Volume 15 Number 1 February 1991
YU ISSN 0350 – 5596

Informatica

A Journal of Computing and Informatics

The Slovene Society INFORMATIKA
Ljubljana

Informatica

A Journal of Computing and Informatics

Subscription Information

Informatica (YU ISSN 0350 – 5596) is published four times a year in Winter, Spring, Summer and Autumn (4 issues).

The subscription price for 1990 (Volume 14) is US$ 30 for companies and US$ 15 for individuals.

Claims for missing issues will be honoured free of charge within six months after the publication date of the issue.

Printed by Tiskarna Tipa, Gosposvetska 13, Ljubljana.

Informacija za naročnike

Informatica (YU ISSN 0350 – 5596) izide štirikrat na leto, in sicer v začetku januarja, aprila, julija in oktobra.

Letna naročnina v letu 1991 (letnik 15) se oblikuje z upoštevanjem tečaja domače valute in znaša okvirno za podjetja DEM 30, za zasebnike DEM 15, za študente DEM 8, za posamezno številko pa DM 10.

Številka žiro računa: 50101 – 678 – 51841.

Zahteva za izgubljeno številko časopisa se upošteva v roku šestih mesecev od izida in je brezplačna.

Tisk: Tiskarna Tipa, Gosposvetska 13, Ljubljana.

Na podlagi mnenja Republiškega komiteja za informiranje št. 23 – 85, z dne 29. 1. 1986, je časopis **Informatica** oproščen temeljnega davka od prometa proizvodov.

Pri financiranju časopisa **Informatica** *sodeluje Republiški komite za raziskovalno dejavnost in tehnologijo, Tržaška 42, 61000 Ljubljana*

Volume 15 Number 1 February 1991
YU ISSN 0350 – 5596

Informatica

A Journal of Computing and Informatics

The Slovene Society INFORMATIKA
Ljubljana

Letnik 15 Številka 1 Februar 1991 YU ISSN 0350-5596

Informatica

Časopis za računalništvo in informatiko

V S E B I N A

MONITORING OF REAL-TIME SYSTEMS

INFORMATICA 1/91

Keywords: Real-time programming, Real-time monitoring, Debugging, Analysis

Viktorija Kobold, Železarna Ravne, Ravne na Koroškem
Marjan Špegel, Institut Jožef Stefan, Ljubljana

ABSTRACT Standard approaches to software performance analysis are not directly applicable to real-time systems owing to their time criticality. Therefore a tool which can monitor or verify if the system meets timing requirements is very useful. A real-time monitor enables us to observe the behavior of a real-time process in its execution. An emphasis have to be done on real-time programming constructs and programming languages for real-time systems. Some important research issues of a programming methodology are presented. Monitoring a process means determining the values of certain parameters associated with a time sequence of events and modifying them. Some possibilities of the implementation of real-time monitoring are presented.

POVZETEK V sistemih z odzivom v realnem času ne moremo neposredno uporabiti standardnih metod za analizo zmogljivosti sistema zaradi njihove časovne kritičnosti. Prav zaradi tega je orodje, ki zna ugotoviti, ali so bile vse časovne zahteve izpolnjene, zelo uporabno. Pri razvoju je potrebno uporabiti pravi način programiranja in izbrati programski jezik, ki bo ustrezal zahtevam glede časovne kritičnosti. Opazovanje procesa med njegovim izvajanjem pomeni določanje vrednosti nekaterih parametrov, ki so povezani s časovnim potekom dogodkov. Sama implementacija orodja za opazovanje zavisi od okolja, v katerem ga bomo uporabili.

1 Introduction

In resent years, there has been rapidly increasing use of computers as passive (monitoring) and active (controlling) components of real-time systems (e.g., air traffic control, aerospace, aircraft, industrial plants, hospital patient monitoring systems) [LS87]. For real-time applications, the problems of safety and reliability are particularly important because a failure may cause a disaster (e.g., destruction of human life and property).

According to a definition, a real-time system is a system which responds to events quickly enough to affect its environment in required response times. Such a system consists of multiple cooperating tasks, which implies detailed consideration of intertask communication and concurrency problems in the program and system design.

Real-time systems are divided into two types: 'soft' and 'hard'. In soft real-time systems, tasks are performed by the system as fast as possible, but they are not constrained to finish by specific times [CSR87]. On the other hand, hard real-time systems are defined as those in which the correctness of the system depends not only on the logical results of computation, but also on the time at which the results are produced [SR87]. In the rest of this paper we concentrate on hard real-time systems denoted as real-time systems.

Real-time systems are further divided into centralized systems and distributed systems. A centralized system is one in which the processors are located at a single point in the system (e.g., multiprocessor system). In contrast, in a distributed system the processors are distributed at different points in the system [CSR87]. An example of such system is a local area computer network. Such system consists of a collection of communicating and cooperating processors or computers (nodes) that

work toward a common goal.

Due to a growing number of real-time applications, there is also a growing need for real-time software to control them and software tools such as timing tools, schedulability analyzers, real-time monitors and real-time debuggers. Timing tools are used to estimate or measure the timing characteristics of execution the given software module or a given function such as the worst-case response time of the system to a particular type of event. A schedulability analyzer can analyze the schedulability of a given real-time task set under the specific scheduling policy. A real-time monitor can monitor and verify if the system meets timing requirements at runtime. Finally, a real-time debugger should be able to capture unseen timing bugs at runtime with the minimum system interference.

It is known that is difficult to design and analyze the software systems for the real-time applications owing to their time criticality [SR87]. Standard approaches to software performance analysis are therefore not directly applicable to real-time systems. Particularly 'timing bugs' are very difficult to capture and eliminate from real-time systems.

In the rest of this paper, we will concentrate on monitoring systems, which can support performance evaluation, testing and debugging of real-time systems. In section 2 we give a description of real-time programming and real-time programming event monitor. In section 3 a definition of a real-time monitor is given and monitoring process is presented. In section 4 some principles of implementation of monitoring system are described. Finally, a schedulability analyzer is introduced. It can analyze whether or not a given real-time task set can meet its timing constraints under a specific scheduling policy.

2 Real-Time Programming

Real-time programming involves the design of multiple program modules concurrently executing and interacting with one another through the sharing of resources and interprogram communication of data and events [PS85]. The multiple program modules or processes may be part of one program (with subtasks created by the program) or separate programs (to be run concurrently by a real-time operating system). These processes execute concurrently at different priority levels and must use real-time programming constructs. Real-time programming constructs are used to control interaction and are often called upon through use of system services of a real-time operating system. For example, they can be used to control concurrent access of data, enforce mutual exclusion of critical sections, perform safe interprocess communication, synchronize scheduling of one another

Real-time programs can not be tested simply by running them with data sources because their execution depends upon asynchronously occurring events. Hence, testing and verification of some critical aspect of the design often requires forcing of time sequences of events. The real-time monitor software permits the addition of 'action' routines which are triggered by event reporting routines. Each action routine must be designed for the specific tests being carried out. However, it does not permit flexible forcing of concurrent tasks to synchronize in such a way that conditions for a particular test are set up and results documented in the ensuing trace report.

The important research issues of a programming methodology for real-time systems include [Sta88]:

- Support for the management of time.
    1. Language constructs should support the expression of timing constraints.
    2. The programming environment should provide the programmer with the primitives to control and to keep track of the resource utilization of software modules.
    3. Language constructs should support the use of scheduling algorithms.
- Schedulability check.
- Reusable real-time software modules. They reduces the software development cost and enhances the quality of resulting software. They have the added difficulty of meeting different timing requirements for different applications.
- Support for distributed programs and fault tolerance.

In conclusion, programming languages should have explicit constructs to express the time realted be-

havior of modules and the semantics must be unambiguous. In the rest of this section the features of monitoring language is introduced and an event monitor is described.

Monitoring languages. A monitoring language is a notation in which a user can specify predicates and actions. It should allow predicates which can access any state variable. In order to enable the user to meet the requirement of completness the monitoring language must depart from the variable accessing mechanisms built into programming language. According to [PN81], the monitoring languages can be classified into:

1. typical high-level programming languages (they were not designed for monitoring)
2. embedded monitoring languages (monitoring statements may be interspersed between programming language statements)
3. separate monitoring languages (predicate and action specifications may range over the entire state space).

Which language a programmer will use depends on why he monitors the program in execution and what he hopes to achieve by doing so.

2.1 Real-time programming event monitor

In order to study and learn real-time programming, it is useful to have the real-time programming event monitor which is described by [Sch88]. The event monitor produces a complete time-stamped event history sequence. This sequence is associated with an execution of the real-time programs with each event identified by a type and triggering program module. For example, events which are monitored include all calls to system services, entrance and exit of routines, the entering and exiting of a critical section, the filling or emptying of a queue, the detection of a value for a variable in an alarm range, etc. [Sch89].

Event records are stored in sequence of occurence and contain a time stamp, identification of the event itself, identification of the routine making the system call or signaling the significant event, and a small amount of additional information related to the particular event.

Standard reports include three different kinds of reports. First of them is a report of all events containing each event, the triggering routine, and a time of occurence in time sequence. The included events are calls of operating system service and user-defined events such as entering and leaving routines, critical sections, etc. The second is the report as the first one, but it is restricted to events associated with a single routine or a set of routines. The last report includes special events such as delays of scheduled tasks, response time, peak queue sizes and the like.

In conclusion, the event monitor can be a useful aid in teaching real-time programming.

3 Program Monitoring and Analysis

The monitoring and analysis of progress of a program in execution has traditionally been part of the program development cycle. There are two reasons why one might want to examine the dynamic aspects of a program. First of them is to evaluate the performance of a program, and thus to assess its overall behavior. The second is to demonstrate the presence of programming errors, to isolate erroneous program code and correct it [Pla84] which is referred to as the "debugging a program". Actually, we can say that monitoring is the extraction of dynamic information concerning an execution of a computational process [Sno88].

Monitoring is an essential part of many program development tools. Monitoring a process means measuring or determining the values of certain parameters associated with a time sequence of events and modifying them. It means also observing its trajectory through the program state space, and if it is needed modifying this trajectory artificially [PN81]. A monitoring system must not affect the timing constraints of the target process (i.e., monitored process). If the monitor satisfies this requirement, it is called a *real-time monitor*.

A monitor is a very useful tool because it makes possible to observe the behavior of a real-time process in its execution, in order to collect genuine data for statistics (system and performance analysis), or for detection of illegal or unexpected process states (identifying erroneous behavior, debugging). This

is particularly important in an environment where simulation of the system behavior is not possible, or in cases where software errors only manifest themselves when production is run [Pla84]. However, it is almost impossible to detect or fix a 'timing' bug for real-time programs [Gla80]. A timing problem further complicates system modification, reconfiguration and maintenance [Mok83].

In the past, most researchers have resorted to 'ad hoc' monitoring of a program written in a high-level language. Eventhough in practice most debuggers operate at a level close to that of the target machine in execution. Another, a more fundamental problem is that software-based debugger which can monitor both control-flow and data-space changes, may impose a heavy overhead on the execution of the target program. Because of this, the software based monitoring is impractical for most kinds of performance analysis, or for debugging in 'live' situations. Performance overhead is the principal factor which constrains the development of modern, high-level language oriented monitoring tools, and their application [CLW90].

3.1 The monitoring process

What is to be monitored is expresed in a monitoring statement which consists of two parts, a predicate and an action. The predicate is a boolean expression which is defined on the state space of the target process. When the predicate becomes true, then the action modifies the target process. For example, actions may consist of operations which are normally associated with performance and data analysis (e.g., saving current variable values for later processing, incrementing a counter) and with debugging (e.g., halting the monitored process).

In general, the predicate identifies individual states or a set of states in the state space of the target program. Predicates may be divided into three classes: a process predicate, a state predicate and a real-time predicate. The process predicate assigns a truth value to a target process. Process predicates arise in program performance analysis (for instance, how often are two specific statements executed in sequence) and in system security (e.g., warn me if any files have been opened but not yet closed at log-out time). The next one, the state predicate is a boolean function defined on the state space of the target program. It divides the state space into two regions, a region in which it yields a true value and another in which it evaluates to false. Finally, the real-time predicate is a state predicate which can be evaluated without delaying the target process.

The monitoring functions can be divided into two classes [Pla84]:

1. Tracing low-level events that change the process state and reconstructing a high-level interpretation of the process state.
2. Executing monitoring statements (i.e. evaluating predicates and executing actions if predicates become true).

As a matter of fact, a monitor process can be designed by two policies. The first one is to process as little information as possible, in order to achieve a short processing time. The other one is to update all dynamic information structures incrementally, assuming that each increment can be executed within a specifiable amount of processing time.

Monitoring requires that the entire state space of a program be accessible, implying two requirements for monitoring. The first is the ability to use selective and close control over the execution of the program. This may be carried down to the finest level. The second, refered to as "completness" of a monitor [PN81], is the ability to define monitoring predicates both in terms of the control flow of the program and of specific changes in its data space. This means that all conditions on the state of the target process that have interpretations at the source language level can be stated and their occurrence detected. Moreover, the requirement for debugging is the ability to inspect and modify details of the program state in execution.
Typical conditions include the following:

- fraction of CPU time spent in supervisor state;
- I/O channel activity, disk accesses;
- addresses generated, paging or cache activity.

Monitoring of conditions may be defined at three levels [CLW90], namely at the primitive level, at the abstract level and at the conditional level.

The primitive level implies the use of machine-level instructions. For instance, conditions at the primitive level would include the execution of an instruction at a particular location, or the accessing of data from a particular location. Completness at the primitive level is achieved by providing an implementation of three types of primitive monitoring functions: the code breakpoint, the data breakpoint and the watchpoint. The code breakpoint defines a point in the execution of the target process. The data breakpoint identifies access to a specific memory location and the watchpoint recognizes a change in value of a specific memory location.
At the abstract level, the notation and semantics of the high-level languages are used. For example, an abstract condition could be a change in the value of some program variable, or an entry to a particular procedure.
The conditional level of monitoring of conditions involves the description of a process state which may or may never be achieved. An example of this type of condition is reaching of a particular program execution path.

A condition defined at the abstract level or at the conditional level can in principle be implemented by translating it into one or more primitives. To do this, it is necessary for the monitoring system to determine absolute addresses of the target process at run-time. In certain cases it will be necessary to mirror the stack operation of the target process. All these problems refered to that kind of monitoring imply a need for dynamic software structures for monitoring.

3.2 Monitoring of single processor systems

Execution monitors (such as debugging aids, software performance measurement tools) may share the processor with the target process. When we have two processes it is not easy to solve the problem of synchronization between the target and monitor system. A monitoring system which slows down the target process is usually useless for real-time applications where the execution monitor must leave the timing behaviour of the target process unchanged.

The problem in implementation of such monitors is in switching between the two processes. This can be solved in two ways. First, the original target program is augmented by the code needed for monitoring.
In the second method, the user can enter the monitoring program step-by-step while the target program is running or ready to run. The target process and monitor have to be implemented as two different processes being executed on the same processor. The multiplexing of the processor between the two processes can be achieved in several ways:

1. The execution monitor assumes the role of the CPU and interprets the target program. In fact, the execution of the target process is slowed down too much [PN81] and that may be unacceptable for many applications.

2. The CPU supports a trace mode in which it generates a trap after each instruction, and an appropriate trap handler can be used to divide the processor between the two processes. Obviously, tracing each instruction considerably delays the target process.

3. Replacing instructions in the target program at locations where the execution monitor should gain control with calls to the execution monitor ('patching') allows full-speed execution in program parts where the execution monitor does not have to intervene. There may come into existence a problem if the number of patches becomes larger.

4. The performance of the monitoring system is improved by increasing the degree of parallelism through the use of additional hardware.

The latter method achieves real-time monitoring if we assume that it is possible to construct a hardware breakpoint device which is a true observer of the target process's activities.

3.3 Monitoring of real-time distributed systems

The monitoring of real-time distributed systems involves the collection and interpretation of information (e.g., event time stamps, synchronization sequences, race conditions, register status, transaction identifications, interrupt activities). This information can be used for improving the perfor-

mance or for testing and debugging of distributed systems.

Nevertheless, monitoring a real-time distributed system is much more difficult than monitoring a centralized, sequential computing system because of multiple asynchronous processes, critical timing constraints and significant communication delays [TFC90]. First of all, asynchronous processes behave nondeterministically and are irreproducible. Therefore, it is hard to understand and interpret the obtained results. Secondly, the correctness of a real-time distributed system is determined by meeting the timing constraints which are imposed by the real-world processes. Finally, geographically dispersed nodes may introduce a significant communication delay. This can cause improper synchronization among the processors. However, the inter-processor communication cost in distributed system is not negligible compared to the processor execution cost.

The noninvasive monitoring system presented by [TFC90] supports process-level activities (e.g., internode and intranode communication), function-level activities (e.g., procedure calls), and finally instruction-level activities (e.g., step-by-step instruction trace). Actually, monitoring process-level activities provides a high-level view of the target system's behavior. For example, typical events are creation and termination of a process, process synchronization, interprocess communication and external signals.

4 Implementation of Monitoring System

The techniques used to implement an execution monitor depend on the environment in which the monitor will be used. The earliest execution monitoring techniques are:

- user-controlled breakpoints;
- tracing, observing and setting values of variables;
- local modification of the program (patching).

Early execution monitors were obviously developed for assembly languages. Then came efforts to integrate debugging and other monitoring facilities into a programming language and its compiler. The simplest technique for dynamic analysis involves the insertion of 'probe' instructions (e.g. print statements) into the program text [CLW90]. This may be performed by preprocessing the program source. But the modifications to the program code may introduce unwanted-side effects.

The monitoring process can be implemented in two different ways [PN81]:

- systems that interleave entities of the monitoring language with the programming language;
- systems that allow the specification of the monitoring task as a separate 'monitoring program'.

The first kind of implementation is not designed for monitoring. In the next type monitoring statements may be interspersed between programming language statements. Hence data references are necessarily relative to the current point of control. In the last type, the predicate and action specifications may have be range over the entire state space.

A further step towards real-time monitoring is done by introducing a second processor which is able to execute the monitor concurrently with the target process. The implementation of monitor which shares the processor with the target process is shown in 4.1. Principles of monitoring on distributed systems are described in the subsection 4.2.

4.1 Principle of Monitoring Implementation on Single Processor System

An implementation of a real-time monitor on single processor system is decribed by [Pla84] and [PN81]. The idea is to use a FIFO (first-in first-out) queue which is inserted between the target processor and the monitor. The FIFO queue is a register which is able to store information from the system bus. The execution monitor should be synchronized with the output of the FIFO queue. It should have nearly the same speed as the target process to prevent the FIFO queue from overflowing. In this case the FIFO queue acts as a delay line. The second device, called "bus listener" or target processor interface is used to listen to the transactions occuring between the target processor and other parts of the system. The data are then fed into a FIFO register. At the output of the FIFO, the temporarily

stored data about memory transactions are used to manipulate the 'phantom' memory. The phantom memory is a dual port memory, which is accessed by the monitor process and the target processor interface. The monitor process may read the content of the phantom memory at any time. It may also lock the output of the FIFO. During that interval the data are kept in the FIFO. The monitor process may also interpret directly the sequence of memory transactions, using the data path from the FIFO. The last building block, a multiple breakpoint register, is added to speed up the monitor process. It is connected to the output of the FIFO, reports to the monitor any memory transactions referencing a location belonging to a previously defined set of memory locations.
Other functions of the monitoring system are implemented in software, such as tracing low-level events, executing monitoring statements, etc.
Processes are written in high-level, block structured programming language.
This kind of monitoring system has the following disadvantageous: first, it limits the feasibility of real-time monitoring to cases where procedure calls and returns do not happen too frequently, and second it limits the complexity and number of monitoring statements that can be submitted to the monitor.

4.2 Principles of Monitoring Implementation on Distributed Systems

Much research has been done and many tools for monitoring have been developed. Even though, they are not yet practical enough for monitoring real-time distributed computing systems due to their invasive nature. [TFC90] have been developed a real-time monitoring system to ensure minimal interference in the execution of a distributed target computing system. Their tool is used to support the testing and debugging as well as to evaluate the performance. This monitoring system extracts information directly from traffic on the internal buses of a target system and is described in the section 4.2.1. The next example of a real-time monitor represents an invasive monitoring system because it needs extra kernel support. Finally we introduce an approach which alleviates the invasive nature of a monitor.

4.2.1 Noninvasive Monitoring

A real-time distributed computing system consists of a hardware part and a software part. The hardware part includes a collection of nodes and a communication network. Each node has its own CPU, peripherals, memory, and communication interface. The software part includes the operating system, the communication module, and a collection of application processes. It is executed on each node of a real-time distributed computing system.

The main purpose of the noninvasive assumptions (e.g., the target system is a distributed/multiprocessor system with a master node and slave nodes, communications among processes via shared variables, asynchronous input to a target distributed system, procedural calls are implemented by system calls, recursive calls of a procedure are not allowed) is to ease the identification of the events of interest (such as interprocess communication, interprocess synchronization, creating and terminating process, input and output operations, procedural and recursive calls), and hence to simplify the trigger conditions and reduce the complexity of the postprocessing mechanism.

The system architecture of the noninvasive monitoring system consists of two major components: the interface module and the development module. The interface module can be considered as the front end of the monitoring system. Its main function is to latch the internal states of the target system based on the predefined conditions set by the user. Otherwise, it copies the internal states of the target node's processor and starts recording data from the buses of the target node onto the memory buffer unit. Second, the development module is the host computer for the interface module. This module is a general-purpose microprocessor-based system. It contains all the supporting software for the initialization of the interface module and postprocessing activities. The development module is basically independent of the target node processor. This is achieved by separating the target-dependent functions into the interface module. The development module provides an interactive interface to the user. It is responsible for all the testing and debugging activities, including

- initialization of the monitoring system,

- controlling the interface module to latch the target node execution history, and
- performing postprocessing on recorded execution history.

The development module consists of the development processor unit and the development memory unit. First, the development processor unit supports functions such as postprocessing, initialization of the interface module and providing a user-friendly interface. Second, the development memory unit consists of two parts: the development processor memory and the memory configuration unit. The program execution history from the target node is latched into the memory configuration unit.

Since the noninvasive monitoring system does not steal CPU time from the target real-time distributed computing system, it does not interfere with target system execution. The noninvasiveness of monitoring is achieved by extracting information directly from traffic on the internal buses of a target distributed system.

4.2.2 ARM

ARM is the example of the Advanced Real-Time Monitor/Debugger which monitors and debugs real-time tasks [Tok90]. It can also analyze the target system's runtime behavior in real-time. Besides it can visualize the system's scheduling decisions or events, analyze the number of tasks, the number of missed and met deadlines, the number of scheduling events, total CPU utilization. This tool has been developing to be used with the ARTS real-time kernel for distributed real-time application. The monitoring can be done directly on the actual target system or it can be run with Scheduler 1-2-3 simulator which will be mentioned in the continuing of the paper.
Their approach is based on monitorability analysis which is used to predict the maximum overhead caused by monitoring/debugging activities [TK88]. As we have seen, particularly in real-time monitoring, it is very important to predict the maximum interference and capability of the monitoring process itself.
As a matter of fact, this real-time monitor is an invasive monitoring system in the sence that it needs extra kernel support.

4.2.3 Relational Approach to Monitoring

Traditional monitoring techniques are inadequite when monitoring complex systems (e.g., multiprocessors or distributed systems). A new approach is described. In this approach a historical database forms the conceptual basis for the information processed by the monitor [Sno88]. Monitoring is concerned with retrieving information and presenting this information in a derived form to the user. Therefore, the monitor is fundamentally an information processing agent, with the information describing time-varying relationships between entities involved in the computations. Otherwise, monitoring is an information-processing activity. The generated information is structured in the *relational model.* The attention is focused on the query of a relational database model so that the user can specify what information is to be collected. This approach also controls the amount of monitoring data collected. To summarize, this approach permits advances in specifying the low-level data collection, specifying the analysis of the collected data, performing the analysis, and displaying the results.

5 Scheduling Analysis

For real-time embedded system analysis, it is necessary to incorporate the timing information into analysis. However, there are several problems in building real-time development tools. For instance, basically correct software actions which are too early or too late can lead to unsafe conditions. Furthermore, the predictability of the analyzer depends on what scheduling policies the target system uses and what types of real-time tasks it has [TK88]. For this reason, real-time software must be verified to adhere to its critical timing constraints before it is used. This verification process is denoted as *schedulability analysis* [Sto87]. A schedulability analyzer can analyze or verify whether or not a given real-time task set can meet its timing constraints under a specific scheduling policy.

In analyzing the scheduling algorithms, different performance metrics can be adopted. In dynamic real-time scheduling, it is very important to determine the percentage of tasks which meet their

deadlines (Weighted Guarantee Ratio) [BS88]. On the other hand, real-time systems must be analyzed for worst-case schedulability. In this case we should know how many events can occur in the worst case. A well-known solution is done by Leinbaugh (Guaranteed response time) [Lei80]. The performance analysis is done by simulating the behavior of the algorithms.

More complex analysis can be done by using the Scheduler 1-2-3, which has been developing for the Arts distributed real-time kernel [Tok90]. It uses analysis methods to determine whether a feasible schedule exists for a given task set and under what conditions deadline can not be met.
The Scheduler 1-2-3 can be used to predict the timing effects due to software and hardware modifications and it can be integrated with other test tools and the real-time monitor. Therefore, it can be used as the close-form analyzer or simulator. For example, the schedulability analysis under the rate monotonic algorithm [Kor90] is done by means of a closed formula analysis, while for other scheduling algorithms Scheduler 1-2-3 provides a simulator.

6 Conclusions

In this paper we introduce monitoring system which can support performance evaluation, testing and debugging of real-time systems. A real-time monitor can monitor and verify if the system meets timing requirements at runtime. It must not affect the timing constraints of the monitored system.
We discussed about real-time programming constructs which are used to control interaction and are often called upon through use of system services of a real-time operating system. Furthermore, we must take into account the role of programming languages in real-time programming. Programming languages should have explicit constructs to express the time realted behavior of modules and the semantics must be unambiguous.

The real-time monitoring is presented as observing a sequence of states of a process (i.e., predicates) and assigning a truth value to each of them. A positive evaluation of a predicate indicates an action, which can be used to record information about the process. A real-time monitor on single processor system shares a processor with a target process. The problem of synchronization between the processes is essential. Furthermore, a monitoring of real-time distributed systems is much more difficult because of multiple asynchronous processes, critical timing constraints and significant communication delays. The inter-processor communication cost is not negligible compared to the processor execution cost. This factor must be explecitely taken into account in scheduling. Some of the various existing implementations of a real-time monitor on a single processor and on distributed system was presented.

References

[BS88] S. R. Biyabani and J. A. Stankovic. The integration of deadline and criticalness in hard real-time scheduling. In *Proceedings of the Real-Time Systems Symposium*, pages 152–160, December 1988.

[CLW90] C. C. Charlton, P. H. Leng, and D. M. Wilkinson. Program monitoring and analysis: software structures and architectural support. *Software–Practice and Experience*, 20(9):859–867, September 1990.

[CSR87] S. C. Cheng, J. A. Stankovic, and K. Ramamritham. Scheduling algorithms for hard real-time systems – a brief survey. July 1987.

[Gla80] Robert L. Glass. Real-time: 'the lost world' of sofware debugging and testing. *Communications of the ACM*, 23(5):264–271, May 1980.

[Kor90] Barbara Koroušić. Real-time executives for embedded microprocessor applications. *Informatica*, 14(4):58–63, 1990.

[Lei80] Dennis W. Leinbaugh. Guaranteed response times in a hard-real-time environment. *IEEE Transactions on Software Engineering*, SE-6(1):85–91, January 1980.

[LS87] Nancy G. Leveson and Janice L. Stolzy. Safety analysis using petri nets. *IEEE Transactions on Software Engineering*, SE-13(3):386–397, March 1987.

[Mok83] A. K. Mok. *Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment.* PhD thesis, Massachusetts Institute of Technology, May 1983.

[Pla84] Bernhard Plattner. Real-time execution monitoring. *IEEE Transactions on Software Engineering*, SE-10(6):756–764, November 1984.

[PN81] Bernhard Plattner and Jurg Nievergelt. Monitoring program execution: a survey. *Computer*, 13(11):76–93, November 1981.

[PS85] J. L. Peterson and A. Silberschatz. *Operating System Concepts. Reading.* MA: Addision-Wesley, 1985.

[Sch88] James D. Schoeffler. A real-time programming event monitor. *IEEE Transactions on Education*, 31(4):245–250, November 1988.

[Sch89] James D. Schoeffler. Real-time programming and its support environment. *IEEE Transactions on Education*, 32(3):377–381, August 1989.

[Sno88] Richard Snodgrass. A relational approach to monitoring complex systems. *ACM Transactions on Computer Systems*, 6(2):157–196, May 1988.

[SR87] J. A. Stankovic and K. Ramamritham. The design of the spring kernel. In *Proceedings of the Real-Time Systems Symposium*, pages 146–157, December 1987.

[Sta88] John A. Stankovic. Real-time computing systems: the next generation. *In Tutorial Hard Rael-Time Systems*, 14–37, 1988.

[Sto87] Alexander D. Stoyenko. A schedulability analyzer for real-time eucklid. In *Proceedings of the Real-Time Systems Symposium*, pages 218–227, December 1987.

[TFC90] J. J. P. Tsai, K. Fang, and H. Chen. A noninvasive architecture to monitor real-time distributed systems. *Computer*, 23(3):11–23, March 1990.

[TK88] H. Tokuda and M. Kotera. A real-time tool set for the arts kernel. In *Proceedings of 9th IEEE Real-Time Systems Symposium*, 1988.

[Tok90] Hideyuki Tokuda. Arts real-time scheduler analyzer/debugger. May 1990.

INFORMATIONAL MODELS OF DICTIONARIES I

INFORMATICA 1/91

Keywords: data, dictionary, electronic dictionary, information, informational dictionary, organization, structure, terminology, understanding

Anton P. Železnikar
Volaričeva ul. 8
61111 Ljubljana

A computer-aided dictionary can be comprehended, structured, and organized in three basic ways: as a data-structured information (classical computer program), as an information-oriented structure (advanced electronic dictionary), and as a complex informational entity in the sense of the informational arising capability. All three concepts can substantially surpass the performance of a book form dictionary which always performs as a fixed, unchangeable data entity.

Computer-aided and information-oriented dictionaries can bring into the foreground the most complexly imaginable semantic and pragmatic connectedness of information items. Under these circumstances it is possible to introduce various aids for construction, development, and understanding of information items constituting an advanced and also to the most pretentious user oriented dictionary. Further, the performance of bilingual and multilingual dictionaries can contain the quality of translation of sentences from one language into another on the user request. In monolingual dictionaries, the demands on syntactic, semantic, pragmatic, and stylistic requirements can be covered strictly and efficiently, delivering in any respect improved and corrected texts in written natural languages. This essay stresses the perspectives of an *informational approach* at the design of future electronic dictionaries.

Informacijski modeli slovarjev I. Računalniško podprti slovar je mogoče razumeti, strukturirati in organizirati na tri različne načine: kot podatkovno strukturirano informacijo (klasični računalniški program), kot informacijsko zasnovano strukturo (napreden elektronski slovar) in kot kompleksno informacijsko entiteto v smislu zmogljivosti informacijskega nastajanja. Vsi trije koncepti zmorejo bistveno preseči zmogljivosti knjižnega slovarja, ki predstavlja le fiksirano, nespremenljivo podatkovno entiteto.

Računalniško podprti in informacijsko zasnovani slovarji lahko upoštevajo tudi najbolj kompleksno semantično in pragmatično povezanost informacijskih enot. Pri tem je mogoče vpeljati različne pripomočke za gradnjo, razvoj in razumevanje informacijskih enot, ki sestavljajo napreden in tudi za najzahtevnejšega uporabnika zasnovan slovar. Razen tega pa lahko ima dvojezikoven ali večjezikoven slovar še lastnost prevajanja stavkov iz enega jezika v drugi na zahtevo uporabnika. V enojezikovnih slovarjih je mogoče dosledno in učinkovito izpolniti zahteve po sintaksnih, semantičnih, pragmatičnih in stilističnih značilnostih jezika in tako izboljšati in korigirati (lektorirati) pisana besedila v naravnih jezikih. Ta spis poudarja prednosti *informacijskega pristopa* pri zasnovi prihodnjih elektronskih slovarjev.

1. Introduction

... The semantic system of a language has to do with meanings and thus with the relation between the conventionalized symbols that constitute language and the external reality about which we need to communicate through language. ...

(NCD) 24

What is an informational dictionary and how does it differ from a common, book form dictionary? But even book form dictionaries can differ essentially among each other in their intention, extensiveness, and purpose. For the most pretentious users of dictionaries who search not only for the minimal meaning, translation, or pure concept of a word, but also for several other attributes like word contexts, phrases, idioms, and whole sentences, paragraphs, etc., who investigate informational items in an epistemological and hermeneutic (semiotic and historical) way, a convenient book form dictionary may not be a satisfactory solution at all. Already the semantic and pragmatic conceptual study of words calls for a careful examination of the meanings belonging to various informational items and the studying of origins, going or descending back into the domain of different languages, the modern and the antique ones, in a recurrent meaning-improving manner. To this type of the searching and construction of meaning, the intention of creating a new, innovative, and adequate meaning has to be taken into the conceptualization and design of an informational dictionary. In this respect, the process of using an informational dictionary, extracting the meaning from it, and constructing a resulting meaning of an informational item approaches, in principle, the problem of understanding as information (UI1, UI2).

The tool one would like to have is a particular, let us say informational expert system which could deliver a complex, linguistic (semiotic), inter-linguistic (multi-linguistic), and particular information concerning the possible and variously related meaning of an informational item (word, phrase, sentence). More precisely, this informational tool should perform as a sufficiently sophisticated informational expert dictionary system (IEDS for short) in any respect. In this essay we will develop concepts and symbolic formalism of IEDS's, rooting in the informational algebra (IIA) and suggesting their informational structure and organization in the proposed algebraic way. In the most cases we shall prefer the so-called formal hermeneutic approach of understanding (UI2), which considers the development of the meaning belonging to an informational item (term, headword) through the item's historicity, but giving also the outlook to the so-called direct (non-historical) parallel-cyclic approach of understanding, which can occur at the conferring of a new arising or arisen meaning in an innovative way.

In the course of the discourse concerning the problem of understanding at the use of informational dictionaries, several approaches are possible. The most direct approach is the so-called spontaneous-circular understanding as used, for instance, by an inexperienced and linguistically unpretentious user of a dictionary. A professional writer, for instance, will insist to use an extremely hermeneutical method based on elementary spontaneous-circular understanding of an informational item, but also on the mostly considered expertise of understanding. While in the first case, the approach to understanding will be characteristically understanding, using a regular informing of understanding, in the second case, this process will take the form of expertizing, giving attention to the specialized emphases, which cannot only improve the content of meaning but can also enrich or innovate it essentially. In an implicit or explicit form, all these approaches of understanding of informational items of a dictionary will concern the moving from one informational item to another inside the information's semantic and pragmatic nets (maybe mutually connected or not) within an informational dictionary. In some respect, a satisfactory informational dictionary will perform as a »well-connected« (completely connected) net of included informational items. Theoretically, the well-connected net will guarantee the access from an arbitrary informational item of the dictionary to any other concerning item. In general, an arising dictionary will always enlarge and enrich its nets and thus improve its well-connectedness in an

informational manner.

The question of building up the process of a dictionary usage goes back to the question, how does a writer or speaker perform in writing or speaking a text. Further, how does a language interpreter or a professional writer search and construct the meaning of words, phrases, and sentences by using single- language, cross-language, and professional (technical) dictionaries, handbooks, lexicons, encyclopedias, scientific, technical, philosophical scriptures, etc. How does someone construct the meaning of an informational item by gathering information from various and semantically different documents, memory, and its own intention? And finally, how does someone invent its own meaning, although considering the usual and existing meaning for an informational item?

The discussion in the previous passage lends our attention to the problem of discourse occurring between a user's metaphysical disposition (as an individual, conscious and unconscious information) and an informational dictionary (as a system information). The moving inside different semantical-pragmatical nets of a dictionary will largely depend on the dialog between the user and the informational expert facilities of the dictionary. With the exception of the problem of understanding and expertness of this bipolar informational game also the problem of discursiveness will come into the focus of our attention.

On the basis of this introductory analysis, the following scheme of our formal informational investigation can be put into the question: What could be a general informational structure of an informational dictionary? Which are the basic questions of the design and usage of a dictionary through the elementary possibilities (approaches) of understanding? Which could be the consequences of the discursive nature of understanding between the user and an informational dictionary (ITD)? How does a dictionary function as a discursive informational expert dictionary system.

What is an informational model? The informational model (IM) of something will be a system of informational formulas describing something as an informational entity (*Železnikar,* IIA). Electronic dictionary, ED for short, will be the term conceptualized by the *Japan Electronic Dictionary Research Institute* (look at OED) and marking a sophisticated project and implementation of dictionaries in natural languages. Thus, an ED is an information-aided or information-oriented structure which essentially surpasses the so-called book form dictionaries and possibly also several performances (functionality) of living human dictionaries. This holds true in particular in cases of inter-lingual dictionaries concerning several natural and ancient languages simultaneously.

In this essay we shall distinguish three main informational models (IM's) of dictionaries, that is, a data-structured dictionary, an information-structured dictionary, and, finally, an informational dictionary.

The second basic aspect of an electronic dictionary we must keep in mind is the question of understanding which concerns an ED, that is, the self-understanding as the quality of the programmed computing system and the so-called other-understanding (understanding of the electronic system users and designers). The system understanding represents the system functionality, its particular performance or intelligent capability in comparison to living and other artificial systems.

2. Data and Information

... No living language stands still, however much we might wish at times that it would. ...

