvirna zbirka CP/M uporabnikov
.SUB	submit zbirka
.TXT	tekstovna zbirka
.$$$	začasna zbirka

Pripone lahko seveda oblikujemo tudi sami, toda nekateri programi (npr. prevajalniki, povezovalniki) zahtevajo natanko določene pripone zbirk, nad katerimi se uporabljajo (BAS, INT, PLI, REL itn.). Neposredno s CP/M sistemom izvajajoči programi morajo imeti pripono COM. Svojim specifičnim zbirkam pa lahko damo tudi številske pripone npr.:

```
IKE.001, IKE.202, IKE.025  itd.
```

Pripono lahko tudi izpustimo in imamo npr. samo ime IKE, ali pa tudi IKE.0, IKE.12 itn.

Kasneje, ko bomo opisovali t.im. prehodne ukaze, bomo spoznali, da je lahko ime zbirke (brez navedbe njene pripone) ukaz, ki se izvede tako, da se ta zbirka, ki ima pripono COM, najprej naloži v hitri pomnilnik (od lokacije 100 navzgor) ter se po naložitvi začne izvajati kot program. Pripone bomo izpuščali tudi pri navedbah zbirk v drugih ukazih, ki sami poiščejo zbirko z navedenim imenom in z ustrezno pripono. Taki primeri se pojavljajo pri kompiliranju, povezovanju zbirk in še kje.

Pri navajanju oz. uporabi zbirk navajamo večkrat tudi ime diskovne enote, ki mu sledi zadaj znak ':'. Imena enot so A, B, C, d, e itd. Osnovna (sistemska) diskovna enota je vedno A, saj se le prek te enote izvede začetni (mrzli) ali ponovni (topli) zagon sistema. Tako lahko izvajamo operacije nad zbirkami iz pozicije ene diskovne enote, čeprav se te zbirke nahajajo na drugih diskovnih enotah.

Poznamo tudi enolične in večlične zbirčne navedbe. Pri večličnih zbirčnih navedbah uporabljamo pri imenu in priponi zbirk znaka '?' in '*'. Če zapišemo v ukazu navedbo zbirke *.*, potem pomeni ta navedba vse zbirke na dani disketi, navedba *.TIP pa vse zbirke s pripono TIP. Podobno pomeni navedba IME.* vse zbirke z imenom IME ne glede na TIP.

Drugo možnost imamo, ko vstavljamo v imena in pripone znake '?'. Tako pomeni PLI?.* vse zbirke PLI.*, PLI0.*, PLI1.*, PLI2.* itd. Navedba *.BA? bi pomenila vse zbirke *.BAK, *.BAS itd.

4. Prehodni ukazi

4.1 Uvod

Doslej smo si ogledali le t.im. vgrajene ukaze. Operacijski sistem CP/M pa dopušča praktično neomejeno razširitev zaloge ukazov. Kako CP/M sistem ugotovi, da so na disku še drugi ukazi?

Ko CP/M ugotovi, da vtipkani ukaz ni vgrajeni ukaz, pogleda v imenik diska, ali se tu nahaja zbirka z vtipkanim imenom (tj. ukaz), ki ima pripono COM. Če je npr. IKE moj novi ukaz, potem mora CP/M najti v imeniku zadevnega diska zbirko IKE.COM. Pri tem se pojavi vprašanje, kakšen je program, ki deluje kot ukaz. Imamo tole osnovno definicijo: Program je zaporedje ukazov, ki je shranjeno na disku v zbirki s pripono COM in tak program se pokliče v izvajanje z vstavitvijo njegovega imena (brez pripone) s konzolę. Vsak tak program se imenuje prehoden program ali prehoden ukaz.

Za t.im. prehodni ukaz imamo tako tole shemo:

```
A> UKAZ parametri 'cr'
|   |        |        |
|   |        |        L- pomik valja označuje ko-
|   |        |             nec ukazne vrstice
|   |        L---------- ukaz ima lahko tudi para-
|   |                      metre
|   L------------------- ukaz, ki bo izvršen
L----------------------- znak pripravljenosti si-
                           stema CP/M
```

Torej lahko posplošimo:

```
A> ime_zbirke parametri 'cr'
|      |          |       |
|      |          |       L- pomik valja na
|      |          |            koncu vrstice
|      |          L--------- možni parametri
|      L-------------------- COM zbirka, ki
|                              bo naložila in
|                              izvršila
L--------------------------- CP/M pripravlje-
                               nost
```

Beseda "prehoden" pomeni, da ukaz oziroma program ni stalno naložen v hitrem pomnilniku, shranjen pa je na disku. Beseda "disk" lahko pomeni upogljivo disketo ali diskovno napravo tipa Winchester. Operacijski sistem CP/M 2.2 omogoča tudi uporabo t.im. trdnih (winchesterskih) diskov, ki lahko shranijo veliko večje število zlogov kot upogljivi diski.

Proizvajalec CP/M sistema (Digital Research) ima svojo zalogo prehodnih ukazov, ki si jih bomo na kratko ogledali v nadaljevanju. Ti ukazi so:

STAT
PIP
ED
DUMP
SYSGEN
MOVCPM
ASM
DDT
LOAD
SUBMIT
XSUB

4.2. STAT: statistika

Ukaz STAT, ki pomeni statistiko, daje podatke o zbirki ali skupini zbirk na disku ter še vrsto drugih podatkov, ki so za uporabnika bistveni. Med take podatke sodijo tudi podatki o napravah. Opišimo najprej pet osnovnih možnosti, nato pa si oglejmo še primere, tako kot se pojavijo na konzolnem terminalu. Imamo tole:

STAT 'cr'
: Pokažejo se pridevki in količina praznega prostora na vseh disketah, ki so bile po zadnjem mrzlem ali toplem zagonu dostopljene

STAT d:'cr'
: Pokaže se količina praznega prostora na disku 'd'

STAT d:ime_zbirke.tip'cr'
: Pokažejo se obseg in pridevki ene ali več zbirk ime_zbirke.tip (v primerih večumnosti, ko uporabljamo znaka '?' in '*'. Če izpustimo 'd:', je mišljen trenutno izbrani disk

STAT d:ime_zbirke.tip$pri'cr'
: Zbirki ime_zbirke.tip se priredi pridevek 'pri' na disku 'd'. Opombe glede večumnosti so enake kot v prejšnjem primeru (podobno tudi za 'd:')

STAT d:*.* 'cr'
: Prikaže se statistika celotnega diska

Ukaz STAT prikazuje taku obsege in veljavne pridevke zbirk in diskov, spreminja pa tudi obstoječe (veljavne) pridevke v nove. Če nam npr. STAT sporoči, da je na disku 114k razpoložljivega prostora, imamo na zadevnem disku prostor za 114 krat 1024 zlogov; to pomeni, da smemo na disk zapisati še 116736 novih zlogov.

Uporabnik CP/M sistema lahko prireja svojim zbirkam na diskih dva para zbirčnih pridevkov: R/O ali R/W in DIR ali SYS. Imamo tale pomen:

R/O ...
Imamo bralno zbirko (Read-Only). Take zbirke ni moč popravljati (zapisovati v njo) ali zbrisati z uporabo ukaza ERA. R/O stanje ščiti zbirko pred možnostjo naključnega brisanja ali njenega spreminjanja

R/W ...
Imamo bralno/pisalno zbirko (Read/Write). Take zbirke lahko popravljamo in brišemo (če seveda sam disk nima materialne pisalne zaščite). Ta pridevek je normalen in se nastavi samodejno.

SYS ...
Imamo sistemsko zbirko. Take zbirke se ne prikazujejo v okviru diskovnega imenika (so zakrite), se torej nikoli ne listajo v imeniku, izjema je le ukaz STAT *.*, ki pokaže vse zbirke. Sistemska zbirka se sicer obnaša kot normalna (R/W) zbirka. Vse zbirke tipa SYS so na voljo vsem uporabnikom (vsem uporabniškim območjem).

DIR ...
Imamo imeniško zbirko, ki se pojavlja v vseh imeniških prikazih trenutnega uporabniškega področja. Ta pridevek je normalen in se nastavi samodejno.

Medsebojno se tako izključujeta pridevka R/O in R/W ter pridevka SYS in DIR. Kadar želimo pisati na R/O disk, se pojavi sporočilo

BDOS ERR ON d:R/O

Operacijski sistem zazna tudi zamenjavo diska in se javlja z enakim sporočilom, če pri zamenjavi diska nismo uporabili toplega zagona, to je znaka 'CTL c'.

4.3. Primeri uporabe STAT ukaza

Imejmo disk, na katerem se nahajajo različne zbirke. Želimo ugotoviti, katere zbirke so na disku, koliko prostora zaseda posamezna zbirka, kakšen je njen pridevek in koliko prostora je še na razpolago za nove zbirke. Imamo tale primer:

```
A>STAT *.*'cr'

 RECS  BYTES  EXT ACC
  126    16K    1 R/W A:CPMINF.BAK
  126    16K    1 R/W A:CPMINF.TXT
   68     9K    1 R/W A:CPMINF1.BAK
   75    10K    1 R/W A:CPMINF1.PRN
   70     9K    1 R/W A:CPMINF1.TXT
   60     8K    1 R/W A:CPMINF2.BAK
   64     8K    1 R/W A:CPMINF2.PRN
   60     8K    1 R/W A:CPMINF2.TXT
   52     7K    1 R/W A:ED.COM
    0     0K    1 R/W A:PESMICA.BAK
    1     1K    1 R/W A:PESMICA.POE
    0     0K    1 R/W A:PLI2.BAK
   31     4K    1 R/W A:PLI2.TXT
   85    11K    1 R/W A:PLIINF1.TXT
   73    10K    1 R/W A:REFM.COM
    0     0K    1 R/W A:REZIME.BAK
    8     1K    1 R/W A:REZIME.PRN
    8     1K    1 R/W A:REZIME.TXT
    0     0K    1 R/W A:SLIKA.BAK
   11     2K    1 R/W A:SLIKA.PRN
   10     2K    1 R/W A:SLIKA.TXT
   41     6K    1 R/W A:STAT.COM
   66     9K    1 R/W A:TEX.COM
   77    10K    1 R/W A:TEX.PRL
   37     5K    1 R/W A:UVOD.BAK
   49     7K    1 R/W A:UVOD.PRN
   43     6K    1 R/W A:UVOD.TXT
    6     1K    1 R/W A:UVOD1.BAK
    6     1K    1 R/W A:UVOD1.PRN
    5     1K    1 R/W A:UVOD1.TXT
BYTES REMAINING ON A: 72K
```

Kot vidimo, so imena zbirk v tem spisku strogo abecedno urejena, kar vobče ne velja za imenik, ki ga izlistamo z vgrajenim ukazom DIR. V prvem stolpcu imamo število zapisov, pri čemer znaša en zapis 128 zlogov, kot ustreza CP/M formatu zbirke. Zbirka se vselej zaokroži na štirikratnik zapisa, kar je razvidno iz stolpca z oznako 'BYTES'. Nadalje označuje 'EXT' število obsegov po 16k zlogov diskovnega prostora. Logični obsegi sodijo v mehanizem, s katerim CP/M upravlja zbirke. Stolpec 'ACC' označuje pridevek dostopa, ki je ali R/W ali R/O.

Z uporabo večumnega znaka '*' dobimo iz zadnjega seznama podseznam vseh tistih zbirk, katerih imena imajo pripono 'TXT', z vsemi pripadajočimi podatki in s podatkom o prostem prostoru na disku:

```
A>STAT *.TXT'cr'

 RECS  BYTES  EXT ACC
  126    16K    1 R/W A:CPMINF.TXT
   70     9K    1 R/W A:CPMINF1.TXT
   60     8K    1 R/W A:CPMINF2.TXT
   31     4K    1 R/W A:PLI2.TXT
   85    11K    1 R/W A:PLIINF1.TXT
    8     1K    1 R/W A:REZIME.TXT
   10     2K    1 R/W A:SLIKA.TXT
   43     6K    1 R/W A:UVOD.TXT
    5     1K    1 R/W A:UVOD1.TXT
BYTES REMAINING ON A: 72K
```

V naslednjem primeru se prikažejo vse zbirke, katerih imena začenjajo s podnizom 'CPM', ko imamo:

```
A>STAT CPM????.*'cr'

 RECS  BYTES  EXT ACC
  126    16K    1 R/W A:CPMINF.BAK
  126    16K    1 R/W A:CPMINF.TXT
   68     9K    1 R/W A:CPMINF1.BAK
   75    10K    1 R/W A:CPMINF1.PRN
   70     9K    1 R/W A:CPMINF1.TXT
   60     8K    1 R/W A:CPMINF2.BAK
   64     8K    1 R/W A:CPMINF2.PRN
   60     8K    1 R/W A:CPMINF2.TXT
BYTES REMAINING ON A: 72K
```

Podatke za posamezno zbirko dobimo v primeru

```
A>STAT CPMINF.TXT'cr'

 RECS  BYTES  EXT ACC
  126    16K    1 R/W A:CPMINF.TXT
BYTES REMAINING ON A: 72K
```

Če zbirke ni na disku, se pojavi sporočilo in imamo npr.

```
A>STAT CPM.TXT'cr'

FILE NOT FOUND
A>
```

Pri treh diskovnih napravah dobimo sporočilo:

```
A>B:STAT'cr'
A: R/W, SPACE: 72K
B: R/W, SPACE: 82K
C: R/W, SPACE: 157K
```

Kot vidimo, ima naš disk še za 72k zlogov razpoložljivega prostora, diska B in C pa imata po 82k in 157k prostora.

Spremenimo satus zbirke PESMICA.POE v bralni, ko dobimo

```
A>STAT PESMICA.POE $R/O'cr'

PESMICA.POE SET TO R/O
```

Nato spremenimo status te zbirke v sistemski, ko imamo

```
A>STAT PESMICA.POE $SYS'cr'

PESMICA.POE SET TO SYS
A>
```

Sedaj preverimo novi stanji zbirke PESMICA.POE, ko dobimo

```
A>STAT PESMICA.POE'cr'

 RECS  BYTES  EXT ACC
    1     1K    1 R/O A:(PESMICA.POE)
BYTES REMAINING ON A: 72K
```

Sistemska zbirka se pri uporabi ukaza STAT zapiše v oklepajih in tudi v celotnem seznamu imamo:

```
A>STAT *.* 'cr'

 RECS  BYTES  EXT ACC
  126    16K    1 R/W A:CPMINF.BAK
  126    16K    1 R/W A:CPMINF.TXT
   68     9K    1 R/W A:CPMINF1.BAK
   75    10K    1 R/W A:CPMINF1.PRN
   70     9K    1 R/W A:CPMINF1.TXT
   60     8K    1 R/W A:CPMINF2.BAK
   64     8K    1 R/W A:CPMINF2.PRN
   60     8K    1 R/W A:CPMINF2.TXT
   52     7K    1 R/W A:ED.COM
    0     0K    1 R/W A:PESMICA.BAK
    1     1K    1 R/O A:(PESMICA.POE)
    0     0K    1 R/W A:PLI2.BAK
   31     4K    1 R/W A:PLI2.TXT
   85    11K    1 R/W A:PLIINF1.TXT
   73    10K    1 R/W A:REFM.COM
    0     0K    1 R/W A:REZIME.BAK
    8     1K    1 R/W A:REZIME.PRN
    8     1K    1 R/W A:REZIME.TXT
    0     0K    1 R/W A:SLIKA.BAK
   11     2K    1 R/W A:SLIKA.PRN
   10     2K    1 R/W A:SLIKA.TXT
   41     6K    1 R/W A:STAT.COM
   66     9K    1 R/W A:TEX.COM
   77    10K    1 R/W A:TEX.PRL
   37     5K    1 R/W A:UVOD.BAK
   49     7K    1 R/W A:UVOD.PRN
   43     6K    1 R/W A:UVOD.TXT
    6     1K    1 R/W A:UVOD1.BAK
    6     1K    1 R/W A:UVOD1.PRN
    5     1K    1 R/W A:UVOD1.TXT
BYTES REMAINING ON A: 72K
```

Če spremenimo status zbirke CPMINF.BAK iz stanja DIR v stanje SYS, tedaj se v imeniku, ki ga dobimo z ukazom DIR, ne bosta pojavili zbirki PESMICA.POE in CPMINF.BAK, kot kaže naslednji primer:

```
A>DIR 'cr'
A: TEX      COM : TEX      PRL : REFM     COM : PLIINF1  TXT
A: CPMINF   TXT : CPMINF2  PRN : CPMINF2  TXT : CPMINF1  TXT
A: CPMINF1  BAK : CPMINF1  PRN : PLI2     BAK : PLI2     TXT
A: STAT     COM : PESMICA  BAK : REZIME   BAK : REZIME   TXT
A: REZIME   PRN : SLIKA    BAK : SLIKA    TXT : SLIKA    PRN
A: UVOD     TXT : UVOD     BAK : UVOD     PRN : UVOD1    BAK
A: UVOD1    TXT : UVOD1    PRN : CPMINF2  BAK : ED       COM
A>
```

Imenik, ki ga dobimo z ukazom DIR, ni abecedno urejen in zbirke se pojavljajo nekako v zaporedju, kot so bile vpisane na disk. Izjema v zaporedju lahko nastopi pri zbirki tipa COM, ki se mora zapisati na strnjenem odseku lokacij (ostale zbirke so lahko razpršene po disku).

4.4. 'Statistika' naprav

Ukaz STAT za naprave daje podatke o fizičnih in logičnih napravah. Imamo tele značilne primere:

STAT dev:'cr'
: Ta ukaz da sporočilo, kako so fizične naprave povezane s štirimi logičnimi napravami

STAT val:'cr'
: Sporočilo vsebuje podatke o možnih prireditvah naprav in o nekaterih STAT ukazih

STAT log:=fiz:'cr'
: S tem ukazom se priredi fizična naprava fiz: logični napravi log: (v vrstici je lahko več prireditev, ločenih z vejico)

STAT usr:'cr'
: Sporoči se trenutno število uporabnikov in tiste uporabniške številke, za katere so zbirke na diskih (na diskih, ki so bili dostopljeni)

STAT d:dsk:'cr'
: Sporoči se karakteristika diska 'd'

STAT d:=R/O'cr'
: Disk 'd' ima s tem pisalno zaščito

CP/M sistem pozna logične in fizične naprave. Logične naprave imenujemo tudi kanale. Kanal ali logična naprava označuje neko splošno funkcijo mikroračunalnika, dočim je fizična naprava neka periferna enota, ki smo jo izbrali za opravljanje te funkcije. CP/M zahteva, da izberemo fizično napravo in jo priredimo kanalu. Prireditev dosežemo z uporabo ustreznega STAT ukaza.

CP/M sistem ima štiri logične kanale (naprave):

CON: je operatorska konzolna funkcija za vstaljanje ukazov in za prikazovanje podatkov
RDR: je bralna funkcija za trak, za sprejem podatkov
PUN: je luknjalna funkcija za trak, za oddajo podatkov
LST: je funkcija za listanje, za listanje podatkov

Nadalje imamo 12 mogočih fizičnih naprav:

TTY: je počasna konzola (teleprinter)
CRT: je hitra konzola (prikazovalnik s katodno elektronko)
BAT: je paketni procesor
UC1: je uporabniško določena konzola
PTR: je bralnik traku
PTP: je luknjalnik traku
UR1: je uporabniški bralnik štev. 1
UR2: je uporabniški bralnik štev. 2
UP1: je uporabniški luknjalnik štev. 1
UP2: je uporabniški luknjalnik štev. 2
LPT: je vrstični tiskalnik
UL1: je uporabniška naprava za listanje

Mogočih je le 16 prireditev fizičnih naprav logičnim kanalom:

Funkcijo CON: lahko realiziramo s fizičnimi napravami TTY:, CRT:, BAT: in UC1:.
Funkcijo RDR: je mogoče opraviti s fizičnimi napravami TTY:, PTR:, UR1: in UR2:.
Za funkcijo PUN: lahko uporabimo fizične naprave TTY:, PTP:, UP1: in UP2:.
Funkcija LST: se lahko uresniči z uporabo fizičnih naprav TTY:, CRT:, LPT: in UL1:.

Seveda pa moramo napisati programe za uporabo fizičnih naprav za vsak poseben mikroračunalniški sistem sami. Te programe napiše navadno proizvajalec mikroračunalniškega sistema. Nadalje se pri zagonu sistema izvede določena inicializacija, ki priredi kanalom standardne fizične naprave. Za tračne naprave velja, da niso le papirne, marveč so lahko tudi magnetne.

Naprava BAT: dejansko ni fizična in ima tale pomen: če priredimo BAT: napravo CON: kanalu, se bo pojavil vhod iz trenutne RDR: naprave, dočim se bo konzolni izhod opravil na trenutno LST: napravo.

V nadaljnem si oglejmo nekaj primerov. Uporabimo ukaz STAT dev:'cr':

```
B>STAT DEV:'cr'
CON: IS TTY:
RDR: IS URI:
PUN: IS UP2:
LST: IS UL1:
```

V levem stolpcu imamo logične kanale, v desnem pa njim prirejene fizične naprave. To pomeni, da smo predhodno uporabili (v našem primeru) STAT ukaz za prireditev (glej kasneje).

Naslednji ukaz naj bo STAT val:'cr', ki prikaže pregled vseh prireditvenih možnosti in rabi kot navodilo uporabniku. Imamo:

```
B>STAT VAL:'cr'

TEMP R/O DISK: D:=R/O
SET INDICATOR: D:FILENAME.TYP $R/O $R/W $SYS $DIR
DISK STATUS  : DSK: D:DSK:
USER STATUS  : USR:
IOBYTE ASSIGN:
CON: = TTY: CRT: BAT: UC1:
RDR: = TTY: PTR: URI: UR2:
PUN: = TTY: PTP: UP1: UP2:
LST: = TTY: CRT: LPT: UL1:
B>
```

Tu imamo:

D:=R/O	Vtipkaj STAT d:=R/O'cr', da postane disk d: bralen (toda ne pisalen)
D:=FILENAME.TYP $pri	Vstavi STAT d:ime_zbirke.tip $ pri'cr' za nastavitev zbirčnega pridevka
DSK D:DSK:	Vstavi STAT d:DSK:'cr' za pregled pomnilniških značilnosti danega diska d: Vtipkaj STAT DSK:'cr' za pregled pomnilniških značilnosti vseh predhodno dostopljenih diskovnih enot
USR:	Vstavi STAT USR:'cr' za prikaz stanja uporabnikov
IOBYTE ASSIGN	Možne prireditve fizičnih naprav logičnim kanalom

Ukaz za prirejanje lahko izpišemo v primeru več prireditev tudi v okrajšani obliki, ko imamo:

```
B>STAT CON:=CRT:,LST:=LPT:,RDR:=PTR:,PUN:=TTY:'cr'
```

Posledica tega ukaza je:

```
B>STAT DEV:'cr'
CON: IS CRT:
RDR: IS PTR:
PUN: IS TTY:
LST: IS LPT:
```

Ukaz STAT usr:'cr' prikaže trenutno število uporabnikov ter izlista uporabniške številke, za katere obstajajo na disku pripadajoče zbirke:

```
B>STAT USR:'cr'

ACTIVE USER : 0
ACTIVE FILES: 0
B>
```