(NCD) 24

Before we start the discourse on data-structured, information-structured, and informational dictionary, it is recommendable to clear the question concerning the concept of data, i.e. data entity, and that of information, i.e. informational entity, conceptually and in an informationally formal (logically informational) way. So we have to point out clearly the conceptual difference of occurring theoretical distinction between the so-called data and information(al) entities.

What is data as an informationally theoretical entity? How does it inform and how can it be not informed? A written text (fixed on a sheet of paper) appears always as data. It is protected against any change or illegal informational arising. A written text can inform any of its readers, certainly in different manners, depending on the informational circumstances concerning the text and the reader. We agree that a written text cannot be changed in a writing way, thus the copies of the same text perform as equivalent data.

Let data as a particular informational operand be marked by δ. We agree upon the fact that data inform in an open way, that an informational receptor can be informed by data. This openness of data δ possibility is symbolically captured by expression $\delta \models$. On the other hand, we have to consider two further facts concerning the lasting (absolute memorizing, steadiness) of data, roughly as $\delta \models \delta$, and the impossibility of data to be informed (informational unchangeableness), roughly as $\not\models \delta$. The last formula, $\not\models \delta$, is closed in regard to δ because of $\delta \models \delta$, otherwise it is open. Further, formula $\delta \models \delta$ has to be particularized because operator $\models$ is a memorizing operator, that is, a non-arising (fixed) informational operator. Thus, instead of $\delta \models \delta$ we can introduce $\delta \models_m \delta$, where $\models_m$ marks a pure memorizing (data storing) informational operator.

After this conceptualization of data δ we can adopt the following informational implication:

(d1) $\quad \delta \Rightarrow (\delta \models; \delta \models_m \delta; \not\models \delta)$

In this formula (system of parallel informational formulas), operators $\models$ and $\not\models$ are the most general informational operators in the sense that

(d2) $\quad (\delta \models) \Rightarrow (\delta \models_\delta \circ \models);$
$\quad (\not\models \delta) \Rightarrow (\models_\delta \circ \models_\delta \delta)$

Here, the operator composition $\models_\delta \circ \models_\delta$ performs as operator $\models_m$, which is closed (non-informing) to any other informational entity except of the memorizing data δ. Further, a more precise denotation of operator $\models_m$ would be $\models_\delta$ to which the operator composition $\models_\delta \circ \models_\delta$ reduces in a logical way. Finally, the entity δ as data implies

(d3) $\quad \delta \Rightarrow (\delta \models; \models_\delta \delta)$

where $\models$ is a general informational operator, whereas $\models_\delta$ is closed against any other operand except δ and, in this way, represents also a general informational operator of non-informing, that is, operator $\not\models$. Additionally, operator $\models_\delta$ is the operator of preserving or strict memorizing of the data entity δ, that is, particularly structured operator, denoted by $\models_{\delta,m}$.

We see how data δ is an informational exception in regard to regular (arising) informational entities. A question which arises on account of formula (d3) is what could the expression

(d4) $\quad \delta \models_\delta; \models_\delta \delta$

represent. By definition, this data entity cannot inform or cannot be informed by any other informational entity except itself. Such an entity would perform as informational noise or concealment for other informational entities. However, entities (processes) $\delta \models_\delta$ and $\models_\delta \delta$ which constitute (d4) could inform and could be informed, for instance, as the processes $(\delta \models_\delta) \models$; $\models (\delta \models_\delta)$; $(\models_\delta \delta) \models$; $\models (\models_\delta \delta)$; etc. A paragon of such a concept could be, for instance, the so-called Being as information (a universal constant or unity of being which never changes, as taught by the Eleatic and some modern existentialists).

If data is information which as an informational entity during an informational process never changes, information is the entity which ever or always changes. By this qualification, changing data perform as information. But this can never take place in the case of a book as seen from the side of the book itself. A book can neither textually change itself nor can it be textually changed by some other informational means. But this does not hold in the case of an electronic book (dictionary) if the user has the access to the text of the book.

What is information as an informationally theoretical entity and what is a clear distinction in

comparison to data? How does information inform and how is it informed? Let information as a particular informational operand be marked by α. We agree upon the fact that information informs in an open way and that by this entity impacted entities can be foreseeable but also unforeseeable. This openness of information α possibility is symbolically captured by $\alpha \models$. On the other hand, we have to consider two further facts concerning the open and spontaneous cycling of information, roughly as $\alpha \models \alpha$, that is the possibility of information to be informed, roughly as $\models \alpha$. The last formula, $\models \alpha$, is unlimitedly open in regard to α as well as in regard to any other informational entity which could impact α. After this conceptualization of information α we can adopt the following informational implication:

(i1) $\alpha \Rightarrow (\alpha \models; \alpha \models \alpha; \models \alpha)$

In this formula (system of parallel informational formulas), operators of the form $\models$ are the most general informational operators in the sense that

(i2) $(\alpha \models) \Rightarrow (\alpha \models_{\alpha} \circ \models)$;
$(\models \alpha) \Rightarrow (\models \circ \models_{\alpha} \alpha)$

As we can see, the concept of information is in some sense controversial to the concept of data. By definition, data never changes or, at least, does not change within a domain of observation. On contrary, information ever changes or, at least, possesses the possibility to change and to be changed by itself and other informational entities. Thus, the so-called Being as a philosophical concept appears as the most universal data, in some respect opposite to the idea of god, which embodies the most pretentiously developing, although self-sufficient informational entity. The advantage of the concept of information lies in the possibility to express ideas being controversial to the concepts themselves.

We can make a direct comparison between the concept of information α and the concept of data δ in the following formal way:

(id1) $\alpha \Rightarrow (\alpha \models; \models \alpha); \delta \Rightarrow (\delta \models)$;
(id2) $\alpha \Rightarrow (\alpha \models \alpha); \delta \Rightarrow (\delta \models_{m} \delta)$

We use operators $\Rightarrow$ and ';' between formulas as informational operators of implication and of 'or/and', respectively. Thus, information α implies $\alpha \models$ or/and $\models \alpha$, i.e. an open informing of α in an outward and inward direction. In this sense, data δ is a sub-concept of information α. However, data δ preserves its form (data information), thus, $\delta \models_{m} \delta$, where $\models_{m}$ is the symbol of an absolute (non-arising) memory operator with the meaning

(d5) $(\delta \models_{m} \delta) \Rightarrow (\delta = \delta)$

The so-called changeable data become information for which formula (d5) holds between two events of their change. On contrary, information can change by itself alone, without any outward influence.

3. A General Informational Structure and Organization of a Dictionary

... in English, word order is a dominant factor in determining meaning, while the use of inflectional endings to mark the grammatical function of individual words within a sentence plays a clearly subordinate role ... Other languages show markedly different patterns, such as Latin with its elaborate set of paradigms for nouns, verbs, adjectives, and pronouns and its highly flexible word order. ...

(NCD) 24

In any case, a dictionary is a kind of expert information dedicated to the correct, that is, commonly (socially) understandable speaking, writing, and language forming, creation, and translation. Usually and formally, dictionaries implement implicitly (rarely explicitly) the so called semantic nets, that is to say, networks of connections among various informational items with semantically and pragmatically related meanings.

A semantic net is an informational connectedness of items which originates out of the pos-

sibilities of distinct informational items to inform other and to be informed by other informational items. An informational net is determined by an informational system with identified and yet not identified, that is, possible forms of informing. Such as it is, a dictionary implements a complex informational system of informational markers (for instance, words) with their attributes and to them corresponding forms of meaning. But meaning of an item is nothing other than an informational formula with its own meaning which points to the meaning of other items occurring in the formula, in an informationally parallel and cyclic way. Thus, informational markers with their actual and possible meaning create an implicit informational net which arises in the process of searching in a dictionary also through the intention or requirements of the user.

Let us introduce the basic formal terms which constitute the informational structure of a dictionary.

An informational dictionary as a ordered collection of informational items, their attributes, and meanings will be marked by ϑ. A dictionary ϑ informs as a regular information and its informing (understanding) will be marked by $\mathfrak{D}$. In principle,

(g1) $\quad \vartheta, \mathfrak{D} \models \vartheta, \mathfrak{D}$

is a direct, straight-forward (potentially developing) relation which says that a dictionary ϑ informs as such (per se), that is, $\vartheta \models \vartheta$, that it informs a kind of understanding $\mathfrak{D}$ (belonging to the dictionary and/or somebody else), that this process, as a form of understanding, can influence the content of a dictionary in the form $\mathfrak{D} \models \vartheta$ (in case of a dynamic changeable dictionary ϑ), and that any understanding of a dictionary can inform as such (per se) in the form $\mathfrak{D} \models \mathfrak{D}$. The hermeneutical approach to this basic situation would be

(g2) $\quad (\vartheta \models \mathfrak{D}) \models \vartheta$ or also
$\quad (\mathfrak{D} \models \vartheta) \models \mathfrak{D}$

We see how informing $\mathfrak{D}$ assumes the role of understanding of item ϑ and vice versa. In a real case, these basic formulas will appear in a much more complex (reasonably decomposed, developed, and/ or arisen) formal context into which several other components, for instance, various attributes and the meaning of items, will enter.

An informational item (word, clause, phrase, sentence, paragraph, etc.) of a dictionary will be denoted by ω. An informational item (word) informs in a specific way, for instance, within an instantaneous user's metaphysical disposition or as an attributed meaning within the dictionary ϑ, as a form of informing, marked by $\mathfrak{W}$. Formally, we have the following formulas, expressing this part of the definition:

(g3) $\quad \omega, \mathfrak{W} \models \omega, \mathfrak{W};$
$\quad (\omega \models \mathfrak{W}) \models \omega; (\mathfrak{W} \models \omega) \models \mathfrak{W};$
$\quad \omega \in \vartheta; \mathfrak{W} \in \mathfrak{D}$

A possible consequence of formulas (g2) and (g3) is, for instance,

(g4) $\quad (((\omega \models \mathfrak{W}) \models \vartheta) \models \mathfrak{D}) \models \omega;$
$\quad (((\mathfrak{W} \models \omega) \models \mathfrak{D}) \models \vartheta) \models \mathfrak{W}$

In these formulas, informing $\mathfrak{W}$ as a component of the item's understanding already hides the meaning of ω. We see how the complexity of hermeneutic understanding of an item in the dictionary grows with the decomposition (or identification) of the concept.

Entity ω is a marker which marks the corresponding attribute(s) and meaning(s), that is, $\alpha(\omega)$ and $\mu(\omega)$, respectively. In a dictionary ϑ, its item ω always assumes the informational triplet (or in general an n-tuple) $(\omega, \alpha(\omega), \mu(\omega))$. This triplet can be read as an informational correspondence of ω, $\alpha(\omega)$, and $\mu(\omega)$ and expressed formally by

(g5) $\quad (\omega \therefore \alpha(\omega)) \therefore \mu(\omega)$

where $\therefore$ is an informational operator with the meaning 'corresponds'. In general,

(g6) $\omega_1, \ldots, \omega_m \in \vartheta;$
$(\omega_i \therefore \alpha(\omega_i)) \therefore \mu(\omega_i);\ i = 1, \ldots, m$

where $\omega_1, \ldots, \omega_m$ are elements of dictionary ϑ. The consequence of this concept is

(g7) $(\omega \in \vartheta) \Rightarrow ((\omega \therefore \alpha(\omega)) \therefore \mu(\omega));$
$((\omega \therefore \alpha(\omega)) \therefore \mu(\omega)) \in \vartheta$

This is certainly only a static definitional expression of the content $((\omega \therefore \alpha(\omega)) \therefore \mu(\omega))$ entering in a dictionary ϑ.

While ω belonging to ϑ is a single informational item, the attribute and the meaning of ω, marked by $\alpha(\omega)$ and $\mu(\omega)$, respectively, are, in general, informational sets of items, that is, sets of formulas depending on other informational items belonging to ϑ. In some respect, an informational dictionary appears as an in-itself closed (completed) information.

The attribute of ω is a set of informationally distinct data entities (for instance, pronunciation, grammatical items, the kind and time of ω's origin, graphical explanation, pictorial illustration, etc.). Certainly, the attribute item can have semantic connections to other items in the dictionary.

The meaning of ω is a set of informationally distinct entities expressed in the form of informational formulas of informational operands and operators (as well as parentheses) and ω within ϑ is nothing else than the marker of this set of formulas. How is the meaning of a marker structured and organized into the so-called semantic-pragmatic net within a dictionary?

If we look into a book dictionary, the meaning of an item appears usually as a non-empty collection of phrases, that is composite structures of words. The formally declared meaning of item ω, marked by $\mu(\omega)$, can be informationally divided, in general, as

(g8) $\mu(\omega) = (\pi_1(\omega), \ldots, \pi_n(\omega))$

However, formula (g8) is an informationally static expression. In fact, partial meanings of ω, that is, phrases $\pi_1(\omega), \ldots, \pi_n(\omega)$, inform the resulting meaning $\mu(\omega)$ of ω and thus instead of formula (g8) the informing of these partial entities to the entire meaning of ω in ϑ is, in general, a more adequate situation:

(g9) $(\pi_1(\omega), \ldots, \pi_n(\omega)) \models \mu(\omega)$

This informing simultaneously satisfies the condition that with the exception of the complete inclusion of phrases in the meaning of an item, phrases of the meaning are in no way isolated or mutually and also otherwise independent items, that is,

(g10) $\pi_1(\omega), \ldots, \pi_n(\omega) \subset \mu(\omega);$
(g11) $\pi_1(\omega), \ldots, \pi_n(\omega) \models \pi_1(\omega), \ldots, \pi_n(\omega)$

In fact, the mutual informing of phrases (g11) produces a resulting phrase of ω's meaning, marked by $\mu_r(\omega)$, also in the user domain, where

(g12)
$(\pi_1(\omega), \ldots, \pi_n(\omega) \models \pi_1(\omega), \ldots, \pi_n(\omega)) \models \mu_r(\omega)$

or in a more complete form

(g13)
$((\omega \therefore \alpha(\omega)) \therefore$
$(\pi_1(\omega), \ldots, \pi_n(\omega) \models \pi_1(\omega), \ldots, \pi_n(\omega))) \models \mu_r(\omega)$

It is to stress that operator '$\therefore$' has to be understood as an informationally active operator and not only as a static relation lacking the dynamics of understanding as information. Operator $\therefore$ is a particularized form of the general operator of informing, that is, operator $\models$. Although the proposed schemes (or scenarios) of meaning of an item concern understanding, we have not introduced an explicit indication of understanding yet. This type of formalization we shall develop in some of the following sections expressing the explicit forms of informing belonging to the distinct items of meaning. Thus, we shall proceed to some concrete structures and organizational schemes of an electronic dictionary.

In general, it is possible to introduce several distinct types of understanding concerning a composite informational item ω of a dictionary and particularly the understanding of the meaning, which corresponds to the dictionary item component μ. Various types of understanding can be constructed as parallel-cyclic, hermeneutic, and hermeneutic parallel-cyclic (mixed) modes of understanding.

4. Terminology, Symbols, and Relations Concerning Dictionaries

... The major systems that make up the broad comprehensive system of language itself are four in number: lexicon, grammar, semantics, and phonology. The one that dictionary editors and dictionary users are most directly concerned with is the vocabulary or lexicon, the collection of words and word elements which we put together in various ways to form larger units of discourse: phrases, clauses, sentences, paragraphs, and so forth. ...

(NCD) 23

A dictionary as informational entity will be denoted by ϑ. Particular dictionaries as well as sub-dictionaries of a dictionary will be denoted by subscripted ϑ's. A subscript will be the integer or the letter combination.

Particularly important dictionaries will be, for instance: the word dictionary, marked by ϑ_w; the concept dictionary, marked by ϑ_c; the co-occurrence dictionary, marked by ϑ_{co}; and the bilingual dictionary, marked by ϑ_b. These particular dictionaries appear as information entities within the Japan EDR electronic dictionary (OED).

Already particularized dictionaries can be split, for instance, in the following way: the word dictionary ϑ_w into the general vocabulary dictionary ϑ_{gv} and the technical terminology dictionary ϑ_{tt}; the concept dictionary ϑ_c into the concept classifications ϑ_{cc} and the concept descriptions ϑ_{cd}; the co-occurrence dictionary ϑ_{co} into the Japanese co-occurrence dictionary ϑ_{Jco}, the English co-occurrence dictionary ϑ_{Eco}, and the Slovene co-occurrence dictionary ϑ_{Sco}, for instance; the bilingual dictionary ϑ_b into the Japanese-English dictionary ϑ_{JE} and the English-Japanese dictionary ϑ_{EJ} or into the Slovene-English dictionary ϑ_{SE} and the English-Slovene dictionary ϑ_{ES}, etc.

Furthermore, the general vocabulary dictionary ϑ_{gv} can be split into the Japanese general vocabulary dictionary ϑ_{Jgv} and the English general vocabulary dictionary ϑ_{Egv}, or into the Slovene general vocabulary dictionary ϑ_{Sgv} and the English general vocabulary dictionary ϑ_{Egv}, etc. Similarly, the technical terminology dictionary ϑ_{tt} can be split into the Japanese technical terminology dictionary ϑ_{Jtt} and the English technical terminology dictionary ϑ_{Ett}, or into the Slovene technical terminology dictionary ϑ_{Stt} and the English technical terminology dictionary ϑ_{Ett}, etc.

By this surface structure of embedded dictionaries we have obtained a rough structure of an electronic dictionary which can be informationally expressed by the following system of operationally interconnected informational entities:

$$
\begin{aligned}
\text{(D1)}\quad & \vartheta = (\vartheta_w, \vartheta_c, \vartheta_{co}, \vartheta_b);\\
& \vartheta_w = (\vartheta_{gv}, \vartheta_{tt});\\
& \quad \vartheta_{gv} = (\vartheta_{Jgv}, \vartheta_{Egv});\\
& \quad \vartheta_{tt} = (\vartheta_{Jtt}, \vartheta_{Ett})\\
& \quad \text{or } \vartheta_{gv} = (\vartheta_{Sgv}, \vartheta_{Egv});\\
& \qquad \vartheta_{tt} = (\vartheta_{Stt}, \vartheta_{Ett});\\
& \vartheta_c = (\vartheta_{cc}, \vartheta_{cd});\\
& \vartheta_{co} = (\vartheta_{Jco}, \vartheta_{Eco})\\
& \text{or } \vartheta_{co} = (\vartheta_{Sco}, \vartheta_{Eco});\\
& \vartheta_b = (\vartheta_{JE}, \vartheta_{EJ})\\
& \text{or } \vartheta_b = (\vartheta_{SE}, \vartheta_{ES})
\end{aligned}
$$

The next step in this system determination is to define the relations (interconnections, operations, also understanding) among the occurring entities (entries) and the deep structure of any of them, which brings new terms and symbols to the surface.

What are the data entries of all these dictionaries, how are they structured, interconnected, and understood, for instance, in the Japanese EDR

electronic dictionaries (OED)?

What is the so-called word entry? The word entry ω is composed of a headword, marked by ω_h, which is the surface representation of a word, its grammatical information ω_g (grammatical features), and the corresponding headconcept ω_{hc}, which is the concept represented by words in the context. Thus,

(h1) $(\omega = (\omega_h, \omega_{hg}, \omega_{hc})) \in \vartheta_w$

is already an inter-dictionary structure. Further, there are relations among headwods ω_h or among headconcepts ω_{hc} which are defined within the word entries ω. Concept entries ω_c define possible relations between headconcepts ω_{hc} with concept relation labels ω_{crl}. Thus,

(h2) $(\omega_c = (\omega_{hc}, \omega_{crl})) \in \vartheta_c$

Co-occurrence entries ω_{co} define all of the possible co-occurrence between headwords ω_h with co-occurrence relation labels ω_{corl}, so,

(h3) $(\omega_{co} = (\omega_h, \omega_{corl})) \in \vartheta_{co}$

Inter-lingual correspondence entries ω_b, in concrete cases, bilingual entries ω_{JE}, ω_{EJ}, ω_{SE}, ω_{ES}, etc., define correspondence between headwords ω_h of different languages, in concrete cases, headwords ω_{hJ}, ω_{hE}, ω_{hS}, etc. For instance,

(h4) $(\omega_{JE} = (\omega_{hJ}, \omega_{hE})) \in \vartheta_{JE}$;
$(\omega_{EJ} = (\omega_{hE}, \omega_{hJ})) \in \vartheta_{EJ}$;
$(\omega_{SE} = (\omega_{hS}, \omega_{hE})) \in \vartheta_{SE}$;
$(\omega_{ES} = (\omega_{hE}, \omega_{hS})) \in \vartheta_{ES}$

etc. Here, ϑ_{JE}, ϑ_{EJ}, ϑ_{SE}, ϑ_{ES}, etc. are bilingual dictionaries.

In formulas (h1), ... , (h4), relations between different entries are implicit. It is possible to make these relations explicit in a general way, with the intention to mark these relations and make them later on completely definite by further particularization of operators and/or composition and decomposition of formulas.

For instance, we can *decompose* the concept dictionary $\vartheta_c = (\vartheta_{cc}, \vartheta_{cd})$ in (D1) which is an informational network structure comprising a set of concept entries $\gamma = (\gamma_n, \gamma_r)$, with nodes γ_n which represent the concepts and arcs γ_r which indicate the relation between one concept and another.

5. Understanding as the Processing Part among the Parallel Dictionaries within an Electronic Dictionary

Understanding in the framework of an ED can be conceived from at least three points of view: as system understanding implemented within the programmed computer system which comprises the ED; as understanding of designers who develop the ED by means of an ED development system and understand also the development system itself; and, at last, as understanding of the ED performance and capability by the ED users who have this system on disposal, however, can still adapt or alter it to some reasonable extent, for instance, by a built-in or outward possibility of learning.

In the process of ED development, the designers and constructors start by a top-down approach of solving the problem of, for instance, the very sophisticated ED not being definitely depicted yet. At the very beginning, they may have in mind the classical data-information system (D1) of the ED, which from this point on can be put into the process of the so-called informational arising, where the occurring entities can be informationally decomposed deeper and deeper to the necessary details. Thus, for instance, the informationally static system (D1) can be put into a formally arising form as

(D2) $\vartheta \models (\vartheta_w, \vartheta_c, \vartheta_{co}, \vartheta_b)$;
$\vartheta_w \models (\vartheta_{gv}, \vartheta_{tt})$;
$\vartheta_{gv} \models (\vartheta_{Jgv}, \vartheta_{Egv})$;
$\vartheta_{tt} \models (\vartheta_{Jtt}, \vartheta_{Ett})$

$$\text{or } \vartheta_{gv} \models (\vartheta_{Sgv}, \vartheta_{Egv});$$
$$\vartheta_{tt} \models (\vartheta_{Stt}, \vartheta_{Ett});$$
$$\vartheta_{c} \models (\vartheta_{cc}, \vartheta_{cd});$$
$$\vartheta_{co} \models (\vartheta_{Jco}, \vartheta_{Eco})$$
$$\text{or } \vartheta_{co} \models (\vartheta_{Sco}, \vartheta_{Eco});$$
$$\vartheta_{b} \models (\vartheta_{JE}, \vartheta_{EJ})$$
$$\text{or } \vartheta_{b} \models (\vartheta_{SE}, \vartheta_{ES})$$

System (D2) is a developing formal system which calls for the missing details, that is, for the decomposition possibilities concerning the data structure of the ED on the one side and the system-functional, developmental, and user-friendly understanding on the other side.

From this point on, understanding as a systematic informational approach can be put into the game of system development. As we have seen, hermeneutic and parallel-cyclic approaches of understanding can be applied simultaneously.

What is the built-in quality of understanding in a dictionary? How could it be developed out of basic, static, already listed informational formulas? A dictionary ϑ as a complex informational entity $\vartheta \models (\vartheta_w, \vartheta_c, \vartheta_{co}, \vartheta_b)$ possesses a quality (mode) of understanding by which its informational components are informed. Thus, it is possible to put into discourse two basic questions: (1) What is the internal understanding $\mathfrak{D}$ of the dictionary ϑ? (2) How can an electronic dictionary ϑ be used (understood) by a user υ through his/her understanding $\mathfrak{U}$? We see how the problem of understanding can become informationally interweaved from the one and the other side.

At the very beginning of the concept we have merely ϑ as an arising information, as the intention of its development, that is, of its design and implementation. A basic concept ϑ informs and is informed, that is, $\vartheta \Rightarrow (\vartheta \models; \models \vartheta)$. This basic formula can be developed, decomposed, particularized, universalized, etc. in several ways. It can be read in the following way: as soon as the intention of a dictionary ϑ informing is given, it is permitted to transit to the system of formulas $(\vartheta \models; \models \vartheta)$. Thus, the first consequence of this implicative informing is, for instance, $\vartheta \models (\vartheta_w, \vartheta_c, \vartheta_{co}, \vartheta_b)$, which from now on becomes the basis, although it possesses a static structure in which understanding of its components is undeveloped, that is, not expressed informationally yet. We conclude that understanding $\mathfrak{D}$ of ϑ is implicit and can be expressed through the development of the concept, by an arising system of formulas $\vartheta \models; \models \vartheta$ where the right $(\vartheta \models)$ and the left informational operands $(\models \vartheta)$ of operator $\models$ are coming into existence, respectively.

In principle, by understanding, informational processes are closed into loops of understanding. The concept of the so-called self-understanding of an informational entity α and understanding of α by the entity β through β's understanding can be the following. Informing of α as its implicit informational process can be marked by $\mathfrak{I}_\alpha$, where $\mathfrak{I}_\alpha$ marks the entire informing of α. Thus, the understanding component of α within its implicit informing $\mathfrak{I}_\alpha$ can be marked by $\mathfrak{U}$ yielding $\mathfrak{U} \subset \mathfrak{I}_\alpha$. Under these circumstances, the understanding part $\mathfrak{U}$ of α's informing $\mathfrak{I}_\alpha$ can be formalized by the system of formulas

$$\text{(D3)} \quad \mathfrak{U} \subset \mathfrak{I}_\alpha; (\alpha \models \mathfrak{U}) \models \alpha$$

Further, if entity β understands α by its understanding $\mathfrak{B}$ as a part of the entire β's informing $\mathfrak{I}_\beta$, then

$$\text{(D4)} \quad \mathfrak{U} \subset \mathfrak{I}_\alpha; (\alpha \models \mathfrak{U}) \models \alpha;$$
$$\mathfrak{B} \subset \mathfrak{I}_\beta; (\beta \models \mathfrak{B}) \models \beta;$$
$$((((\beta; \alpha \models \mathfrak{U}) \models \alpha) \models \mathfrak{B}) \models \beta$$

In this scheme, entity β considers the understanding $\mathfrak{U}$ of α within its own understanding $\mathfrak{B}$. Usually, the understanding of something as an informational process produces a particular information called the meaning μ of something. For instance, in case of self-understanding $\mathfrak{U}$ of α, we have the meaning $\mu_{\mathfrak{U}}(\alpha)$. In case of the so-called other-understanding, the meaning $\mu_{\mathfrak{B}}(\alpha)$ or even $\mu_{\mathfrak{B}}(\mu_{\mathfrak{U}}(\alpha))$, $\mu_{\mathfrak{B}}(\beta, \mu_{\mathfrak{U}}(\alpha))$, etc. can come into existence. In a discourse between entities α and β, system (D4) extends into

(D5) $\mathfrak{A} \subset \mathfrak{F}_\alpha$; $(\alpha \models \mathfrak{A}) \models \alpha$;
$\mathfrak{B} \subset \mathfrak{F}_\beta$; $(\beta \models \mathfrak{B}) \models \beta$;
$((((\alpha; \beta \models \mathfrak{B}) \models \beta) \models \mathfrak{A}) \models \alpha$;
$((((\beta; \alpha \models \mathfrak{A}) \models \alpha) \models \mathfrak{B}) \models \beta$

After this discussion, it is possible to answer the question concerning the understanding occurring among parallel dictionaries within an electronic dictionary. The basic formula $\vartheta \models (\vartheta_w, \vartheta_c, \vartheta_{co}, \vartheta_b)$ in (D2) and the subsequent formulas of (D2) give a firm orientation in which direction the system design of the system understanding should develop. Here, understanding concerns the necessary processing among dictionaries guarantying the achievement of goals of an ED, that is, to be functional, efficient, user-friendly, and innovative.

The decomposition of $\vartheta \models (\vartheta_w, \vartheta_c, \vartheta_{co}, \vartheta_b)$ can begin by the following steps:

(D6)

(1) $(\vartheta \models \vartheta_w, \vartheta_c, \vartheta_{co}, \vartheta_b) \Rightarrow$
$(\vartheta_w, \vartheta_c, \vartheta_{co}, \vartheta_b \models \vartheta_w, \vartheta_c, \vartheta_{co}, \vartheta_b)$;

(2) $((\vartheta_w \models \vartheta_c) \Rightarrow$
$(\omega \in \vartheta_w, \gamma \in \vartheta_c)) \Rightarrow$
$(\omega \models_{poi} \gamma; \gamma \models_{del} \mu(\omega))$;

(3) $(((\omega \models \mathfrak{W}) \models \gamma) \models \mathfrak{G}) \models \mu(\omega)$; ...

In formula (2) of (D6), the particularized operators $\models_{poi}$ and $\models_{del}$ are read as »points« (or »marks«) and »delivers«, respectively. Entities $\mathfrak{W}$ and $\mathfrak{G}$ in (3) mark to the word entity ω and to it corresponding concept γ belonging understandings, respectively, informing the meaning $\mu(\omega)$.

From the last system of formulas one can see the course of a dictionary development considering the so-called functions of understanding. In the next sections of the essay we shall point to some further informational details of an electronic dictionary disclosing the structure of some very concrete concepts. In these concepts, problems of monolingual, bilingual and multilingual dictionaries will be treated from the developmental, operational and implementing point of view. Further, a clear conceptual distinction among various concepts of dictionaries has to be developed, for instance, concerning formal models of data-structured (classically programmed), information structured (advanced conceptualized), and the so-called informational dictionaries. In this point, a question of developing tools for these different sorts of dictionaries can arise. Certainly, several questions will remain unanswered, but a new informational way for the development of the most sophisticated and advanced dictionaries will begin to become its specific trace.

References

(OED) *An Overview of the EDR Electronic Dictionary*, Technical Report TR-024, Japan Electronic Dictionary Research Institute, Tokyo (1990).

(NCD) *Webster's Ninth New Collegiate Dictionary*, Merriam-Webster Inc., Publishers, Springfield, Ma (1986)

(ITD) A.P. Železnikar, *An Informational Theory of Discourse I*, Informatica **13** (1989) *4*, 16-37.

(IIA) A.P. Železnikar, *An Introduction to Informational Algebra*, Informatica **14** (1990) *1*, 7-28.

(UI1) A.P. Železnikar, *Understanding as Information (Part One: A General Philosophy and Theory of Understanding as Information)*, Informatica **14** (1990) *3*, 8-30.

(UI2) A.P. Železnikar, *Understanding as Information (Part Two: A Formal Theory of Understanding as Information)*, Informatica **14** (1990) *4*, 5-30.

STRUCTURED OBJECT-ORIENTED SYSTEM DECOMPOSITION

INFORMATICA 1/91

Stevan Mrdalj
Eastern Michigan University
Ypsilanti, Michigan 48197, USA
Vladan Jovanović
University of Detroit
Detroit, Michigan 48221, USA

Keywords: object-oriented design, structured system design, object-oriented models

ABSTRACT: The aim of this paper is to present a structured method for object-oriented system design with special emphasis on discovering objects within the system and creating an object-oriented model of the system. First, a technique for decomposing an object into its components according to the actions required by the operations of that object is introduced. Second, system decomposition is presented as a coherent top-down process based on an object-oriented model. Lastly, reusability and extensibility of such an approach to system design are discussed.

REZIME: U ovom radu je predstavljen strukturan metod za objektno-orijentisano projektovanje sistema sa posebnim naglaskom na način na koji se objekti otkrivaju u sistemu i kako se kreira objektno-orijentisani model sistema. Prvo je dat opis tehnike za dekomponovanje objekata na komponente u zavisnosti od akcija neophodnih za realizaciju njihovih operacija. Nakon toga je predstavljen proces dekompozicije sistema kao koherentan proces od vrha nadole koji se bazira na upotrebi-objektno orijentisanog modela. Na kraju je diskutovana mogućnost ponovne upotrebe objekata i mogućnost nadgradnje objekata izmenom, odnosno dodavanjem operacija.

1. INTRODUCTION

The object-oriented paradigm has broader impact on system development than the traditional, functional or data oriented paradigms. The objective of object-oriented design is not only to create a model of the system, but to do so by reusing existing objects. Thus, object-oriented design requires more than just choosing objects and arranging them in class hierarchies. It needs a structured technique to: (1) avoid confusion in defining objects, (2) arrange objects into a system model, and (3) reuse already defined objects from the object library. Hence, here we propose a formalized approach for structured object-oriented system decomposition (SOOSD).

In considering the context of creating the object-oriented model of the system, most of the existing methods [13,1,3,8,5,15,16,2] are intuitive. They provide some informal rules for identifying the objects and their operations and can be categorized as direct decomposition methods. Also, they place little emphasis on the object's complexity and decomposition, and do not support different levels of either object or system abstractions.

On the contrary, SOOSD focuses on the discovery and arrangement of the objects of interest in the real world and creating an object-oriented model of reality. It allows us to develop an object-oriented model of a system as a leveled and incremental top-down decomposition process in which already existing objects can be reused. SOOSD is based on: (1) a specification using an object-oriented model and (2) a structured object decomposition technique.

2. OBJECT-ORIENTED MODEL

We use an object-oriented model called Abstract Object Model (AOM) as a formal specification tool to naturally and efficiently design the structure of a complex system. In AOM all things or concepts, in the designer's work environment, that are visible or otherwise tangible to the designer, are modeled as abstract objects. An abstract object encapsulates the problem space inside a set of pre-defined operations that manipulate and access that space. We refer the reader to [10] for a full description of AOM. Here we concentrate only on the following characteristics that are used in the object decomposition process:

1. AOM recognizes two kinds of objects: simple and composite. Simple objects occupy coherent space which cannot be further decomposed into meaningful objects, or one is not interested in their further decomposition. Conversely, composite objects are an *aggregation* of simple and/or other composite objects. Figure 1 illustrates a graphi-

cal representation of the object aggregation structure.

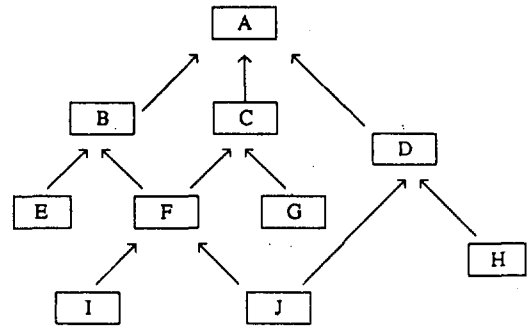


Figure 1: Object Aggregation Diagram.

2. The state of the composite object is a collection of its components' states. Thus, Figure 2 illustrates the assumption that the composite object state can be changed or accessed only by changing or accessing the state of at least one of its components.

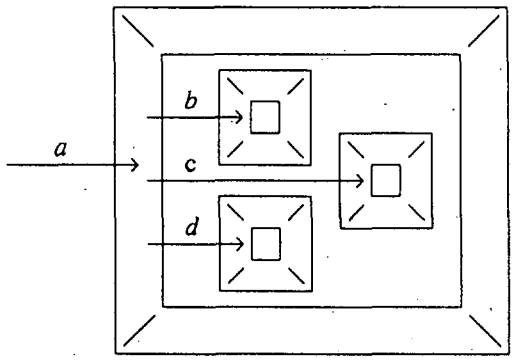


Figure 2: Access to the object space.

3. As a consequence of the previous characteristics, the operations of the composite objects are composed of the messages sent only to the components of that object. Figure 2 depicts that the response to message *a* are messages *b*, *c*, and *d*.

3. STEPWISE OBJECT DECOMPOSITION

We use the technique called Stepwise Object Decomposition (SOD) to discover "new" objects and to start the specification of such discovered lower level abstract objects [11]. SOD is based on the following two principles:

1. Actions required to construct operations of the given object determine components of that object.
2. The operations of an object are determined by the needs of the objects in which the given object is aggregated.

This is illustrated in Figure 3 where the actions required for the operations of object *X* determine that objects *Y* and *Z* become components of object *X*, while messages received by objects *Y*, *Z* and *W* determine their own operations.

The usage of these principles in the process of object decomposition is summarized into the following algorithm:

1. List all operations *O* of object *X*.
2. For each operation from *O*:
   2.1. List actions *A* required for that operation;
   2.2. For each action *a* from *A*:
      2.2.1. Associate *a* with corresponding object *Y*;
      2.2.2. Assign *Y* to the list *C* of *X*'s components;
      2.2.3. Assign *a* to *O* of *Y*.
3. For each object from *C* repeat steps 1 through 3.

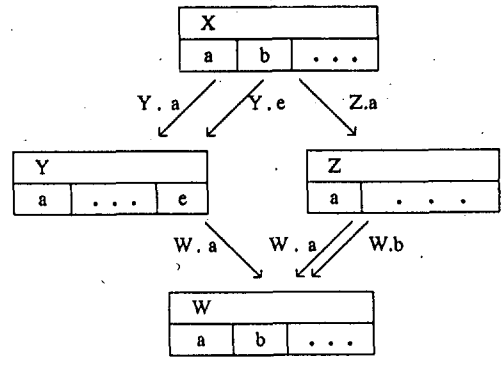


Figure 3: Message Flow Diagram.

The object that the designer chooses to decompose with this algorithm becomes the context of the decomposition process. Obviously, if the starting object is the system itself, the result will be the entire system specification.

4. SYSTEM DECOMPOSITION

AOM allows us to view any individual level of system abstraction as an abstract object, as well as to view the entire system as an abstract object on the highest level. For example, consider a system which maintains customers' requests for transportation and the assignment of taxis to customers as an abstract object called TAXI-DISPATCH.