Tu pomeni:

ACTIVE USER : 0	Število trenutno aktivnih uporabnikov je nič
ACTIVE FILES : 0	Na disku so zbirke z uporaniško številko 0, drugih pa ni (1, 2, 3, 4)

Z ukazom STAT x:DSK:'cr' prikažemo, kako so podatki shranjeni na disku x:, ko imamo npr.

```
B>STAT A:DSK:'cr'

   A: DRIVE CHARACTERISTICS
1944: 128 BYTE RECORD CAPACITY
 243: KILOBYTE DRIVE  CAPACITY
  64: 32  BYTE DIRECTORY ENTRIES
  64: CHECKED  DIRECTORY ENTRIES
 128: RECORDS/ EXTENT
   8: RECORDS/ BLOCK
  26: SECTORS/ TRACK
   2: RESERVED TRACKS
```

Tu nastavimo x: z izbiro (A, B, C, ...), če pa ga izpustimo, se prikažejo pregledi za vse predhodno dostopljene enote (po hladnem ali toplem zagonu sistema), ko imamo npr.

```
B>STAT DSK:'cr'

   A: DRIVE CHARACTERISTICS
1944: 128 BYTE RECORD CAPACITY
 243: KILOBYTE DRIVE  CAPACITY
  64: 32  BYTE DIRECTORY ENTRIES
  64: CHECKED  DIRECTORY ENTRIES
 128: RECORDS/ EXTENT
   8: RECORDS/ BLOCK
  26: SECTORS/ TRACK
   2: RESERVED TRACKS

   B: DRIVE CHARACTERISTICS
1944: 128 BYTE RECORD CAPACITY
 243: KILOBYTE DRIVE  CAPACITY
  64: 32  BYTE DIRECTORY ENTRIES
  64: CHECKED  DIRECTORY ENTRIES
 128: RECORDS/ EXTENT
   8: RECORDS/ BLOCK
  26: SECTORS/ TRACK
   2: RESERVED TRACKS
```

Ti podatki bržkone ne bodo posebno zanimivi za uporabnika, njihov pomen pa je tale:

128 BYTE RECORD CAPACITY
: to je maksimalno število 128-zložnih zapisov, ki se lahko shranijo na disku

KILOBYTE DRIVE CAPACITY
: to je maksimalno število kilozlogov, ki se lahko shranijo na disku

32 BYTE DIRECTORY ENTRIES
: je maksimalno število zbirk, ki se lahko shranijo na disku

CHECKED DIRECTORY ENTRIES
: je enako kot za 32 BYTE DIRECTORY ENTRIES pri zamenljivih diskih, toda enako '0' za trdne diske

RECORDS/ EXTENT
: je maksimalno število zapisov na vstop v zbirčni imenik

RECORDS/ BLOCK
: je minimalna količina prostora na disku, ki se lahko dodeli zbirki

SECTORS/ TRACK
: je število sektorjev na stezi diska

RESERVED TRACKS
: je število diskovnih stez, ki jih ni mogoče uporabiti za shranjevanje zbirk

Disk je mogoče zaščititi pred nekontroliranim zapisom, ko imamo ukaz in učinek tega ukaza:

```
B>STAT D:=R/O 'cr'

B>STAT
A: R/W, SPACE: 72K
B: R/W, SPACE: 157K
D: R/O, SPACE: 229K
```

V povezavi z ukazom STAT se lahko pojavi tudi več vrst sporočil o napakah. Tako imamo:

INVALID DISK ASSIGNMENT
: se lahko pojavi, če smo za diskovnim določilom (npr. B:) postavili nekaj drugega kot =R/O

BDOS ERR ON d: SELECT
: se lahko pojavi pri navedbi neobstoječega diska (d:). Napako odpravimo z znakom 'cr' ali 'CTL c'. Računalnik lahko pride v neskončno zanko, če mu navedemo prazno diskovno enoto

FILE NOT FOUND
: se pojavi, če smo pozabili dvopičje

BDOS ERR ON d:R/O
: se pojavi pri poskusu zapisa na pisalno zaščiten disk

4.5. Kopiranje zbirk z ukazom PIP

PIP je okrajšava za Peripheral Interchange Program, s katerim kopiramo zbirke iz enega mesta na drugo. Imamo tri osnovne skupine PIP ukazov, in sicer:

PIP 'cr'
: PIP se naloži v pomnilnik in se javi z z znakom pripravljenosti '*'

PIP d:novo.tip=e:staro.tip p 'cr'
: Ta ukaz kopira zbirko staro.tip z diska 'e' v zbirko novo.tip na disku 'd'

PIP d:novo.tip=e:staro1.tap p f:staro2.tep q
: Oblikuje se zbirka novo.tip na disku d: iz vsebin zbirk staro1.tap z diska e: pri parametru 'p' in staro2.tep z diska f: pri parametru 'q'

Imena zbirk in njihove pripone so lahko dvoumne ali enoumne. Navedbe diskov se lahko izpustijo; v teh primerih se upošteva trenutno aktivna diskovna enota. Parametri niso obvezni, zaprti pa so v oglate oklepaje, če jih uporabimo.

PIP ukaz sprožimo torej na dva načina. Kadar imamo predvideno eno samo PIP operacijo, uporabimo ukaz

A PIP pip_ukazna_vrstica'cr'

Če pa predvidevamo več PIP operacij, imamo

```
A PIP'cr'

*pip_ukazna_vrstica'cr'

*pip_ukazna_vrstica'cr'

*pip_ukazna_vrstica'cr'

*
```

V primeru napake se pri uporabi prvega PIP ukaza vrnemo v CP/M, pri uporabi drugega ukaza pa v stanje PIP (ko imamo znak pripravljenosti '*'). Pri uporabi enačaja velja za zbirke splošno tole:

prihodna_zbirka = odhodna_zbirka

kjer je prihodna zbirka nova, odhodna zbirka pa stara zbirka. Podatki se vselej kopirajo iz odhodne na prihodno zbirko (gibanje podatkov iz desne strani enačaja na njegovo levo stran). Če prihodna zbirka že obstaja, se njena prvotna vsebina zbriše oziroma je za uporabnika zgubljena.

Pri kopiranju lahko uporabljamo več parametrov (hkrati) in imamo tale osnovni seznam:

B	imamo "bločni" prenos
Dn	zbrišejo se vsi znaki za stolpcem 'n'
E	imamo odmev na konzolo, tako kot kopiranje poteka
F	med prenosom se odstranijo oblikovnopomični znaki
Gn	PIP kopira zbirke iz drugih uporabniških območij
H	preizkuša se pravilnost HEX formata
I	ignorirajo se zapisi ':00' pri prenosu zbirk v HEX formatu
L	velike črke se pretvorijo v male
N	doda se vrstična številka k vsaki prenešeni vrstici
O	prenaša se objektna zbirka (ignorirajo se znaki konca zbirke)
Pn	izda se pomik za stran na vsakih 'n' vrstic
Qs'CTL z'	določa konec kopiranja po nizu 's'
R	omogočeno kopiranje s sistemske zbirke
Ss'CTL z'	določen je začetek kopiranja za nizom 's'
Tn	nastavijo se tabulirne pozicije na vsakem 'n-tem' stolpcu
U	male črke se pretvorijo v velike
W	omogoči se kopiranje na R/O zbirko
Z	izniči se bit parnosti ASCII znakov

Tako imamo npr. uporabo parametrov v naslednjih dveh primerih:

```
PIP B:=A:ime_zbirke.tip BEF 'cr'
```

ali

```
PIP B:=A:ime zbirke.tip B E F 'cr'
```

kjer smo uporabili parametre B, E in F.

I N F O R M A T I C A

časopis za tehnologijo računalništva in probleme informatike

Parmova c. 41 61000 Ljubljana Tel.(061)312-988

****

CP/M*

****

programirni priročnik

Anton P. Železnikar

urednik
časopisa INFORMATICA

ž.ž.

* cp/m je zaščitni znak podjetja Digital Research

ljubljana 1981

VGRAJENI UKAZI

DIR	Prikaži zbirčni imenik z diska
DIR c:	Prikaži zbirčni imenik z diska 'c'
DIR ime_zbirke.tip	Poišči imenovano zbirko na disku
DIR *.tip	Prikaži vse zbirke imenovanega tipa
DIR ime_zbirke.*	Prikaži vse zbirke z imenom ime_zbirke s trenutno izbranega diska
DIR a?????.*	Prikaži vse zbirke z dolžino imena 6, ki začenjajo s črko 'a' ali 'A'
DIT ime_zbirke.b??	Prikaži vse zbirke z imenom ime_zbirke in z začetkom tipa 'b' ali 'B'
TYPE ime_zbirke.tip	Prikaži ASCII zbirko z diska
TYPE c:ime_zbirke.tip	Prikaži ASCII zbirko z diska 'c'
SAVE n ime_zbirke.tip	Reši imenovano zbirko na disk
SAVE n c:ime_zbirke.tip	Reši imenovano zbirko na enoto 'c' 'n' je število strani, stran je 256 zlogov, začetek je na naslovu 100H
ERA ime_zbirke.tip	Zbiši imenovano zbirko z diska
ERA *.*	Zbriši vse zbirke z diska
ERA *.tip	Zbriši vse zbirke danega tipa
ERA ime_zbirke.*	Zbriši vse zbirke z imenom ime_zbirke
ERA c:ime_zbirke.tip	Zbriši imenovano zbirko z diska 'c'
REN no_ime.tip=st_ime.tip	Preimenuj zbirko z diska v no_ime
REN c:no_ime.tip=st_ime.tip	Preimenuj zbirko z diska 'c' v no_ime
c:	Preklopi na označeno diskovno enoto, kjer je c = A, B, C, D, ... , P
USER n	Spremeni uporabniško območje

PREHODNI UKAZI

DDT	Pokliči program za dinamično popravljanje programov in podatkov
DDT ime_zbirke.tip	Pokliči DDT in naloži imenovano zbirko za popravljanje
ASM ime_zbirke	Prevedi zbirko ime_zbirke tipa .ASM iz zbirnega v procesorski jezik
ASM c:ime_zbirke	Prevedi zbirko ime_zbirke z diska 'c'
ASM ime_zbirke.xyz	Prevedi zbirko ime_zbirke z uporabo stikal x, y in z: x - ime diska z ime_zbirke (.ASM)

ž.ž.

	y - ime diska, na katerega se bo naložila ime_zbirka.HEX y = Z pomeni preskok parametra z - ime diska, na katerega se bo naložila ime_zbirka.PRN z = X pomeni izpis ime_zbirka.PRN na konzolo z = Z pomeni preskok parametra Preskok parametra: ustrezna zbirka se ne bo generirala
LOAD ime_zbirke	ime_zbirke.HEX naloži kot ime_zbirke.COM (z diska v pomnilnik)
LOAD c:ime_zbirke	Naloži ime_zbirke.HEX z diska ´c´
DUMP ime_zbirke.tip	Prikaži zbirko z diska v HEX formatu
DUMP c:ime_zbirke.tip	Prikaži zbirko z diska ´c´ v HEX for.
MOVCPM n	Oblikuj in izvedi nK-zložni CP/M sistem
MOVCPM n *	Oblikuj preslikavo nK-zložnega CP/M sistema
MOVCPM * *	Oblikuj preslikavo maksimalnega CP/M sistema glede na dani pomnilnik
SYSGEN	Pokliči program za generiranje CP/M sistema
SUBMIT ime_zbirke parametri	Izvedi SUB zbirko z uporabo pripadajočih parametrov
XSUB	Izvedi razširjeni SUBmit program
ED ime_zbirke.tip	Izvedi ED (urejevalni) program za oblikovanje in urejanje imenovane zbirke
ED c:ime_zbirke.tip	Izvedi ED program za oblikovanje in urejanje imenovane zbirke z diska ´c´
STAT	Prikaži razpoložljivi prostor na vseh vstavljenih diskih
STAT c:	Prikaži razpoložljivi prostor na disku ´c´
STAT *.*	Prikaži posamično zasedenost za vse zbirke na disku in razpoložljivi prostor
STAT c:*.*	Prikaži posamično zasedenost za vse zbirke in razpoložljivi prostor na disku ´c´
STAT *.tip	Prikaži posamično zasedenost za vse zbirke določenega tipa in razpoložljivi prostor na disku
STAT c:*.tip	Prikaži posamično zasedenost za vse zbirke določenega tipa in razpoložljivi prostor na disku ´c´