Decomposition of the TAXI-DISPATCH object starts from the messages that it receives from its environment. Let us say that these messages define the following list of operations {AnswerCall, BuyCar, Hire}. These operations can be compared with basic functions of the TAXI-DISPATCH object that are required by its environment.

In order to discover components of the TAXI-DISPATCH object, one needs to define actions required to accomplish operations of the TAXI-DISPATCH object. Let us start from the AnswerCall operation for which the list of all required actions is {FindAvailable, ReceiveRequest, MonitorTransport}. Next, one associates all these actions with the objects which should be responsible to perform them. For example, by associating the ReceiveRequest action with the object which has to perform it, one

discovers the DISPATCHER object as one of the TAXI-DISPATCH's components. Of course, at the same time ReceiveRequest becomes the DISPATCHER's operation. After specification of all TAXI-DISPATCH's operations one may have its aggregation structure as it is displayed in Figure 4.

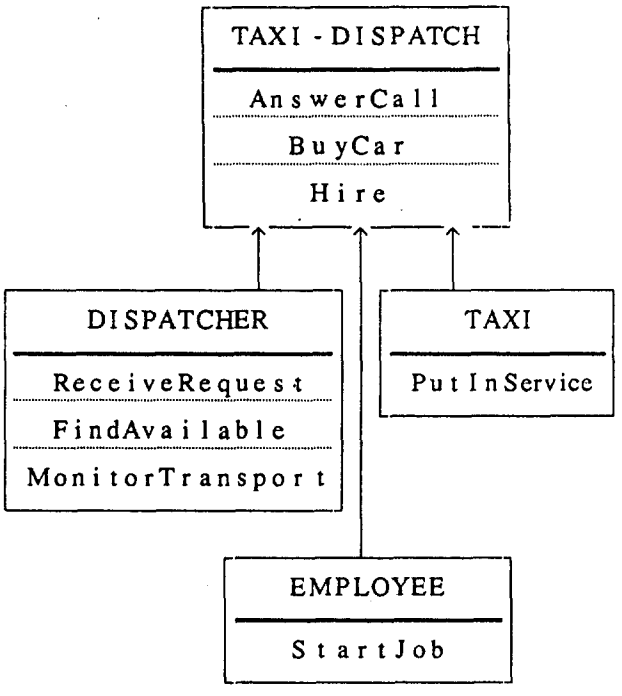


Figure 4: TAXI-DISPATCH's components.

Next, we decompose the system components into their sub-components. For example, let us decompose the DISPATCHER object with the following list of actions required for operation ReceiveRequest: {GiveName, GiveRoute, FindAvailable, Assign}. Now one has to assign again all these actions to the corresponding objects. In that way one discovers the CUSTOMER, TAXI, and DISPATCH objects and their operations. Just as before, one may keep decomposing components of the previously discovered composite objects until they are composed only of simple objects.

After specification of all TAXI-DISPATCH's components one may have the complete aggregation structure of the system as it is shown in Figure 5. Notice that the system has been simplified and its structure is partially presented so that the basic ideas of the decomposition process can be emphasized.

5. REUSABILITY AND EXTENSIBILITY

Within a framework of developing complex and long-lasting systems, we identify two objectives that SOOSD should provide support for: (1) Reusability - ability of objects to be reused, in whole or in parts, for the construction of an existing system or a new system; (2) Extensibility - changes of the objects to accommodate modifications of their requirements.

The simplest kind of reusability is the use of an object type as it already exists. For example, CUSTOMER object in Figure 5, gets its specification through the specification of DISPATCHER object. Later on, it is reused in the aggregation of the DISPATCH object. This kind of reusability is limited because object types can only be reused in the construction of the system if there is a need for exactly the same behavior as provided by that object type. Thus, it does not solve the reusability problem in general.

Very often object types need to be extended or modified to fit in a new aggregation. Object types

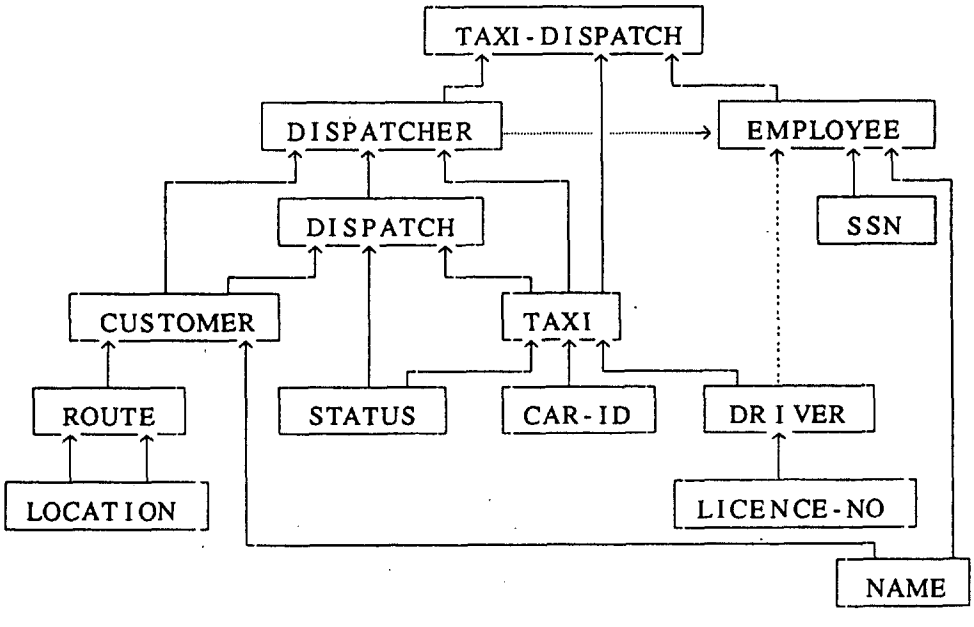


Figure 5: TAXI-DISPATCH's aggregation diagram.

can be extended and modified by means of incremental design and inheritance.

Incremental design. Since an object can be aggregated into more then one composite object, it can obtain its specification according to the needs of more than one object. For example, TAXI object from Figure 5, gets its initial specification through the specification of the TAXI-DISPATCH object. But then, by decomposing the DISPATCHER object, action FindAvailable is required from the TAXI object. That operation has to be added to the TAXI object prior to its aggregation into the DISPATCHER object. Later on during the decomposition of DISPATCH object, action AssignDrive is required from the TAXI object. That operation again has to be added to the previous specification of the TAXI object.

Inheritance. Another case of reusability is the usage of inheritance to define new object types out of existing ones by adding new components and operations. For example, EMPLOYEE object from Figure 5 gets its initial specification through the specification of the TAXI-DISPATCH object. But then one may discover the operation Drive for EMPLOYEE object which is required by TAXI object. Instead of adding that operation to the EMPLOYEE object, one defines a new object DRIVER as a subtype of the EMPLOYEE object shown in Figure 5 by the dashed line. The newly required operation Drive can now be attached to the DRIVER object as its specialized behavior. At this point we may also reconsider the DISPATCHER object and make it a subtype of the EMPLOYEE object. In this case the DISPATCHER and DRIVER objects inherit EMPLOYEE's components SSN and NAME, as well as its operation StartJob.

Conceptually lower-level objects could be the subtypes with specialized functionalities or even the special cases of the more abstract objects. An object library consisting of a set of object type hierarchies in the background may significantly reduce effort in the decomposition process.

As can be seen, an open design architecture with open-ended sets of extensions to an existing design is promoted. This is important for long-lasting systems because a system's functionality changes over time. However, the types of objects from which the system is composed will probably be more or less the same over time. The changes are most likely to occur when an object gets a new operation, an existing operation changes behavior, or a new object arises. At that time, it will only be necessary to add new operations, modify existing ones, or aggregate existing objects into new ones. Therefore, SOOSD enables maintenance of a system model as a process of reusing objects across time, and not only across applications.

6. RELATION TO OTHER WORK

There are a few works which explore the merging of structured system analysis and design techniques [4,17] and object-oriented design [12,14]. But none of them view object-oriented system design as a leveled process that starts from an entire system as an object or from any high complexity object and that decomposes them into lower level objects as it is in our case. This is the essential difference between the existing object-oriented design methods and our object-oriented top-down system decomposition. Although we use a top-down decomposition approach, the solution to system decomposition is a digraph that combines one object aggregation diagram and many object inheritance diagrams.

We also concentrate on making object types reusable through aggregation, incremental design and inheritance, three powerful mechanisms for sharing specification and promoting their reuse. That is what makes our decomposition approach distinct from most object-oriented designs which use only inheritance as a tool for reusability [9].

Finally, SOOSD is build upon the abstraction mechanisms (such as encapsulation, classification, aggregation, and generalization/specialization) from semantic data models [7] and object-oriented programming languages.

6. CONCLUSION

Structured object-oriented system decomposition represents a refinement of the structured system analysis and design using object-oriented principles. It allows us to start system decomposition from the system-object and work top-down to the complete design solution using the objects' operations to discover their components.

We have now had about three years of successful experience in using SOOSD to design and implement vastly large systems. We have found SOOSD to be extremely useful in the initial design of the systems. That is because it provides system decomposition according to currently existing object operations. But as with any software project, we have done as much re-design as design. In such cases, SOOSD continues to play an important role in reusing object types through aggregation and inheritance.

7. REFERENCES

[1] G. Booch "Object-Oriented Development", *IEEE Trans. on Software Eng.*, Vol.SE-12, No.2, 1986, pp. 211-221.

[2] P. Coad and E. Yourdon, *Object-Oriented Analysis*, Prentice Hall, 1990

[3] W. Cunningham and K. Beck "A Diagram for Object-Oriented Programs", *Proc. of the OOPSLA'86*, Sep. 29 - Oct. 2, 1986, pp. 361-367.

[4] T. DeMarco *Structured Analysis and System Specification*, Prentice-Hall, 1979.

[5] R.M. Ladden "A Survey of Issues to be Considered in the Development of an Object-Oriented Development Methodology for ADA", *Software Engineering Notes*, Vol.13, No.3, 1988, pp. 24-31.

[6] P. Lyngbaek and W. Kent "A Data Modeling Methodology for the Design and Implementation of Information Systems", *Proc. of the Inter. Workshop on Object-Oriented Database Systems*, September, 1986, pp. 6-17.

[7] R. King and D. McLeod "Semantic Data Models", in S.B. Yao (ed.) *Principles of Database Design*, Vol.I, Prentice-Hall, 1985.

[8] B. Meyer "Reusability: The Case for Object-Oriented Design", *IEEE Software*, March 1987, pp. 50-64.

[9] J. Micallef "Encapsulation, Reusability, and Extensibility in Object-Oriented Programming Languages", *Journal of Object-Oriented Programming*, Vol.1, No.1, April/May 1988, pp. 12-35.

[10] S. Mrdalj "Abstract Object Model: Data Model for Object-Oriented Information System Design", *Informatica*, Vol.14, No.2, April 1990, pp. 1-11.

[11] S. Mrdalj "Stepwise Object-Oriented System Design", *Proc. of the IEEE International Conf. on Computer Systems and Software Engineering - CompEuro'90*, May 7 - 9, 1990, pp. 520-521.

[12] E. Seidewitz and W. Stark "Towards a General Object-Oriented Software Development Methodology," *SIGAda Ada Letters*, July/Aug. 1987, pp. 54-67.

[13] R.F. Sincovec and R.S. Wiener "Modular Software Construction and Object-Oriented Design Using Ada," *J. of Pascal, Ada & Modula-2*, March/April 1984, pp. 29-34.

[14] P.T. Ward, "How to Integrate Object Orientation with Structured Analysis and Design," *IEEE Software*, March 1989, pp. 74-82.

[15] A.I. Wasserman, P,A. Pircher and R.J. Muller "An Object-Oriented Structured Design Method for Code Generation", *ACM SIGSOFT*, Vol.14, No.1, January 1989, pp. 32-55.

[16] R. Wirfs-Brock and B. Wilkerson, "Object-Oriented Design: A Responsibility-Driven Approach," *Proc. of the OOPSLA'89*, October 1-6, 1989, pp. 71-75.

[17] E. Yourdon and L. Constantine *Structured Design: Fundamentals of a Discipline of Computer Program Design*, Prentice-Hall, 1975.

NEURAL NETS: SURVEY AND APPLICATION TO WAVEFORM PROCESSING

INFORMATICA 1/91

Ivan Bruha
Dept. Computer Science and Systems,
McMaster University, Hamilton, Ont.
Canada L8S4K1

Keywords: neural nets, waveform processing

Abstract

This paper surveys Artificial Neural Nets as essential tools for pattern recognition. The author concentrates on the characteristics of well-known types of currently exploited neural nets and implementation of a multilayer perceptron. Aspects, specification, and comparison of various types of neural nets will be presented, as well.

Furthermore, a decision-supporting system for neurological diagnosis will be described. The actual input to our system is an evoked potential waveform. It is analyzed by a syntax pattern recognition algorithm based on a regular attributed grammar. Its semantic functions return a list of numerical features. The second step of the waveform processing includes a two-layer perceptron which processes the above list of numerical features. The rules of thumb for optimal adjustment of perceptron's parameters will be discussed, too.

1. Pattern Recognition: A Survey

Pattern recognition is one of the oldest and most widely investigated areas of artificial intelligence (AI). Although some researchers do not recognize pattern recognition as a part of AI, there are some others (including the author of the paper) who accept Minsky's definition of AI as 'the science of making machines do things that would require intelligence if done by men' and consider artificial intelligence as a 'club' of several theoretical and experimental disciplines, including pattern recognition. Pattern recognition as a discipline has been thoroughly described, analyzed, and surveyed in a great number of books and monographs. For examples the reader might look at e.g. [Du73], [Fu82], [Tou74], [Wa85], or [Bru80]. Nevertheless, in this section we present the necessary terminology.

The principal function of a pattern recognition system is to make a decision concerning the class membership of the patterns received as input. *Patterns* are facts, observable statements, situations, events, or objects of any type that are to be recognized. The patterns are grouped into two or more sets, so-called *classes*, according to the given problem. Any such system (usually a computer) cannot, of course, observe real-world objects, since it is only able to read in the data of its world, i.e. numbers and symbols. Therefore, before the actual classification, we have to represent patterns by a suitable *formal description*. There are in fact two distinct possibilities for pattern representation:

(a) numerical (statistical) description: a pattern is represented by a sequence of numbers, called features;
(b) symbolic description: a pattern is described either by a string of (terminal) symbols, or by more general structures such as relational structures, semantic nets, frames, formulas of predicate calculus, etc.

Perceptrons and other neural nets we discuss in this paper have strictly numerical character; therefore, we will focus on the numerical approach only. A pattern is represented by so-called *feature vector*

$$\mathbf{x} = [\, x_1 , x_2 , \ldots , x_N \,]$$

of numbers, called *features*. The number of features is usually fixed, and therefore all feature vectors form an N-dimensional space, called the *feature space*.

Let a pattern recognition problem comprise R classes; the r-th class will be designated by the symbol z_r , r = 1,...,R . Then, a pattern recognition system with numerical representation of patterns (called *numerical classifier*) is a device with N inputs (one for each feature) and one output, which classifies an input feature vector $\mathbf{x}$ to one of the given classes, i.e. it yields one of the symbols z_r . The relation between its input and output is called *decision rule* and can be written as the following function

$$z = d(\mathbf{x})$$

The feature space is divided by the decision rule into R disjoint subsets R_r (called *decision regions*) where R_r contains the features $\mathbf{x}$ such that $z_r = d(\mathbf{x})$. The boundaries between decision regions are called *decision boundaries*. Their adjustment is the principle problem of a classifier design.

There are several ways of describing decision boundaries. Among them, a most general approach uses so-called *discriminant* (or *decision*) *functions*. The R discriminant functions $g_r(\mathbf{x})$ have to be selected so that

$$g_s(\mathbf{x}) > g_r(\mathbf{x}) \quad , \quad r = 1,...,R \; , \; r \neq s$$

for all $\mathbf{x} \in R_s$. Consequently, the decision boundary between adjacent regions R_r and R_s is defined by the equation

$$g_r(\mathbf{x}) - g_s(\mathbf{x}) = 0$$

The decision rule for a classifier with discriminant functions thus has the form: for a given $\mathbf{x}$ compute all $g_r(\mathbf{x})$, r = 1,...,R and classify $\mathbf{x}$ to the class z_s for which

$$g_s(\mathbf{x}) = \max_r g_r(\mathbf{x}) \qquad (1)$$

see Fig. 1.

The most common type of a numerical classifier is the one with linear discriminant functions:

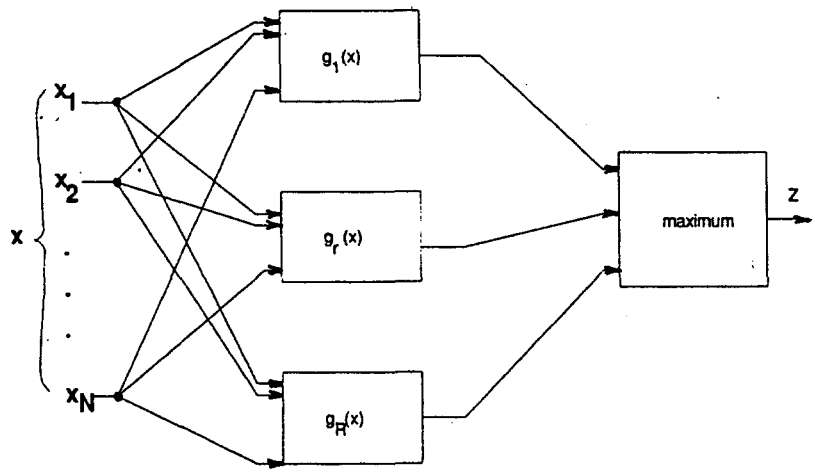


Fig. 1. Classifier with discriminant functions

$$g_r(x) = \sum_{n=0}^{N} w_{nr} x_n \quad , \quad r = 1,...,R \qquad (2)$$

where w_{nr} , $n = 0,1,...,N$, $r = 1,...,R$, are so-called *weights*, and $x_0 = 1$. This latter condition allows w_{0r} to be interpreted as a *threshold* of the discriminant function. The structure of a linear classifier is illustrated in Fig. 2. Decision boundaries formed by a linear classifier are parts of hyperplanes (see Fig. 3).

The problem of finding decision boundaries is thus reduced to that of finding optimal values for all the weights of the linear classifier. One possible and often used way of a classifier adjustment is to utilize a *learning (training)* process. In such a case, a teacher (a designer of the classifier) has to collect a reasonable set (*training set*) of typical, representative examples (*training patterns*) including the correct class membership of each training pattern (*desired class*). During the learning process, training patterns with their desired classes are presented (usually many times) by the teacher to the classifier which performs as a student, i.e. it modifies (utilizing the information involved in the training set) its weights according to an appropriate learning algorithm. Among a large collection of learning algorithms for linear classifiers, we introduce the simplest one, called *Rosenblatt's algorithm* (also known as *perceptron* algorithm, *fixed-increment*, *delta*, or *LMSE* algorithm):

1. Initialize weight vectors

   $w_r = [\, w_{0r} , w_{1r} , \ldots , w_{Nr} \,]$, $r = 1,...,R$

   arbitrarily.

2. For each training pattern $x = [\, x_0, x_1 , ..., x_N \,]$ (where we added $x_0 = 1$ for a convenience) with its desired class $Z = z_s$ do:

   2.1. Compute $g_r(x)$, $r = 1,...,R$ according to (2).

   2.2. If the equation (1) is satisfied, i.e. the pattern x is correctly classified, do nothing. However, if

   $$g_s(x) < g_r(x) \quad \text{for some } r \qquad (3)$$

   then change the weights vectors as follows (c is a positive constant):

   w_s += c x

   w_r -= c x for all r satisfying (3)

   (where A += B means: add B to A ; similarly -= relates to a decrement).

3. If the weights were modified for at least one training pattern, return to the step 2, i.e. repeat the training for the entire training set. If not, save the weights and terminate the learning process.

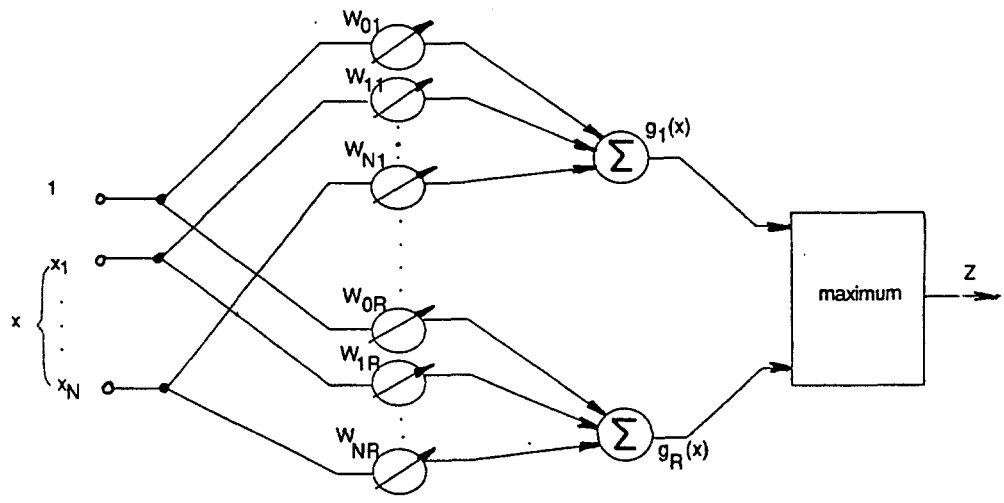


Fig. 2. Linear classifier

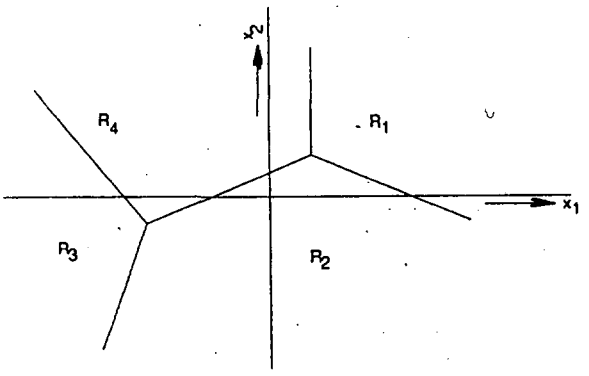


Fig. 3. Decision boundaries of a linear classifier

A simple model of a learning linear classifier was developed by Widrow and Hoff [Wid60]. Their *adaline* (adaptive linear neuron) is able to recognize $R = 2$ classes only (using a single discriminant function) where features can be only equal to +1 or -1, but otherwise its structure and learning is equivalent to those presented above. Adaline has found many useful application in technical areas.

2. Rosenblatt's Perceptron

Development, investigation, and theory of neural nets is not a new discipline at all. The first attempts to model the behaviour of biological nerve cells were realized in 1940's. Their results encouraged the foundation of cybernetics, a discipline (or a 'club' of disciplines), which attempted to combine concepts from biology, psychology, engineering, and mathematics. Nevertheless, more serious research and experimentation with neural nets started in the 1960's with the publication of Rosenblatt's book [Ros61].

Rosenblatt developed a cognitive model of the brain, which was called the *perceptron*. Its structure is shown in Fig. 4. It is very similar to a linear classifier (Fig. 2); however, the features are not input directly to its weights, but instead are preprocessed by a so-called Φ-processor which is represented by M functions $\Phi_m(x) = \Phi_m(x_1,...,x_N)$, $m = 1,...,M$. The Φ-processor transforms the original features to the new ones x'_m , $m = 1,...,M$, which are then processed by a linear classifier. Hence, the discriminant functions of Rosenblatt's perceptron are

$$g_r(x) = w_{0r} + w_{1r}\,\Phi_1(x) + ... + w_{Mr}\,\Phi_M(x) \quad , r = 1,...,R \qquad (4)$$

Consequently, the decision boundaries of the original feature space need not be just hyperplanes but may consist of more general shapes that are transformed by the Φ-processor to linear ones. Another difference between the perceptron and a linear classifier is that both original and new features of the perceptron can only equal to 0 or 1 . Rosenblatt's algorithm is used for the training.

The functions of the Φ-processor can be of any shape, but the following cases are the most interesting ones:

- The Φ-processor consists of quadratic functions; thus decision boundaries are quadratic surfaces.
- The functions are boolean ones; the case with randomly arranged connections (bonds) and boolean gates was extensively studied by Rosenblatt's group since they had expected that this would be the best model of the brain.
- If the Φ-processor is an identity (or is eliminated) we get the 'standard' linear classifier or the Widrow's adaline.

The perceptron is a technical model for the following hypothesis of a nerve system's performance. This hypothesis states that the nerve system of a living organism is more less chaotically (randomly) organized when it arises. Its organization takes place during the life of the organism as a consequence of its adaptation to its environment and learning. The system creates suitable bonds (connections) within its Φ-processor and modifies its weights. As a metaphor one could say that a living organism is a general computer whose program is formed during its life. After the program has been created (the system has learned), a negligible amount of computer hardware is utilized; this is the cost of the versatility of the computer.

Perceptrons were applied to solve many problems in many technical disciplines. In some cases they were 100% successful, in others, however, they did not yield satisfactory results. The fundamental disadvantage of a perceptron consists in its low ability to generalize acquired knowledge; that is the principle difference between perceptrons and living organisms. For instance, a perceptron trained to recognize squares and circles at a certain place on its input retinal matrix, is not able to satisfactorily recognize the same squares and circles situated at another place on the matrix. Precise theoretical investigation of perceptron's limitations were presented by Minsky and Papert [Min69]. Analysis of the perceptron's behaviour revealed that the above hypothesis is not quite correct. Some bonds already seem to be determined when the organism arises, i.e. in our metaphorical example, the designer of a computer does not propose only its hardware, but also some parts of its software as well. In spite of these limitations, the perceptron became one of the fundamental models of neural nets, has had many technical applications, and, at present, has successful successors.

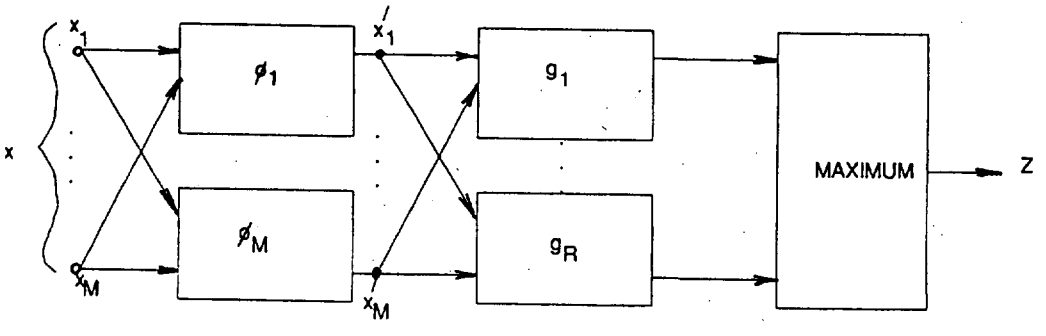


Fig. 4. Perceptron. g_1 to g_R are linear discriminant functions

3. Neural Nets

3.1. Paradigm of Connectionism

Minsky and Papert's criticism of perceptrons strongly diminished interest in their investigation, and consequently research in the field of perceptrons, their applications, and generalization was neglected for many years. Moreover, research activities in 1970's were focused on symbolic approaches and knowledge-intensive modelling. However, a few research centres continued to investigate the neural (or connectionist) approach and parallel processing systems, especially in computer vision; see e.g. Hough, Duda and Hart [Du72], Grossberg [Gro78], Anderson [And72], Kohonen [Koh77], and others (more references are in [Lip87], [Mat87], [Om87]).

After more than a decade of intensive research, however, the apparent limitations of the knowledge-based and symbolic approaches to artificial intelligence have been realized, returning the interest of AI researchers to neural net modelling. Consider just the following. The standard serial von Neumann computers are excellent in multiplication; they can multiply two large numbers in a fragment of one microsecond, but - even in late 80's - any computer with the best available knowledge base and the best available algorithm needs hours to recognize objects in a picture. That is evidently one of the reasons for the recent explosion of interest in parallel distributed processing and neural network models. Neural nets have exhibited promising results for a number of problems such as pattern recognition, speech processing, computer vision, association etc. (see e.g. Kohonen [Koh84], Sejnowski and Rosenberg [Sej86], Hopfield and Tank [Hop86], Rummelhart and McClelland [Ru86a], Hinton and Sejnowski [Hin86], Carpenter and Grossberg [Car86], etc.). A fair survey of types of neural nets and related problems can be found in [Gro88], [Lip87], [Sim90].

The paradigm of connectionism can be specified as follows: *Intelligence is an emergent property of a large parallel net of simple uniform nodes*. Indeed, although there are various types of neural nets, they all attempt to get fast and perfect behaviour by massive interconnections among their elementary units (nodes), and by exploring promising cases (hypotheses) simultaneously, in parallel. An elementary unit (node) usually has a large number of inputs, either from the environment that is to be processed or from the outputs of other nodes. The input signals to a unit are usually weighted and their sum is processed by a nonlinearity whose output is either a binary number (0 or 1) or a real number in a certain interval. The weights of nodes are usually adjustable by a learning (training) algorithm.

In the following, we will discuss all relevant aspects of neural nets and their specifications. We will then survey a few famous types of neural nets, and focus on the multilayer perceptron, one of the best models for the pattern recognition purposes (particularly, it is the best model for our task of doing waveform analysis).

3.2. Aspects of Neural Nets

Hecht-Nielson [He88] defines a neural network as a parallel, distributed information processing structure consisting of processing elements interconnected together with connections; each processing element has a *local* memory, a single output, and carries out *localized* information processing operations. Another definition can be found in [Sim90]: a neural net is a nonlinear directed graph with weighted edges that is able to *store* patterns by changing the edge weights, and is able to *recall* patterns from incomplete and unknown data.

Particularly, the latter definition reflects not only the internal structure of a neural net, but also the procedures (actions) that are carried out by the net. Now, following the paradigm of connectionism and the above definitions, we can observe the following aspects of any artificial neural net.

(1) Set of processing elements (neurons, units, nodes)

A neural net consists of a large numbers of simple elements (neurons, units, nodes) of the same type. Each neuron n has its *activity (state, output)* x_n . It is either a real number, usually from the close interval $< 0; 1>$, or an integer from the set $\{ 0,1,...,K \}$, or a binary number 0, 1, or +1, -1 .

(2) Topology of neural nets

Neurons are connected within a net by means of a large number of connections (synapses). Each connection is accompanied by a certain strength or weight. Therefore, the topology of a neural network is characterized by connections, their weights, and so-called interconnection schemes. We will briefly discuss each of these factors.

(a) Connections (synapses). Neurons of a net are connected together by oriented synapses. We can observe a neural net as a graph with oriented edges. There are several types of connections (graphs), among them the most typical ones are:

- total connection: each node is connected with all nodes;
- uniform local connection: each neuron is connected with its neighbours only;
- layered networks: neurons are grouped into layers which are ordered; neurons can be connected to neurons of the same layer, those of the next 'higher' layer, or any 'higher' layer, or 'lower' layer etc.; some of the possible layered types are on Fig. 5.

(b) Weights (connection strengths). If the neuron m is connected to the neuron n , then the corresponding connection (synapsis) has a certain strength or weight w_{mn} . A weight is usually characterised by a real number from a certain interval. All weights of a neural net form a matrix $w = [w_{mn}]$ which is called *(long-term) memory* of the net.

Weights can be either positive numbers (in such a case the connection is called *excitatory*), or negative (*inhibitory* connection). If a weight has zero value, then we have to distinguish either *static* zero (there exists no connection between the two nodes at all) or *dynamic* zero (the initial zero value of the weight has not been modified yet).

If $w_{mn} = w_{nm}$ for all connections, then the neural net is called symmetrical. If $w_{mn} * w_{nm} = 0$ for all connections, then we have one-way connection network. Otherwise, the net is of a general asymmetrical type.

(c) Interconnection schemes. We distinguish feedforward and feedback schemes. If information (signals) flows in one direction only, the net exhibits so-called *feedforward* scheme. If information flows in either direction and/or information flows recursively the net has the *feedback* scheme. Most feedback systems exploit recursive information flow with the following *stopping condition*: the information flows until the output of the network ceases to change.

(3) Environment

A neural net is working in a certain environment, processes environmental stimuli and returns its responses (output signals). The nodes which are directly connected to the environment (i.e. they process the environmental inputs, stimuli) are called *input* nodes. The nodes whose outputs (activities) are returned to the environment are called *output* nodes. Other nodes (i.e. those whose inputs and/or output are not directly connected to the environment) are called *hidden* ones; see Fig. 6.

Consequently, we can characterize a neural net as a system with the mapping $\mathbf{b} = S(\mathbf{a})$ where $\mathbf{a}$ is a vector of all environmental inputs, $\mathbf{b}$ a vector of outputs (activities) of all output nodes (i.e. the response of the neural system to the given environmental input).

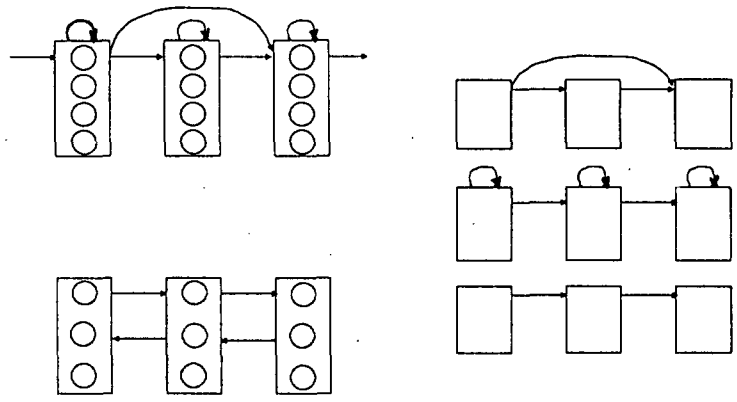


Fig. 5. Layered networks

(4) Time

A neural network can work either in continuous or discrete time. The neural systems we will present here are working in discrete time only. However, most of them have corresponding continuous variant.

(5) Rule of activity

A detailed structure of a node (neuron) is on Fig. 7. The neuron n has N inputs $a_1, a_2, \ldots, a_N$, which are either outputs of some neurons or environmental inputs (stimuli). All inputs are weighted by weights $w_{1n}, w_{2n}, \ldots, w_{Nn}$, summed, and led through a nonlinearity function (or threshold function) f. The common types of the threshold function are also shown on Fig. 7. A neuron usually has a threshold which symbolizes the level of the summed input signals above which the neuron is 'activated'. To simplify the formula for the neuron's output we consider its threshold as a weight w_{0n} and formally introduce the 0-th input a_0 identically equal to 1. Hence, the output (state, activity) of the neuron n is

$$x_n = f\left(\sum_{m=0}^{N} w_{mn}\, a_m\right)$$

To incorporate discrete time we should write rather

$$x_n(t+1) = f\left(\sum_{m} w_{mn}\, a_m(t)\right)$$

which indicates that the input signals at time t form the output for time t+1.

The neuron can process its own output x_n as one of its inputs, i.e. a feedback loop can be incorporated. Fig. 7 depicts this situation by means of the dotted connection with the weight w_{nn}.

(6) Learning

Learning is characterized by any change of neural net's memory w (i.e. matrix of all its weights). Methods of learning can be classified along several different dimensions. However, the presence or absence of a teacher seems to be a most important attribute. We will thus distinguish two types of learning for neural systems: supervised learning (with a teacher) and unsupervised one (without a teacher).

Supervised learning consists in that the teacher supplies so-called training set of typical exemplars (representatives, prototypes, etalons) $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_K$ with the desired output behaviour of the net. One of the simplest, but commonly used method of supervised learning is so-called *error-correction learning* (or *Widrow-Hoff*, or *delta*). A change of the weight w_{mn} between the neuron m and n is done by the formula

$$\Delta\, w_{mn} = \eta\, x_m\, (X_n - x_n)$$

where x_m is the input from the neuron m, x_n is the actual output of the neuron n, X_n is the desired output of the neuron n, provided by the teacher, and η is a parameter, called learning rate. The Rosenblatt's algorithm, discussed in Section 1, is a generalization of the above.

The simplest method of *unsupervised learning* (without a teacher) is *Hebbian learning*. A change of the weight w_{mn} between the neuron m and n is done by the formula

$$\Delta\, w_{mn} = \eta\, x_m\, x_n$$

When studying learning algorithms we should also distinguish competitive and cooperative learning. The *competitive* learning uses

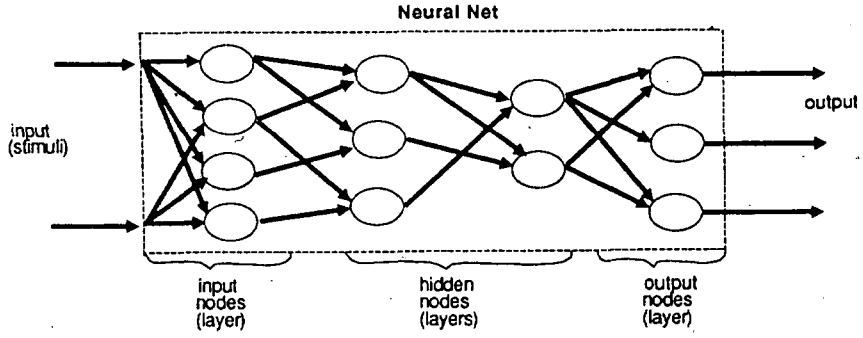


Fig. 6. A neural net in an environment

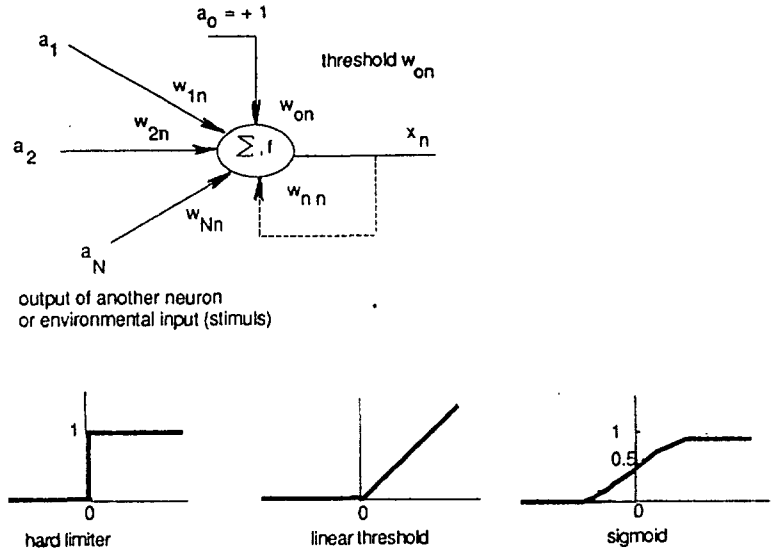


Fig. 7. Rule of activity and threshold functions

neighbour-inhibiting strategy, i.e. activities of all neighbour neurons of a given neuron are lessened, and the activity of the given neuron is increased. One simple competitive learning method is called *winner-take-all*; we will discuss it in the Section 3.5. The *cooperative* learning, on the other hand, uses neighbour-exciting strategy, i.e. activities of all neighbours including the given neuron are reinforced.

(7) Mapping mechanism

A neural network can be observed as a certain type of an associative memory. We distinguish two types of associativity:

- *autoassociative* neural net: its memory w stores input patterns (environmental stimuli) a_1, a_2, ..., a_K ;
- *heteroassociative* neural net: its memory w stores input as well as output pattern pairs (a_1, b_1), ..., (a_K, b_K) .

(8) Rule for changing topology

This optional attribute grants a neural net to modify its topology, which is usually done by adding new connections between existing nodes, or even by creating a new node and its new connections to existing nodes. Carpenter-Grossberg classifier (Section 3.6) is one example of the neural net which is able to modify its topology.

3.3. Specification of neural nets

We observe from the above aspects of neural nets that there exist two (or three) interacting dynamic regimes: activities (outputs) of nodes are changed, the weights (connection strengths) are adjusted, and, optionally, the net's topology is modified. Since the weight adjustments and topology modifications are usually much slower processes than the activity changes, we can specify the above processes separately.

(1) Active regime is specified by

- the space X of state vectors $\mathbf{x} = [\, x_1, x_2, ..., x_N \,]$, where x_n is the state (activity, output) of the neuron n , n=1,2,...,N ;
- the activity function

  $$\mathbf{x}(t+1) = \mathbf{F}(\mathbf{x}(t), \mathbf{w}(t))$$

  which determines the state vector for the time t+1 if the state vector and the net's memory are given for the time t ;
- the energy function $E(\mathbf{x})$.

Its motion consists in that, for given external (environmental) input **a**, the state vector is initialized to

$\mathbf{x}(0) = \mathbf{a}$ for input nodes,
$\mathbf{x}(0) = \mathbf{0}$ (or a random vector), otherwise

and the net is trying to find a stable state $\mathbf{x}(t^*)$ for which the energy function E reaches its minimum (so-called *stability* of the net). The over-all output (i.e. the response of the neural network to the given environmental input) is

$\mathbf{b} = \mathbf{x}(t^*)$ for output nodes.

A typical active regime is called *nearest-neighbour* one: for a given unknown input pattern **a** , it finds the stored input pattern (exemplar, etalon) $\mathbf{a}_k$ which most closely matches **a** , see Section 3.4 for details.

(2) Training (learning) regime is specified by

- the space W of the weight matrices $\mathbf{w} = [\, w_{mn} \,]$;
- the training function (learning algorithm)

  $$\Delta\, \mathbf{w} = \mathbf{G}(\mathbf{w},\mathbf{a})$$

  which determines how the weight matrix w is changed according to the training example a ;
- the error function $J(\mathbf{w})$.

Its motion consists in that, for a given training set of training examples $\mathbf{a}_1$, $\mathbf{a}_2$,..., $\mathbf{a}_K$ and $\mathbf{w}(0) = \mathbf{0}$ (or random), it finds an optimal weight matrix $\mathbf{w}^*$ for which the error function reaches its minimum (so-called *convergence* of learning).

(3) Optional *configuration regime* could be considered as a special type of the training regime. If the modification of the weight matrix is not efficient for some reasons then the configuration (topology) of the network could be optionally changed: either new connections are added or new nodes are incorporated to the existing network. Specification of such a regime is not, however, so uniform as that of the other regimes. We have to specify first of all the conditions under which the training mode has to be replaced by the configuration regime. Second, rules for adding new connections and/or nodes and their connections have to be specified as functions of existing topology and training examples.

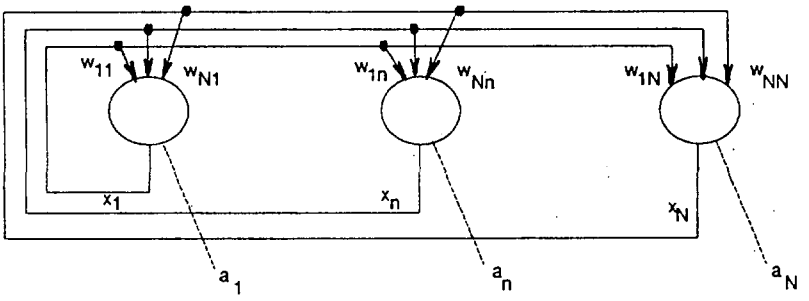


Fig. 8. Hopfield net

3.4. Hopfield net

The *Hopfield net* [Hop82] (see Fig. 8) is a single-layer, symmetric neural network, with the feedback scheme, working in discrete time. It uses the nearest neighbour active regime and Hebbian algorithm for (unsupervised) learning. It is a simple but very efficient network exhibiting the autoassociative mapping mechanism. Therefore, it is also called *autocorrelator*, or just *autoassociative memory*. Its input is binary only.

Training (learning) of the Hopfield net looks as follows. Given R training examples $\mathbf{a}_r = [\, a_{r,1}, \ldots, a_{r,N} \,]$, r=1,...,R, which are to be stored in the memory of the network, the weight from the node m to n is simply given by

$$w_{mn} = \sum_{r=1}^{R} a_{r,m}\, a_{r,n} \quad , \qquad m,n = 1,\ldots,N \ , \ r = 1,\ldots,R$$

However, if we prefer sequential formulas for (incremental) learning we can simply derive Hebbian learning algorithm:

$$w_{mn}(0) = 0$$
$$\Delta w_{mn} = a_{r,m}\, a_{r,n}$$

for the r-th example on the input, r=1,...,R

In the active regime, an unknown (new) pattern $\mathbf{a} = [a_1,\ldots,a_N]$ is imposed onto the net's input at time 0 and the node states (outputs) are changed in the feedback scheme according to the following formula (Fig. 8):

$$\mathbf{x}(0) = \mathbf{a}$$

$$x_n(t+1) = f\left(\sum_{m=1}^{N} w_{mn}\, x_m(t) \right) \quad , \qquad n = 1,\ldots,N$$

until the states (outputs) no longer change on successive iterations (i.e. stopping condition of the active regime is reached). The pattern $\mathbf{x}(t^*)$ specified by the node outputs after the stability is achieved represents the exemplar $\mathbf{a}_r$ which best matches the unknown pattern $\mathbf{x}$ (i.e. the nearest-neighbour active regime).

The energy function $E(\mathbf{x})$ used for the Hopfield net is the Lyapunov function

$$E(\mathbf{x}) = - \sum_{m,n} w_{mn}\, x_m\, x_n$$

It reflects a disharmony (chaos) of the system. Learning can be portrayed as a decrease of the energy function, i.e. 'relaxation' of the system. Stability and capacity (i.e. number of exemplars stored in the network) is discussed e.g. in [Sim90]. [Lip87] states that $R < 0.15\ N$. Here we just mention that local minima of the energy function are either the right ones (i.e. correspond to stored exemplars), or spurious ones, so-called phantoms. If a phantom φ is identified then it can be 'unlearned' by

$$\Delta\, w_{mn} = - b\, \varphi_m\, \varphi_n$$

where b is a positive constant.

3.5. Hamming net

The *Hamming net* (Fig. 9) is a two-layer neural network, with the feedback scheme, working in discrete time. Its input is binary only. Learning regime of the Hamming net is very simple, because it is in fact done by these assignment statements for the weights of the first layer:

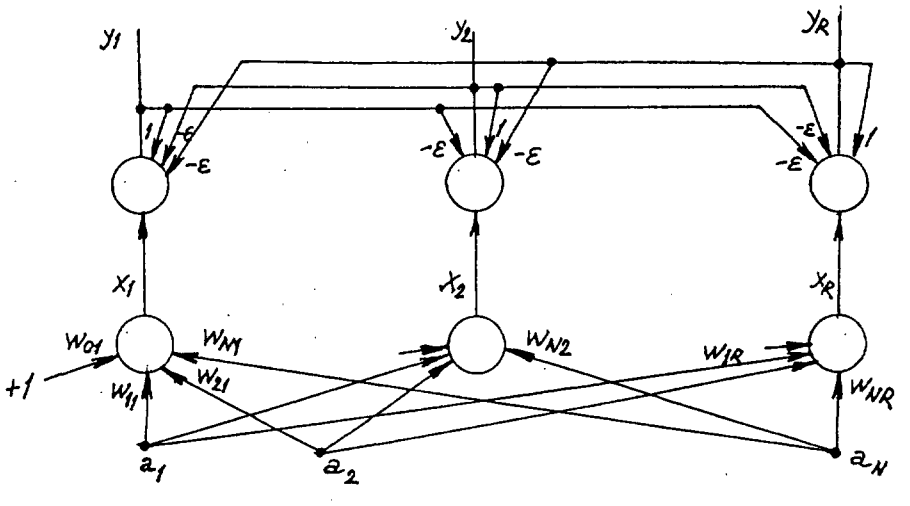


Fig. 9. Hamming net

$$w_{rn} = \frac{1}{2} a_{r,n} \quad , \quad r = 1,...,R \ , \ n = 1,...,N$$

$$w_{0r} = \frac{1}{2} N$$

where $\mathbf{a}_r = [\ a_{r,1}\ ,...,\ a_{r,N}\ ]$, r=1,...,R, are training examples. The second layer has predefined weights, see Fig. 9.

In the active regime, an unknown pattern $\mathbf{a} = [\ a_1\ ,...,\ a_N\ ]$ is imposed onto the net's input layer and is propagated through the first layer. The outputs of the first-layer nodes are (see Fig. 9):

$$x_r = f\left(\sum_{n=0}^{N} w_{rn}\, a_n\right) \quad , \quad r = 1,...,R$$

where f is the linear-threshold nonlinearity. The first layer in fact calculates *Hamming distance* for each training example stored in the network, which is defined as a number of (binary) features of the input pattern **a** which do not match the corresponding components (features) of the stored example $\mathbf{a}_r$.

The second layer of the Hamming net depicts the *winner-take-all* technique which picks up the maximum value among its inputs x_1 ,..., x_R . This is done in a feedback regime according to the following iterative formulas:

$$y_r(0) = x_r \quad , \quad r = 1,...,R$$

$$y_r(t+1) = f\left(\ y_r(t) - \varepsilon \sum_{s \neq r} y_s(t)\ \right)$$

where $0 < \varepsilon < 1/R$ is a priori given constant. The above formula is executed until the outputs y_r no longer change. At that time only one output is positive and the others are zero. This positive output indicates the 'winner'; it is the training example which is the closest to the input pattern **a** , i.e. whose Hamming distance to the input pattern is minimum. If the Hamming net is used for classification then the input pattern will be classified to the class of the winner. Other attributes are discussed e.g. in [Lip87].

3.6. Carpenter-Grossberg classifier

The *Carpenter-Grossberg classifier* [Car86] (Fig. 10) is a two-layer neural network, with the feedback regime, working in a discrete time. It uses the nearest neighbour active regime and competitive unsupervised learning. Its input is binary only. This net achieves a clustering algorithm which is similar to the traditional sequential leader clustering algorithms (see e.g. [Har75]).

As we can see on Fig. 10, the network consists of both 'bottom-up' connections with weights w_{rn} and 'top-down' connections with weights w'_{rn} . The net of 'bottom-up' connections is equivalent to the Hamming net (section 3.5), the 'top-down' connections propagate so-called matching exemplar to the input nodes where it is compared with the input pattern. In the following we will give a flow-chart of the entire procedure. Let there be N input nodes and R output nodes. Assume the all weights have been adjusted according to previous learning sweeps. Let a new input pattern be $\mathbf{a} = [\ a_1\ ,...,\ a_N\ ]$. Then its processing by the Carpenter-Grossberg classifier is outlined as follows:

1. Present an unknown input pattern $\mathbf{a} = [\ a_1\ ,...,\ a_N\ ]$ to the input nodes. Let the set of allowable winners comprise all output nodes.

2. Send signals through w_{rn} to these output nodes y_r which are members of the set of allowable winners.

3. Find y_s with the maximum value among y_r , r=1,...,R, using the winner-take-all strategy (equivalent to the Hamming net).

4. The winning output node y_s sends the top-down signal along the weights w'_{sn} back to the input nodes, forming the matching exemplar $\mathbf{a}' = [\ w'_{s1}\ a_1\ ,...,\ w'_{sN}\ a_N\ ]$.

5. The input pattern is compared with the matching exemplar by so-called *vigilance*

$$\rho = \frac{|a|}{|a'|}$$

where the norm |a| of the pattern **a** equals to the sum of its components (features) a_n .

5.1. If $\rho \geq \rho_{max}$, where ρ_{max} is a vigilance threshold (a number between 0 and 1), then the winning node y_s represents the proper class of the given input pattern **a** , and **a** is *merged* into this class, i.e. all w_{ns} and w'_{sn} of the winning matching exemplar are modified:

$$w'_{sn}(t+1) = w'_{sn}(t)\ a_n$$

$$w_{ns}(t+1) = \frac{w'_{sn}(t)\ a_n}{0.5 + |a'|}$$

and the learning sweep for the given input pattern is terminated.

5.2. However, if $\rho < \rho_{max}$ then the winning node y_s does not represent the proper class of the input pattern **a** , and y_s is removed from the set of allowable winners. If there are still some allowable winners, go to Step 2. Otherwise the pattern **a** forms a new class, i.e. a new node y_{R+1} is created and **a** is encoded to it.

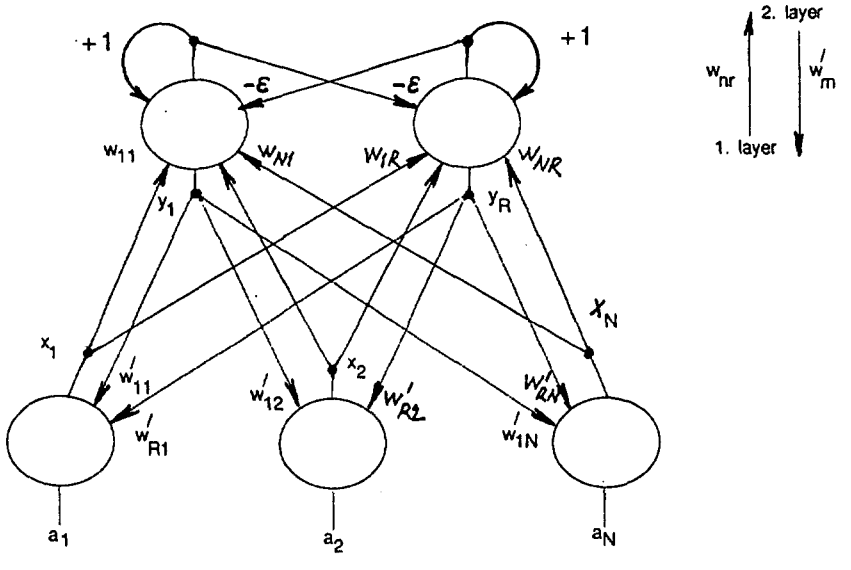


Fig. 10. Carpenter-Grossberg classifier

With no noise, the vigilance threshold can be set such that two patterns which are most similar are considered different. In noise, however, this level may be too high and the number of new stored exemplars can rapidly grow. One possibility is changing vigilance threshold during training, see e.g. [Car86].

3.7. Kohonen's self-organizing feature maps

The *Kohonen's self-organizing feature map* [Koh84] is a one-layer neural network with the feedforward scheme, working in discrete time. It exploits the nearest-neighbour active regime and cooperative learning. Its behaviour is similar to K-means clustering algorithm. It incorporates a kind of space topology (Fig. 11), since all its nodes are organized into a two-dimensional array. Particularly, neighbourhood of a node is taking into consideration.

Training (learning) of Kohonen's feature map looks as follows. Let there be R nodes in the two-dimensional topology, and each pattern have N features. For a training example $a = [a_1,...,a_N]$ the following distances between a and the weight vectors w_r for each node are computed

$$d_r = | a - w_r |^2 = \sum_{n=1}^{N} (a_n - w_{nr})^2 \quad , \quad r = 1,...,R$$

Then the output s with minimum distance is selected among r=1,...,R . Afterwards, the weights of this winner and its neighbours are updated:

$$w_{nr}(t+1) = w_{nr}(t) + \eta(t) \, (a_n - w_{nr}(t)) \qquad \text{for } r \in N_s(t)$$

where $N_s(t)$ is the neighbourhood of the node s at time t , $\eta(t)$ is the learning rate at time t . The other weights are not modified. The important fact is that both the size of any neighbourhood and the learning rate decrease in time. E.g. if the node s is a winner for the time t then the weights of all 25 nodes in its neighbourhood (including the winner itself) are modified at the time t , only 9 nodes at the time t+1 , and just the winner itself at time t+2 , see Fig. 11.

After enough training patterns have been presented, the weights will specify cluster or pattern (vector) centers that sample the feature space such that the density function of the pattern centers tends to approximate the probability density function of the training patterns. The weights will be organized such that topologically close nodes are sensitive to the input patterns that are physically similar. The algorithm performs relatively well in noise. Other aspects as well as areas of applications can be found in [Sim90].

3.8. Boltzmann machine

The *Boltzmann machine* [Hin86] is a two-layer network with feedforward scheme and binary inputs, working in discrete time. It exploits the nearest-neighbour active regime and a combination of Hebbian learning and stochastic learning.

The entire principle of the Boltzmann machine is described in detail e.g. in [Sim90]. Here we highlight the idea of *stochastic learning* only. A learning sweep looks as follows:

1. Add a new training example to the network by applying the Hebbian learning formula.
2. Make a random weight change.
3. Determine the change of energy function ΔE after the weight was randomly changed.

4a. If $\Delta E < 0$ then keep the change.

4b. If $\Delta E > 0$ select a random number p and calculate

$$P = e^{-\Delta E/T(t)}$$

where T(t) is the *temperature* of the Boltzmann process. If $p < P$ then accept the weight change; otherwise return to the original value.

We can observe that larger T(t) causes the random weight change would be accepted more likely. The temperature T(t) decreases in time according to

$$T(t) = T_0 / (1 + \log t)$$

The authors of this method have proven that the random character of the weight change allows to escape local energy minima and reach thus the absolute minimum of the energy function.

3.9. Multilayer perceptron

As we have already mentioned, a linear classifier, i.e. Rosenblatt's perceptron without its Φ-processor (or: a *single-layer perceptron*, in the neural net terminology) can only create linear decision boundaries (Fig. 12a). A *two-layer* perceptron can form convex decision boundaries (see Fig. 12b) while a *three-layer* one can generate boundaries of any shape (Fig. 12c) [Lip87]. This means that no more than three layers are required because a three-layer perceptron can generate arbitrarily complex decision regions.

The structure of a three-layer perceptron is shown in Fig. 12c; it has two hidden layers and one output layer. A two-layer

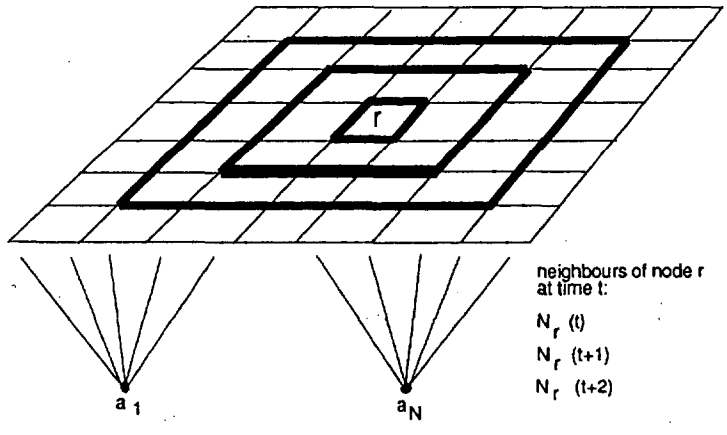


Fig. 11. Kohonen's self-organizing feature map

perceptron has only one hidden layer and, of course, one output layer (Fig. 12b). Thanks to the hidden layer(s), the multilayer perceptrons overcome many limitations found in a single-layer perceptron. However, they were not generally used in the past because effective learning algorithms that would adjust their weights in an optimal way were not available. Three years ago, the so-called back propagation learning algorithm for multilayer perceptrons was discovered [Ru86b]. Although it cannot be proven that this learning algorithm generally converges (as with the single-layer perceptron), it has been proven to be useful and efficient for many pattern recognition problems.

Since the structure and the back propagation learning algorithm are almost identical for two- and three-layer perceptron, we will only describe the structure and behaviour of the latter. An input pattern of a multilayer perceptron is represented by a N-dimensional feature vector

$$\mathbf{x} = [\, x_1, \ldots, x_N \,]$$

where x_n, $n = 1,\ldots,N$ are features, i.e. numbers in the range 0 to 1.

The feature vector is input to each of M nodes of the first hidden layer so that the linear weighted sum of all features is computed and the sigmoid nonlinearity

$$f(\alpha) = \frac{1}{1 + e^{-\alpha}}$$

converts it to an output x'_m, $m = 1,\ldots,M$, that is thus between 0 and 1. Hence (see Fig. 12c)

$$x'_m = f\left(\sum_{n=0}^{N} w_{nm}^{(1)} x_n \right) , \quad m = 1,\ldots,M \qquad (5)$$

where f is the above nonlinearity, $w_{nm}^{(1)}$ is the weight connecting the n-th input to the m-th node of the first hidden layer, $x_0 = 1$ allows the weight $w_{0m}^{(1)}$ to be considered as a threshold of the m-th hidden node.

In the same way, the second hidden layer has H nodes and the output x''_h of the h-th node of the second hidden layer is

$$x''_h = f\left(\sum_{m=0}^{M} w_{mh}^{(2)} x'_m \right) , \quad h = 1,\ldots,H \qquad (6)$$

where $w_{mh}^{(2)}$ is the weight connecting the m-th node of the first hidden layer to the h-th node of the second hidden layer, $x'_0 = 1$ processes the weights $w_{0h}^{(2)}$ as thresholds.

Similarly, the output layer has R nodes and the outputs x''_h of the second hidden layer are linearly weighted and the sum is converted by the sigmoid nonlinearity. The output signal of the r-th output node is thus

$$y_r = f\left(\sum_{h=0}^{H} w_{hr}^{(3)} x''_h \right) , \quad r = 1,\ldots,R \qquad (7)$$

where $w_{hr}^{(3)}$ is the weight connecting the h-th node of the second hidden layer to the r-th output node, $x''_0 = 1$ processes the weights $w_{0r}^{(3)}$ as thresholds.

The multilayer perceptron, when used for classification, can classify input patterns to R classes z_r, $r = 1,\ldots,R$. The discriminant functions are given by formulas (7) and the decision rule of the classifier is equivalent to (1): the pattern (more precisely, its feature vector) $\mathbf{x}$ is classified to the class z_s iff the output y_s is the maximum among y_r, $r = 1,\ldots,R$.

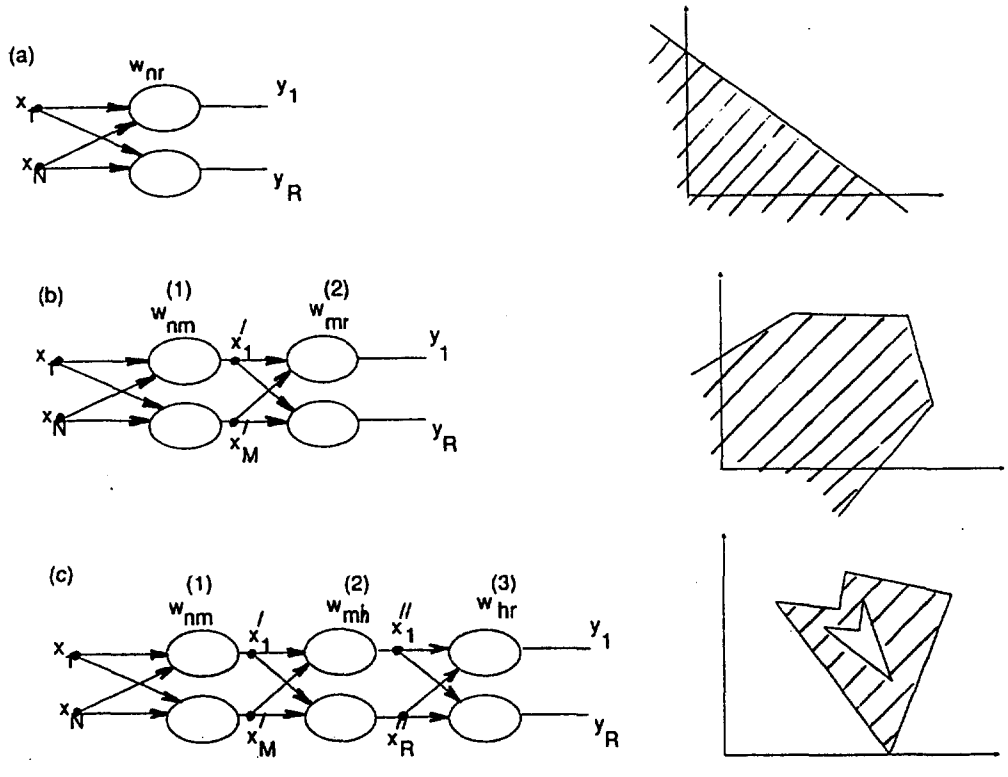


Fig. 12. Structure and shape of decision regions for (a) single-layer perceptron, (b) two-layer perceptron, (c) three-layer perceptron. The arrows depict weights, a circle represents the sum including nonlinearity

Since we want the multilayer perceptron to have the optimal classification performance we have to adjust its weights according to a criterion. As in case of a single-layer perceptron, one possible and commonly used way of such an optimal adjustment is to use a learning (training) process. As we have already stated one of the popular learning algorithm for multilayer perceptrons is the *back propagation training algorithm*. It propagates an output error signal back through the network and modifies its weights accordingly.

The *error signal* is defined as

$$eps = \sum_{r=1}^{R} (y_r - Y_r)^2 \qquad (8)$$

where Y_r is the desired output of the r-th output node.

For the purposes of classification, if a training pattern x belongs to the desired class $Z = z_s$ then

$$Y_s = 1 \qquad (9)$$
$$Y_r = 0 \quad \text{for} \quad r \neq s$$

Hence, the error signal for the training pattern x of the (desired) class $Z = z_s$ is

$$eps = \sum_{r \neq s} y_r^2 + (1 - y_s)^2 \qquad (10)$$

Training patterns are presented several times to the training algorithm until the error signal for all training patterns is less than an a priori given maximum epsmax . This maximum must be below 0.5 , since it is the error signal value of the worst case: if e.g. a training pattern belongs to $Z = z_1$ and if

$$y_1 = 0.5 \ , \ y_2 = 0.5 \ , \ y_r = 0 \quad \text{for} \quad r \neq 1, 2$$

then

$$eps = 0.5^2 + (1 - 0.5)^2 = 0.5$$

The flow chart of the back propagation algorithm is given below (written in the same fashion as Rosenblatt's learning algorithm):

1. Initialize all weights to random values in the range -Winit to +Winit where Winit is a parameter of the learning algorithm that shrinks or expands the range of the initial values of weights.
2. Set Number_of_learning_sweeps to 0 .
3. For each training pattern $x = [x_1 ,..., x_N]$ with its desired class $Z = z_s$ do:
   - 3.1. Compute y_r , r = 1,...,R for the given pattern according to (5) to (7).
   - 3.2. Compute the error signal eps according to (8) and (9).
   - 3.3. If eps > epsmax (a given maximum) then modify the weights as follows: compute

     $$\delta_r^{(3)} = y_r (1 - y_r) (Y_r - y_r) \ , \ r = 1,...,R \qquad (11)$$

     and change the weights of the output layer:

     $$w_{hr}^{(3)} \mathrel{+}= \eta \ \delta_r^{(3)} \ x''_h \ , \ h = 0,1,...,H \ , \ r = 1,...,R \qquad (12)$$

     where η is the *learning rate* (*gain term*), another parameter of the learning algorithm.

     Similarly, change the weights of the second hidden layer:

     $$\delta_h^{(2)} = x''_h (1-x''_h) \sum_{r=1}^{R} \delta_r^{(3)} w_{hr}^{(3)} \qquad (13)$$

     $$w_{mh}^{(2)} \mathrel{+}= \eta \ \delta_h^{(2)} \ x'_m \ , \ m = 0,1,...,M \ , \ h = 1,...,H \qquad (14)$$

     and those of the first hidden layer:

     $$\delta_m^{(1)} = x'_m (1-x'_m) \sum_{h=1}^{H} \delta_h^{(2)} w_{mh}^{(2)} \qquad (15)$$

     $$w_{nm}^{(1)} \mathrel{+}= \eta \ \delta_m^{(1)} \ x_n \ , \ n = 0,1,...,N \ , \ m = 1,...,M \qquad (16)$$
4. If we have modified the weights for at least one training pattern, increment Number_of_learning_sweeps by 1 and return to the step 3, i.e. repeat the training for the entire training set.
   If not, save all weights, print Number_of_learning_sweeps and terminate the learning process.

Convergence is sometimes faster and the danger of oscillation is diminished if a so-called *momentum term* α is added to the formulas for changing weights [Lip87].

The two-layer perceptron has just one hidden layer with weights $w_{nm}^{(1)}$ and the output layer with weights $w_{mr}^{(2)}$. Otherwise, the structure is identical to the three-layer perceptron, and the formulas (11) to (16) have to be slightly changed.

4. Application: waveform processing

4.1. Introduction

Multilayer perceptrons have found many useful and efficient applications in pattern recognition. Our research group is using a multilayer perceptron as one component of a larger decision-supporting system for neurological diagnoses. The actual input to our system is an evoked potential waveform. It is processed by these subsystems [Bru88], [Bru89] (see Fig. 13 for illustration):

(1) Filter preprocessing. The evoked potential waveform is preprocessed by a digital filter. This eliminates noise and allows the use of simple recognition grammars in the syntax analysis stage.

(2) Extracting a string of symbols. The potential waveform is segmented and each segment is described by one (terminal) symbol. Thus, the entire input waveform is formally described by a string of (terminal) symbols [Mad86].

(3) Syntax analysis. An attributed regular grammar with semantic functions [Fu82] is used for the syntactic analysis of an input waveform represented by a string of symbols. It recognizes the start of the relevant waveform as well as each peak of the waveform. It returns through its semantic functions the latency of the beginning of the relevant waveform, number of hills, and the peak latency of each hill found (see [Bru88] for details).

(4) Numerical classification. The features extracted by the attributed grammar are further processed by a two-layer perceptron that classifies the given input waveforms (using the above features) into two classes: normal and abnormal. Note the semantic functions of our attributed grammar form an interface between strictly syntax subsystem (a grammar) and strictly numerical processing (a multilayer perceptron).

(5) The knowledge-based subsystem. We analyze methods of incorporating a knowledge-based subsystem into our decision-supporting system in order to obtain more reliable results. The need for such a knowledge-intensive device is discussed in the conclusion of this paper.

After analyzing a large number of experiments that have been conducted to obtain the performance characteristics of a multilayer perceptron, we have found out that the three-layer perceptron is quite unstable for this recognition task. Therefore, we have focused on a two-layer perceptron. Similar conclusion can be found in e.g. [Wie87].

The training method used in our experiments is the back propagation training algorithm. By analyzing the above algorithm one can easily discover that the following five parameters strongly affect both learning and classification performance of the multilayer perceptron:

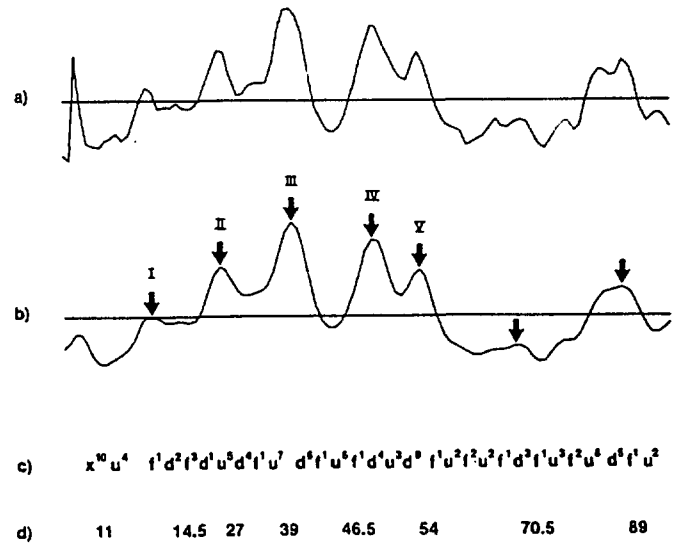


Fig. 13. Waveform processing: (a) evoked potential waveform, (b) filtered data, (c) string of symbols (u upward slope, d downward slope, f flat line, x irrelevant data), (d) list of features

- the range of initial weights Winit ,
- the learning rate (gain term) eta ,
- the momentum term alpha ,
- the maximum allowed value of the error signal eps ,
- the number of hidden nodes M .

However, there is no formal method for obtaining optimal values of the above parameters. Therefore, we have carried out a large number of experiments, analyzed them, and obtained a few heuristics (thumb-rules) for optimal adjustments of the above parameters. The following sections discuss both experiments and the heuristics obtained.

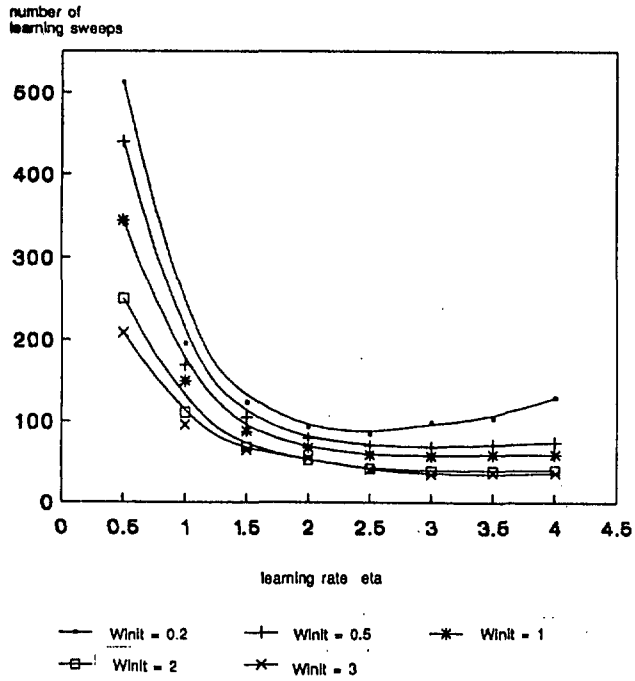


Fig. 14. Number of learning sweeps versus the learning rate (gain term) eta and the initial weight range Winit (M = 9 , eps = 0.30 , alpha = 0)

4.2. Optimal adjustment of parameters: experiments

A two-layer perceptron emulator used in the experiments has been implemented in C because of its portability and flexibility. The emulator has been tested under BSD 4.3, SunOS, and DOS using MS-DOS Microsoft C and Turbo C.

We have used a training set of 50 evoked potentials (training patterns) of normals (class z_1) and 50 potentials of abnormals (class z_2) for training the two-layer perceptron. Additional set of 40 patterns with unknown classes has been used for testing. The number of features (inputs) has always been N = 4 , and the number of classes R = 2 . We have chosen the maximum error signal eps = 0.3 for most experiments since it corresponds to the reasonable difference of 0.4 between the actual output (y_r) and the desired one (Y_r) which has been accepted in many experiments (see e.g. [Bur88], [Bru89]); if a training pattern belongs to the class say z_1 then we allow

$$y_1 = 0.6 \quad , \quad y_2 = 0.4$$

thus approximately

$$eps = (1 - 0.6)^2 + 0.4^2 = 0.3$$

We have executed a large number of experiments for various sets of parameters. The efficiency of the learning algorithm has been symbolized by the following factors [Bur88]:

1. the number of learning sweeps,
2. the accuracy in classifying unknown (testing) patterns, measured as percentage of correct classifications.

Following the Kolmogorov's mapping theorem and its interpretation to the neural nets [He87], we have set the number of hidden nodes to

$$M = 2 * N + 1 = 9$$

Afterwards, we have carried out four sets of experiments, running the back propagation algorithm for the given set of 100 training patterns (50 normals and 50 abnormals). To achieve comparable statistical results, we have made 200 runs for each configuration.

The first set of our experiments has been run to find optimal values of the learning rate eta . Therefore, we have run the back propagation algorithm for the two-layer perceptron with eps = 0.30, M = 9 , alpha = 0 , for various learning rates (gain terms) eta (from 0.5 to 4.0) and various initial weight range Winit (from 0.2 to 3.0). The results are in Fig. 14. We observe that the learning algorithm has required *larger* number of learning sweeps for *smaller* values of eta , since less information is learned during each sweep. The performance has not changed substantially for learning rate eta between 2.5 and 4.0 . It is also seen that *the larger* the range of initial weights, *the faster* the training is completed.