STAT ime_zbirke.*	Prikaži posamično zasedenost za vse
STAT c:ime_zbirke.*	imenovane zbirke in razpoložljivi prostor na disku oziroma disku ´c´
STAT ime_zbirke.tip	Prikaži podatke o imenovani zbirki in
STAT c:ime_zbirke.tip	razpoložljivi prostor na disku oziroma na disku ´c´, kjer dobimo z uporabo stikala ´s´ več podatkov
STAT ime_zbirke.tip $s	
STAT c:ime.zbirke.tip $s	
STAT DEV:	Prikaži dejansko prireditev naprav
STAT VAL:	Prikaži možne prireditve naprav
STAT DSK:	Prikaži značilnosti diska
STAT USR:	Prikaži trenutno uporabniško področje
STAT c:=R/O	´c´ se spremeni v bralni disk, na katerega ni moč zapisovati
STAT ime_zbirke.tip $R/O	Zbirka postane bralna (nepisalna)
STAT ime_zbirke.tip $R/W	Zbirka postane bralna in pisalna
STAT ime_zbirke.COM $SYS	Zbirka postane sistemska in v imeniku nečitljiva (zakrita)
STAT ime_zbirke.COM $DIR	Zbirka postane nesistemska in v imeniku čitljiva (odkrita)
STAT sn:=fn:	Splošni (logični) napravi (sn) se priredi fizična naprava (fn), in sicer: sn = CON, LST, PUN, RDR fn = TTY, CRT, PTP, PTR, LPT, UP1, UR1, BAT, UL1, UP2, UR2, UC1 (glej odstavek o V/I zlogu)

PIP UKAZI

PIP	Pokliči program za periferno izmenjavo podatkov
*a:=b:ime_zbirke.tip	Kopiraj imenovano zbirko z diska ´b´ na disk ´a´
a:no_ime.=b:st_ime.tip	Kopiraj in spremeni ime zbirke iz starega (st) na disku ´b´ v novo (no) na disku ´a´
PIP a:=b:ime_zbirke.tip	Pokliči PIP in kopiraj imenovano zbirko iz diska ´b´ na disk ´a´
PIP a:=b:*.*	Kopiraj vse zbirke iz diska ´b´ na disk ´a´
PIP a:=b:ime_zbirke.*	Kopiraj vse imenovane zbirke različnih tipov iz diska ´b´ na disk ´a´
PIP a:=b:*.tip	Kopiraj vse zbirke določenega tipa iz diska ´b´ na disk ´a´
PIP LST:=ime_zbirke.tip	Izlistaj zbirko na LST napravi (npr. na kvalitetnem tiskalniku)
PIP PUN:=ime_zbirke.tip	Izluknjaj zbirko na PUN napravi
PIP CON:=ime_zbirke.tip	Izpiši zbirko na konzolni napravi
PIP ime_zbirke.tip=RDR:	Kopiraj podatke iz bralne naprave v imenovano zbirko

```
*no_ime.tip=a_ime.tip,b_ime.tip,c_ime.tip
                          Kopiraj staknjene ASCII zbirke v
                            zbirko  no_ime.tip
*a:no_ime.tip=b:a_ime.tip,b:b_ime.tip,b:c_ime.tip
                          Kopiraj staknjene ASCII zbirke z di-
                            ska ´b´ v zbirko na disku ´a´
*no_ime.tip=a_ime.tip[X],b_ime.tip[X]
                          Kopiraj staknjene neASCII zbirke v
                            zbirko no_ime.tip
```

```
PIP LST:=a_ime.tip,b_ime.tip
                          Izlistaj zbirki na LST napravi
PIP LST:=c:a_ime.tip,c:b_ime.tip
                          Izlistaj zbirki iz diska ´c´ na LST
                            napravi
```

PIP PARAMETRI

PIP parametri so zaprti v oglata oklepaja za PIP ukazom, npr:
*ime_zbirke.tip=RDR:[B]

B	Beri podatkovni blok do znaka ´CTL s´
Dn	Zbriši znake za stolpcem ´n´
E	Vse kopije pošlji tudi na konzolo
F	Odstrani oblikovne pomike
Gn	Vzemi zbirko iz uporabniškega področja ´n´
H	Preizkusi pravilnost HEX formata
I	Enako kot parameter H vendar z izpustitvijo nizov ´:00´
L	Spremeni vse velike znake v male znake
N	Dodaj številke vrstic brez vodečih ničel
O	Prenesi objektno zbirko brez upoštevanja konca zbirke
P	Vstavi oblikovalne pomike na vsako 60. vrstico
Pn	Vstavi oblikovalne pomike na vsako n-to vrstico
Qniz	´CTL z´ Prenehaj s kopiranjem, ko je bil najden niz
Sniz	´CTL z´ Začni s kopiranjem, ko je bil najden niz
R	Beri SYS zbirko
Tn	Razširi TAB presledke na vsakih ´n´ stolpcev
U	Spremeni vsa male znake v velike znake
V	Verificiraj kopirane podatke
W	Zbriši R/O zbirke ob prihodu
X	Kopiraj neASCII zbirke
Z	Postavi ničti parni bit pri vseh zlogih zbirke

PIP KLJUČNE BESEDE

CON:	Konzolna naprava (definirana v BIOSu)
EOF:	Pošlji ´konec_zbirke´ (ASCII znak ´CTL z´) v napravo
INP:	Vhodni vir (se nastavi v PIPu)
LST:	Naprava za listanje (definirana v BIOSu)
NUL:	Pošlji 40 znakov NUL v napravo
OUT:	Izhodna točka (se nastavi v PIPu)
PRN:	Enako kot LST:; TAB položaji so pri vsakem osmem znaku, oštevilčijo se vrstice in izdajajo strani na vsakih 60 vrstic po začetni izdaji
PUN:	Naprava za luknjanje (definirana v BIOSu)
RDR:	Naprava za branje (definirana v BIOSu)

Dodatne fizične naprave so določene v odstavku, ki obravnava V/I zlog

UKAZNI KRMILNI ZNAKI

znak	funkcija	kod
´CTL c´	Naloži CP/M (topla naložitev)	03H
´CTL e´	Začni z novo vrstico	05H
´CTL h´	Pomik za mesto nazaj z brisanjem znaka	08H
´CTL i´	TAB pomik na stolpni osemkratnik	09H
´CTL j´	Vrstični pomik	0AH
´CTL m´	Pomik valja	0DH
´CTL p´	Vključitev in izključitev tiskalnika	10H
´CTL r´	Ponovi trenutno vrstico	12H
´CTL s´	Ustavi prikazovanje izhoda Vsak znak z izjemo ´CTL c´ sproži ponovno prikazovanje izhoda	13H
´CTL u´	Zbriši vrstico	15H
´CTL x´	Zbriši vrstico in pomakni kurzor na začetek vrstice	18H
´CTL z´	Konec vhoda s konzole (pri ED in PIP)	1BH
´CTL _´	Zbriši in prikaži zadnji znak (odmev)	7FH

NEKATERI ZBIRČNI TIPI

ASC	ASCII tekstovna zbirka, tudi BASIC izvirna zbirka
ASM	Zbirka v zbirnem jeziku (ASM izvirni program)
BAK	Kopirana prejšnja zbirka (z urejevalnikom)
BAS	Zbirka z BASIC izvirnim programom
COM	Ukazna zbirka (prehodni izvršljivi program)
DAT	Podatkovna zbirka
DOC	Dokumentacijska zbirka
FOR	Zbirka s FORTRANskim izvirnim programom
INT	Zbirka z vmesnim BASIC programom (izvršljiva)

HEX	Zbirka s heksadecimalnim formatom (za LOAD ukaz)
LIB	Knjižnična zbirka, ki se uporablja v makrozbirniku
PLI	PL/I izvirna zbirka
PRN	Tiskalna zbirka (proizvod zbirnika, prevajalnika)
REL	Premestljivi modul
SAV	Sistemska zbirka (V2.X)
SUB	Tekstovna zbirka, ki jo izvrši SUBMIT program
SYM	Simbolna zbirka
TEX	Tekstovna zbirka
XRF	Zbirka prečnih referenc
$$$	Začasna zbirka

Dolžina zbirčnega imena: največ 8 znakov
Dolžina zbirčnega tipa : največ 3 znaki ali 0 znakov
Nedopustni imenski in tipski znaki:
. , ; : = ? < > [ ]

DDT UKAZI

A zna	Prevedi zbirni kod, začetek pri naslovu 'zna'
D	Izpiši RAM od 'tna' (trenutni naslov), 16 vrstic
D zna	Izpiši RAM od 'zna' (začetni naslov), 16 vrstic
D zna,kna	Izpiši RAM od 'zna' do 'kna' (končni naslov)
F zna,kna,kon	Napolni RAM od 'zna' do 'kna' z vrednostjo 'kon'
G	Izvedi program pri shranjenem PŠ (progr. števnik)
G zna	Izvedi program pri 'zna'
G zna,prl	Izvedi program pri 'zna' s prekinitvijo pri 'prl'
G zna,prl,pr2	Izvedi program pri 'zna' s prekinitvijo pri 'prl' ali pri 'pr2'
G, prl,pr2	Izvedi program pri trenutnem naslovu s prekinitvijo pri 'prl' ali pri 'pr2'
H a,b	Prikaži heksadecimalno a+b in a-b
I ime_zbirke	Postavi ZKB (5CH) za uporabniški kod
I ime_zbirke.tip	Postavi ZKB (5CH) za R ukaz (HEX ali COM zbirka) ZKB: zbirčni krmilni blok
L	Prevedi inverzno-zbirno od 'tna', 12 vrstic
L zna	Prevedi inverzno-zbirno od 'zna', 12 vrstic
L zna,kna	Prevedi inverzno-zbirno od 'zna' do 'kna' iz RAMa 'tna' je trenutni naslov
M zna,kna,nna	Kopiraj RAM blok od 'zna' do 'kna' na 'nna' 'nna' je novi naslov
R	Beri zbirko, določeno z I ukazom v RAM pri normalnem (nalagalnem) naslovu

R pomik	Beri zbirko, določeno z I ukazom v RAM pri naslovu nalagalni naslov + pomik
S zna	Vstavi v RAM, začenši pri 'zna'
T n	Izvedi 'n' ukazov z izpisom registrov po vsakem ukazu (tj. sledilni ukaz)
U n	Izvedi 'n' ukazov z izpisom registrov po zadnjem ukazu (pri n = 1 uporabi ukaz U brez 'n')
X r	Poglej/spremeni vsebino registrov in zastavic
X	Poglej vsebino registrov (zastavični registri so: C=prenos, Z=ničla, M=predznak, E=parnost, I=pomožni prenos)

tna = trenutni naslov zna = začetni naslov
nna = novi naslov kna = končni naslov
? = napaka, ki pomeni: zbirka ne more biti odprta, napaka parnosti v HEX zbirki ali zbirnik/ inverzni zbirnik se prekriva

ED UKAZI

nA	Pridruži 'n' vrstic k vmesniku (n = 0: uporabi polovico vmesnika)
B	Pomakni kazalec na začetek zbirke
-B	Pomakni kazalec na konec zbirke
nC	Pomakni kazalec naprej za 'n' znakov
nD	Zbriši 'n' znakov za kazalcem
E	Končaj urejevanje, zapri zbirko in se vrni v CP/M
nFs	Poišči n-to pojavitev niza 's' v tekstu za kazalcem
H	Končaj urejevanje in pomakni kazalec na začetek zbirke
I	Vstavljaj tekst za kazalcem dokler ne vtipkaš 'CTL z'
Is	Vstavi niz 's' za kazalcem
nK	Zbriši 'n' vrstic, začenši pri kazalcu
nL	Pomakni kazalec za 'n' vrstic
nMx	Izvedi ukazni niz 'x' n-krat
nNs	Globalni F ukaz, do konca zbirke
O	Ukini urejevanje in začni z začetno zbirko
nP	Izlistaj naslednjih 'n' strani s po 23 vrsticami (n = 0: trenutna stran)
Q	Prenehaj z urejevanjem brez spremembe vhodne zbirke
Riz	Včitaj iz.LIB v vmesnik za trenutnim kazalcem
nSx'CTL z'y	Substituiraj niz 'y' pri 'n' pojavitvah niza 'x'
nT	Izpiši 'n' vrstic
U	Spremeni male znake v velike (pri vpisu)
V	Vključi notranje oštevilčevanje vrstic
nW	Zapiši 'n' vrstic v izhodno zbirko (začni na začetku vmesnika)
nX	Izpiši naslednjih 'n' vrstic v zbirko 'X$$$$$$$.LIB'

UPORABNI PROGRAMI

Poziv časopisa Informatica bralcem, da objavljajo svoje uporabne programe, se nadaljuje. Katera so programirna uporabna področja, ki nas še posebej zanimajo?