Therefore, we have chosen Winit = 3.0 and eta = 2.5, 3.0, 4.0 as promising values for the second set of experiments that were to reveal an optimal number of hidden nodes M . Again, eps = 0.30 , alpha = 0 , but the number of hidden nodes has been changed from 4 to 25 . The results are in Fig. 15a and 15b. The figures indicate that the number M = 9 chosen according to Kolmogorov's mapping theorem yields the optimal performance for the number of learning sweeps only. On the other hand, the accuracy of classifying unknown patterns (evoked potential waveforms) is satisfactory for M > 9 . Therefore, we conclude that the *number obtained by Kolmogorov's theorem* should be considered as the *lower bound* of the recommended number of hidden nodes.

The objective of the third set of experiments has been to confirm the previous two tests that Winit equal 3.0 was an optimal parameter. We have set eps = 0.30 , eta = 4.0 , alpha = 0 , M has been changed from 7 to 16 . The results are presented in Fig. 16a and 16b; they confirm the predicted optimal value for Winit as well as the above thumb-rule for M .

The momentum term alpha may speed up the convergence of the back propagation learning algorithm. It is expected to smooth the weight changes during the learning. Therefore, we have run the last set of experiments in order to observe the effect of the momentum term to the perceptron's performance. The results are in Fig. 17a and 17b. Fig. 17a indicates that as the value of the momentum factor *increases*, the optimal value for the learning rate (gain term) *decreases*. However, the accuracy of classification *decreases* with *larger* momentum factor (Fig. 17b).

We have run the same set of experiments for a three-layer perceptron for the same learning rates, initial weights range, and maximum error signal. As for the number M of nodes in the first hidden layer, we have again incorporated Kolmogorov's theorem, and have followed the advice [6] that there should be more than three times as many nodes in the second hidden layer as in the first one, i.e. H >= 3 * M . However, we have discovered that the back propagation learning algorithm does not converge over a great number of runs, and the weights oscillate, especially in the second hidden layer.

4.3. Optimal Adjustment of Parameters: Heuristics

This section discusses the heuristics (thumb-rules) for optimal adjustment of multilayer perceptron's parameters we have observed after analyzing the results of our experiments.

- The initial weight range Winit should be a larger value, greater than 1.0 . Our recommendation is Winit = 3.0 .
- The learning rate (gain term) eta should be a larger value, greater than 1.0 . Intuitively, a larger momentum term together with a larger learning rate would speed up the learning and avoid the oscillation. However, our experiments have revealed that the optimal learning rate for minimum number of learning sweeps becomes *smaller* for *larger* momentum factor. Our recommendation is to choose eta between 1.0 and 4.0 and alpha = 0 as a starting point. If the learning does not converge, one should increase alpha and decrease eta .
- The number obtained by Kolmogorov's theorem should be considered as the lower bound of the recommended number of hidden nodes. However, [Guy89] introduces another thumb-rule for an optimal number of hidden nodes:

  M = (N + R) / 2

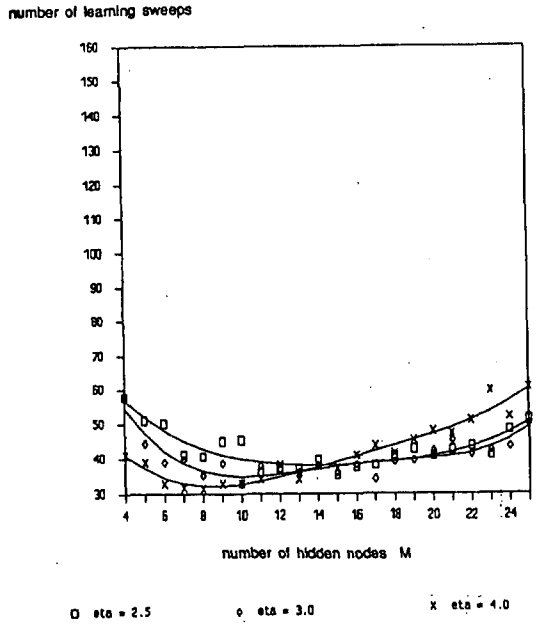


Fig. 15a. Number of learning sweeps versus number of hidden nodes M (eta = 2.5, 3.0, 4.0 , Winit = 3 , eps = 0.30 , alpha = 0)

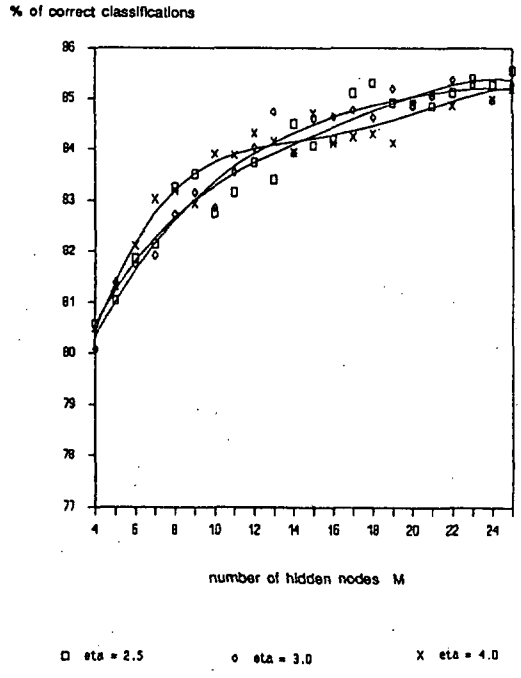


Fig. 15b. Percent correct recognitions versus number of hidden nodes M (eta = 2.5, 3.0, 4.0 , Winit = 3 , eps = 0.30 , alpha = 0)

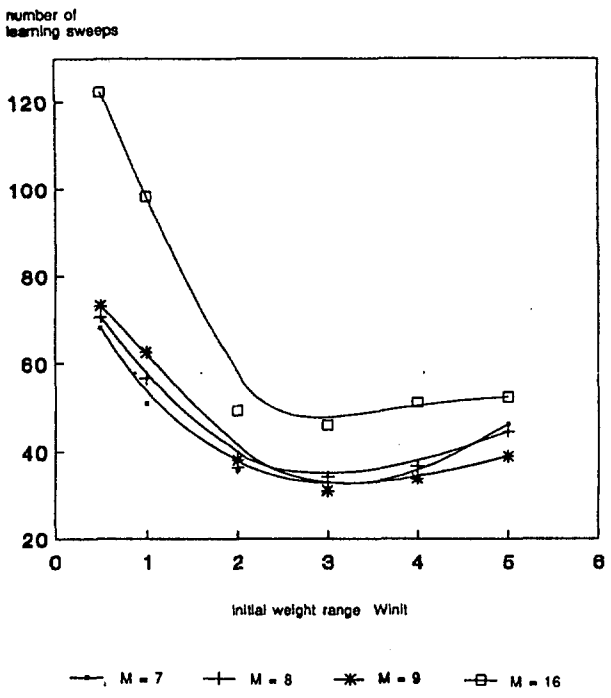


Fig. 16a. Number of learning sweeps versus initial weight range Winit and the number of hidden nodes M (eta = 4.0 , eps = 0.30 , alpha = 0)

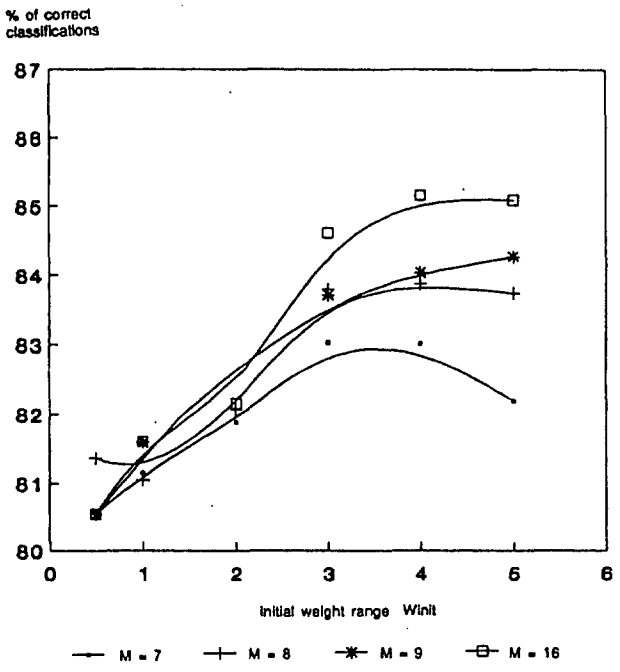


Fig. 16b. Percent correct recognitions versus initial weight range Winit and the number of hidden nodes M (eta = 4.0 , eps = 0.30 , alpha = 0)

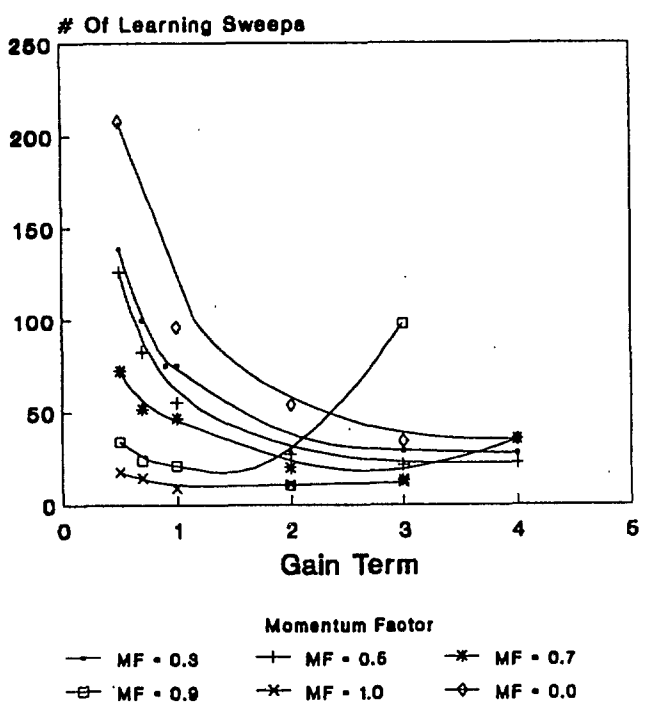


Fig. 17a. Number of learning sweeps versus learning rate (gain term) eta and momentum factor alpha (M = 9 , Winit = 3 , eps = 0.30)

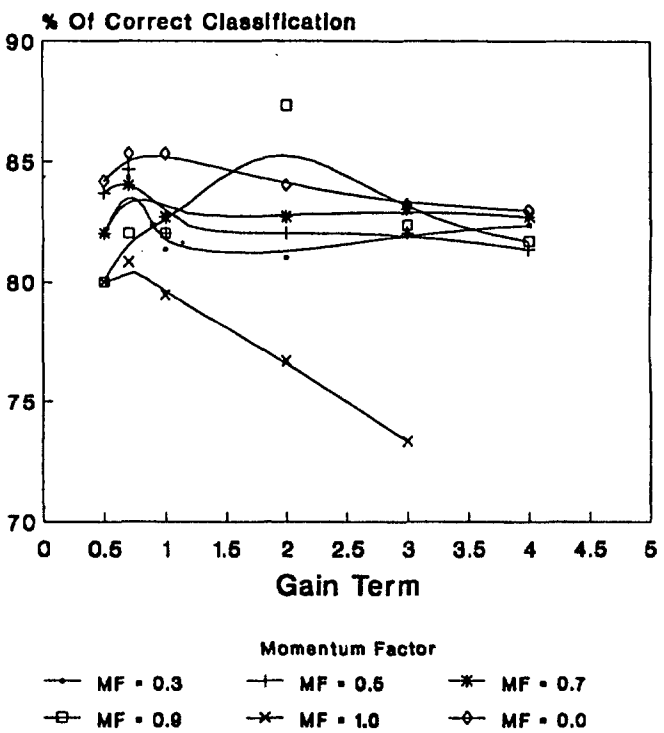


Fig. 17b. Percent correct recognitions versus learning rate (gain term) eta and momentum factor alpha (M = 9 , Winit = 3 , eps = 0.30)

where N is the number of features (input nodes) of the perceptron, R is the number of output nodes (i.e. classes in a classification problem). Our recommendation is to try Kolmogorov's mapping theorem for smaller number (units) of features, and the above formula for larger number (tens) of features[1].

Let us summarize the entire procedure of finding optimal values of the parameters:

(1) The user of a multilayer perceptron should firstly select a relatively small subset of representative training examples (patterns). The error signal should be set to a larger value (less than 0.5), say eps = 0.49 . The first heuristic should be applied for the initial weight range.

(2) Following the second and the third heuristics, the user should run a few experiments (using the above small representative set of training patterns) in order to find the optimal values of eta , alpha and M .

(3) As our experiments have revealed, the above optimal values can be afterwards used for the entire training set, with the error signal set to e.g. eps = 0.30 , which will guarantee more precise results of learning.

5. Conclusion: the need for a knowledge-based subsystem

This paper surveys the fundamental tools for pattern recognition: perceptrons and neural nets. We have exhibited that neural nets are powerful tools for image, speech, and waveform processing, as well as general recognition systems. The greatest potential of neural nets is in the high-speed processing that can be achieved by means of parallel VLSI implementations. Massively parallel hardware is evidently one of the reasons why so many researchers are investigating this field so broadly at present. Current research is aimed at analyzing new types of neural net structures and their learning (or self-learning) algorithms. Thus after twenty years the enthusiasm of artificial intelligence research and development has returned to connectionism. Nevertheless, we should bear in mind that the marvellous powers of the brain emerge not from any single, uniformly structured connectionist network but from highly evolved arrangements of smaller, specialized networks which are interconnected in very specific ways [Min88].

Furthermore, the paper introduces our decision-supporting system for neurological diagnoses comprised of several subsystems, with all but last one having been developed, implemented, and thoroughly tested. Using attributed grammars in our recognition system seems to be quite adequate because they provide a simple way of expressing semantic results of the syntactic recognition. The features extracted by the attributed grammar are further processed by a multilayer perceptron. A large number of experiments have revealed that the three-layer perceptron is quite unstable. Therefore, we have focused on the two-layer perceptron, and indicated four parameters of the learning algorithm that have to be adjusted by the user. We have found that the ranges of these parameters depend on the classification problem. Thumb-rules have been derived from these results.

After considerable experience with combined syntax and neural net classification, we have found that 'well-behaved' waveform processing reaches acceptable rates of correct classification. By 'well-behaved', we mean waveforms whose peaks are identifiable after the initial filter processing. However, there are quite a few clinical cases which are not 'well-behaved'. For example, the even peaks of waveforms may be difficult to detect even in some normal subjects [Chi83]. Therefore, we have decided to incorporate a knowledge-based subsystem to our decision-supporting system, since our present combined syntax and neural net system would not be able to handle such situations in an elegant way. The knowledge-based subsystem will be able:

(a) to handle 'non-well-behaved' data as mentioned above,
(b) to combine different modalities of evoked potential waveforms (to improve its behaviour the system will have to process and analyze three or more separate types of evoked potential waveforms),
(c) to incorporate other clinical information (for a complete diagnostic system, rules for incorporating other patient related factors such as age, sex, etc. will have to be developed).

There are three possible ways of incorporating a knowledge-based subsystem into our decision-supporting system:

(i) A knowledge-based system as a 'high-level' processing subsystem can be placed at the end of the entire evoked potential processing. In this traditional case, the knowledge-based subsystem does not influence the 'low-level' processing at all. However, this configuration fails if the feature list represents a 'non-well-behaved' situation (pattern) which would be processed by the neural net in an improper fashion and should, therefore, be recognized in advance by the knowledge-based subsystem.

(ii) A knowledge-based subsystem can be placed before the neural net, recognize 'non-well-behaved' cases (patterns) and process them separately; the 'well-behaved' cases could be directly processed by the neural net. We are going to investigate this promising configuration thoroughly but its success will depend on the knowledge acquisition, i.e. co-operation with human experts in the field of neurology. As for the knowledge representation we are going to use a rule-based model with uncertainties, the expert system environment called McESE [Fr89].

(iii) A knowledge-based subsystem can co-operate with the neural net so that it can call the multilayer perceptron and continue the processing according to its results. This feed-back loop configuration seems to be the most promising model. However, we cannot directly use the conventional representation of knowledge as a set of production rules, which is the case in the model (ii), since the rule-based system has to modify its inference process according to the results of the neural net.

[1] based on results of experiments done with another set of real data [Ho89]

References

[And72] J. Anderson, "A simple neural network generating an interactive memory", Math. Biosciences, 14, pp. 197-220, 1972

[Bru80] I. Bruha, J. Jelinek, Z. Kotek, Adaptive and Learning Systems, SNTL Prague, 1980 (In Czech)

[Bru88] I. Bruha and G.P. Madhavan, "Use of attributed grammars for pattern recognition of evoked potentials", IEEE Trans. System, Man and Cyber., Dec. 1988

[Bru89] I. Bruha and G.P. Madhavan, "Combined syntax - neural net method for pattern recognition of evoked potentials", Proc. International Conference Computing and Information, North Holland, May 1989

[Bur88] D.J. Burr, "Experiments on neural net recognition of spoken and written text", IEEE Trans. Acoustics Speech Signal Proc., Vol. 36, No. 7, pp. 1162-8, July 1988

[Car86] G.A. Carpenter and S. Grossberg, "Neural dynamics of category learning and recognition: attention, memory consolidation, and amnesia", in: J. Davis, R. Newburgh, and E. Wegman (eds.), Brain Structure, Learning, and Memory, 1986

[Chi83] K.H. Chiappa, "Evoked potentials in clinical medicine", Raven Press, New York, 1983

[Du72] R.O. Duda and P.E. Hart, "Use of Hough transformation to detect lines and curves in pictures", Communications of the ACM, 15, 1972

[Du73] R.O. Duda and P.E. Hart, Pattern Classification and Scene Analysis, Wiley, New York, 1973

[Fr89] F. Franek and I. Bruha, "McESE - McMaster Expert System Environment", International Conference Computing and Information, North Holland, May 1989

[Fu82] K.S. Fu, "Syntactic Pattern Recognition and Applications", Prentice-Hall, New Jersey, 1982

[Gro78] S. Grossberg, "Adaptive pattern classification and universal recording: parallel development and coding of neural feature detectors", 3rd European Conference Cybernetics and Systems Research, Halstead Press, 1978

[Gro88] S. Grossberg, "Nonlinear neural networks: principles, mechanisms, and architectures", Neural Networks, Vol. 1, pp. 17-61, 1988

[Guy89] I. Guyon, "Neural Network Systems", Proc. of INME Symposium, Lausanne, Sept. 1989

[Har75] J.A. Hartigan, Clustering Algorithms. John Wiley, New York, 1975

[He87] R. Hecht-Nielsen, "Kolmogorov's mapping neural network existence theorem", IEEE International Conference Neural Networks, San Diego, 1987

[He88] R. Hecht-Nielson, "Applications of counterpropagation networks", Neural Networks, 1, pp. 131-140, 1988

[Hin86] G.E. Hinton and T.J. Sejnowski, "Learning and relearning in Boltzmann machines", in [Ru86a]

[Ho89] R. Ho, "A Neural Network System for Recognition of Evoked Potentials", M.Sc. Thesis, Dept. Computer Science and Systems, McMaster Univ., 1989

[Hop82] J.J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities", Proc. National Academy of Sciences, 79, 2554-8, 1982

[Hop86] J.J. Hopfield and D.W. Tank, "Computing with Neural Circuits: a model", Science, Vol. 233, pp. 625-633, 1986

[Koh77] T. Kohonen, Associate memory - a system theoretical approach, Springer Verlag, New York, 1977

[Koh84] T. Kohonen, Self-organization and associative memory, Springer Verlag, Berlin, 1984

[Lip87] R.P. Lippman, "An introduction to computing with neural nets", IEEE ASSP magazine, pp. 4-22, April 1987

[Mad86] G.P. Madhavan, H. de Bruin, A.R.M. Upton, M.E. Jernigan, "Classification of Brain-stem Auditory Evoked Potential by Syntactic Methods", Electroencephalography and Clinical Neurophysiology, 65, 289-296, 1986

[Mat87] C.J. Matheus and W.E. Hohensee, "Learning in artificial neural systems", Techn. Report 87-1394, Dept. Computer Science, Univ. Illinois, Dec. 1987

[Min69] M. Minsky and S. Papert, The perceptrons, MIT Press, 1969

[Min88] M. Minsky and S. Papert, The perceptrons: expanded edition, MIT Press, 1988

[Om87] S.M. Omohundro, "Efficient algorithms with neural network behaviour", Techn. Report 87-1331, Dept. Computer Science, Univ. Illinois, Dec. 1987

[Ros61] R. Rosenblatt, Principles of neurodynamics, Spartan Books, Washington, 1961

[Ru86a] D.E. Rummelhart and J.L. McClelland, Parallel distributed processing: exploration in the microstructure of cognition, MIT Press, 1986

[Ru86b] D.E. Rummelhart, G.E. Hinton, and R.J. Williams, "Learning internal representation by error propagation", in [Ru86a]

[Sej86] T. Sejnowski and C.R. Rosenberg, "NETtalk: a parallel network that learns to read aloud", Johns Hopkins Univ., Techn. Report EECS-86/01, 1986

[Sim90] P.K. Simpson, Artificial Neural Systems. Pergamon Press, 1990

[Tou74] J.T. Tou and R.C. Gonzalez, Pattern Recognition Principles, Addison-Wesley, London, 1974

[Wa85] S. Watanabe, Pattern Recognition: Human and Mechanical, Wiley, New York, 1985

[Wid60] B. Widrow and M.E. Hoff, "Adaptive switching circuits", IRE WESCON Convention Record, Part 4, pp. 96-104, 1960

[Wie87] A. Wieland, R. Leighton, "Geometric analysis of neural network capabilities", IEEE International Conference Neural Networks, San Diego, 1987

Nevronske mreže - pregled in uporaba v procesiranju signalov

V prispevku je podan pregled umetnih nevronskih mrež kot bistvenih orodij za razpoznavanje vzorcev. Avtorjeva pozornost gre predvsem značilnostim dobro znanih tipov trenutno uporabljanih nevronskih mrež in implementaciji večnivojskega perceptrona. Podane so specifikacije, različni vidiki in primerjave različnih vrst nevronskih mrež. Opisan je tudi sistem za podporo pri odločanju za nevrološko diagnostiko. Vhod v ta sistem je signal evociranega potenciala, ki ga analizira algoritem za sintaktično razpoznavanje vzorcev. Algoritem temelji na atributni regularni gramatiki in njegove semantične funkcije izračunajo seznam njegovih numeričnih značilk. Druga faza procesiranja signala vključuje dvonivojski perceptron, ki procesira omenjene numerične značilke. Nekaj besed bomo namenili tudi empiričnim pravilom za optimalno nastavitev parametrov pri perceptronu.

CASE TOOLS-SOFTWARE DEVELOPMENT AND MAINTENANCE AID: A SURVEY

INFORMATICA 1/91

Keywords: CASE methodology, CASE tools, life cycle, maintenance, reverse engineering

Saša Bošnjak
Laslo Šereš
Faculty of Economics,
Subotica, Moše Pijade 9

SADRŽAJ. U radu je data osnovna podela CASE proizvoda s posebnim osvrtom na njihovu upotrebu u pojedinim fazama životnog ciklusa softverskih proizvoda. Za razliku od uobičajenog pristupa kojim se ističe značaj CASE proizvoda u fazi razvoja softvera, ovde je posebna pažnja posvećena korišćenju CASE prizvoda u završnoj fazi, tj. pri održavanju softverskih proizvoda. S obzirom na probleme koji se javljaju u održavanju softvera namera je da ukažemo na neke nove tehnike koje treba da doprinesu smanjenju angažovanih ljudskih i materijalnih resursa u ovim poslovima.

ABSTRACT. This article deals with the basic classification of CASE products, with a special review on their usage in various phases of software products life cycle. A particular attention has been paid to software maintenance and the help of CASE products in these activities, being a quite opposite approach to the usual ones. Usually, one can find informations about the usage of CASE in the development of software, but very rarely anything about its usage in software maintenance. Our intention was to present some of the new techniques which have been developed to decrease the costs and increase the efficiency of maintenance in IS.

1. INTRODUCTION

CASE, or computer-aided software engineering, is a tool for programmers, analysts, and system engineers, as well as for bussiness planners and executives at all levels. It provides software tools to help planners to plan and document their work; to support systems analysts in analyzing and designing systems, and in documenting those tasks; and to take some of the drudgery out of programming while documenting it.

As the notion of CASE evolved over the last five years, the definition of a CASE tool broadened from meaning simple systems analysis and documentation tools to include fullfunction tools providing automated support for the entire software life cycle process. A CASE toolkit specializes in automating particular software tasks. CASE tool users frequently use several different toolkits as well as outside tools, such as fourth-generation languages, dictionaries, and DBMSes, during the software process. Selecting a CASE workbench involves matching hardware, development methodologies, and target systems that the workbench supports to the CASE user's development style and target system requirements.

Most companies spend 60 to 80% of their software budgets on maintaining existing applications. It's expected that fixing and reengineering existing applications will consume 90% of IS software resources by 1995. It's no wonder that companies are taking a hard look at their

software maintenance operations. They are specially interested in reverse-engineering products that assist in analyzing existing code to reconstruct original source information and specifications that have been lost or forgotten. Once reconstructed, those specifications can be fed into a code generator, which would reproduce the application in a new and improved structured form. Although reverse engineering sounds easy, it represents a major technical challenge.

2. CASE - SOFTWARE AUTOMATION

The crise in the development of software (SW) has made many people think about the ways of increasing productivity in this field. Current demands in dealing with the design of a new SW greatly surpasses the possibilities of the traditional approach in management, SW creation and documentation, and the cost of such development is not acceptable. The increase in the number of designers is not a good solution, since there are not enough designers. Even if one provides enough designers, the work on a large projects by using manual techniques would show that the managing of such a project is difficult, documentation incomplete, and it would be impossible to achieve that all the parts of the project fit. All these things become even worse if the number of designers working on the project is increased. Until recently, the main method used in the development of SW systems was the method of trying and making mistakes. The disadvantages of the traditional approach showed that well known engineering techniques should be used in the development of a SW product. SW engineering includes a systematic work in all stages: dealing with development, usage, maintenance, and withdrawal from usage of a SW product. Such product is characterised by the idea of life cycle and structural approach to the development. The idea of life cycle as the basic philosophy of SW engineering derives from the fact that every SW product goes through certain life phases. There are many opinions about the number and content of certain phases of life cycle, but disregarding the selected division, the quantity of required work on a particular project and the quantity of required documentation is constant. Among the others we have choosen the following division:

1. Project planning
2. Analyzing
3. Design
4. Programming
5. Testing
6. Application
7. Maintenance

The result of the computer aided SW engineering is the automation of SW development. Simply defined, CASE is SW automation. Most often, CASE means all the SW tools used in the development of SW. CASE may also be regarded as a combination of SW tools and structured methodology of SW development in which the tools automate SW processes, while the methodology defines the processes which are to be automated. The aim of CASE systems is not to reach the complete automatic development of SW, but only to automate this process in order to help people who work with them. There are many criteria for division of CASE systems, but we will classify them according to:

— universality and

— function of CASE system.

3.CLASSIFICATION OF CASE SYSTEMS ACCORDING TO THEIR UNIVERSALITY CASE

CASE Tools

CASE tools are a kind of graphic-oriented SW tools whose hardware (HW) environments are mainly microcomputers. In a broad sence, these are SW tools that provide automatic aid for development activities. CASE tools use very powerful graphics in order to describe and document SW systems as well as to promote user interfaces. CASE tools are so integrated that the data transmission between several tools is made possible without difficulties. Yet, there is no CASE tool that can provide complete automatic aid to development and maintenance of various SW systems.

CASE Toolkit

CASE Toolkit is a collection of integrated SW tools that provide complete automation of some phases of life cycle or automation of one function through several phases (as for example system analyses, logical database design, program design and implementation). Tools that constitute CASE Toolkit, use common information that are necessary for building and maintenance of SW systems. They also share the same user interface and the same interface between certain tools. According to the phase of development and the function which they automate, CASE Toolkits can be divided to those dealing with:

— analyses and design
These tools are used for the creation of SW systems specification. They include screen generating and report pointing, simulation and prototyping. Tools for analyses and design that have been written for IBM PC compatible computers are based on the following structural methodologies: De Marco's or Gane/Sarson's structured analyses, Martin's information engineering, etc.

— database design
This toolkit provides the automation of database design, as for example, making the logical database model,

generation of database scheme and description of database relation scheme.

— programming
These tools automate the steps in programming and testing. Some CASE Toolkits provide code generation based on informations from program specifications. Such CASE Toolkits are called code generators and they automatically produce completely documented programs ready to be used.

— maintenance and reengineering
These CASE Toolkits are directed to the existing SW systems. Such toolkits are made from tools for program analyses, reverse engineering, reconstruction, restructuring programs in use, and documentation. This toolkits are capable of maintaining a large number of program lines.

— project management
This type of CASE Toolkits support the planning function, controlling, managing and reporting, which are the characteristics of SW development and maintenance.

CASE Workbenches

CASE Workbench is an integrated collection of tools that support SW development through all development phases. When CASE Workbenches are combined with hardware (HW), the result is a workstation for SW development. While choosing CASE Workbenches, attention should be paid to system for which CASE Workbench is for. Also, it is important to take into account the demands of CASE Workbench users in the respect of development methodology and demands of the aim system.

4.CLASSIFICATION OF CASE SYSTEMS ACCORDING TO THEIR FUNCTIONS

In such a classification, serious problems arise due to lack of standards for CASE Systems division. Under these circumstances, different producers of the same type of CASE Systems put them in different categories or they put different types of CASE Systems in the same category according to their own criteria. Each producer tend to give his criteria as a standard. Despite the discussions about the reliability of functional division of CASE Systems, the following division can be accepted:

1. UPPER CASE
2. MIDDLE CASE
3. LOWER CASE

UPPER CASE

We refer to Upper CASE (UC) as a computer aided planning. Managers spend much of their time trying to understand the company, create plans for its activities, generate strategies for achieving the prescribed company's goals. They also establish standards for acceptable level of performance of the activities, which should result in fullfilment of the goals. Graphic diagrams are used to describe the company and its plans. According to these diagrams, the company's important aspects are decomposed and strategies planned. The planning methodology provides the structure into which the planning attributes have to be entered. For different plans, the structure of the company and its requirements remain the same, only the planning attributes values change. Every plan depends on timely informations to ensure its success. The use of UC products in planning gives a wide range of opportunities in reusing the earlier description and shortens the time required for designing a new or revising the existing plan.

MIDDLE CASE

A Middle CASE (MC) products are very helpfull in system analyses and design and belong to the first generation of CASE products as well as the UC. MC analyzes the problems and designs the solution for them. Most MC systems consist of diagraming and dictionary components, which are based on different methodologies. These systems are also used for symbolic description of the real world objects, using graphic diagrams. MC stores the type of knowledge that usually resides only in the minds of system analysts, e.g. the understanding how the company functions and what are its information needs. Thus the experience gained through years and the knowledge about companies functioning become more accessible. One can get these informations simply by retrieval of design specifications. Since MC specifications predominantly involve documenting a company's activities and the ways in which information serves it, only a small percentage of these specifications is directly mapped into Lower CASE systems (25-30%).

LOWER CASE

Lower CASE (LS) products are used in later phases of system development, to create a program specifications, the programming codes and the documentation related to the program specifications. LC products rarely provide a graphic component, since they are oriented on programming. However, it contains a data dictionary which enables one to enter specifications of the modelled real world objects. An active dictionary comprises three major components:

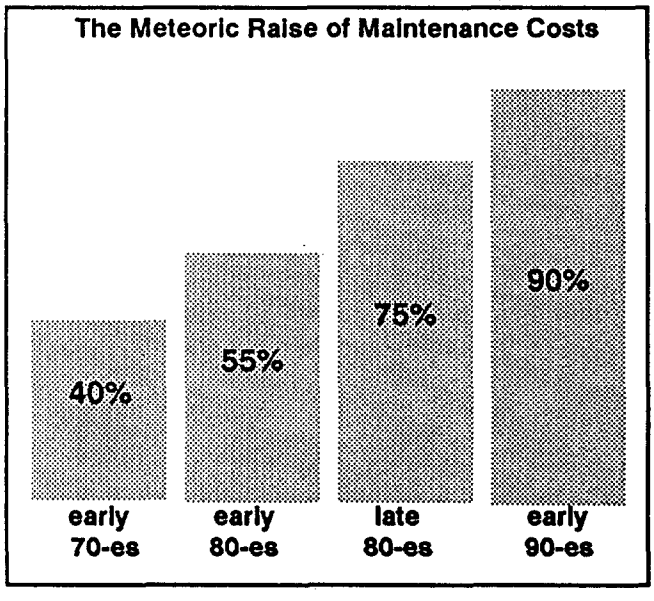
—a database in which to store the characteristics of the computing environment and application systems;
—frameworks for procedural logic and specific types

of procedural commands and modules contained within typical programs in the application systems; and — an activator capable of combining environmental and application characteristics with selected procedural - command frameworks and models to produce application programs.
LC products can generate both development and user documentation and make it available in various formats.

The usage of any category of CASE products demands a great deal of intellectual work, but the additional efforts and time, needed for completion of the SW design, are reversely proportional to the length and intensity of the CASE product implementation.

5.CASE SYSTEMS IN THE MAINTENANCE OF SOFTWARE

Majority of the SW development cost goes for its maintenance (up to 80 %). In USA this percentage is equal to $ 30 billions.



From this point of view, the efforts being made for reducing the maintenance costs are understandable, (especially in some branches as insurance, finance, telecommunications, etc., which are information-packed). We must not forget the fact that the maintenance quality is equally important for effectiveness and competitiveness of SW product as the development itself. Once this attitude is accepted, maintenance can't be regarded as a task of uninventive programmers or unsuccesful developing personell. New relations toward maintenance often can be seen through the following:

1. employment of the best team in maintenance jobs (since the savings made in this way are much bigger than the additional costs that could appear in the development field because of the migration of good teams)

2. the improvement of work organization related to maintenance, and

3. development of CASE systems that include tools for maintenance and reverse engineering.

Most of the existing CASE systems can not satisfy the needs of the designers and programmers working on maintenance: to provide the products that will enable reverse engineering of the existing application, using the advantages of CASE design, analysis and code generation. Even the producers of CASE systems themselves (IBM, TI, ORACLE Corp., etc.) admit that they still do not have the way to help the maintainers of the existing applications, although they have been working on such CASE tools designs.

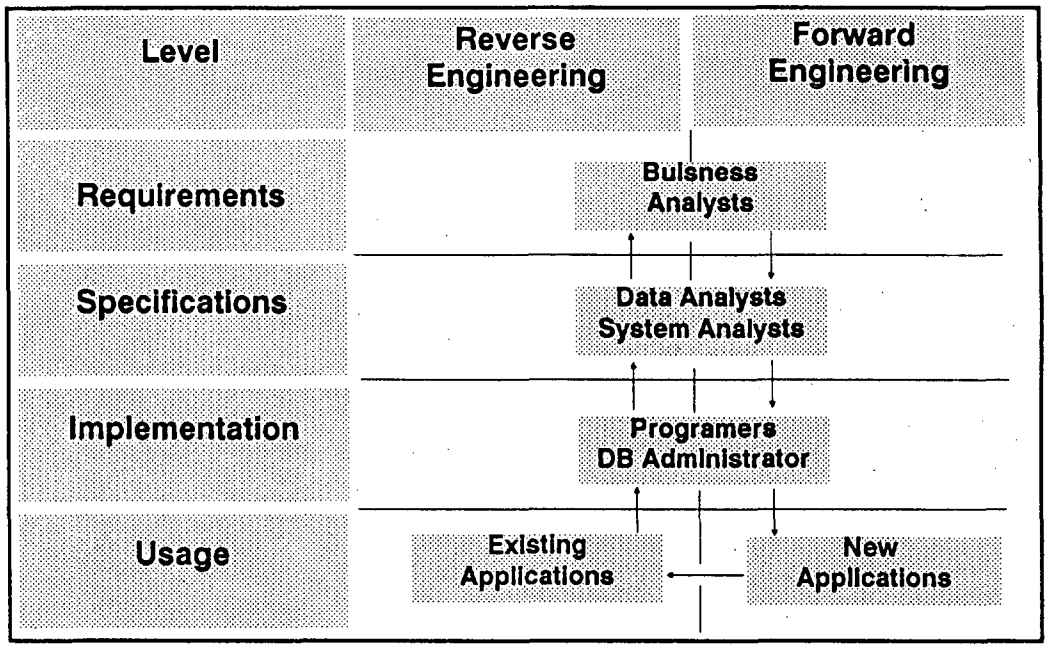
In these efforts their resources are, first of all, aimed to the design of tools for reverse engineering, that help in analyzing the existing source code. All this is aimed towards the reconstruction of original program specifications and the basic informations that have been lóst or forgotten long time ago. In this way reconstructed program specification can be put into code generator that will make the application in its new and better form, in the optional language of third or fourth generation.

Basic CASE concepts and products are directed towards the development of new applications and they are applied in scientific institutions and in a number of large systems. On the contrary, the new generation of CASE tools should include the possibility of fast change of existing applications, in accordance to the demands of market changes and new computer technologies. Such demands impose the spread of CASE concepts in respect of maintenance to include both improvement and upgrading the existing applications. IBM applications themselves, written in COBOL, include 77 million lines of source code, which speaks well enough about the dimension of maintenance and improvement of existing applications.

Reverse engineering enables separation of rules from all applications and uses them as a base for maintenance and improvement of these applications. Very often source code of an old application is not enough for the reverse engineering and additional informations about conditions of object system is also needed. It is impossible to expect from an individual or even a group to find out all the incomplete or missing components of the IS in use, so an Expert System as a part of a CASE tools could be of some help in solving this problem.

Classical CASE approach is based on top-down methodology of SW engineering, in which the information demands of a new IS have been worked out and documented. First step in this methodology includes carefull study and documentation of information demands, which are to satisfy information needs in a new IS. In the second step, the specifications are designed on the basis of study from previous phase. In the next step, programming and modelling of data are accomplished. Designer of database creates a database scheme that cor-

Re- Engineering Cycle



responds to the real system limits. Creation of program includes writing or generating a source code, compiling and testing. New CASE approach should enable an universal, simplier, and more effective way of IS development. Forward as well as reverse engineering, with the possibility of change of all informations in every common point would thus be included in CASE. Both of these engineerings together make a cycle of reengineering making possible reprojecting and improving the previously developed applications in an effective way. We expect these concepts to decrease the time needed for SW design.

6.WHY TO USE CASE SYSTEMS?

The benefits of CASE system's usage are great and they are the result of the CASE philosophy compounds. Models created by this systems allow better understanding of its functioning. The plans are more reliable, the decisions are more valid. The automation with CASE systems enables fast "what-if" analyses and choosing the scenario for better or worse case. During the system analysis, there is an opinion exchange on the relation designer end-users, so the diagrams and data dictionaries suit the end-user needs. The Middle CASE products give the possibility of prototyping by creating screens and reports which simulate the inputs and outputs of new application. Making prototypes in the phase of analysis and design, enables a simulation of searching and updating future database. It means that one can perceive the needs for information that some components of a system will satisfy, already in the phase of design. The Lower CASE products generate 60-80% of source code and considerably decrease the time required for SW development. Also these products make the program modification easier. The greatest benefit from Lower CASE usage, after all, is the possibility of prototyping during the development of complete application which is functioning as an independent system.

7. REFERENCES

MOGIN P. "*Karakteristike i uloga CASE alata u razvoju savremenih IS*", IIS - prolećni seminari informaciono-upravljačkih sistema '89

LAZAREVIĆ B. "*Razvoj i upotreba CASE alata*", *Metode i alati za projektiranje IS*, Opatija 1989

BACHMAN Ch. "*A CASE for Reverse engineering*", Datamation, July 1988

MCCLURE C. "*The CASE Experience*", Byte, April 1989

MCCLURE C. "*CASE is Software Automation*", Prentice Hall 1988

GIBSON M. L. "*The CASE Philosophy*", Byte, April 1989

GANE C., SARSON T. "*Structured Systems Analisys: Tools and Techniques*", Englewood Cliffs, Prentice Hall 1979

BEST L. J. "*Building Software Skyscrapers*", Datamation, March 1990

MOAD J. "*The Software Revolution*", Datamation, February 1990

MOAD J. "*Maintaining the Competitive Edge*", Datamation, February 1990

IMPLEMENTATION OF DENOTATIONAL SEMANTICS

INFORMATICA 1/91

M. MERNIK, V. ŽUMER
Technical Faculty of Maribor
University of Maribor
Smetanova 17, 62000 MARIBOR

Keywords: formal methods, semantics, rapid prototyping, functional languages

ABSTRACT : In the paper formal methods for semantics definition and reasons for their appearance are briefly described. Denotational approach to semantics definition is more detail explained in next chapter. Implementation of denotational semantics lead us to operational approach and to rapid prototyped interpreters. We represent one of the possible implementation of denotational semantics using functional language LISP.

KEYWORDS : formal methods, semantics, rapid prototyping, functional languages

1. INTRODUCTION

A programming language is a notation for describing computations and is defined with syntax, semantics and pragmatics. The structure of statements is given by syntax. The area of syntax has been intensively studied and Backus - Naur form (BNF) is widely used for defining syntax, because there exists a close correspondence between the BNF definition and parser. The meaning of statements is given by semantics and language implementation on specific computer is described by pragmatics. In this paper we briefly describe semantics definition methods. Unfortunately, the semantic area is not as well developed as the syntax area, because semantic features are much more difficult to define and describe. Semantics of early developed programmming languages were described in natural languages. This description has the advantage that semantics was easily understood but also many disadvantages such as: ambiguity, inaccuracy, misunderstanding, etc. So we need formal semantics definitions which, ensures us :-

- precise standards for a computer implementation, which guarantes that the language implementation is exactly the same on all machines ;
- usefull user documentation ;
- a tool for design and analysis ;
- input to compiler generator, which maps semantic definitions to a guaranted correct implementation of the language.

The effort, on formal methods for semantics definition, done in 70-ties have brought us three distinct and complementary description methods : operational, denotational and axiomatic semantics [3,4,6,8].

The operational semantics method uses an interpreter to define a language. The meaning of a program is the evaluation history that the interpreter produces when it interprets the program.

The denotational approach to semantics makes use of mappings, which are called semantic valuation functions. These map syntax constructs into their abstract mathematical counter part ; thus numerals are mapped into numbers, procedures are mapped into mathematical functions, and so on. The denotational definitions are more abstract then operational.

With the axiomatic semantics method, the meaning of program is not explicitly given at all. Instead, properties about language constructs are defined. These properties are expressed with axioms and inference rules. Axiomatic definitions are more abstract then denotational and operational.

All three methods together provide a set of tools for language development. Designers might first define properties that they wish the system

to have, so axiomatic definition is constructed first. Next, a denotational semantics is defined to give the meaning of the language. Finally, the denotational definition is implemented using an operational definition. These complementary semantic definitions of a language support systematic design, development and implementation. In the paper we describe denotational approach and one of its possible implementation using functional programming language LISP.

2. DENOTATIONAL SEMANTICS

Denotational semantics [1,2,3] is a methodology for giving mathematical meaning to programming languages. To make things more understandable an example of denotational semantics for a simple language of binary numbers is given first. Its syntax is defined with :

$$\nu ::= \nu\ \delta \mid \delta$$
$$\delta ::= 0 \mid 1$$

The language of binary numbers is composed by sequence of digits 0 and 1. To define the meaning of statement a valuation semantic function V which maps sequence of digits into their meaning is constructed. We say that the domain of V is a sequence of digits and the codomain are integers which represents meanings of digits. Mathematical notation for the function V is :

V : Bin -> Integer

We must define an instruction, which maps the domain into the codomain for each BNF rule :

V[0] = 0
V[1] = 1
V[νδ] = 2 * V[ν] + V[δ]

The next example, shows the semantic valuation function V on work :

V[1001] = 2 * V[100] + V[1]
= 2 * 2 * V[10] + V[0] + 1
= 2 * 2 * 2 * V[1] + V[0] + 0 + 1
= 2 * 2 * 2 * 1 + 0 + 0 + 1
= 9

Therefore the denotational approach to languages definition must make clear several sets of quantities :

- the syntax rules,
- the definition of the various semantic functions,
- the syntax domains,
- the semantic domains.

For describing syntax rules the abstract syntax is used. It specifies the relations between logical parts of the language and can be simpler than concrete syntax, and may not contain enough information to parse the language unambiguously. On contrary, concrete syntax contains sufficient information to parse a language.

λ-notations is used for describing semantic functions. To show the advantage of a λ-notation following definitions must be stated first. Function f(x) = y is equivalently described in λ-notation with f = λx.y . In this notation all functions usually have only one argument or one parameter. Consider a function f with two variables x and y. A function g with the property g(x)(y) = f(x,y) can be defined and is in some sense equivalent to f. The function g is called the curried version of f. Function which produced another function on its output is called the high order function. λ-notation has following advantages :

- a high order function is easier to describe ;
  example :
  add : ℕ -> ℕ -> ℕ
  standard notation :
  add(n) = f , where f(m) = n+m
  λ-notation :
  add = λn.λm.n+m

- a function can be defined without giving it a name. Such function is called the anonymous function.

Several additional notations may be used in expressions of semantic equations, such as "update" expressions of the form [a ↦ b]f, which denotes a function f which is the same as the function f except, that it maps the argument a into the value b. Conditional expressions are written in the form "t -> e_1 □ e_2" where the value of the expression e_1 is evaluated if and only if the boolean expression t is true and the value e_2 is evaluated otherwise.

Syntax domains of imperative languages are usually identifiers, expression, commands and definitions. Semantic domains are most conventionally described with an environment and a store. The environment is a finite set of associations of identifiers with values they denote (denotable values) and the store is a

finite set of locations with values they contain (storable values). The store will be sufficient for simple languages where equal identifier can not be used for different objects.

And now, let us define the meaning of identifiers. Identifiers may stands for location, label, procedure and theirs meaning are represent by semantic domain environment which is a function from identifiers to denotable values. If identifier stands for location , a second semantic domain is used to get value they contain. The store is therefore a function from locations to storable values.

The meaning of commands is to change the store, the meaning of a definition is to change the environment and the meaning of an expression is to produce a value (expressible value).

3. IMPLEMENTING DENOTATIONAL SEMANTICS USING LISP

A great advantage of the denotational approach is that can be mechanized and interpreted on a computer [1,2,7]. In this chapter a small and a simple example is described with the denotational semantics first and then implemented using functional language LISP. LISP is selected because high order functions are relatively simple to express using it.

The abstract syntax for a simple imperative language is given first :

P ∈ Program
B ∈ Block
D ∈ Declaration
C ∈ Command
E ∈ Expression
I ∈ Identifier
N ∈ Numeral

P ::= B.
B ::= begin D & C end
D ::= D_1 & D_2 | var I
C ::= C & C_2 | I:=E |
if E then C_1 else C_2 | B
E ::= E_1 + E_2 | I | N

Program is a block of statements and each block consists of a declaration part and a command part. Commands are : assignment statement, if statement and block of statements. Expressions are : identifiers, numerals and composed expressions. Delimiter symbol between statements is a character &.

Semantics are described as follows :
Domains and operations on it are :

I. Natural numbers n ∈ N
II. Identifiers i ∈ Id
III. Location l ∈ Loc
reserve_loc : Loc
reserve_loc = random 1000
IV. Denotable values d ∈ D_values = Loc
V. Environment e ∈ Env = Id -> D_values
emptyenv : Env
emptyenv = λi.0
accessenv : Id -> Env -> D_values
accessenv = λi.λe.e(i)
updateenv : Id -> Value -> Env -> Env
updateenv = λi.λd.λe.[i ↦ v]e
VI. Storable values v ∈ S_value = N
VII. Store s ∈ Store = Loc -> S_value
emptystore : Store
emptystore = λi.0
access : Loc -> Store -> S_value
access = λi.λs.s(i)
update : Loc -> S_value ->
Store -> Store
update = λi.λv.λs.[i ↦ v]s
VIII. Expressible values x ∈ Exp_vaules = N

Semantic valuation functions is given next :
P: Program -> Store
P[B] = B[B] emptyenv emptystore

The meaning of a program is defined with a store, which is the output from function B. The function B needs environment and store on the input. The function B is applied on initialized environment and store first.

B: Block -> Env -> Store -> Store
B[begin D & C end] = λe.λs.C[C] (D[D] e) s

The meaning of a block is the meaning of commands, which is evaluated on environment and store. The environment is changed by function D first.

D: Declarations -> Env -> Env
D[D_1 & D_2] = λe.D[D_2] (D[D_1] e)
D[var I] = λe.updateenv [I] reserve_loc e

The meaning of a declaration is to change the environment. When an identifier declaration is evaluated, new location is produced first and then new updated environment is returned. This updated environment has a property that it maps an identifier to a new location.

```
C: Command -> Env -> Store -> Store
C[B] = λe.λs.B[B] e s
C[C1 & C2] = λe.λs.C[C2] e (C[C1] e s)
C[I:=E] = λe.λs.update (accessenv [I] e)
                       (E[E] e s) s
C[if E then C1 else C2] = λe.λs.E[E]s = 0 ->
                                  C[C1]e s □
                                  C[C2]e s
```

The meaning of a command is to change the store. The meaning of an if statement is to evaluate the expression E first and then appropriate command is selected and evaluated. The meaning of an assignment statement is :

- a location for an identifier is evaluated (accessenv [I] e) .
- an expression E is evaluated (E[E] e s) .
- a location is updated with value produced by the function E
  (update (accessenv [I] e) (E[E]e s) s) .

```
E: Expression -> Env -> Store -> Exp_value
E[E1+E2] = λe.λs.E[E1]e s + E[E2]e s
E[I] = λe.λs.access (accessenv [I] e) s
E[N] = N[N]
```

The meaning of an expression is to produce a new value. Semantic valuation function N is similar to function V , which is introduced in chapter 2. Now, we can apply syntax semantic valuation functions to syntax construct to get their meaning.

Example :

```
P[begin var i; i:=10; begin var i; i:=20 end;
   i:=i+1 end] =
B[begin var i; i:=10; begin var i;  i:=20  end;
  i:=i+1  end]  emptyenv emptystore =
C[i:=10; begin var i; i:=20 end; i:=i+1]
  (D[var i]emptyenv) emptystore =
C[i:=10; begin var i; i:=20 end; i:=i+1]
 (updateenv [i] reserve_loc emptyenv)
   emptystore =
C[i:=10; begin var i; i:=20 end; i:=i+1]
  [i ↦ 1]emptyenv emptystore =
C[begin var i; i:=20 end; i:=i+1]
  [i↦ 1]emptyenv (C[i:=10] [i ↦ 1]emptyenv
    emptystore) =
C[begin var i; i:=20 end; i:=i+1]
  [i↦ 1]emptyenv (update (accessenv [i]
    [i ↦ 1] emptyenv) (E[10] [i↦ 1]emptyenv
      emptystore) emptystore) =
C[begin var i; i:=20 end; i:=i+1]
[i↦ 1]emptyenv (update 1 10 emptystore) =
C[begin var i; i:=20 end; i:=i+1] [i↦ 1]emptyenv
  [1 ↦ 10]emptystore =
 let e1= [ i↦ 1]emptyenv
 let s1= [1 ↦ 10]emptystore
C[i:=i+1] e1 (C[begin var i; i:=20 end] e1 s1 ) =
C[i:=i+1] e1 (B[begin var i; i:=20 end] e1 s1 ) =
C[i:=i+1] e1 (C[i:=20] (D [var i] e1) s1) =
C[i:=i+1] e1 (C[i:=20]
  (updateenv [i] reserve_loc e1) s1 ) =
C[i:=i+1] e1 (C[i:=20] [1 ↦ 2]e1 s1 ) =
C[i:=i+1] e1 (update (accessenv [i] [1 ↦ 2]e1)
  (E[20] s1) s1 ) =
C[i:=i+1] e1 (update 2 20 s1 ) =
C[i:=i+1] e1 [2 ↦ 20]s1 =
  let e2= [ i↦ 2]emptyenv
  let s2= [1 ↦ 10, 2 ↦ 20]emptystore
update ( accessenv [i] e1 ) (E[i+1] e1 s2) s2 =
update 1 ((acess (accessenv [i] e1 )s2) + 1) s2 =
update 1 (access 1 s2)+1 s2 =
update 1 10+1 s2 =
update 1 11 s2 =
[1 ↦ 11, 2 ↦ 20]emptystore
```

The meaning of the above program is the store, which map location 1 to value 11 and location 2 to value 20.

Implementation of semantic valuation functions is shown in appendix . Now, we can analyze any semantic valuation function.

Examples :

```
(setq new_env
 (funcall (D '(var i & var j & var k)) emptyenv))

        =>  ( is evaluated to )

   ( (i 599) (j 41) (k 855) )
```

Function D changed the environment. The new environment maps identifier i to location 599, identifier j to location 41 and identifier k to location 855.

```
(funcall (funcall (C '( ( i := (1)) &
                        ( j := (2)) &
                        ( k := ((i) + (j)))
                       )
                  ) new_env ) emptystore)

        =>  ( is evaluated to )

   ( (599 1) (41 2) (855 3) )
```

Function C changed the store. The new store maps location 599 to value 1, location 41 to value 2 and location 855 to value 3.

```
(setq prog '(begin (var i & var j & var k) &
             ( (i := (1))  &
               (j := ((i) + (1))) &
               ( begin (var i & var l & var m) &
                       ( (i := (20)) &
                         (m := ((i) + (1))) &
                         (l := ((j) + (1)))
                       )
                   end ) &
               (k := ((i) + (2)))
             )
            end ))


 ( P prog )

    =>  ( is evaluated to )

( (599 1) (41 2) (784 20) (503 21) (981 3)
 (855 3) )
```

We can understand the store only with the environment. The environment for that example is :

((i 599)(j 41)(k 855)(i 784)(l 981)(m 503))

4. CONCLUSION

The implementation of semantic valuation functions has many advantages [1,2,5,7] :

- each semantic valuation function can be separately tested and analyzed,
- a rapid prototyped interpreter for the language is produced
- all advantages of prototyped systems such as :
  - greater user participation,
  - early detection of errors,
  - better user - developer communications,
  - early delivered executable systems to the users,
  - help by suppressing uncertainties in user requirements (I'll know what I want, when I see it !).

5. REFERENCES

1. Schmidt D.A., "Denotational semantics", Allyn and Bacon, Newton, 1986.

2. Allison L., "A practical introduction to denotational semantics", Cambridge university press, Melbourne, 1986.

3. Tennent R.D., "Principles of Programming Languages", Prentice Hall International, London, 1981.

4. Horowitz E., "Fundamentals of Programming Languages", Computer science press, Rockville, 1983.

5. Agresti W.W.,"New paradigms for software development",IEEE Computer society press, New York, 1986.

6. Mernik M., Kokol P., Žumer V.,"Formalne metode za opis semantike programskega jezika", XIV. Simpozijum o informacionim tehnologijama, Sarajevo - Jahorina 1990, pp. 181-1 - 181-10.

7. Mernik M., Kokol P., Žumer V.,"Π-BOSS: A new rapid prototyping paradigm for language design", Proceedings of International Conference on Computing and Information, Niagara Falls, pp. 504 - 507, 1990.

8. Marcotty M., Ledgard H.F., Bochmann G.V., "A Sampler of Formal Definitions", Computing Surveys, Vol. 8., No. 2., 1976.

APPENDIX : SEMANTIC VALUATION FUNCTIONS IN LISP

```
;; Env = Id -> D_value
;; Environment is modeled with association list
;;  ( (id1 loc1) ... (idn locn) )

;; accessenv: Id -> Env -> D_values
 (defun accessenv ()
  #'(lambda (i)  #'(lambda (e) (cadr (assoc i e))))

;; updateenv: Id -> D_value -> Env -> Env
 (defun updateenv ()
  #'(lambda (i)
   #'(lambda(d)
    #'(lambda (e) (putassoc e i d)))))

;; Store = Loc -> S_value
;; Store is modeled with association list
;;    ( (loc1 value1) ... (locn valuen) )

;; access : Loc -> Store -> S_value
 (defun access ()
  #'(lambda (l)
   #'(lambda (s) (assocl l s))))

;; update : Loc -> S_value -> Store -> Store
 ( defun update ()
  #'(lambda (l)
   #'(lambda (v)
    #'(lambda (s) (putassoc s l v)))))
```

```
;; reserve_loc is modeled with random numbers
 (defun reserve_loc ()  ( random 1000 ))

;; environment initialization
 (defun emptyenv ()   nil )

;; store initialization
 (defun emptystore ()  nil )

;; P: Program -> Store
 ( defun P (Program)
   (funcall (funcall (B  Program) (emptyenv))
     (emptystore)))

;; B: Block -> Env -> Store -> Store
 ( defun B (Block)
  #'( lambda ( e )
   #'(lambda ( s ) (funcall (funcall (C (cadddr
    Block)) (funcall (D (cadr Block)) e)) s))))

;; D: Dec -> Env -> Env
 (defun D (Dec)
  #'(lambda (e)
     (cond ((equal (caddr Dec) '&)
        ;; composed declaration
       (funcall (D (cdddr Dec)) (funcall
        (D (list (car Dec)  (cadr Dec))) e)))
      ;; identifier declaration
      (t (funcall (funcall (funcall
  (updateenv) (cadr Dec)) (reserve_loc)) e )))))
;; C: Comm -> Env -> Store -> Store
 (defun C (Comm)
  #'(lambda (e)
   #'(lambda (s)
     (cond ((not (equal (cdr Comm) nil))
       ;; composed commands
       (funcall (funcall (C (cddr Comm)) e)
        (funcall (funcall (C (list (car Comm)))
          e) s)))
      ;; if statements
      ((equal (caar Comm) 'IF)
         (cond ((equal (funcall (funcall
                (E (cadar Comm)) e) s) 0)
           (funcall (funcall (C (list
             (cadddr (car Comm)))) e) s))
          (t (funcall (funcall (C (list
              (caddddr (car Comm)))) e) s))))
       ;; block statements
       ((equal (caar Comm) 'BEGIN)
          (funcall (funcall (B (car Comm)) e) s))
        ;; assignment statement
        (t (funcall (funcall (funcall
          (update) (funcall (funcall
          (accessenv) (caar Comm)) e))
          (funcall (funcall (E
            (caddar Comm)) e ) s)) s))))))

;; E: Exp -> Env -> Store -> Exp_value
 (defun E (Exp)
  #'(lambda (e)
   #'(lambda (s)
     ;; number
     (cond ((numberp (car Exp)) (car Exp))
     ;; Exp + Exp
     ((equal (cadr Exp)  +)
        (+ (funcall (funcall (E (car Exp)) e) s)
          (funcall (funcall (E (caddr Exp))
           e) s)))
      ;; identifier
      (t (funcall (funcall (access)
              (funcall (funcall (accessenv )
               (car Exp)) e)) s))))))
```

SIGNIFICANCE LEVEL BASED CLASSIFICATION WITH MULTIPLE TREES

INFORMATICA 1/91

Keywords: machine learming, artificial intelligence, level, ASSISTANT.

Aram Karalič, Vlado Pirnat
Jožef Stefan Institute, Ljubljana

ABSTRACT: Usually, algorithms for machine learning during the classification return a single class for a given object. Many of the systems do not estimate a reliability of their answer. In the article a method is presented that returns multiple classes as possible. The method also gives the user an estimation of the answer's reliability. Additionally, the method enables also classification in domains where one example can belong to more than one class. The described ideas are tested on a real medical domain — rheumatology. The results are compared with the results of the classical algorithms for machine learning and with the results of general practitioners.

POVZETEK: KLASIFIKACIJA Z VEČ DREVESI BAZIRANA NA STOPNJI ZAUPANJA. Algoritmi za avtomatsko učenje ponavadi pri klasifikaciji neznanega primera podajo samo en razred. Mnogo sistemov ne oceni zanesljivosti svojega odgovora. V članku je podana metoda, ki poda več razredov kot možne. Metoda poda tudi zanesljivost svojega odgovora. Dodatna prednost opisanega pristopa je, da omogoča učenje tudi v domenah, kjer en učni primer lahko pripada več razredom hkrati. Opisane ideje so preverjene na realni medicinski domeni — revmatologiji. Podana je tudi primerjava rezultatov metode z rezultati splošnih zdravnikov.

1 INTRODUCTION

The task of machine learning from examples is usually defined as follows:

> Given: A set of learning examples, described in terms of attributes and their values. Every example belongs to one class. Attributes have symbolic values (discrete attributes) or real values (continuous attributes).
>
> Find: A decision rule that fits the learning set and maps every (previously unseen) example into probability distribution:
>
> $$y = (p_1, p_2, \ldots, p_n) \quad (1)$$
>
> where component p_i of the vector $y \in \Re^n$ is an estimation of the probability that the example belongs to class C_i.

An algorithm for machine learning gives an estimation of the probability distribution (1) over classes. It also assumes that one example belongs to exactly one class.

A decision rule usually consists of a *knowledge base* and a *classification algorithm*. The knowledge base is constructed by a *learning algorithm* in a process of learning. The classification algorithm *uses* the knowledge base to obtain a class-probability distribution of an unseen example. Thus, with different classification algorithms the same knowledge base can be interpreted in several ways.

In this article only the algorithms that construct knowledge base in the form of decision trees (e.g. ASSISTANT [5,3,7], ID3 [8], etc.) are discussed. Nevertheless, the described ideas can be generalized also to algorithms with different knowledge representations.

The two situations that can often occur in the context of machine learning in real-world domains and that are not treated properly by the existing learning systems are the following:

1. It is often the case that class probabilities have to be estimated from relatively small number of examples. Let us consider the case with 3 classes. It is possible that when classifying an unseen example the algorithm classifies it in the leaf which corresponds to three learning examples, all belonging to class C2. The system's answer is: (0.00 1.00 0.00), saying that our new example belongs to class C2 with probability 1. But it is obvious that this probability estimate is extremely inaccurate, because it was calculated only from three learning examples. It was also stated in [2] that the probability estimation is one of the crucial tasks in certain subareas of machine learning.
2. In many of the problems suitable for the application of machine learning example can belong to more than one class. In medicine, for example, a patient can have more than one disease simultaneously. In such domains systems act with incorrect assumption, that the classes are disjoint, which can decrease the performance of the system.

To solve the above situations we propose the following:

1. We believe that it would be more convenient for a user if a system, in the case of inability to give an accurate answer, explicitly answers "I don't know" instead of giving inaccurate information. An appropriate measure of the answer's reliability is necessary to accomplish this task.
2. In domains where classes are not necessary disjoint it is reasonable to drop the assumption of disjoint classes. This means that the relation

$$p_1 + \ldots + p_n = 1$$

no longer holds. The system should rather return the probability for each class independently.

In this article a method called MULTI-α is described. It was designed so as to overcome the above mentioned weak points. It is described in section 2. Section 3 describes experiments in the real-life medical domain — rheumatology. The results of computer-based diagnosing are compared with those achieved by general practitioners. The testing of performance of physicians is also described in section 3. In section 4 advantages of MULTI-α method are summarized and some ideas presented for its practical use.

2 DESCRIPTION OF MULTI-α METHOD

2.1 Learning Algorithm

The basic idea of our approach is to generate a decision tree (decision rule) with classes "C_i" and "$\neg C_i$" for each class C_i separately. That is how the assumption about disjoint classes can be avoided. In fact, the problem is divided into n (n is the number of classes) subproblems. Knowledge about the domain now consists of n decision trees $T_1, \ldots, T_n$. Every decision tree tells something about one of the classes. Similar approaches to building a knowledge base are described in [1,6]. Our approach differs from the above-mentioned ones in a way of how it classifies unseen examples.

2.2 Classification Algorithm

A version of learning algorithm should be used, which produces trees, whose leaves (besides probability distribution) also include the number of learning examples. When classifying example E each class C_i is marked with $\oplus$, $\ominus$ or $\odot$. The meaning of this labels is the following:

- $\oplus$ — "It is possible that the example belongs to this class.",
- $\ominus$ — "It is not possible that the example belongs to this class.",
- $\odot$ — "Nothing can be accurately said about this class.".

This three claims are made with certain degrees of reliability. The whole answer is composed of answers of individual trees, and looks as follows:

$$(x_1, x_2, \ldots, x_n), \alpha$$

Where n is number of classes, $x_n \in \{\oplus, \ominus, \odot\}$ and α is level of significance, described later. Such an answer enables a user to judge the value of the system's answer, which is especially important in "soft" domains like, for example, medicine.

The assignment of $\oplus$, $\ominus$ or $\odot$ to class C_i is made by the following algorithm:

- Classify example with a tree T_i.
- In a leaf, in which an example E is classified, let k be the number of examples belonging to class "C_i" and let the number of all examples in the leaf be N. The relative frequency R_i of the class C_i is $R_i = k/N$ and the system's answer is:

$$(N|R_i, 1 - R_i)$$

- If $R_i > 0.5$ we suspect that the example belongs to class C_i. Let p be the actual probability of class C_i in a given population. The hypothesis $H_\ominus$:"$p < 0.5$" is formulated, saying: "probability of the example belonging to the class is less than 0.5". Interpretation of $H_\ominus$ could be: "the example probably doesn't belong to the class". Now, we try to reject the hypothesis. If we succeed in rejecting it we can, with a certain level of significance, believe that the example belongs to the class. In this case we mark the class with $\oplus$. If we can not reject the hypothesis we believe that we can not make any statistically significant claim about the example belonging to the class. We inform the user of this by marking the class with a $\odot$.
- If $R_i < 0.5$ we formulate the hypothesis $H_\oplus$:"$p > 0.5$" (meaning: "example belongs to the class") and try to reject it. If we succeed it means that we can (with certain level of significance) claim that the example does not belong to the class, so we mark the class with $\ominus$. If we can not reject the hypothesis we can not make any statistically significant claim about the example belonging to the class and inform the user of this by marking the class with a $\odot$
- If $R_i = 0.5$ the class is simply marked with $\odot$.

Let us now explain the statistical test that is based on the principles of statistical tests explained in [4]. The test is explained only for the case when $R_i > 0.5$. The other situation ($R_i < 0.5$) is handled analogously.

Recall that $R_i = k/N$. The leaf can be seen as a series of N experiments, where event C_i occured k times (C_i = "example belongs to class C_i"). We formulate a hypothesis about the probability of event C_i:

$$H_\ominus : p(C_i) < 0.5$$

and try to reject it.

Let R be a random variable denoting a relative frequency of event C_i in a series of experiments (R_i is its value in our series of experiments). The hypothesis can be rejected with risk α when:

$$p(R \geq R_i / H_\ominus) < \alpha$$

Where "/" indicates conditional probability. Let us calculate the value of $p(R \geq R_i/H_\ominus)$:

$$\begin{aligned} p(R \geq R_i/H_\ominus) &= p(R \geq R_i/(p < 0.5)) \\ &= \sum_{i=k}^{N} \binom{N}{i} p^i (1-p)^{N-i} \end{aligned}$$

The actual probability p is not known. But we know, that it is between 0 and 0.5. Considering that

$$(0 \leq p < 0.5) \wedge (N/2 < i \leq N) \Rightarrow$$

$$\begin{aligned} p^i(1-p)^{N-i} &= p^{2i-N}p^{N-i}(1-p)^{N-i} \\ &= [p(1-p)]^{N-i}p^{2i-N} \\ &\leq [1/4]^{N-i}p^{2i-N} \\ &< (1/2)^{2N-2i}(1/2)^{2i-N} \\ &= (1/2)^N \end{aligned}$$

$p(R \geq R_i/H_\Theta)$ can be estimated in the following way:

$$\begin{aligned} \sum_{i=k}^{N}\binom{N}{i}p^i(1-p)^{N-i} &< \sum_{i=k}^{N}\binom{N}{i}0.5^N \\ &= \sum_{i=k}^{N}\binom{N}{i}0.5^i(1-0.5)^{N-i} \\ &= p(R \geq R_i/(p=0.5)) \end{aligned}$$

The last expression can be easily calculated as follows:

$$p(R \geq R_i/(p=0.5)) = 2^{-N}\sum_{i=k}^{N}\binom{N}{i}$$

So, we computed the upper bound for $p(R \geq R_i/H_\Theta)$. Let's denote it with $p'(R \geq R_i/H_\Theta)$:

$$p(R \geq R_i/H_\Theta) < p'(R \geq R_i/H_\Theta) = 2^{-N}\sum_{i=k}^{N}\binom{N}{i}$$

If $p'(R \geq R_i/H_\Theta) \leq \alpha$ (which means $p(R \geq R_i/H_\Theta) < \alpha$) then we reject the hypothesis and declare the diagnosis as "possible" (mark it with $\oplus$).

3 EXPERIMENTS

ASSISTANT [5,3,7], a system for inductive learning of decision trees, was used for generating the classification trees. ASSISTANT classifies an example into class C_i with the highest p_i in the corresponding probability distribution (1). An experiment with the described method of classification was performed and the obtained results were compared with the results of the classical use of ASSISTANT. Parameter settings for the ASSISTANT, which are described in [3], are displayed in Table 1. When describing experiments in medicine a term "patient" is often used instead of "example" and the a term "diagnosis" instead of "class".

ALL Instances Selected	
Instances for Testing	: 30 %
Pruning Factor	: 3.0 x
Best Class Threshold	: 100 %
Weight Threshold	: 0 %
Post Pruning	: YES

Table 1: ASSISTANT's learning parameters.

3.1 Domain Description And Experimental Data

The data for the patients were collected at the Rheumatological Clinic of the University Clinical Center in Ljubljana. If a diagnosis after the first examination of a patient was unclear the patient was re-examined many times during one year to obtain the reliable diagnosis.

Experiments were performed on four different diagnostic problems. The first (and the easiest) problem is to classify a patient into one of the three possible diagnoses. The other three problems have six, eight and twelve possible diagnoses. The percentage of majority class for each problem is given in Table 2. Each example is described in terms of 16 anamnestical attributes, 37 clinical, 4 laboratory and 1 radiological attribute.

number of diagnoses	percentage of majority class
3	66.45
6	61.90
8	34.20
12	34.20

Table 2: Percentage of majority class for each diagnostic problem.

The data for 462 patients were collected. Data for anamnestical and clinical attributes were missing only for 10 attribute values. Laboratory data were missing in 44 cases and radiological in 211 cases.

Experiments were made for all four diagnostic problems (classification into one of 3, 6, 8 and 12 diagnostic groups). For every problem one tree was built for each diagnosis. Learning was performed on 70% of the patients, the rest (30%) was used for testing. Each experiment was repeated 10 times, so the final results are averages of 10 runs.

3.2 Testing Of MULTI-α Method

On the basis of the results of the statistical tests a set of possible diagnoses (SPD) was constructed. An answer of the system was defined as correct if SPD included the patient's diagnosis. SPD was constructed using three different strategies, called A, B and C.

When using strategy A all diagnoses marked with $\oplus$ or $\odot$ were included into SPD. When using strategy B diagnoses marked with $\oplus$ were included into SPD and when using strategy C only the diagnose marked with $\oplus$ whose hypothesis was rejected with the smallest α was included into SPD (this corresponds to the most probable diagnosis). In all cases the SPD was checked for emptiness. If it was found empty, the "most probable" diagnose was added to the SPD.

Strategy A typically produces the largest SPD, resulting in the highest percentage of correct answers. Strategy C is the most similar to the classic use of the ASSISTANT (select the most probable diagnosis). They differ only when several diagnoses were assigned the same risk factor. The SPD then contains more than one element. Strategy B is a compromise between the strategies A and C. If there are many examples in the domain that belong to more than one class, it is wiser to use strategy A; otherwise one should use strategies B or C.

We refer to strategies A, B and C also as MULTI/A, MULTI/B and MULTI/C respectively and to all three of them collectively as MULTI.

Three different levels of significance were used: 1%, 5% and 10%, but the results differed only slightly so only results for $\alpha = 5\%$ are presented in the article.

When comparing the results of ASSISTANT and MULTI-α, a problem can arise due to a fact that ASSISTENT always classifies in one class. The measure of success of ASSISTANT classifi-

cation is classification accuracy, which is defined with an assumption, that the system always suggests only one diagnosis. But in MULTI-α classification the system can suggest more diagnoses to be the possible ones. So, the classification accuracy as defined in ASSISTANT classification does not exist. Therefore one can not make any exact comparison (e.g. with t-test), between the results. Nevertheless, results of ASSISTANT classification can be compared with the results of C strategy of the MULTI classification (which is the most similar to ASSISTANT classification) just to get the impression about the performance.

To further reduce the dissimilarity between the two measures of performance, each correct answer was weighted with a $1/|SPD|$. This technique is also implemented in original ASSISTANT [5].

In the experiments the *weighted* percentage of correct answers, which we denoted with *acc*, the size of the SPD (denoted with *siz*) and the percentage of cases when |SPD| was more than one (denoted with *MT1*) were measured. The mean values and standard deviations were measured for each parameter. Results of the experiments are presented in Table 3.

strategy		3 diagnoses	6 diagnoses	8 diagnoses	12 diagnoses
A	acc	70.5 ± 2.7	64.8 ± 2.7	48.9 ± 2.0	47.9 ± 2.1
	siz	1.2 ± 0.1	1.2 ± 0.0	1.4 ± 0.1	1.4 ± 0.1
	MT1	20.0 ± 9.4	15.1 ± 4.2	30.1 ± 7.8	29.2 ± 5.3
B	acc	71.7 ± 3.2	65.7 ± 3.6	51.2 ± 3.0	50.85 ± 3.5
	siz	1.1 ± 0.1	1.1 ± 0.0	1.2 ± 0.1	1.2 ± 0.0
	MT1	12.5 ± 6.1	8.9 ± 3.4	17.7 ± 4.0	16.4 ± 2.3
C	acc	71.6 ± 3.4	65.5 ± 3.5	51.4 ± 3.5	51.2 ± 3.7
	siz	1.1 ± 0.1	1.1 ± 0.0	1.2 ± 0.1	1.2 ± 0.0
	MT1	11.8 ± 6.4	7.6 ± 3.2	14.1 ± 3.8	13.8 ± 2.1

Table 3: Multiple tree classification.

The comparison between ASSISTANT and MULTI/C is summarized in Table 4. It can be seen that the weighted number of correct answers using MULTI/C method is typically larger than the number of correct answers using ASSISTANT.

number of diagnoses	ASSISTANT	MULTI/C
3	70.4 ± 3.2	71.6 ± 3.4
6	64.7 ± 3.0	65.5 ± 3.5
8	49.9 ± 3.5	51.4 ± 3.5
12	47.8 ± 3.6	51.2 ± 3.7

Table 4: Comparison between ASSISTANT and MULTI/C method.

The classification accuracy on testing examples was compared with the classification accuracy on learning examples. The results are summarized in Table 5. The Table shows that the classification on learning examples is better than on the testing examples. This indicates that the system did not have enough learning examples to accurately learn decision rules. With increasing number of learning examples we believe that the system could achieve similar results on testing examples as on learning examples.

number of diagnoses	MULTI/C, $\alpha = 0.05$		
	Lrn	Tst	Lrn/Tst
3	78.8 ± 2.2	71.6 ± 3.4	1.10
6	72.9 ± 2.2	65.5 ± 3.5	1.11
8	71.2 ± 2.0	51.4 ± 3.5	1.39
12	69.8 ± 1.9	51.2 ± 3.7	1.36

Table 5: Number of correct answers when classifying learning (Lrn) and testing (Tst) examples.