Naštejmo le nekaj področij:

poslovnometodološki programi
ekonometrični izračuni
napovedovanje dogodkov
vrednotenje proizvodnje
planiranje proizvodnje
mrežno planiranje
računovodski programi
statistični izračuni
poslovodne igre
programi za preizkušanje sposobnosti
obdelava podatkovnih zbirk
tehnološki programi
bančna in finančna aritmetika
programi za učenje
itd.

Konkretni uporabni programi so npr. programi za mali poslovni sistem, za glavno poslovno knjigo, za evidenco prihodkov in izdatkov, za obračunavanje osebnih dohodkov, za izračun različnih statističnih kazalcev, tudi krajši zabavni programi (igre), dalje programi za preizkušanje računalniških virov in naprav, za kriptografiranje (šifriranje podatkov), za slikovito prikazovanje rezultatov na zaslonu, za urejanje in upravljanje podatkovnih zbirk itn.

Rubrika "Uporabni programi" naj bi vzpodbujala bralce, da pišejo kratke prispevke v dokaj standardni obliki, ki obsega

kratek opis področja, na katerem se program uporablja, s pripadajočo metodologijo (pojasnili)

programsko listo s komentarji v visokem programirnem ali zbirnem jeziku (PL/I, FORTRAN, COBOL, PASCAL, BASIC, ALGOL, FORTH, LISP, ADA, MODULA in zbirni jeziki mikroprocesorjev)

izvajanje (izvršitev) programa na dovolj slikovitem primeru, po možnosti v realnem okolju

Uredništvo bo vsakemu prispevku dodalo določen označevalnik. Tekst in formati programskih list se morajo prilegati avtorskim formularjem, ki jih dobite v uredništvu časopisa Informatica, Parmova 41, 61000 Ljubljana.

```
**********************
*  IZRAČUN BIORITMA  *
**********************

****************************
*  Informatica UP 3        *
*  Bioritem                *
*  avgust 1981             *
*  Izvor neznan            *
*  modif. A.P.Železnikar   *
*  sistem CP/M, CBASIC2    *
****************************
```

1. Področje uporabe programa

Uporaba diagramov bioritma v poslednjem času narašča zlasti v sportu in pri poklicnih dejavnostih. Tako so npr. Viktorju Korčnoju napovedali slab začetek dvoboja za svetovnega šahovskega prvaka prav prek njegovega bioritemskega diagrama. Nekatera podjetja uporabljajo te diagrame za vnaprejšnjo oceno zmogljivosti svojih delavcev, npr. poklicnih pilotov, astronavtov in voznikov. Čeprav obstaja dvom o koristnosti bioritemskih diagramov, si s prikazanim programom lahko sami določimo diagrame bioritma za krajše ali daljse razdobje.

Program za izračun bioritma je zanimiv s programirnega vidika, saj uporablja za izpis diagramov znakovno "grafiko" in oblikuje diagrame s tiskanjem znakov. Ta program je napisan v jeziku CBASIC2 za operacijski sistem CP/M, z manjšimi spremembami pa ga lahko uporabimo na vsakem računalniškem sistemu s prevajalnikom za jezik BASIC.

2. Kratek opis programa

Program v listi 1 izračuna najprej tabele, ki so potrebne pri kasnejšem izračunavanju diagramov bioritma. V manjših sistemih je za določitev teh tabel potrebnih nekaj deset sekund, zato imamo na začetku opozorilo (vrstica 90). Ko so tabele določene se s konzole vtipka ime (s priimkom) ter datum rojstva, tako da bomo imeli diagrame označana s temi podatki. Naslednji parametri programa, ki se vstavijo s konzole, so še začetni mesec in leto biokoledarja ter število mesecev, za katere želimo imeti diagrame izpisane. Po vstavitvi vseh teh podatkov se začnejo izračunavati in pisati posamezne točke bioritemskega diagrama. Diagram se izpisuje vsakokrat za tri krivulje (označene s črkami F, E in I), ki predstavljajo fizično, čustveno in inteligenčno zmogljivost. Relativna vsota (bioindeks) teh zmogljivosti je v intervalu (0, 100000) ter se izpisuje na levi strani diagrama (glej kasneje). Vsakokratna vrednost treh diagramov se izračunava v vrsticah 790, 800 in 810 liste 1.

3. Izvajanje programa

Na listi 2 je prikazana vstavitev začetnih podatkov in diagram bioritma. Diagram je izpisan za mesec oktober, seveda pa bi ga lahko izpisali za večje število mesecev. Po izvedbi programa za posamezen primer, je program pripravljen na sprejetje novih podatkov in tako na nov izpis. Program je moč še dopolniti tako, da izpišemo diagram v zbirko na disku ter ga tako lahko priložnostno čitamo z zaslona.

```
B>TYPE BIO2.BAS
90 PRINT TAB(20);"POCAKAJ, DA SE IZRACUNAJO TABELE !"
100 DIM P1(23),P2(23),E1(28),E2(28),I1(33),I2(33)
110 DIM W$(7),L$(41),M$(12)
120 DATA NED,PON,TOR,SRE,CET,PET,SOB
130 DATA JAN,FEB,MAR,APR,MAJ,JUN,JUL,AVG,SEP,OKT,NOV,DEC
140 FOR N=1 TO 23
150 P2(N)=SIN(6.283185307*(N-1)/23)
160 P1(N)=INT(21.5+20*P2(N))
170 NEXT N
180 FOR N=1 TO 28
190 E2(N)=SIN(6.283185307*(N-1)/28)
200 E1(N)=INT(21.5+20*E2(N))
210 NEXT N
220 FOR N=1 TO 33
230 I2(N)=SIN(6.283185307*(N-1)/33)
240 I1(N)=INT(21.5+20*I2(N))
250 NEXT N
260 FOR N=1 TO 7
270 READ W$(N)
280 NEXT N
290 FOR N=1 TO 12
300 READ M$(N)
310 NEXT N
320 FOR N=1 TO 41
330 L$(N)=" "
340 NEXT N
350 PRINT"VSTAVI IME"
360 INPUT N$
370 PRINT"VSTAVI DATUM ROJSTVA: DAN, MES, LETO:"
390 INPUT D,M,Y
400 PRINT"VSTAVI ZACETNI MESEC IN LETO BIOKOLEDARJA:"
420 INPUT M4,Y4
421 D8=D
422 M8=M
423 Y8=Y
424 PRINT"VSTAVI STEVILO MESECEV BIOKOLEDARJA:"
424.1 INPUT Z8
425 FOR Z0=1 TO Z8
426 D=D8
427 M=M8
428 Y=Y8
430 GOSUB 1350
440 M=M4
450 D=1
460 Y=Y4
470 GOSUB 980
480 S1=J
490 GOSUB 1350
500 L1=31
510 IF M4=12 THEN 570
520 GOSUB 1110
530 S3=N3
540 M=M4+1
550 GOSUB 1110
560 L1=N3-S3
570 B=J-S1+1
580 E=B+L1-1
591 PRINT
590 PRINT"  BIO";TAB(57);M$(M4)
600 PRINT" INDEKS";TAB(55);Y
610 PRINT TAB(8);"--.....0.....++"
630 V=0
640 FOR I=B TO E
650 V=V+1
660 J3=I-1
670 K1=INT(J3/23)
680 K2=J3-(K1*23)+1
690 K3=INT(J3/28)
700 K4=J3-(K3*28)+1
710 K5=INT(J3/33)
720 K6=J3-(K5*33)+1
730 P=P1(K2)
740 Q=E1(K4)
750 R=I1(K6)
760 O=P2(K2)+E2(K4)+I2(K6)
770 O=INT(16666*(O+3))+1
780 L$(21)=":"
790 L$(P)="F"
800 L$(Q)="E"
810 L$(R)="I"
820 PRINT O;TAB(10);
830 FOR N=1 TO 41
840 PRINT L$(N);
850 NEXT N
860 PRINT TAB(54);V;TAB(58);W$(N2)
870 L$(P)=" "
880 L$(Q)=" "
890 L$(R)=" "
900 N2=N2+1
910 IF N2<8 THEN 930
920 N2=1
930 NEXT I
940 PRINT TAB(8);"--.....0.....++"
960 PRINT TAB(10);N$;", ROJEN-A ";D8;M8;Y8;TAB(57);M$(M4)
970 GOTO 1451
980 IF M<3 THEN 1020
990 M1=M-2
1000 Y1=Y
1010 GOTO 1040
1020 M1=M+10
1030 Y1=Y-1
1040 C=INT(Y1/100)
1050 D1=Y1-(C*100)
1060 N4=INT((13*M1-1)/5)+D+D1+INT(D1/4)
1070 N=N4+INT(C/4)-2*C+77
1080 N1=INT(N/7)
1090 N2=N-(N1*7)+1
1100 RETURN
1110 Y2=INT(Y/4)
1120 Y3=Y-(Y2*4)
1130 IF Y3=0 THEN 1150
1140 GOTO 1250
1150 Y2=INT(Y/100)
1160 Y3=Y-(Y2*100)
1170 IF Y3=0 THEN 1190
1180 GOTO 1230
1190 Y2=INT(Y/400)
1200 Y3=Y-(Y2*400)
1210 IF Y3=0 THEN 1230
```

Lista 1. Program na tej listi (napisan v jeziku CBASIC2) izračunava in "riše" bioritmične diagrame v odvisnosti od rojstnega datuma in meseca ter leta začetka biokoledarja. Najprej se izračunajo tabele, potem se vstavljajo različni vhodni podatki (kot je razvidno z liste 2), nakar program izračunava podatke ter "riše" sestavljen diagram bioritma.

```
1220 GOTO 1250
1230 L1=1
1240 GOTO 1260
1250 L1=0
1260 N1=INT((3055*(M+2))/100)-91
1270 L=0
1280 IF M<3 THEN 1330
1290 IF L1=0 THEN 1320
1300 L=1
1310 GOTO 1330
1320 L=2
1330 N3=N1+D-L
1340 RETURN
1350 IF M<3 THEN 1390
1360 M1=M-3
1370 Y1=Y
1380 GOTO 1410
1390 M1=M+9
1400 Y1=Y-1
1410 C=INT(Y1/100)
1420 D1=Y1-(C*100)
1430 N=INT((146097*C)/4)+D+INT((1461*D1)/4)
1440 J=N+1721119+INT((153*M1+2)/5)
1450 RETURN
1451 M4=M4+1
1452 IF M4<13 THEN 1459
1453 M4=1
1454 Y4=Y4+1
1459 NEXT Z0
1460 GOTO 350
1470 END
```