3.3 Comparing MULTI-α Method With Physicians

Because we wanted to compare the results of our system with the results that can be obtained from general practitioners 10 general practitioners were tested. Their task was to classify 30 randomly chosen patients from our set of patients. For each patient, the practitioners were presented with a description of the patient (attribute values). They had to assign one of the numbers 1, 2, 3, 4, 5 to every possible diagnose. Interpretation of number 1 was that this diagnose was surely incorrect, number 2, that there was a small possibility of this diagnose. Number 3 meant, that the doctor couldn't say anything accurate about the diagnose. Interpretation of number 4 was that this diagnose was very probable and number 5 that this diagnose was certain. Numbers 1 and 2 were mapped to $\ominus$, number 3 to $\odot$ and numbers 4 and 5 to $\oplus$. Patients were chosen so that their class distribution was as close as possible to the distribution of the whole set of patients. Our system classified the same 30 patients with knowledge, learned from the *remaining* 432 patients.

The results of the testing of physicians' performance are summarized in Table 6. Columns marked with *physicians* are the results of the physicians and columns marked with *MUL* are the results of our method with $\alpha = 0.05$. The computer-based classification always outperformed the practitioners.

strategy		3 diagnoses		6 diagnoses		8 diagnoses		12 diagnoses	
		physicians	MUL	physicians	MUL	physicians	MUL	physicians	MUL
A	acc	48.6 ± 14.8	61.7	51.9 ± 13.5	58.8	30.8 ± 7.1	41.7	17.4 ± 6.9	88.8
	siz	1.4 ± 0.4	1.1	2.8 ± 0.8	1.1	2.7 ± 1.0	1.8	3.4 ± 1.4	1.4
	MT1	85.7 ± 28.1	18.8	60.8 ± 84.4	10.0	74.0 ± 26.2	26.7	84.0 ± 19.7	80.0
B	acc	49.7 ± 14.8	61.7	57.9 ± 11.8	58.8	36.1 ± 7.4	41.7	22.7 ± 7.1	88.9
	siz	1.8 ± 0.2	1.0	1.6 ± 0.8	1.1	1.7 ± 0.4	1.8	1.7 ± 0.4	1.4
	MT1	25.0 ± 19.4	8.8	82.7 ± 21.9	6.7	46.0 ± 18.0	26.7	46.0 ± 17.8	80.0
C	acc	49.9 ± 14.8	61.7	58.1 ± 11.9	58.8	36.4 ± 7.2	44.7	28.1 ± 7.6	48.9
	siz	1.8 ± 0.2	1.0	1.8 ± 0.4	1.1	1.8 ± 0.4	1.3	1.4 ± 0.3	1.8
	MT1	24.0 ± 19.2	8.8	26.7 ± 19.8	6.7	36.7 ± 16.9	16.7	20.8 ± 10.0	20.0

Table 6: Results of the testing of physicians' performance.

4 CONCLUSIONS

We developed a method which enables learning in domains where one example can belong to *several classes*. We also improved the weak points of the systems that do not *estimate reliability of their answers*. Additional advantage of the method is that it does not contain any *probability estimates* and therefore avoids all problems arising when estimating probabilities from small samples. Let us at this place point out that parameter α is the *upper bound* for a probability of making an improper decision *not an approximation* of the probability. Only exact statistical assertions are used in our approach. The method was developed on the basis of the system ASSISTANT that expresses the induced knowledge in the form of a decision tree. However, the method can be used by a wide number of today known systems for empirical learning.

Results of the physicians clearly showed that with equal information about the patient general practitioners correctly diagnose less patients than our method. However, in real life physicians have much more information about the patient than just the 58 attributes that were used for our experiments. But in spite of this we think that physicians could benefit from computer-based classification. The discrepancy between system's and physician's classification could serve as a warning to the physician to re-examine the patient and possibly take some additional tests.

To the general practitioner, the most interesting problem is our first diagnostic problem (3 diagnoses). General practitioner examines the patient and sends him to the rheumatologist, to the orthopedist or to another specialist. Some of the general practi-

tioner's decisions are not correct and the patient has to queue for the improper doctor and wait for some time. Meanwhile, a disease can make a considerable progress. The use of an expert system to assist the general practitioner's decisions would decrease the number of such cases which would result in better functioning of medical systems as well as in smaller medical expenses.

ACKNOWLEDGMENTS

We would like to thank all members of Artificial intelligence laboratory at Jožef Stefan Institute, Ljubljana and Faculty of Electrical and Computer engineering, Ljubljana, especially Igor Kononenko and Bojan Cestnik. Thanks to Dunja Mladenić for incorporating extensions to system ASSISTANT, which enabled us to perform experiments. Thanks also to Renata Zupanc and Alen Varšek for careful proofreading.

REFERENCES

[1] Catlett, J., Christopher, J.: *"Is it better to learn each class separately?"*, technical report, University of Sydney, Australia, 1987.

[2] Cestnik, B.: "Estimating Probabilities: A Crucial Task in Machine Learning", *Proceedings of ECAI-90*, Stockholm, Sweden, august 1990.

[3] Cestnik, B., Kononenko, I., Bratko, I.: "ASSISTANT 86: A Knowledge elicitation tool for sophisticated users", in: Bratko, I., Lavrač, N. (eds.): *Progress in Machine Learning*, Sigma Press, Wilmslow, 1987.

[4] Ferguson, G.A., *"Statistical Analysis in Psychology and Education"*, Chapter 11, McGraw-Hill, London, 1959.

[5] Kononenko, I.: *"The design of the ASSISTANT Inductive Learning System"*, M.Sc. Thesis, E.Kardelj University, Faculty for Electrical and Computer Engineering, Ljubljana, 1985 (in Slovene).

[6] Michie D., Al Attar A.: "Use of sequential Bayes with class probability trees", to appear in: J.E.Hayes – Michie, D.Michie, E.Tyugu (eds.), *Machine Intelligence 12*, Oxford University Press.

[7] Mladenić, D.: *"Magnus Assistant: a system for machine learning"*, B.Sc. Thesis, E.Kardelj University, Faculty for Electrical Engineering and Computer Science, Ljubljana, 1990 (in Slovene).

[8] Quinlan, J.R.: "Induction of Decision Trees", *Machine Learning 1*, Kluwer Academic Publishers, Boston, 1986.

O VZPOREDNEM MNOŽENJU MATRIK

INFORMATICA 1/91

Borut Robič, Polona Blaznik
Institut Jožef Stefan
Ljubljana

Keywords: parallel algorithm, matrix multiplication, parallel computer architecture.

Opisani so trije časovno optimalni algoritmi za vzporedno množenje kvadratnih matrik na modelih SIMD-MC^2, SIMD-CC ter SIMD-PS vzporednega računanja. Podane so ustrezne časovne zahtevnosti. Izpeljan je formalni dokaz pravilnosti algoritma na SIMD-PS.

ON PARALLEL MATRIX MULTIPLICATION: We describe three parallel matrix multiplication algorithms which are known to be time optimal on underlying models of parallel computation: SIMD-MC^2, SIMD-CC, and SIMD-PS. Their time complexities are analized. Also, a formal proof of the algorithm correctness for SIMD-PS is derived.

Keywords: parallel algorithm, matrix multiplication, parallel computer architecture.

1. Uvod

Množenje matrik je operacija, ki se zelo pogosto pojavlja v numeričnih ter nenumeričnih problemih, zato je bila in še vedno je predmet intenzivnih raziskav.

Osnovni zaporedni algoritem za množenje dveh kvadratnih matrik reda n ima časovno zahtevnost $O(n^3)$ aritmetičnih operacij. Slednji je veljal za najboljšega vse do odkritja Strassenovega algoritma leta 1969 s kompleksnostjo $O(n^{2.81})$, kar je sprožilo plaz raziskav, usmerjenih v iskanje časovno še učinkovitejših algoritmov. Poznan je že algoritem s kompleksnostjo $O(n^\omega)$, kjer je $\omega < 2.4955$ [3]. Ker ni znano, ali je $\Omega(n^2)$ hkrati tudi *največja* spodnja meja za časovno kompleksnost zaporednega množenja matrik, je iskanje časovno optimalnega zaporednega algoritma še vedno aktualno. Žal pa se prednost teh algoritmov pokaže šele pri razmeroma velikih matrikah, zato imajo predvsem teoretični pomen.

Po drugi strani pa hkrati z razvojem vzporednih arhitektur postajajo zanimivi tudi vzporedni algoritmi za množenje matrik. Vzporedno množenje matrik se pojavlja celo v problemih, ko jih običajno ne rešujemo z uporabo zaporednega matričnega množenja [4]. V nadaljevanju se bomo osredotočili na družino SIMD računalnikov (Single Instruction stream, Multiple Data stream).

Družina SIMD modelov vzporednega računanja obsega dve poddružini. V prvo spada model SIMD-SM (Shared Memory) skupaj z različnimi variantami, ki se med seboj razlikujejo po stopnji omejevanja in izvedbe sočasnega čitanja/vpisovanja v skupni pomnilnik. Modeli iz te poddružine omogočajo lažji razvoj vzporednih algoritmov, ker se ni potrebno ozirati na arhitekturne značilnosti. Uporabni so predvsem pri določanju spodnjih meja za časovne kompleksnosti algoritmov.

V drugo poddružino pa spadajo realnejši modeli, pri katerih so izpostavljene nekatere arhitekturne značilnosti - predvsem način povezovanja procesorjev ter porazdelitev pomnilnika. Mednje spadajo tudi modeli SIMD-MC (Mesh Connected), SIMD-CC (Cube Connected), ter SIMD-PS (Perfect Shuffle), ki so poimenovani po različnih medprocesorskih povezovalnih shemah. Pri razvoju algoritma za reševanje danega problema moramo upoštevati vse značilnosti modela, na katerem bo potekalo računanje. Zato lahko obstaja za dani problem več časovno različnih optimalnih algoritmov, ki so pač razviti za različne računske modele.

V nadaljevanju bomo opisali problem vzporednega množenja dveh kvadratnih matrik na omenjenih modelih. Matriki označimo z A in B; obe sta reda $n \times n$. Produkt naj bo matrika C z elementi $c_{i,j} = \sum_{k=0}^{n-1} a_{i,k} b_{k,j}$.

2. Množenje na SIMD-SM-CRCW

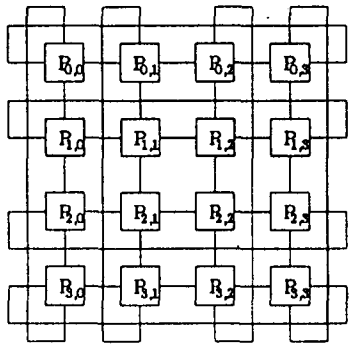
Najprej določimo spodnjo mejo za časovno kompleksnost vzporednega množenja matrik. Vzemimo model SIMD-SM-CRCW, kjer imajo vsi procesorji skupni pomnilnik, v katerega lahko sočasno vpisujejo ali iz njega čitajo. Reševanje konfliktnih situacij pri sočasnem posegu (vpisovanju, čitanju) v skupni pomnilnik je prepuščeno modelu (nekateri pristopi so opisani v [1]); algoritem bo zato enostavnejši, model pa toliko bolj skrivnosten. Pri analizi ne upoštevamo operacije prenašanja podatka iz enega procesorja v drugi. Prav tako predpostavljamo, da število procesorjev ni vnaprej omejeno: če se pri razvoju algoritma pojavi potreba po večjem številu procesorjev, lahko smatramo, da so že na voljo. Kljub svoji nerealnosti pa ta model omogoča določitev spodnje meje za časovno kompleksnost vzporednega množenja matrik, saj se pri razvoju algoritma osredotočimo le na bistvene operacije.

Denimo torej, da je na voljo vsaj n^3 procesorjev. V prvem koraku se sočasno izračuna vseh n^3 produktov $a_{i,k} b_{k,j}$, $0 \leq i,j,k \leq n-1$. Nato se sočasno izračuna vseh n^2 vsot $c_{i,j} = \sum_{k=0}^{n-1} a_{i,k} b_{k,j}$, $0 \leq i,j \leq n-1$. Vsako vsoto lahko izračunamo z $\frac{n}{2}$ procesorji v $\lceil \log n \rceil$ korakih, če izvršimo seštevanje v obliki binarnega drevesa. Skupaj je potrebnih $\lceil \log n \rceil + 1$ korakov, kar je spodnja meja za časovno kompleksnost vzporednega množenja matrik [2].

Za razvoj praktično uporabnega algoritma pa moramo izbrati enega izmed realnejših modelov vzporednega računanja. Obravnavali bomo vzporedno množenje matrik na modelih SIMD-MC^2, SIMD-CC in SIMD-PS.

3. Množenje matrik na SIMD-MC^2

SIMD-MC^2 vsebuje n^2 procesorjev $P_{i,j}$, $0 \leq i,j \leq n-1$, ki so ciklično povezani v dvodimenzionalno mrežo (Slika 1). Vsak procesor $P_{i,j}$ ima štiri sosede $P_{i\oplus 1,j}$, $P_{i,j\oplus 1}$, $P_{i\ominus 1,j}$ in $P_{i,j\ominus 1}$, kjer smo s $\oplus$ in $\ominus$ označili operaciji seštevanja in odštevanja po modulu n. Vsak $P_{i,j}$ vsebuje tri registre $A_{i,j}$, $B_{i,j}$ ter $C_{i,j}$.



Slika 1

Ob ustreznem ukazu lahko procesorji sočasno pošljejo vsebine izbranega registra v dani smeri svojim sosedom. Na primer, kadar se prenašajo vsebine registrov A v levo (oziroma navzgor), to zapišemo z $A_{i,j} \leftarrow A_{i,j\oplus 1}$ (oziroma $A_{i,j} \leftarrow A_{i\oplus 1,j}$).

3.1 Spodnja časovna meja

Gentleman je v [5] pokazal, da je treba pri analizi vzporednih algoritmov upoštevati tudi operacije prenašanja podatkov med procesorji; analize, ki tega ne upoštevajo, dajejo nerealne rezultate. Računski model, na katerem temelje Gentlemanove ugotovitve, združuje večje število procesorjev, ki imajo lahko tudi lokalne pomnilnike. Vsak procesor lahko neposredno pošlje podatek *omejenemu* številu sosednjih procesorjev. Procesor lahko pošlje podatek v enem koraku le *enemu* sosedu. Gentlemanov računski model je dovolj splošen, da je v njem zajet tudi SIMD-MC^2.

Razpršilna funkcija $\rho(k)$: N $\mapsto$ N naj bo definirana kot največje število procesorjev, ki lahko po k korakih prenašanja podatkov prejmejo podatek iz izbranega začetnega procesorja. Tu seveda predpostavljamo, da procesorji, ki so podatek že prejeli, sodelujejo pri njegovem nadaljnem širjenju v okolico. Funkcija ρ je seveda odvisna od povezovalne sheme procesorjev, torej od modela vzporednega računalnika. Neodvisno od povezovalne sheme pa velja sledeči

Izrek [5] *Na opisanem modelu računanja zahteva množenje dveh matrik reda $n \times n$ vsaj s korakov prenašanja podatkov, tako da je $\rho(2s) \geq n^2$* .

Dokaz. Opazujmo poljuben element $c_{i,j}$; nahaja se v procesorju $P_{i,j}$ in je enak $c_{i,j} = \sum_{k=0}^{n-1} a_{i,k}b_{k,j}$. Vsak izmed faktorjev $a_{i,k}$ in $b_{k,j}$, $0 \leq k \leq n-1$, ki vstopajo v ta izraz, se na začetku nahaja v lastnem procesorju in med računanjem vrednosti $c_{i,j}$ opravi neko pot do procesorja $P_{i,j}$. Naj bo $s(i,j)$ dolžina najdaljše izmed poti, ki jo opravijo ti faktorji pri računanju $c_{i,j}$ in naj bo $s = \max_{i,j} s(i,j)$. Torej je pri računanju produkta matrik potrebnih vsaj s prenosov podatkov. Izberimo dva poljubna procesorja P_x in P_y. Element matrike A, ki se nahaja v procesorju P_x, naj bo $a_{i,j}$. Zanima nas zgornja meja za število potrebnih korakov pri prenašanju tega elementa v procesor P_y. Denimo, da se v P_y nahaja element $b_{u,v}$ matrike B. Tedaj lahko pri prenašanju elementa $a_{i,j}$ uberemo pot preko procesorja P_z, v katerem pričakujemo element $c_{i,v}$ produkta matrik. Takšna pot namreč mora obstajati, saj v nasprotnem ne bi mogli izračunati $c_{i,v}$: prvi del je pot, ki jo pri računanju $c_{i,v}$ opravi $a_{i,j}$ do P_z, drugi del pa pot, ki jo pri tem (v obratni smeri) opravi $b_{u,v}$. Iz definicije števila s sledi, da obe poti skupaj merita kvečjemu $2s$. Sledi, da je za prenos podatka iz enega procesorja v drugi potrebnih kvečjemu $2s$ korakov. Seveda pa mora biti s zadosti velik, da lahko iz danega procesorja podatek pride do kateregakoli izmed n^2 procesorjev. To pomeni, da mora biti $2s$ korakov dovolj za razpršitev podatka v vseh n^2 procesorjev, oziroma $\rho(2s) \geq n^2$. S tem je izrek dokazan. Opazimo, da dokazovanja nismo oprli na nobeno od medprocesorskih povezovalnih shem. □

Za model SIMD-MC^2 je enostavno videti, da je moč v k korakih začetni podatek razpršiti v $\rho(k) = 2k^2 + 2k + 1$ procesorjev. Od tod in iz zgornjega izreka sledi, da je za ta model $s \geq \frac{1}{2}(n-\frac{1}{4})^{\frac{1}{2}} - \frac{1}{4}$. Sledi, da je spodnja časovna meja za množenje dveh matrik reda $n \times n$ na modelu SIMD-MC^2 enaka $\Omega(n)$. Vsak algoritem s časovno kompleksnostjo $\Theta(n)$ je zato s stališča asimptotičnih časovnih kompleksnosti *optimalen na SIMD-MC^2*. Tak algoritem je opisan v nadaljevanju.

3.2 Algoritem

Videli smo, da se na modelu SIMD-MC^2 dveh matrik reda $n \times n$ ne da zmnožiti hitreje kot v linearnem času glede na n. L.E.Cannon je razvil algoritem, ki to stori v času $\Theta(n)$ in je zato časovno optimalen na modelu SIMD-MC^2 [4].

Na začetku velja $A_{i,j} = a_{i,j}$, $B_{i,j} = b_{i,j}$ ter $C_{i,j} = 0$, $0 \leq i,j \leq n-1$. Na koncu pa je $C_{i,j} = c_{i,j} = \sum_{k=0}^{n-1} a_{i,k}b_{k,j}$, $0 \leq i,j \leq n-1$, torej element produkta matrik. Algoritem se glasi:

```
1  begin
2   for k = 1 to n-1 do
3    forall P(i,j) do
4     if i≥k then A(i,j) ← A(i,j⊕1)
5     if j≥k then B(i,j) ← B(i⊕1,j)
6    endforall
7   endfor;
8   forall P(i,j) do
9    C(i,j) := A(i,j)*B(i,j)
10  endforall;
11  for k = 1 to n-1 do
12   forall P(i,j) do
13    A(i,j) ← A(i,j⊕1)
14    B(i,j) ← B(i⊕1,j)
15    C(i,j) := C(i,j)+A(i,j)*B(i,j)
16   endforall
17  endfor
18 end.
```

Tu pomeni znak $\leftarrow$ prenašanje podatkov med registri različnih procesorjev, znak := pa znotraj procesorja.

Algoritem sestavljajo trije deli. V prvem, ki ga sestavljajo vrstice 2...7, se vsebine registrov $A_{i,j}$ in $B_{i,j}$ premestijo: v i-ti vrstici se začetne vsebine registrov A i-krat premestijo ciklično v levo, v j-tem stolpcu pa se začetne vsebine registrov B j-krat ciklično premestijo navzgor. Po izvršitvi prvega dela velja torej za vsak $P_{i,j}$: $A_{i,j} = a_{i,i\oplus j}$ in $B_{i,j} = b_{i\oplus j,j}$. Vsebini registrov $A_{i,j}$ in $B_{i,j}$ se lahko takoj pomnožita, saj se nahajata v istem procesorju $P_{i,j}$, njun produkt pa je eden od členov v končnem $c_{i,j}$. Sočasno množenje v vseh procesorjih opisujejo stavki 8...10, ki tvorijo drugi del algoritma. V tretjem delu algoritma (vrstice 11...17) prištejemo trenutni vsebini registra $C_{i,j}$ še ostalih $n-1$ produktov, ki so členi končnega $c_{i,j}$. V vsakem procesorju dobimo nova faktorja, ki sestavljata takšen produkt, če vsebine vseh registrov A preslikamo enkrat ciklično v levo, vsebine vseh registrov

B pa enkrat ciklično navzgor. Na primer, po k izvršitvah zanke 11 velja za vsak $P_{i,j}$, $0 \le i,j \le n-1$:

$$A_{i,j} = a_{i,i\oplus j\oplus k}$$
$$B_{i,j} = b_{i\oplus j\oplus k,j}$$
$$C_{i,j} = c_{i,j} = \sum_{p=0}^{k} a_{i,i\oplus j\oplus p} b_{i\oplus j\oplus p,j}$$

Po $k = n-1$ ponovitvah zanke 11 se v $C_{i,j}$ nahaja rezultat $c_{i,j}$.

Kakšna je časovna kompleksnost algoritma? Prvi del zahteva $2(n-1)$ operacij sočasnega premikanja vsebin registrov. Drugi del algoritma zahteva eno sočasno množenje po vseh procesorjih. Tretji del pa zahteva $2(n-1)$ operacij sočasnega premikanja vsebin registrov, ter $2(n-1)$ operacij sočasnega množenja/seštevanja. Skupaj je potrebnih $4(n-1)$ operacij sočasnega premikanja vsebin registrov ter $2(n-1)+1$ aritmetičnih operacij. Algoritem ima časovno kompleksnost $\Theta(n)$ in je zato asimptotično časovno optimalen na SIMD-MC^2.

4. Množenje matrik na SIMD-CC

Matriki reda $n \times n$ lahko teoretično zmnožimo v času $\Theta(\log n)$, če se ne oziramo na model vzporednega računanja. Z izborom modela vzporednega računanja, na katerem želimo množiti matriki, pa je potrebno pri snovanju algoritma upoštevati tudi arhitekturne značilnosti modela. Prav te pa lahko zelo poslabšajo časovno kompleksnost algoritma, pa čeprav je ta optimalen na izbranem modelu. Takšen primer smo videli v prejšnji točki, kjer je imel optimalni algoritem na SIMD-MC^2 časovno kompleksnost samo $\Theta(n)$. V tej točki pa bomo opisali model SIMD-CC, ki dovoljuje zasnovo algoritma s časovno kompleksnostjo $\Theta(\log n)$. Model SIMD-CC imenujemo tudi *hiperkocka*.

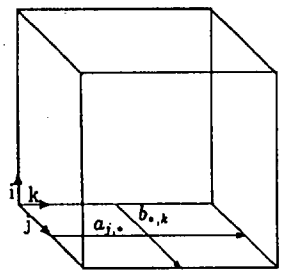
V tej točki naj bo dimenzija matrik $n = 2^q$, za nek $q \in \mathbf{N}$. V modelu SIMD-CC naj bo $p = n^3 = 2^{3q}$ procesorjev $P_0, \ldots, P_{p-1}$. V vsakem procesorju P_r se nahajajo registri A_r, B_r in C_r. Vsak procesor P_r je povezan z $\log(p) = 3\log n = 3q$ procesorji, katerih indeksi so števila $r^{(0)}, r^{(1)}, \ldots, r^{(3q-1)}$, kjer $r^{(b)}$ dobimo tako, da v r invertiramo bit r_b.

Če indeks r, $0 \le r \le p-1$, zapišemo v binarni obliki (s $3q$ biti) in zaporedne q-terke bitov opazujemo kot števila i,j,k, $0 \le i,j,k \le n-1$,

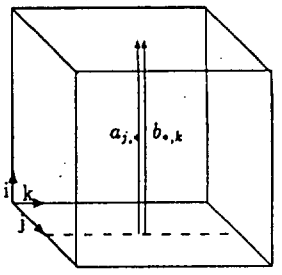
$$r = \underbrace{r_{3q-1}\cdots r_{2q}}_{i}\ \underbrace{r_{2q-1}\cdots r_{q}}_{j}\ \underbrace{r_{q-1}\cdots r_{0}}_{k}$$

lahko procesor P_r označimo tudi s $P_{[i,j,k]}$, pripadajoče registre pa z $A_{[i,j,k]}$, $B_{[i,j,k]}$ in $C_{[i,j,k]}$. Oba zapisa sta ekvivalentna in ju bomo uporabljali izmenično po potrebi. Prednost drugega zapisa je, da si lahko SIMD-CC predstavljamo kot tridimenzionalno polje procesorjev; števila i,j,k povedo, kje v polju se nahaja $P_{[i,j,k]}$. Opozoriti pa moramo, da procesorja, ki sta v tem polju sosednja, v splošnem nista povezana, saj so povezave med procesorji vspostavljene tako, kot je bilo opisano zgoraj.

Na začetku predpostavljamo, da sta matriki A in B vpisani v registre A oziroma B "na dnu" tridimenzionalnega polja, tako da je $A_{[0,j,k]} = a_{j,k}$ ter $B_{[0,j,k]} = b_{j,k}, 0 \le j,k \le n-1$. Zahtevamo, da nam algoritem vrne $C_{[0,j,k]} = c_{j,k} = \sum_{m=0}^{n-1} a_{j,m} b_{m,k}, 0 \le j,k \le n-1$. Opišimo postopek najprej v splošnem. Vektorja $a_{j,*}$ in $b_{*,k}$, katerih skalarni produkt je $c_{j,k}$, sta na dnu polja pravokotna eden na drugega (Slika 2a.). Zato ju "dvignemo" tako, da so komponente $a_{j,i}, b_{i,k}, 0 \le i \le n-1$ paroma v istih procesorjih (Slika 2b.). To storimo sočasno za vse pare vektorjev $a_{j,*}, b_{*,k}, 0 \le j,k \le n-1$. Istoležne komponente sočasno pomnožimo v vseh n procesorjih, ter v vsakem stolpcu seštejemo produkte tako, da se njihove vsote "sesedejo" na dno v $C_{[0,j,k]}$.

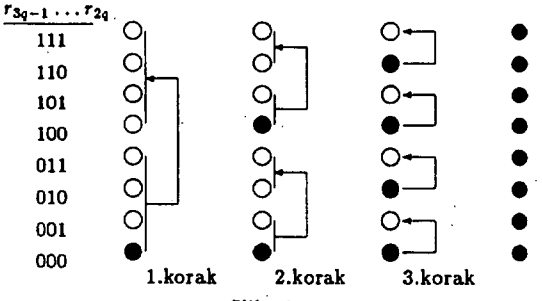


Slika 2a



Slika 2b

Sedaj pa podrobneje opišimo algoritem. V prvem delu preslikamo obe matriki z dna še na vse ostale nivoje tridimenzionalnega polja, tako da na koncu dobimo $A_{[i,j,k]} = a_{j,k}$ ter $B_{[i,j,k]} = b_{j,k}$, za vse nivoje i, $0 \le i \le n-1$. Na prvi pogled kaže, da je za to potrebnih $\Theta(n)$ korakov, saj je vsak procesor $P_{[0,j,k]}$ navzgor neposredno povezan le s $q-1 = \log(n)-1$ procesorji $P_{[2^m,j,k]}, 1 \le m \le q-1$, poleg tega pa lahko v enem koraku procesor pošlje podatek le enemu sosedu. Toda v SIMD-CC so procesorji med seboj tako povezani, da je možno v vsakem koraku podvojiti število tistih procesorjev v stolpcu, ki že imajo začetni podatek. Zato preslikavanje zahteva $q = \Theta(\log n)$ korakov. Za $n = 8$ je preslikavanje v enem stolpcu ilustrirano na sliki 3.



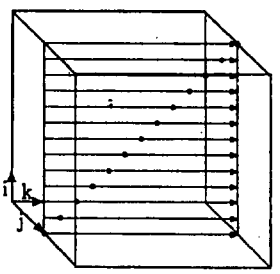
Slika 3

V splošnem pa za to poskrbijo stavki 1...8:

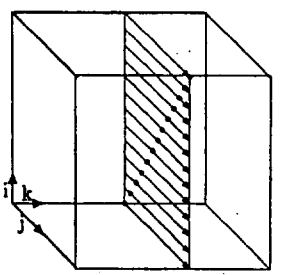
```
1  for m = 3q − 1 downto 2q do
2    forall P_r, 0 ≤ r ≤ p − 1 do
3      if r_m = 0 then A_{r(m)} ← A_r
4    endforall;
5    forall P_r, 0 ≤ r ≤ p − 1 do
6      if r_m = 0 then B_{r(m)} ← B_r
7    endforall
8  endfor;
```

Na koncu velja $A_{[i,j,k]} = a_{j,k}$ ter $B_{[i,j,k]} = b_{j,k}, 0 \le i,j,k \le n-1$.

Pri $k = i$ dobimo $A_{[i,j,i]} = a_{j,i}$; ti elementi se nahajajo na diagonali, kot kaže Slika 4a. Podobno dobimo $B_{[i,i,k]} = b_{i,k}$, če vzamemo $j = i$ (Slika 4b.). Vsebino registra $A_{[i,j,i]}$ prenesemo v vse ostale registre $A_{[i,j,k]}$, $0 \le k \le n-1$ (Slika 4a.). Vsebino registra $B_{[i,i,k]}$ pa prenesemo v vse registre $B_{[i,j,k]}$, $0 \le j \le n-1$ (Slika 4b.).



Slika 4a



Slika 4b

Za to poskrbijo stavki 9...12 oziroma 13...16:

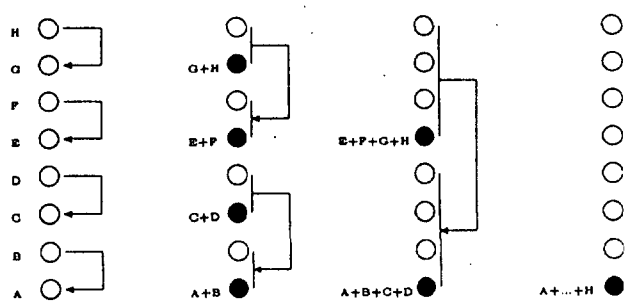
```
9  for m = q − 1 downto 0 do
10   forall P_r, 0 ≤ r ≤ p − 1 do
11     if r_m = r_{2q+m} then A_{r(m)} ←A_r
12   endforall;
13 for m = 2q − 1 downto q do
14   forall P_r, 0 ≤ r ≤ p − 1 do
15     if r_m = r_{q+m} then B_{r(m)} ←B_r
16   endforall;
```

Na koncu je $A_{[i,j,k]} = a_{j,i}$ ter $B_{[i,j,k]} = b_{i,k}$, $0 \le i,j,k \le n-1$, (Slika 2b).

Sedaj pa lahko sočasno pomnožimo vsebine registrov $A_{[i,j,k]}$ ter $B_{[i,j,k]}$ v vseh p procesorjih, kot to opisujejo stavki 17...19:

```
17 forall P_r, 0 ≤ r ≤ p − 1 do
18   C_r := A_r*B_r
19 endforall;
```

V zanjem koraku moramo sešteti produkte, ki se nahajajo v stolpcih, tako da se njihove vsote $c_{j,k}$ pojavijo na dnu tridimenzionalnega polja. Postopek seštevanja v enem stolpcu je za $n = 8$ ilustriran na Sliki 5, v splošnem pa je opisan s stavki 20...24.



Slika 5

```
20 for m = 2q to 3q − 1 do
21   forall P_r, 0 ≤ r ≤ p − 1 do
22     if r_m = 0 then C_r ← C_r+C_{r(m)}
23   endforall
24 endfor;
```

Časovno kompleksnost algoritma sestavlja $5q = 5\log(n)$ korakov sočasnega prenašanja podatkov med procesorji, eno sočasno množenje in $\log(n)$ sočasnih seštevanj. Opisani algoritem ima torej časovno kompleksnost reda $\Theta(\log n)$.

5. Množenje matrik na SIMD-PS

Naj bo dimenzija matrik enaka $n = 2^q$, za nek $q \in \mathbf{N}$. V modelu SIMD-PS naj bo $p = n^3 = 2^{3q}$ procesorjev $P_0, \ldots, P_{p-1}$. V vsakem procesorju P_r se nahajajo registri A_r, B_r in C_r. Vsak procesor P_r je povezan s tremi sosednjimi procesorji $P_{sh(r)}$, $P_{ex(r)}$ in $P_{un(r)}$, kjer so indeksi $sh(r)$, $ex(r)$ ter $un(r)$ števila, katerih binarni zapis dobimo iz binarno zapisanega indeksa

$$r = r_{3q-1}r_{3q-2}\ldots r_{2q}r_{2q-1}\ldots r_q r_{q-1}\ldots r_1 r_0$$

na naslednji način:

$$sh(r) = r_{3q-2}\ldots r_{2q}r_{2q-1}\ldots r_q r_{q-1}\ldots r_1 r_0 r_{3q-1}$$

$$ex(r) = r_{3q-1}r_{3q-2}\ldots r_{2q}r_{2q-1}\ldots r_q r_{q-1}\ldots r_1 \bar{r}_0$$

$$un(r) = r_0 r_{3q-1}r_{3q-2}\ldots r_{2q}r_{2q-1}\ldots r_q r_{q-1}\ldots r_1$$

Tu je $\bar{r}_0$ komplement bita r_0. Funkcije sh, un in ex se imenujejo "shuffle", "unshuffle" ter "exchange" in izvršijo ciklično rotacijo bitov argumenta v levo, desno oziroma komplementiranje zadnjega bita.

Podobno kot pri modelu SIMD-CC bomo tudi tu uporabljali za indekse izmenično oznaki r in $[x,y,z]$, kjer je $0 \le r \le p-1$, $0 \le x,y,z \le n-1$ ter $r = 2^{2q}x + 2^q y + z$. Prednost drugega zapisa je, da si lahko SIMD-PS predstavljamo kot tridimenzionalno polje procesorjev; števila x,y,z povedo, kje v polju se nahaja $P_{[x,y,z]}$. Zopet pa moramo opozoriti, da procesorja, ki sta v tem polju sosednja, v splošnem nista povezana, saj so medprocesorske povezave vzpostavljene tako, kot je bilo opisano zgoraj.

Sedaj pa opišimo algoritem za množenje matrik na modelu SIMD-PS. Najprej preslikamo matriki A in B v registre A oziroma B na dno polja: $A_{[0,i,t]} := a_{i,t}$ in $B_{[0,i,t]} := b_{i,t}$, $0 \le i,t \le n-1$. Od tu matriki preslikamo še na vseh ostalih $n-1$ vzporednih plasti. Slednje opisujejo stavki 1...8:

```
1 for b = 0 to q − 1 do
2   forall P_r, 0 ≤ r ≤ p − 1 do
3     A_{sh(r)} ←A_r;
4     B_{sh(r)} ←B_r;
5     if r_0 = 0 then A_{ex(r)} ←A_r;
6     if r_0 = 0 then B_{ex(r)} ←B_r;
7   endforall
8 endfor;
```

Trditev: *Po izvršitvi stavkov* 1...8 *velja* $A_{[i,t,s]} = a_{i,t}$ *in* $B_{[i,t,s]} = b_{i,t}$, $0 \le i,t,s \le n-1$.

Dokaz: Register $A_{[0,i,t]}$ z vsebino $a_{i,t}$ se nahaja v procesorju P_r, kjer je $r = \underbrace{00\ldots0}_{0}\underbrace{r_{2q-1}\ldots r_q}_{i}\underbrace{r_{q-1}\ldots r_0}_{t}$.

V prvi izvedbi zanke 1 se vsebina $a_{i,t}$ registra $A_{[0,i,t]}$ z ukazom 3 preslika v $A_{sh(r)}$, kjer $sh(r) = 0\ldots0r_{2q-1}\ldots r_q r_{q-1}\ldots r_0 0$, z ukazom 5 pa iz tega registra še v register $A_{ex(sh(r))}$, kjer je $ex(sh(r)) = 0\ldots0r_{2q-1}\ldots r_q r_{q-1}\ldots r_0 1$. Predpostavimo, da so po b ponovitvah zanke 1 vrednosti $a_{i,t}$ v vseh $s = 2^b$ registrih A_r, kjer je $r = \underbrace{00\ldots0}_{0}\underbrace{r_{2q-1}\ldots r_q}_{i}\underbrace{r_{q-1}\ldots r_0}_{t}\underbrace{r_{3q-1}\ldots r_{3q-1-(b-1)}}_{j}$ in $0 \le j \le 2^b - 1$. Po naslednji ponovitvi zanke je $a_{i,t}$ zaradi 3 in 5 v vseh $s = 2^{b+1}$ registrih A_r, kjer je $r = \underbrace{00\ldots0}_{0}\underbrace{r_{2q-1}\ldots r_q}_{i}\underbrace{r_{q-1}\ldots r_0}_{t}\underbrace{r_{3q-1}\ldots r_{3q-1-b}}_{j}$ in $0 \le j \le 2^{b+1} - 1$. Ker je $n = 2^q$, se po q ponovitvah zanke 1 $a_{i,t}$ nahaja v vseh n registrih $A_{[i,t,s]}$, kjer $0 \le s \le n-1$. Ker je bil par i,t poljuben, je s tem prvi del trditve dokazan. Dokaz za registre B je podoben. □

V naslednjem koraku premestimo vsebine registrov A:

```
9  for b = 0 to q − 1 do
10   forall P_r, 0 ≤ r ≤ p − 1 do
11     A_{sh(r)} ←A_r
12   endforall
13 endfor;
```

Trditev: *Po izvršitvi vrstic* 9...13 *je* $A_{[i,t,s]} = a_{s,i}, 0 \le i,t,s \le n-1$.