Lista 1 (nadaljevanje). Program uporablja vrsto vgrajenih funkcij jezika CBASIC2 (sin, int, tab), tako da izračunava in izpisuje ustrezne diagrame. Program je prav gotovo zanimiv tudi s programirnega vidika in če ga boste pozorno "brali", se bo moč iz njega tudi česa naučiti.

```
B>CRUN2 BIO2

CRUN VER 2.07P
                    POCAKAJ, DA SE IZRACUNAJO TABELE !
VSTAVI IME
? ANTON P. ZELEZNIKAR
VSTAVI DATUM ROJSTVA: DAN, MES, LETO:
? 8,6,1928
VSTAVI ZACETNI MESEC IN LETO BIOKOLEDARJA:
? 10,1981
VSTAVI STEVILO MESECEV BIOKOLEDARJA:
? 1

  BIO                                                        OKT
 INDEKS                                                      1981
      --.....................0.....................++
 49102       F              :    E      I             1  CET
 45796     F                E                 I        2  PET
 43239   F                E :                   I      3  SOB
 41610   F          E       :                      I    4  NED
 41006     F      E         :                       I   5  PON
 41436       EF              :                        I  6  TOR
 42820     E      F          :                        I  7  SRE
 45003     E             F   :                       I   8  CET
 47766   E                   F                    I     9  PET
 50842     E                 :     F            I       10 SOB
 53945     E                 :           F I              11 NED
 56785       E               :        I     F           12 PON
 59100           E           :     I           F         13 TUR
 60669             E         : I                   F     14 SRE
 61333                 E I   :                       F   15 CET
 61007                   I   E                       F   16 PET
 59686               I       :   E              F        17 SOB
 57445                 I     :           E   F            18 NED
 54434         I             :         F   E              19 PON
 50865       I               :   F              E         20 TOR
 46995     I                 F :                    E     21 SRE
 43111   I                 F   :                      E   22 CET
 39500   I       F           :                        E  23 PET
 36435     I F               :                       E   24 SOB
 34150     FI                :                      E    25 NED
 32826   F   I               :                   E        26 PON
 32571   F     I             :               E            27 TOR
 33422     F       I         :           E                28 SRE
 35333         F       I     :   E                        29 CET
 38185               F     I   E                          30 PET
 41794                   FE  I                            31 SOB
      --.....................0.....................++
          ANTON P. ZELEZNIKAR, ROJEN-A  8  6  1928          OKT
VSTAVI IME
? ^C
B>
```

Lista 2. Ta lista kaže dva dela programskega izvajanja: začetni del, v katerem se pojavi sporočilo o računanju tabel ter se vstavljajo začetni podatki s konzole ter nadaljevalni del, v katerem se izračunavajo in izpisujejo diagrami. Prikazani primer je izvajan za en sam mesec biokoledarja, kot vhodni podatki pa se upoštevajo le datumi rojstva in začetka biokoledarja. Žal se v tem izračunu ne upoštevajo pravi biološki (fiziološki, biokemični, bioelektrični) podatki.

NOVICE IN ZANIMIVOSTI

NOVICE IZ RAZVOJNEGA PROGRAMA PODJETJA MOTOROLA

Kot pravijo odgovorni pri podjetju Motorola je razvoj mikroprocesorjev cikličen s pet ali šest letno dobo med vpeljavo novih procesorjev.Tako naj bi se pojavila nova generacija mikroprocesorjev(32 bitni)v letu 1984/5 - Redwood-68020.Nov mikroprocesor naj bi bil zmogljivejši in hitrejši,vendar združljiv s programsko opremo družine 68000.Med posebnimi lastnostmi je omeniti zlasti poudarek na razširjenem adresnem prostoru in na ustrezni podpori za virtualni pomnilnik ter vpeljavi koprocesorskih enot.Osnovna frekvenca delovanja bo med 10 in 12 MHz.

Zanimivo je pogledati kakšen je razvoj perifernih čipov serije 68000.Na tržišču je že možno dobiti naslednje čipe:

68120-intelligent terminal controller(kompat. vodilu serije 6800),
68122-cluster terminal controller,
68540-error detect and correct.

V nekaj mesecih bodo dosegljivi naslednji čipi:

68450-DMA controller(kompatibilen vodilu6800),
68451-memory management unit(razvit je predvsem za virtualni pomnilnik za Redwood),

Predvidoma v letu 1982/3 bodo dosegljivi naslednji čipi:

68340-dual port RAM,
68454-hard disc controller,
68455-advanced CRT controller,
68560-16-bit serial DMA processor,
68561-serial I/O multiprotocol communication controller,
68XXX-local area network controller.

Prvi koprocesor bo enota s plavajočo vejico (FPCP) z oznako 68881 in bo deloval v povezavi s procesorjem Redwood (68020).

VISOKO INTEGRIRANA VEZJA ZA DETEKCIJO IN KOREKCIJO

Za povečanje zanesljivosti dinamičnih pomnilnikov vpeljujemo posebna vezja za korekcijo in detekcijo napak.Na tržišču so se že pojavila visoko integrirana vezja,ki nadomestijo več deset čipov srednje integracije.Večina omenjenih čipov omogoča priključitev pomnilnikov 64Kx1bit in 16-bitnih mikroprocesorjev.Nekateri od njih so zelo hitri(napako javijo v 23nS)in imajo zmogljivost detekcije in korekcije enojnih napak ter detekcije dvojnih in skupinskih napak.

Proizvajalec ADVANCED MICRO DEVICES: Razvita je serija čipov Am2960.Jedro tvori čip Am2960,t.j.16-bitna enota za detekcijo in korekcijo napak.V družino spadajo še ojačevalniki treh stanj Am2962,ojačevalniki Am2966 in krmilnik dinamičnega pomnilnika Am2964.Za neposredno priključitev na mikroprocesor Am Z8000 je na voljo čip AmZ8163(EDC krmilnik in osveževalnik),dobavljiv bo v prvem četrtletju 1982.

ProizvajalecNATIONAL SEMICONDUCTOR: Osnovni čip je DP 8400,čip za krmiljenje DRAMov DP 8408(DP 8409 je za pomn.256K X 1 bit).Na voljo so še ojačevalniki DP 8420.Čipi bodo na voljo čez nekaj mesecev.

Proizvajalec MOTOROLA nudi osnovni čip z oznako MC 68540 in proizvajalec TEXAS INSTRUMENTS čip LS 630. Čipi še niso na voljo.

Prednost Am serije je v tem,da opravi koregiran vpis neposredno,brez programske pomoči.

Proizvajalec INTEL napoveduje osnovni čip,ki bo znatno zmanjšal število potrebnih čipov za detekcijo in korekcijo napak pri statičnih in dinamičnih pomnilnikih.Osnovni čip ima oznako 8206 in še ni dobavljiv.Z njim lahko detektiramo in koregiramo vse enojne napake in detektiramo vse dvojne napake ter nekatere večkratne napake.En osnovni čip lahko deluje nad 8 ali 16 bitnimi podatkovnimi besedami.V primeru,da povežemo v kaskado 5 čipov pa z 80 bitnimi besedami.Glede na povezavo CPE enot z osnovnim čipom ločimo dva sistema korekcije,prvi je označen z kot "correct always"ali"flow-through"sistem,drugi pa kot "check only"ali "parallel"sistem.

Čas za detekcijo je največ 50nS in za korekcijo največ 65nS (16-bitni sistem).Ima ločeno vhodno in izhodno vodilo.Čitanje je lahko z ali brez korekcije,zapis pa z parcialnim(byte) zapisom in Read-Modify zapisom.

Izveden je v HMOS tehnologiji in je v ohišju z 68 pini.Napajalna napetost samo +5V.

Poleg osnovnega čipa potrebujemo samo še en čip-krmilnik dinamičnih pomnilnikov.Proizvajalec naslednje čipe krmilnikov DRAMov:

- 8202A za 4K,16K DRAMe,
- 8203 za 64K DRAMe in
- ADRC (Advance DynamicRAM Controller) za 64K in 256K (dual port) DRAMe.

Krmilnik 8203 tvori vse signale potrebne za krmiljenje 64K(2164),16K(2117) in 4K(2104) DRAMov.Neposredno naslavljanje do 64 naprav brez zunanjih ojačevalnikov.Omogoča adresno multipleksiranje in generira potrebne impulze.Vsebuje osveževalni časovnik in osveževalni števnik,dekodiraCPE statuse in je popolnoma kompatibilen z Intelovimi mikroprocesorji 8080A,8085A,iAPX88 in iAPX86.

ADRC krmilni čip predstavlja popolnoma novo tehnologijo in je razvit za aplikacije s pomnilniki velikih dimenzij.Kompatibilen je z mikroprocesorji kot so:iAPX88,86,186,286.Z njim lahko neposredno naslavljamo do 2 Mbytov.Izveden je v HMOS II tehnologiji in nadomesti ca.30 čipov.Vzgrajen je v 68 pinsko ohišje.V povezavi z osnovnim čipom 8206 omogoči avtomatsko korekcijo napak med osveževanjem,kar je novost v primerjavi z drugimi sistemi za korekcijo in detekcijo napak.

SEZNAM NEKATERIH POMEMBNEJŠIH PROIZVAJALCEV WINCHESTER POGONOV

PROIZVAJALEC	14 col	8 col	5 col
Ampex	9150	Napovedan	
BASF	6150	6171/6172	Napov.
CDC	9730serija		
	33502		
Dastek	4835	2080	
FujitsuAmerica	FG411,F496		
	F493,M2253	2311,2312	
IBM	3370	3310	
Memorex	601,602,612	101,201	
	3650/52		
Nippon El.Corp.	D1200serija	D2210-2230	
Olivetti		HD830	HD561, HD512
Pertec		D-8000	
Shugart Associates	SA4000	SA10002	Napov.
	SA4100	SA1004	
Shugart Technology			ST 506
Siemens Corp.	PS5-8		
Toshiba	MK300F		

R. Murn

SREČANJA

19-20 januar, Geneve, Švica
Applications industrielle d'intelligence artificielle
IMAG-Quatriemes journees francophones sur l'informatique

ORg.:IMAG
Informacije:IMAG, B.P. 53 38041 Grenoble,France

27-29, Eindhoven,Nizozemska
Microelectronica-2nd European microelectronic Congress
Org. in informacije: Microelectronica, P.O.B. 428, Los Altos, CA 94022, USA

2-4 februar, London, Velika Britanija
Electronic OEM Assemblies 82
Org. in informacije: Trident Int.Exh.Ltd.
81 Plymouth Road, Tavistock, Devon, PL19 8AU, UK

24-26 februar, London, Velika Britanija
Microsystems 82
Org. in informacije: IpC Exh. Ltd., Surrey House,
1 Throwley WaySutton, Surrey SM1 4Q, UK

30 marec 1 april, Brithon, Velika Britanija
5th International Conference on Computers in Design Engineering
Org. in informacije: JudyVare, Conference Secretary,
IPC Science and Technology Press Ltd. POB 63, Westbury House BurySTreet , Guildford GU2 5BH, UK

6-8 april, Torino, Italija
5th International Symposium on Programming
Org.Instituto di Scienze dell'Informazione
Informacije: S.Ronchi della Rocca, Instituto dell'Informazione, C.M. D'Ageglio 42 - 10125 Torino, Italy

3-5 maj, Paris, Francija
Septieme Conference Internationale sur L'Acoustique,
la Parole et le Traitement du Signal
Org. in informacije:C. Gueguen, ENST, 46 rue Barrault 75013 Paris, France

24-28 maj, Berlin, ZR Nemčija
9th World Conference IMEKO
Org. in informacije: Gesellschaft Mess und Regelungstechnik, Graf-Recke Strasse 84, B.P. 1139, D 4000 Duesseldorf 1, BDR

11-14 maj, Paris, Francija
Congres Bureautique
Org. in informacije: AFCET, 156 Boulevard Periere, Paris, France

16-18 maj, Leningnad , ZSSR
PROLAMAT 81
Org. in informacije:Leningrad Researche Computer Centre Mendeleyevskaya Linia 1, USSR Academy of Science, 199164 Leningrad, USSR

9-11 juni, Paris, Francija
12e Congres International sur les Robots Industriel
Org. in informacije: AFRI - c/o SEPIC, 40 rue du Colisee, 75381 Paris, France

16-18, juni,, Bruxelles, Belgija
7th international Conference on Dynamic Systems
Org.Universite du Paris Dauphine
Informacije:B.Paulre, Universite du Paris Dauphine
Place de Mal. de Lattre de Tassigny, 75735 Paris, France

16-18 juni ,Dublin, Irska
Second International Conference on Boundary and Interior Layers - Computational and Asymptotic Methods
Org. in InformacijeRens.Bail Conference - 39 Trinity College-Dublin 2 Ireland

25-27 maj, Anvers, Belgija
Chemical Process Analysis and Design Using Computers
Org. in informacije:CHEMCOMP-c/o K. VIV Jan Van Rijswijckl aan, 58, B - 2000 Anvers , Belgique

29 juni - 2 juli, Toulouse, Francija
3rd IFAC Symposium on Control of Distributed Parameter Systems
Org. in informacije:INRIA, Service de relations exterieures Domaine de Voluceau, B.P. 105, 78153 Le Chesnay, France

22-24 juni, Jerusalem, Izrael
2nd International Confrence on Data Base
Org. : Hebrew University and Northwestern University
Informacije: Michael Hanani, University of Negev, Beer Sheba, Israel

5-10 juli,Praha, ČSSR
9th Conference on Computational Linguistic
Org.: International Committee on Computational Linguistic
Informacije: COLING 82, MFF UK, Linguistics, Malostranske n. 25, 118 00 Praha 1, ČSSR

11-16 juli, Warszawa, Poljska
3rd IFAC/IFORS Symposium on Large Scale Systems
Org.:IFAC/IFORS
Informacije: Z.Nahorski, System Research Institute, Polish Academy of Sciences, ul. Newelska 6 - 01 447 Warszawa, Poland

AVTORJI IN SODELAVCI

Wladyslaw M. Turski was born on 17th October 1938 in Krakow, Poland. After graduating from the Zamoyski Lyceum in Warszawa he enrolled in the Lomonosov University in Moskow, where he studied astronomy, specializing in celestial mechanics /1955-1960/. In 1960/61 W.M.Turski was a Leverhulme Research Fellow at Jodrell Bank /Radio-observatory of the Manchester University/ where he worked on large-scale computer simulation of astronomical phenomena. Having returned to Poland in 1961 he joined the just-created Computation Centre of the Polish Academy of Sciences, where he worked until 1972 first as the head of a unit concerned with space computation and then as the chief of the software laboratory. In 1962 W.M.Turski was granted a doctorate in mathematical physics /Warsaw University/ and in 1966 he was given the degree of a "doctor habilitatus" by the Academy of Mines and Metallurgy for his research on computational methods in control engineering /the results were reported also at the IFIP Congerss in New York, 1965/.

In 1965 W.M.Turski was elected a member of IFIP Working Group 2.1 on Algol and became its Secretary in 1966. In 1969 he co-founded the IFIP W.G. 2.3 on Programming Methodology.

In 1972 W.M.Turski left the Academy for an industrial job in MERA Industries, where he was the Director for Software and Applications until 1977. In this capacity he directed much of the Polish software developments for Riad range of computers, as well as several major software projects in operating systems, compilers and data bases.

Starting from 1977 W.M.Turski is a Professor of computing science in Warsaw University and the Director of the Informatics Institute in this university.

Professor Turski has hold visiting positions in many universities in US, UK, Canada and DDR; frequently he gave invited lectures in many countries on all continents. He was a member of Program Committee for IFIP Congress 1974 and 1980 and its Chairman for 1977 Congress in Toronto.

W.M.Turski has published nearly 100 papers and several books including Datenstrukturen /Academie Verlag, Berlin/, Computer Programming Methodology /Heyden, London/ and Programming Teaching Techniques /North Holland, Amsterdam/.

He is an editor of many journals, including Acta Informatica, Information Processing Letters and Annals of Computer History.

W.M.Turski is a Fellow of the British Computer Society and President of the Polish Informatics Society.

SAŠA PREŠERN (1951), je diplomiral na Fakulteti za naravoslovje in tehnologijo, Odsek tehnične fizike, v Ljubljani (1975). Magistriral je na Operacijskih raziskavah s temo simulacije in optimalnega upravljanja sistemov vodovodnega omrežja, ki jo je izdelal na Katedri za sisteme, avtomatiko in kibernetiko Fakultete za elektrotehniko v Ljubljani (1978). Na oddelku za uporabno fiziko v Delftu na Nizozemskem (1977) je sodeloval pri izdelavi procesorja za simulacijo rasti kristalov. Leta 1978 se je vključil v delo v IBM računskem centru v Philadelphiji v ZDA. Na Inštitutu Jožef Stefan se je zaposlil leta 1980 in je prevzel razvoj senzorskih sistemov ter mikroračunalniško vodenje robotskih in avtomatskih sistemov. V letu 1980/81 se je na oddelku za računalništvo na Univerzi v Torontu ukvarjal s senzorskimi sistemi, računalniškim vidom in modeliranjem računalniških sistemov.

Peček Dušan, rojen 1952. leta, je diplomiral v letu 1977 na Fakulteti za elektrotehniko v Ljubljani, smer Računalništvo in informatika. Tema diplomskega dela je bila implementacija časovnega simulatorja digitalnih vezij, s posebnim poudarkom na simulacijskem algoritmu in pripadajočih kodnih nizih. Po diplomi se je zaposlil na Institutu "Jožef Stefan" v Ljubljani, na Odseku za računalništvo in informatiko. Na raziskovalnem področju se ukvarja s problematiko zanesljivosti, diagnostike in simulacije digitalnih sistemov. Glavna tema aplikativne dejavnosti pa je razvoj materialne opreme za 8 in 16 bitne mikroračunalnike.

Franc Novak je diplomiral leta 1975 na Fakulteti za elektrotehniko v Ljubljani s tematiko s področja diagnostike mikroračunalnikov. Magistriral je leta 1977 na isti fakulteti z delom Funkcionalna diagnostika mikroračunalnikov. Zaposlil se je na Institutu "Jožef Stefan", na Odseku za računalništvo in informatiko. Ukvarjal se je z razvojem aparaturne opreme v mikroračunalniških aplikacijah, v zadnjih letih pa se ukvarja predvsem z razvojem aparaturne in programske opreme za diagnostiko mikroračunalniških sistemov.

Drago Novak se je rodil 18.8.1951 v Ptuju. Diplomiral je 1975 leta na Fakulteti za elektrotehniko Univerze Edvarda Kardelja v Ljubljani. V okviru diplomskega dela se je ukvarjal z merjenjem parametrov hoje in sprotnim izračunavanjem karakterističnih parametrov, ki omogočajo ocenjevanje normalnosti hoje. Magistriral je na isti fakulteti leta 1978 z delom Oblikovanje večprocesorskih sistemov na osnovi mikroprocesorske tehnologije.

Po diplomi se je zaposlil na Institutu Jožef Stefan v odseku za računalništvo in informatiko. Ukvarjal se je z razvojem aparaturne in programske opreme za mikroračunalnike. Sodeloval je pri razvoju mikroračunalnika Iskradata 1680 in pri razvoju programske opreme za telefonsko centralo Iskra 2000. V okviru raziskovalnega dela se je ukvarjal s problematiko multiprocesorskih sistemov. V zadnjem času pa se je usmeril na področje porazdeljenih sistemov oz. porazdeljene kontrole.

Leta 1980 se je zaposlil v DO Delta v Ljubljani. Sodeloval je pri razvoju sistema DELTA 323/M1.

V Slovenskem društvu Informatika aktivno sodeluje kot urednik področja mikro-računalniki v našem časopisu in kot predsednik programskega odbora simpozijev Informatica '81 in Informatica '82.

NAVODILO ZA PRIPRAVO ČLANKA

Avtorje prosimo, da pošljejo uredništvu naslov in kratek povzetek članka ter navedejo približen obseg članka (število strani A 4 formata). Uredništvo bo nato poslalo avtorjem ustrezno število formularjev z navodilom.

Članek tipkajte na priložene dvokolonske formularje. Če potrebujete dodatne formularje, lahko uporabite bel papir istih dimenzij. Pri tem pa se morate držati predpisanega formata, vendar pa ga ne vrišite na papir.

Bodite natančni pri tipkanju in temeljiti pri korigiranju. Vaš članek bo s foto postopkom pomanjšan in pripravljen za tisk brez kakršnihkoli dodatnik korektur.

Uporabljajte kvaliteten pisalni stroj. Če le tekst dopušča uporabljajte enojni presledek. Črni trak je obvezen.

Članek tipkajte v prostor obrobljen z modrimi črtami. Tipkajte do črt - ne preko njih. Odstavek ločite z dvojnim presledkom in brez zamikanja prve vrstice novega odstavka.

Prva stran članka:
a) v sredino zgornjega okvira na prvi strani napišite naslov članka z velikimi črkami;
b) v sredino pod naslov članka napišite imena avtorjev, ime podjetja, mesto, državo;
c) na označenem mestu čez oba stolpca napišite povzetek članka v jeziku, v katerem je napisan članek. Povzetek naj ne bo daljši od 10 vrst.
d) če članek ni v angleščini, ampak v katerem od jugoslovanskih jezikov izpustite 2 cm in napišite povzetek tudi v angleščini. Pred povzetkom napišite angleški naslov članka z velikimi črkami. Povzetek naj ne bo daljši od 10 vrst. Če je članek v tujem jeziku napišite povzetek tudi v enem od jugoslovanskih jezikov;
e) izpustite 2 cm in pričnite v levo kolono pisati članek.

Druga in naslednje strani članka:
Kot je označeno na formularju začnite tipkati tekst druge in naslednjih strani v zgornjem levem kotu.

Naslovi poglavij:
naslove ločuje od ostalega teksta dvojni presledek.

Če nekaterih znakov ne morete vpisati s strojem jih čitljivo vpišite s črnim črnilom ali svinčnikom. Ne uporabljajte modrega črnila, ker se z njim napisani znaki ne bodo preslikali.

Ilustracije morajo biti ostre, jasne in črno bele. Če jih vključite v tekst, se morajo skladati s predpisanim formatom. Lahko pa jih vstavite tudi na konec članka, vendar morajo v tem primeru ostati v mejah skupnega dvokolonskega formata. Vse ilustracije morate (nalepiti) vstaviti sami na ustrezno mesto.

Napake pri tipkanju se lahko popravljajo s korekcijsko folijo ali belim tušem. Napačne besede, stavke ali odstavke pa lahko ponovno natipkate na neprozoren papir in ga pazljivo nalepite na mesto napake.

V zgornjem desnem kotu izven modro označenega roba oštevilčite strani članka s svinčnikom, tako da jih je mogoče zbrisati.

Časopis INFORMATICA
Uredništvo, Parmova 41, 61000 Ljubljana

Naročam se na časopis INFORMATICA. Predplačilo bom izvršil po prejemu vaše položnice.

Cenik: letna naročnina za delovne organizacije 500,00 din, za posameznika 200,00/100,00/50,00 din

Časopis mi pošiljajte na naslov ☐ stanovanja ☐ delovne organizacije.

Priimek.....

Ime.....

Naslov stanovanja

Ulica.....

Poštna številka_____ Kraj.....

Naslov delovne organizacije

Delovna organizacija.....

.....

Ulica.....

Poštna številka_____ Kraj.....

Datum..... Podpis:

.....

INSTRUCTIONS FOR PREPARATION OF A MANUSCRIPT

Authors are invited to send in the address and short summary of their articles and indicate the opproximate size of their contributions (in terms of A 4 paper). Subsequently they will receive the outor's kits.

Type your manuscript on the enclosed two-column-format manuscript paper. If you require additional manuscript paper you can use similar-size white paper and keep the proposed format but in that case please do not draw the format limits on the paper.

Be accurate in your typing and through in your proof reading. This manuscript will be photographically reduced for reproduction without any proof reading or corrections before printing.

Use a good typewriter. If the text allows it, use single spacing. Use a black ribbon only.

Keep your copy within the blue margin lines on the paper, typing to the lines, but not beyond them. Double space between paragraphs.

First page manuscript:
a) Give title of the paper in the upper box on the first page. Use block letters.
b) Under the title give author's names, company name, city and state - all centered.
c) As it is marked, begin the abstract of the paper. Type over both the columns. The abstract should be written in the language of the paper and should not excesed 10 lines.
d) If the paper is not in English, drop 2 cm after having written the abstract in the language of the paper and write the abstract in English as well. In front of the abstract put the English title of the paper. Use block letters for the title. The lenght of the abstract should not be greater than 10 lines.
e) Drop 2 cm and begin the text of the paper in the left column.

Second and succeeding pages of the manuscript:
As it is marked on the paper, begin the text of the second and succeeding pages in the left upper corner.

Format of the subject headings:
Headings are separated from text by double spacing.

If some characters are not available on your typwriter write them legibly in black ink or with a pencil. Do not use blue ink, because it shows poorly.

Illustrations must be black and white, sharp and clear. If you incorporate your illustrations into the text keep the proposed format. Illustration can also be placed at the end of all text material provided, however, that they are kept within the margin lines of the full size two-column format. All illustrations must be placed into appropriate positons in the text by the author.

Typing errors may be corrected by using white correction paint or by retyping the word, sentence or paragraph on a piece of apaque, white paper and pasting it nearly over errors

Use pencil to number each page on the upper-right-hand corner of the manuscript, outside the blue margin lines so that the numbers may be erased.

INFORMATICA, Journal Headquarters
Parmova 41, 61000 Ljubljana, Yugoslavia

Please enter my subscription to INFORMATICA and send me the bill.

Annual subscription price: companies US $ 22, individuals US $ 7,5.

Send journal to my ☐ home address ☐ company's address.

Surname.....

Name.....

Home address

Street.....

Postal code _____ City.....

Company address

Company.....

.....

Street.....

Postal code _____ City.....

Date..... Signature

.....

CENIK OGLASOV

Ovitek - notranja stran (za letnik 1981)

2 stran	28.000 din
3 stran	21.000 din

Vmesne strani (za letnik 1981)

1/1 stran	13.000 din
1/2 strani	9.000 din

Vmesne strani za posamezno številko

1/1 stran	5.000 din
1/2 strani	3.300 din

Oglasi o potrebah po kadrih (za posamezno številko)

2.000 din

Razen oglasov v klasični obliki so zaželjene tudi krajše poslovne, strokovne in propagandne informacije in članki. Cene objave tovrstnega materiala se bodo določale sporazumno.

ADVERTIZING RATES

Cover page (for all issues of 1981)

2nd page	1300 $
3rd page	1000 $

Inside pages (for all issues of 1981)

1/1 page	790 $
1/2 page	520 $

Inside pages (individual issues)

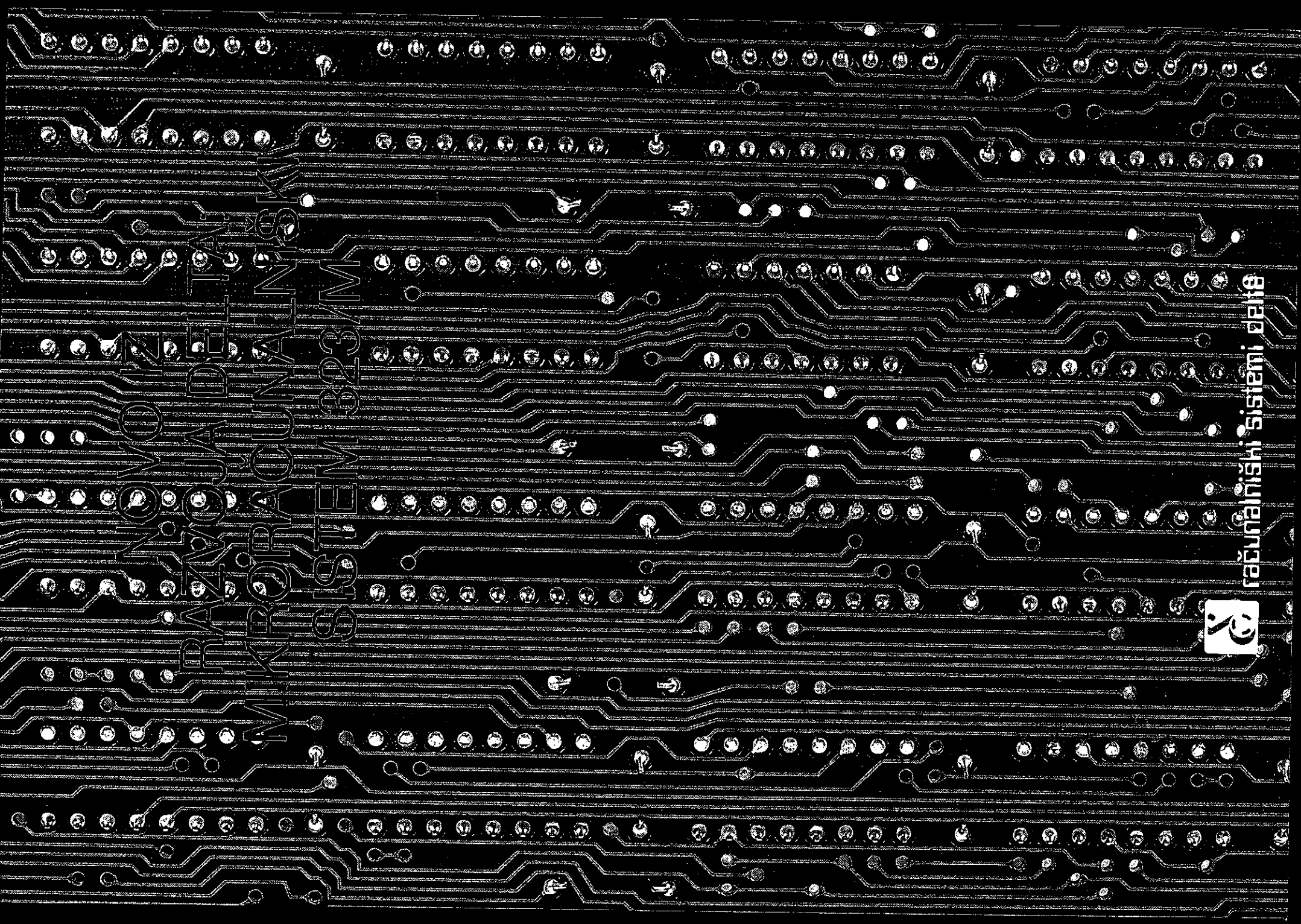
1/1 page	260 $
1/2 page	200 $

Rates for classified advertizing:

each ad	66 $

In addition to advertisment, we welcome short business or product news, notes and articles. The related charges are negotiable.

Introduzione ai sistemi digitali



DELTA 323/M
mikroračunalniški sistem

računalniški sistemi delta

Delavci DO DELTA proizvajamo najpopolnejšo jugoslovansko družino računalnikov, katera obsega celotno območje, od mikro računalnikov do največjih 32-bitnih sistemov.
Poseben pomen dajemo aplikacijski programski opremi. Ob izbiri segmentov je bila naša skrb posvečena povečanju produktivnosti in čimvečjemu prihranku energije ter surovin. Programski moduli za področja procesne kontrole, planiranja in upravljanja proizvodnje ter finančnega poslovanja, predstavljajo integralen pristop v izgradnji informacijskega sistema proizvodne delovne organizacije.
Naši računalniki so narejeni tako, da niso element prestiža delovnih organizacij, ki jih kupujejo, temveč so orodje razvojnega inženirja, projektanta, delavca v skladišču in drugih. S takim načinom dela vstopa DELTA, skupaj s svojimi uporabniki, v informatizirano družbo prihodnosti.
Našo družino računalnikov dopolnjujemo z malimi poslovnimi sistemi 323/M, ki so plod lastnega razvoja, delo strokovnjakov DO DELTA.

Značilnosti družine 323/M so:

- aplikativna usmerjenost računalniških paketov, ki vsebujejo vso potrebno aparaturno in programsko opremo za določeno uporabo
- enostavna razširitev in rekonfiguracija
  sistem se lahko enostavno širi in dopolnjuje, tako kot raste vaša delovna organizacija
- komunikativne sposobnosti
  računalnik 323/M omogoča komuniciranje z drugimi računalniki iz družine sistemov DELTA, kot tudi simulacijo protokolov IBM.
- delo s podatkovnimi zbirkami
  upravljanje in preoblikovanje podatkovnih zbirk, sistem za zaščito podatkovnih zbirk

Za mikroračunalniški sistem DELTA 323/M je značilno sodobno oblikovanje. Sistem se zato lepo vključuje v delovno okolje in ker zanj ne potrebujemo klimatiziranih prostorov, ga lahko priključimo tam, kjer ga potrebujemo.
Področja uporabe so številna, saj ga lahko uporabljamo kot mali poslovni sistem, pisarniški in šolski sistem, inteligentni terminal v računalniški mreži ter v distribuirani obdelavi podatkov.

Tehnični podatki:

— 8-bitni mikroprocesor
— 64 K zložni hitri pomnilnik (RAM)
— ura realnega časa
— vhodno/izhodni kanali
  serijski: 2 asinhrona dupleksna kanala
  4 asinhroni/sinhroni dupleksni kanali
  (progr. nastavljiva hitrost prenosa do 38,4 K baud)
  paralelni: 4 kanali (8-bitni, s po dvema krmilnima signaloma)
— do štirje gibljivi diski obsega 256 K ali 512 K zlogov
— matrični tiskalnik (300 baud)
— vrstični tiskalnik (600 vrstic/min.)
— lepopisni tiskalnik (45 znakov/sek.).
— **sistemska programska oprema vsebuje**
  prevajalnik za PL/1, COBOL, BASIC, PASCAL, FORTRAN in ALGOL;
  zbirnik, urejevalnik, program za preizkušanje in sledenje uporabniških programov, program za kopiranje zbirk.
— aplikacijska programska oprema
  glavna knjiga
  saldakonti kupcev in dobaviteljev
  osebni dohodki
  osnovna sredstva
  linearno programiranje
  optimizacija vzorčnih pravil
  mrežno planiranje
  poslovna statistika
  razpošiljanje pošte
  urejanje dokumentacije
  oblikovanje dopisov

ELEKTROTEHNA, DO DELTA, LJUBLJANA, PARMOVA 41