Dokaz: Pred izvršitvijo teh vrstic je $A_{[s,i,t]} = a_{s,i}, 0 \le s,i,t \le n-1$. Po q izvršitvah stavka 11 se $a_{s,i}$ iz $A_{[s,i,t]}$ prenese v $A_{[i,t,s]}$, saj velja $sh^q([s,i,t]) = [i,t,s]$. □

Po izvršitvi stavkov 1...13 sta v registrih $A_{[i,t,s]}$ in $B_{[i,t,s]}$ procesorja $P_{[i,t,s]}$ vrednosti $a_{s,i}$ in $b_{i,t}$. Njun produkt $a_{s,i}b_{i,t}$ je eden od n členov v končnem $c_{s,t}$, zato ju je smiselno pomnožiti. To seveda velja za vse procesorje P_r, $0 \le r \le p-1$, saj lahko vsak r zapišemo v obliki $r = [i,t,s]$, kjer $0 \le i,t,s \le n-1$. Sočasno

množenje v vseh procesorjih opisujejo stavki 14 ... 16:

```
14 forall P_r, 0 ≤ r ≤ p − 1 do
15   C_r := A_r*B_r
16 endforall;
```

Rezultat stavkov 14 ... 16 je $C_{[i,t,s]} = a_{s,i}b_{i,t}$. Ker želimo na koncu dobiti $c_{s,t} = \sum_{i=0}^{n-1} a_{s,i}b_{i,t}$, moramo za vsak par s,t sešteti vsebine vseh registrov $C_{[i,t,s]}, 0 \le i \le n-1$:

```
17 for b = 0 to q − 1 do
18   forall P_r, 0 ≤ r ≤ p − 1 do
19     C_sh(r) ← C_r;
20     C_r ← C_r + C_ez(r)
21   endforall
22 endfor;
```

Trditev: *Po izvršitvi stavkov* 17...22 *velja* $C_{[t,s,i]} = c_{s,t} = \sum_{k=0}^{n-1} a_{s,k}b_{k,t}$ *za vse* t,s,i, $0 \le t,s,i \le n-1$.

Dokaz: Izberimo poljuben par števil t,s, $0 \le t,s \le n-1$ in pišimo $f_i = a_{s,i}b_{i,t}$. Vsebina registra $C_{[i,t,s]}$ je f_i. Naj bosta τ in σ binarna zapisa števil t in s, torej $\tau,\sigma \in \{0,1\}^q$. Naj pomeni $\sum_{j_1\dots j_b}$ seštevanje po vseh $j_1\dots j_b \in \{0,1\}^b$. Naprimer, $\sum_{j_1 j_2}$ pomeni seštevanje, kjer indeks preteče binarna zaporedja 00, 01, 10 in 11.

Tedaj opazimo, da po prvi izvršitvi zanke 17 velja za vse $i_{q-1} \in \{0,1\}$:

$$C_{i_{q-2}\dots i_0\tau\sigma i_{q-1}} = \sum_{j_1} f_{j_1 i_{q-2}\dots i_0}.$$

Naj po b ponovitvah zanke 17 velja za vse $i_{q-1}\dots i_{q-1-(b-1)} \in \{0,1\}^b$:

$$C_{i_{q-1-b}\dots i_0\tau\sigma i_{q-1}\dots i_{q-1-(b-1)}} = \sum_{j_1\dots j_b} f_{j_1\dots j_b i_{q-1-b}\dots i_0}. \quad (*)$$

Po vnovični izvršitvi stavka 19 je za vse $i_{q-1}\dots i_{q-1-(b-1)} \in \{0,1\}^b$:

$$C_{i_{q-1-(b+1)}\dots i_0\tau\sigma i_{q-1}\dots i_{q-1-(b-1)}0} = \sum_{j_1\dots j_b} f_{j_1\dots j_b 0 i_{q-1-(b+1)}\dots i_0}$$

in

$$C_{i_{q-1-(b+1)}\dots i_0\tau\sigma i_{q-1}\dots i_{q-1-(b-1)}1} = \sum_{j_1\dots j_b} f_{j_1\dots j_b 1 i_{q-1-(b+1)}\dots i_0}.$$

S stavkom 20 se obe vsoti seštejeta in priredita obema registroma, zato velja za vse $i_{q-1}\dots i_{q-1-b} \in \{0,1\}^{b+1}$:

$$C_{i_{q-1-(b+1)}\dots i_0\tau\sigma i_{q-1}\dots i_{q-1-b}} = \sum_{j_1\dots j_{b+1}} f_{j_1\dots j_{b+1} i_{q-1-(b+1)}\dots i_0}.$$

Torej velja enakost $(*)$ za vse $b, 0 \le b \le q-1$. Posebej za $b = q-1$ je

$$C_{\tau\sigma i_{q-1}\dots i_0} = \sum_{j_1\dots j_q} f_{j_1\dots j_q} = \sum_{j=0}^{n-1} f_j$$

za vse $i_{q-1}\dots i_0 \in \{0,1\}^q$, kar je isto kot $C_{[t,s,i]} = \sum_{j=0}^{n-1} a_{s,j}b_{j,t}$, $0 \le i \le n-1$. □

S tem je produkt matrik izračunan: element $c_{s,t}$ se nahaja v vseh n registrih $C_{[t,s,i]}$, $0 \le i \le n-1$.

Navadno pa želimo, da se rezultati nahajajo v registrih C tistih procesorjev, v katere sta bili na začetku vpisani matriki A in B. Potemtakem želimo, da se elementi $c_{s,t}$ pojavijo "na dnu" tridimenzionalnega polja: $C_{[0,s,t]} = c_{s,t}$, $0 \le s,t \le n-1$. Ta popravek izvršimo v dveh korakih.

Najprej opazimo, da po izvršitvi stavkov 1 ... 22 v posebnem primeru velja $C_{[t,s,t]} = c_{s,t}, \le 0 \le s,t \le n-1$. Vsebine teh registrov prenesemo v registre $C_{[s,t,0]}$:

```
23 for b = 0 to q − 1 do
24   forall P_r, 0 ≤ r ≤ p − 1 do
25     C_sh(r) ←C_r;
26     if r_0 = r_q = 1 then C_ez(r) ←C_r
27   endforall
28 endfor;
```

Trditev: *Po izvršitvi* 23 ... 28 *velja* $C_{[s,t,0]} = c_{s,t}$, *za vse* $0 \le s,t \le n-1$.

Dokaz: Zapišimo indeks $[t,s,t]$ binarno: $t_{q-1}\dots t_0 s_{q-1}\dots$ $\dots s_0 t_{q-1}\dots t_0$. Po prvi izvršitvi stavka 25 se $c_{s,t}$ prenese v register C z binarno zapisanim indeksom $t_{q-2}\dots t_0 s_{q-1}\dots s_0 t_{q-1}\dots t_0 t_{q-1}$. Pri stavku 26 je za ta indeks $r_0 = r_q = t_{q-1}$. Če je $t_{q-1} = 1$, se $c_{s,t}$ prenese še v register C z indeksom $t_{q-2}\dots t_0 s_{q-1}\dots s_0 t_{q-1}\dots t_0 0$. Torej se $c_{s,t}$ gotovo nahaja v registru C, katerega binarno zapisani indeks je $t_{q-2}\dots t_0 s_{q-1}\dots s_0 t_{q-1}\dots t_0 0$. Predpostavimo, da je po b izvršitvah zanke 23 vrednost $c_{s,t}$ v registru C z binarno zapisanim indeksom $t_{q-1-b}\dots t_0 s_{q-1}\dots s_0 t_{q-1}\dots t_0 0\dots 0$. Ob naslednji ponovitvi zanke se s stavkom 25 vrednost $c_{s,t}$ prenese v register C z binarnim indeksom $t_{q-1-(b+1)}\dots t_0 s_{q-1}\dots s_0 t_{q-1}\dots t_0 0\dots 0 t_{q-1-b}$. Za ta indeks je $r_0 = r_q = t_{q-1-b}$. Če je $t_{q-1-b} = 1$ se vrednost $c_{s,t}$ s stavkom 26 prenese še v register C z binarnim indeksom $t_{q-1-(b+1)}\dots t_0 s_{q-1}\dots s_0 t_{q-1}\dots t_0 0\dots 0\bar{t}_{q-1-b}$. Vidimo, da se po q ponovitvah zanke 23 vrednost $c_{s,t}$ nahaja v registru C z binarnim indeksom $s_{q-1}\dots s_0 t_{q-1}\dots t_0 0\dots 0$, torej v registru $C_{[s,t,0]}$. Opisano velja za vsak $0 \le s,t \le n-1$.□

Končno vsebine registrov $C_{[s,t,0]}$ prenesemo v registre $C_{[0,s,t]}$:

```
29 for b := 0 to q − 1 do
30   forall P_r, 0 ≤ r ≤ p − 1 do
31     C_un(r) ← C_r
32   endforall
33 endfor.
```

Kakšna je časovna kompleksnost opisanega algoritma ? Vrstice 1 ... 33 zahtevajo $10q = 10\log(n)$ operacij sočasnega premeščanja vsebin registrov, eno sočasno množenje v vseh procesorjih ter $q = \log(n)$ sočasnih seštevanj. Opazimo, da je algoritem približno dvakrat počasnejši od algoritma na modelu SIMD-CC ter približno desetkrat počasnejši od teoretično najhitrejšega. Če pa se zadovoljimo z okrnjeno inačico algoritma (vrstice 1 ... 22), pa je potrebnih $7\log(n)$ sočasnih premeščanj ob enakem številu seštevanj in množenj. Asimptotična časovna kompleksnost je v obeh primerih $\Theta(\log n)$, torej je algoritem asimptotično časovno optimalen.

6. Zaključek

Opisani so bili nekateri algoritmi za vzporedno množenje kvadratnih matrik. Najprej je bil opisan postopek, katerega časovna kompleksnost je $\lceil \log n \rceil + 1$, kjer je n dimenzija matrik. Ta algoritem se ne ozira na nobenega od realnih modelov vzporednega računanja - njegov pomen je v določitvi spodnje meje za časovno kompleksnost vzporednega množenja matrik. Če pa izberemo nek računski model, se lahko časovna kompleksnost optimalnega algoritma (na izbranem modelu) poslabša. Pri modelu SIMD-MC^2 časovna kompleksnost optimalnega algoritma poskoči na $\Theta(n)$. Model, ki dovoljuje razvoj asimptotično absolutno optimalnega algoritma (s časovno kompleksnostjo $\Theta(\log n)$) je SIMD-CC (hiperkocka). Slabosti hiperkocke pa se pokažejo pri njeni VLSI implementaciji, saj število medprocesorskih povezav prehitro narašča. Teh slabosti nima model SIMD-PS, pri katerem je vsak procesor povezan vedno

le s tremi sosedi. Tudi ta model dovoljuje zasnovo algoritma za množenje matrik, ki je sicer približno dvakrat počasnejši, a seveda še vedno asimptotično časovno optimalen.

V prispevku nismo uporabljali pojma *cene* vzporednega algoritma, tj. produkta med številom procesorjev in časom izvrševanja, čeprav je cena primernejše merilo za primerjavo algoritmov. Iz opisanega je cene moč enostavno določiti.

Literatura

[1] S.G.Akl, *The Design and Analysis of Parallel Algorithms*, Prentice-Hall Int'l Editions, 1989.

[2] A.Borodin, I.Munro, *The Computational Complexity of Algebraic and Numeric Problems*, Elsevier Publishing Comp., 1975.

[3] D.Coppersmith, S.Winograd, *On the asymptotic complexity of matrix multiplication*, SIAM J.Comput., Vol.11, No.3, 1982, str.472-492.

[4] E.Dekel, D.Nassimi, S.Sahni, *Parallel matrix and graph algorithms*, SIAM J.Comput., Vol.10, No.4, 1981, str.657-675.

[5] W.M.Gentleman, *Some complexity results for matrix computations on parallel processors*, J.ACM, Vol.25, No.1, 1978, str.112-115.

HIERARHIČNI VEČPROCESORSKI SISTEM

INFORMATICA 1/91

Keywords: Parallel processor system, Token data flow, Transputer, Topology, Communication message, Static scheduling, Processor allocation, Speedup.

Peter Kolbezen,
Institut Jožef Stefan, Ljubljana
Peter Zaveršek, Gorenje Servis,
Velenje

POVZETEK. V članku je predstavljen večnivojski rekonfigurabilni večtransputerski sistem. Izdelan je koncept izmenjave sporočil med transputerji na nivoju procesiranja uporabniških programov. Nivo procesiranja elementarnih uporabniških programov je mogoče s pomočjo spremenljive topologije povezav med transputerji povezati v hierarhično vgnezdene zanke. Razvrščanje procesov je prilagojeno tej topologiji in je zasnovano na matematičnem zapisu hierarhičnega acikličnega grafa pretoka podatkov. Predhodna analiza uporabniškega programa na sistemskem nivoju računalnika omogoča odkriti sočasne procese, določiti aciklične dele programa in jih transformirati v ekvivalentne dele programa, ki so predstavljivi s hierarhičnimi grafi pretoka podatkov. Komunikacija med transputerji, ki so povezani v zankah, in avtomatično dodeljevanje procesov potekata asinhrono s podajanjem žetonov. Predlagana topologija omogoča časovno optimirano razvrščanje, ki je pogojeno z usklajenostjo topologije in hierahičnimi acikličnimi grafi pretoka podatkov.

A HIERARCHICAL MULTIMICROPROCESSOR SYSTEM. The architecture of a multiprocessor transputer system and a communications shell for a multiple ring topology network of transputers for efficient execution of parallel programs are proposed in the paper. The system consists of three layers dedicated to elementary user program process execution, global control in programs and system functions. The elementary user program process execution layer network ring topology is hierarchically arranged. The process allocation is dinamically adjusted to the topology of intreprocess communications. It is based on the analysis of user program and on the description of the program with the hierarchical acyclic data flow graph (HADFG). The prime requirement is that any processor in the network must be capable of passing a message to its successor, with the communications shell automatically. The communications shell passes each message packet through the ring network asynchronously. Proposed topology enables time-optimized scheduling by harmony of the network topology.

1. UVOD

Procesiranje v realnem času na področjih, kot so vodenje zahtevnih procesov, robotizacija, znanstveno - raziskovalno delo, obdelava slik in signalov ter podobno, zahteva vse večje računalniške zmogljivosti, ki jih s klasičnimi računalniki ni več mogoče doseči.

Znano je, da je edina pot, ki vodi k večjim zmogljivostim, možna preko bistveno drugačnih računalniških organizacij, kot so se uporabljale v preteklosti. Med uspešnejše organizacije sodijo predvsem takšne, ki izkoriščajo paralelizem. Zato zasluži paralelno računanje danes še posebno pozornost. Že sorazmeroma cenena in hitra tehnologija zelo visoke integracije (VLSI) omogoča učinkovitejše računalniško podprto načrtovanje in uporabo paralelnih VLSI računalnikov. Zato so dosežki v mikroelektronski tehnologiji, in bodo še bolj v bodoče, pobudniki številnih inovacij v paralelnih računalniških arhitekturah in v uporabi t.im. polj procesorjev. V svetu je ta trend deležen velike pozornosti v vladnih, industrijskih in univerzitetnih okoljih, saj je v zadnjem desetletju zaslediti viden vzpon raziskav in vlaganj v razvoj tovrstne računalniške tehnologije.

2. PARALELNO PROCESIRANJE

Med najbolj preprostimi in najbolj obetavnimi tipi paralelnih računalnikov je dobro poznana večprocesorska arhitektura. Od vseh tehnik, ki večajo moč procesiranja, je paralelno procesiranje v pogledu načrtovanja aparaturne opreme ena od najpreprostejših tehnik, hkrati pa tudi najbolj izzivalna v pogledu načrtovanja programske opreme.

Paralelizem je lahko drobno ali grobo zrnat. Prvi se pojavlja pri vektorskih procesorjih, drugi pa v računalniških mrežah. Obstaja pa tudi srednje zrnat paralelizem, kakršnega srečujemo v transputerskih sistemih.

Večprocesorskimi sistemi imajo običajno centralni mikroprocesor, obdan s procesorji, pomnilniki in

komunikacijskimi potmi. Ti viri se centralno dodeljujejo in preko njih potekajo komunikacije med procesi. V sklop sistema mora biti vključena tudi posebna aparaturna oprema, ki odloča o večkratnih sočasnih zahtevah za dostop do dodeljenih virov, in programska oprema, ki mora pri dodeljevanju virov zagotavljati vzajemno izključevanje izvajanj kritičnih področij programa, v katerih se viri uporabljajo. Tako se v zvezi z uporabo večprocesorskih sistemov pojavlja vrsta problemov, ki so težje narave. Znano je, da sta ob večjem številu procesorjev večja tudi cena upravljanja in zmogljivost sistema. Ta odvisnost je pri manjšem številu procesorjev skoraj linearna in spočetka dopušča tudi dovolj veliko prepustnost sistema. Z nadaljnim večanjem števila procesov pa prične prepustnost sistema kljub večanju števila procesorjev vse hitreje upadati. Zato je zmogljivost takega sistema omejena.

V večprocesorskih sistemih, ki so osnovani na transputerski tehnologiji, se dodeljevanje virov bistveno poenostavi. Problemi spornih točk in vzajemnega izključevanja skoraj v celoti odpadejo. Komunikacije med procesorji in procesi zamenjajo medprocesorske zanke. Zlasti pa je pomembno, da v primerjavi s klasičnimi sistemi prepustnost transputerskih sistemov neomejeno narašča skoraj sorazmerno s številom transputerjev. Komunikacija med procesorji poteka zaporedno preko dveh enosmernih linij na asinhroni način. Vsaka zanka predstavlja enosmerni kanal, preko katerega dva procesa med seboj komunicirata. S takšno komunikacijo je omogočena tudi sinhronizacija med posameznimi procesi. Transputerski večprocesorski sistem predstavlja torej skupek procesorskih vozlišč s ponori in izvori, ki si v naključnostnih intervalih izmenjujejo sporočila med seboj.

Dasi razvoj in gradnja večprocesorskih sistemov napredujeta z dramatično naglico, to ne velja za razvoj dovolj učinkovitih postopkov programiranja. Tudi programiranje na nivoju srednje zrnatosti je še vedno povezano s prekomernim naporom in često zahteva povsem nove postopke. Prenašanje sporočil v transputerskem sistemu ne predstavlja tolikšnih režijskih stroškov kot sistem, ki zahteva dekompozicijo programa v drobno zrnat program. Po drugi strani pa program ne sme biti preveč grobo zrnat, saj kot tak preprečuje, da se njegovo izvajanje porazdeli na pametno število procesorjev.

3. PROGRAMIRANJE VEČPROCESORSKIH SISTEMOV

Programer se pri programiranju večprocesorskih sistemov sooča s tremi glavnimi problemi:

1. Kako v danem programu ugotoviti dele programa, ki se lahko izvajajo sočasno, in kako originalni program modificirati tako, da se zmanjša ali čas računanja ali število potrebnih virov?

2. Kako dodeljevati zgoraj omenjene različne dele kode v večprocesorskem sistemu?

3. Kako za dano arhitekturo določiti optimalno število procesorjev, ki so potrebni za kar se da učinkovito izvajanje programa?

Pričujoče delo je posvečeno predvsem vprašanju dodeljevanja procesov neregularnih algoritmov na večnivojskem transputerskem sistemu. Za reševanje problemov iz tč.1 je potrebno orodje, ki omogoča skrbno analizo in dekompozicijo programa za paralelno procesiranje na večtransputerskem sistemu. Sistem naj bi programerju dopuščal uporabo dovolj splošnih metod za reševanje zgoraj omenjenih problemov.

Program, ki naj bi se kolikor je mogoče optimalno izvajal na večprocesorski arhitekturi tako, da bi izkoriščal ves možni paralelizem, mora biti predhodno skrbno analiziran in razstavljen na množico procesov. Seveda je eden prvih pogojev za paralelno procesiranje, da program vsebuje dovolj paralelnih poti, ki so programerju pogosto tudi prikrite. Pristop k reševanju problema določanja sočasnosti pa lahko poteka po dveh poteh. Po prvi poti je določanje sočasnosti neposredno prepuščeno programerju. Znano je namreč, kako določi programer sočasnost pri pisanju programov v jezikih, ki podpirajo paralelno programiranje. Po drugi poti pa je določanje sočasnosti implicitno, neodvisno od programerja na osnovi analize izvornega programa. Pri tem je odločilnega pomena podatkovna odvisnost med procesi. Ker so sekvenčni programi razbiti na različne nivoje, je osnovni korak k paralelnemu procesiranju razpoznavanje procesov znotraj posameznih nivojev, predvsem takšnih, ki se lahko že po svoji naravi izvajajo paralelno, pa tudi takšnih, katerih paralelnost je več ali manj prikrita programerju. Vsekakor je druga pot zanimiva tudi v primeru, da programer v prvi fazi programira v paralelnem programskem jeziku, v drugi fazi pa izdelani program še dodatno časovno analizira in ga na osnovi te analize optimira in testira.

4. TRANSPUTERSKI VEČPROCESORSKI SISTEM

Sistem je zasnovan na podobnem konceptu, kakršnega sta predlagala M.Szturmowicz in M.Tudruj v delu /1/. Vsebuje tri nivoje transputerskega vezja: krmilni, sistemski in delovni nivo. Karakteristike načrtovanega sistema so:

- hierarhična struktura, ki je zgrajena iz več procesorskih nivojev, od katerih je vsak namenjen izvajanju posebne vrste opravil
- namenska komunikacijska struktura na vsakem nivoju procesorjev
- rekonfigurabilne procesorske povezave delovnega nivoja, ki omogočajo dinamično preslikavo tekočega programa na polje procesorjev z ustreznimi komunikacijskimi povezavami
- procesiranje podatkov, globalno krmiljenje v programih in izvajanje sistemskih funkcij poteka paralelno

- transputerji imajo možnost medsebojnega povezavanja s pomočjo posebne materialne opreme (C004)
- med seboj neodvisni procesi istega nivoja se izvajajo sočasno
- sistem je učinkovit za srednje zrnato paralelno procesiranje
- omogoča tudi dovolj učinkovito drobno zrnato procesiranje

Procesorsko polje na delovnem nivoju je razdeljeno v grupe (t.im. grozde) transputerjev. Transputerje v vsaki grupi je mogoče povezati v mrežo poljubne topologije. Sistem med izvajanjem nekega posla, ki je sestavljen tako iz regularnih kot neregularnih postopkov, ustrezno prilagaja polje procesorjev vsakokratnim potrebam izvajanja z rekonfiguracijo posameznih grup procesorjev delovnega nivoja. Tako je lahko v nekem trenutku izvajanja ena grupa procesorjev povezana v hiper-kocko, na kateri se izvaja nek acikličen paralelni postopek, druga grupa pa v valovnofrontno polje (wavefront array), na katerem se izvaja nek matrični račun /8/.

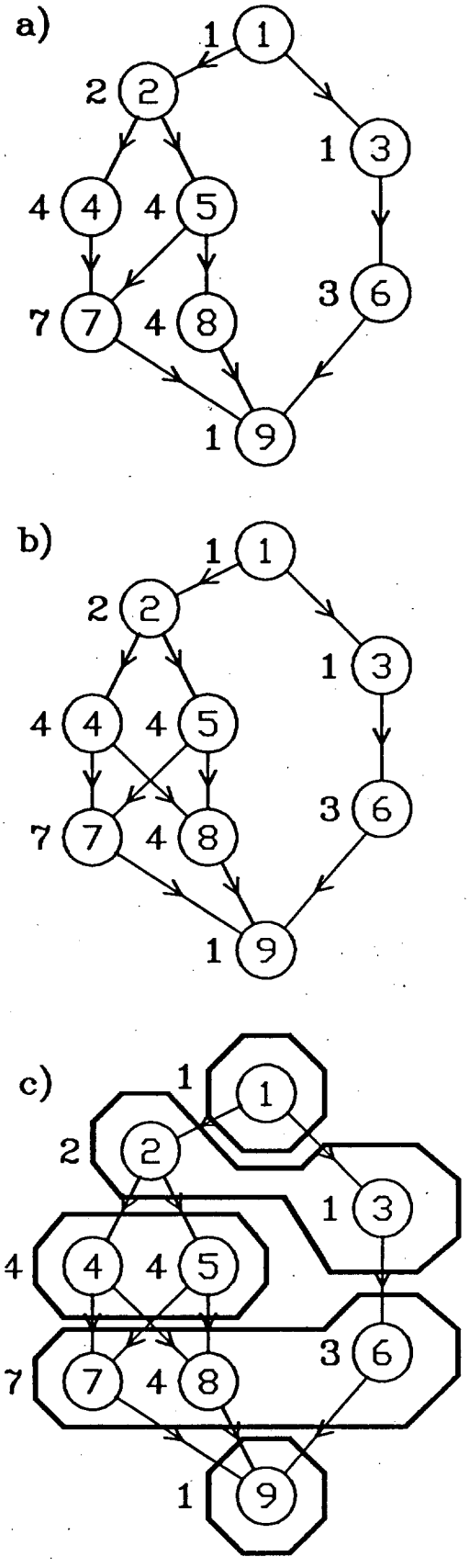
5. TOPOLOGIJA SISTEMA IN PRESLIKAVA ALGORITMA

V izdelanem konceptu rekonfigurabilnega večprocesorskega sistema topologija procesorskega polja kar najbolje ustreza zapisu algoritmov, ki določajo izvajanje opravila. Koncept kaže na dobro usklajenost topologije procesorskega polja s preslikavo algoritma.

Pri snovanju sistema je bilo upoštevano dejstvo, da je mogoče poljuben neregularen algoritem, ki ne vsebuje ciklov, imenovan "**blok**" programa, predstaviti z acikličnim grafom pretoka podatkov (ADFG). Primer takšnega grafa kaže slika 1a.

Vozlišče grafa predstavlja proces, povezave med vozlišči pa tokove podatkov. V splošnem je mogoče poljubni ADFG transformirati v hierearhični DFG (HDFG), ki predstavlja drevo (Slika 2b). ADFG je lahko preprost ali ne. Definicijo preprostega ADFG najdemo v naslednjem razdelku 5.1 in v delu Hwanga /2/. Transformacija ADFG v HADFG lahko poteka na dva načina. Prvi način transformacije ohranja podatkovno vodeno računanje in vodi v računanje, ki smo ga imenovali "**mehko sinhronizirano podatkovno vodeno računanje**". V postopku transformacije se neprepreosti grafi postopoma transformirajo v preproste le z dodajanjem novih (namišljenih) podatkovnih odvisnosti. Takšen primer je prikazan na grafu (Slika 1b), ki ga dobimo iz grafa na sliki 1a s povezavo med vozliščema 4 in 8.

Drugi način transformacije sicer ohranja princip podatkovno vodenega računanja, vendar odstopa od vodenja, ki je določeno z izvirnim ADFG. Računanje, določeno s takšno transformacijo, smo imenovali "**trdo sinhrozirano podatkovno vodeno računanje**". V postopku te transformacije se



Slika 1. ADFG in drevo grafa
a) Aciklični graf pretoka podatkov (ADFG)
b) Preprost ADFG
c) Možni sočasni procesi v ADFG

na osnovi skrbne analize programa, ki sloni na časovnih karakteristikah izvajanj posameznih procesov, v izvirnem ADFG odvzamejo nekatere povezave med vozlišči grafa. Primer takšne preslikave predstavlja preslikava grafa na sliki 1b v graf na sliki 1a. Pri uporabi takšnega načina transformacije morajo biti brezpogojno na voljo natančni podatki o dolžini ali možnih časovnih intrevalih trajanja vsakega procesa posebej.

Implementacija v tem prispevku zasnovanega transputerskega sistema in razvrščanja procesov je omejena na mehko sinhronizirano podatkovno vodeno računanje. Omejitev je pogojena z organizacijo komuniciranja in razvrščanjem procesov pri izbrani večzančni arhitekturi sistema.

Zaradi mehko sinhroniziranega podatkovnega vodenja na večzančni arhitekturi procesorskega polja, v katerem so vsi procesni elementi med seboj enakopravni, je omogočeno:

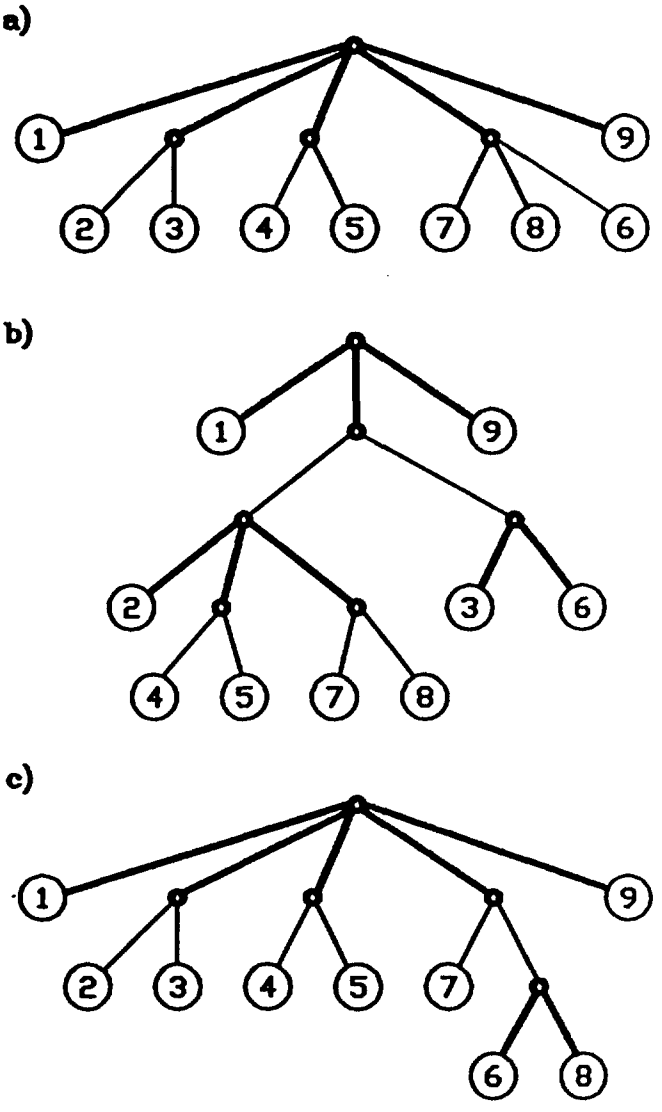
- dinamično dodeljevanje procesov
- sočasno izvajanje med seboj neodvisnih algoritmov
- medsebojno prepletanje različnih HDFG na polju procesorjev

Procesi na isti stopnji paralelnosti med seboj ne komunicirajo. Programerju ni treba skrbeti, na katerem procesorju se bo proces izvajal. Procesi se dodelujejo dinamično. Da ni del procesorskega polja preobremenjen, drugi del pa neizkoriščen, je topologija simetrična za več možnih osnovnih vstopnih mest. To pomeni, da se lahko različni algoritmi (ADFG) začnejo izvajati na različnih delih sistema. Ti deli sistema so, gledani posamično, topološko identični. Ker sistem omogoča medsebojno prepletanje različnih HDFG v procesorskem polju, je lahko bolje izkoriščen.

5.1. ADFG in HADFG

Graf pretoka podatkov opisuje izvajanje nekega programa (algoritma), ki je sestavljen iz medsebojno povezanih procesov. V nadaljevanju bomo predpostavili, da program nima povratnih zank. Graf, ki ustreza takšnemu programu, je acikličen in je prikazan na sliki 1a. Utež ob vozlišču grafa pomeni čas izvajanja procesa, ki ga vozlišče predstavlja. Ta čas je celoštevilčen večkratnik neke poljubne realne časovne enote.

Proces (vozlišče grafa) je zaključena celota. Začetek procesa je pogojen z določitvijo nabora vseh vhodnih podatkov, ki so potrebni za izvajanje procesa. Rezultat, kot izhodni podatek procesa, je navadno vhodni podatek za naslednji proces. Proces se lahko začne izvajati šele, ko ima na voljo vse potrebne podatke. Procesi, ki so med seboj neodvisni, se lahko izvajajo sočasno tako, kot kaže slika 1c. Izvajanje algoritma, ki je določen z grafom na tej sliki, je mogoče nazorno pokazati tudi z drevesno strukturiranim grafom na sliki 2a. Poudarjene razvejitve označujejo zaporedje, manj poudarjene pa sočasnost procesov.



Slika 2. Drevo grafa
a) Drevo grafa iz slike 1c
b) HADFG grafa iz slike 1b
c) Optimirano drevo grafa iz slike 1b,c

Aciklični usmerjeni graf, ki je DFG postopka, je mogoče predstaviti tudi z matematičnim zapisom. Ta predstavlja vsoto vseh možnih nizov vozlišč, po katerih je mogoče priti iz posameznih začetnih vozlišč do končnega vozlišča. Matematični zapis grafa na sliki 1b je potem:

$$X_1X_2X_4X_7X_9 + X_1X_2X_4X_8X_9 + X_1X_2X_5X_7X_9 + \\ + X_1X_2X_5X_8X_9 + X_1X_3X_6X_9 = \\ = X_1(X_2(X_4 + X_5)(X_7 + X_8) + X_3X_6)X_9$$

Graf je **preprost**, če ga je mogoče zapisati tako, da vsaka spremenljivka v njegovem matematičnem izrazu nastopa le enkrat. Iz takega matematičnega zapisa, v katerem pomeni, da se X_iX_j izvajata zaporedno, $X_k + X_m$ pa paralelno, je mogoče preprosto razbrati vse možne sočasnosti v grafu (i, j, k in m so elementi množice 1, 2, ..., 9).

Preprost acikličen DFG lahko prevedemo v **hierarhični ADFG (HADFG)**. Oblika HADFG je povsem primerljiva z

matematičnim zapisom ADFG. Preslikava ADFG na sliki 1b v odgovarjajoči HADFG na sliki 2b je določena z izpeljanim matematičnim izrazom.

Tudi tu, v HADFG je zaporedno izvajanje procesov prikazano z debelejšimi, vzporedno izvajanje pa s tanjšimi črtami. Vrstni red izvajanj je natančno določen. Poteka od leve strani drevesa proti desni strani in od zgoraj navzdol.

5.2. Analiza sočasnosti v ADFG

Algoritem, ki ga določa ADFG, je mogoče optimirati glede na čas izvajanja tako, da se v vsakem trenutku izvaja sočasno prav toliko procesov, kolikor je možno. Kot je bilo že omenjeno, tansformacija ADFG v optimirani graf (in dalje v HADFG) zahteva podatke o času trajanja posameznih procesov in splošno vodi k trdo sinhroniziranemu podatkovnemu vodenju. Pri takem vodenju lahko pride med procesiranjem algoritma do konfliktnih situacij v primerih, če se spremenijo časovne karakteristike posameznih procesov preko časovnih meja, ki so bile upoštevane v postopku transformacije.

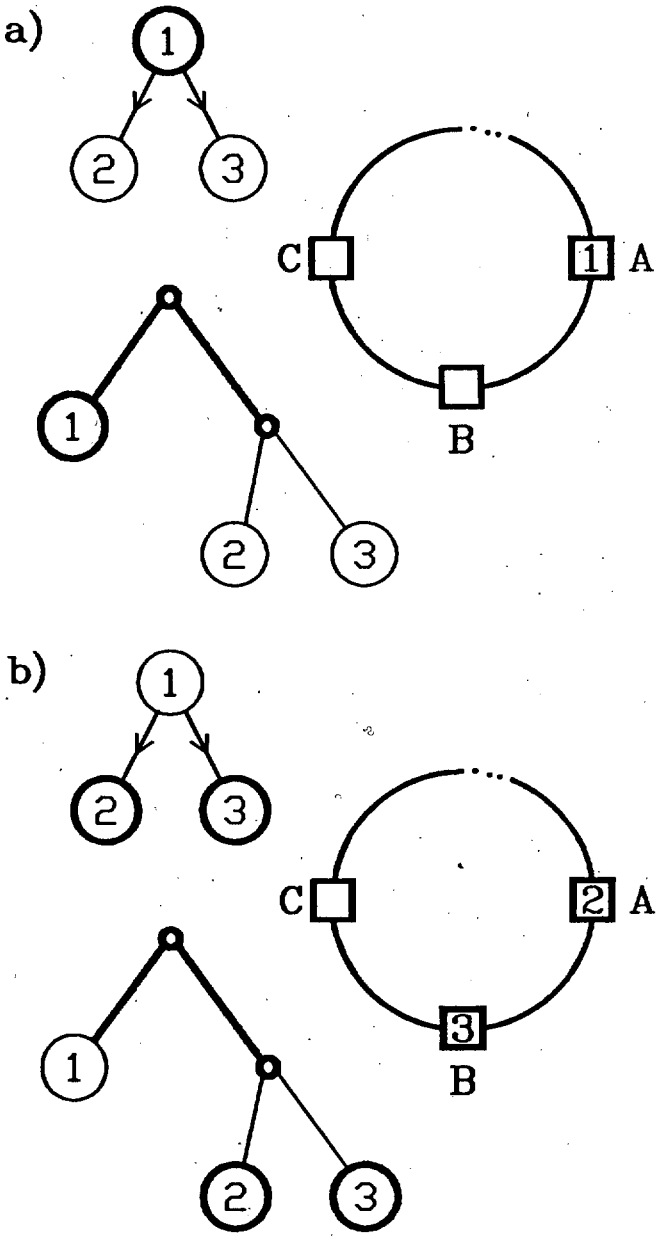
V nekaterih primerih je število procesorjev, ki je potrebno za časovno optimirano izvajanje algoritma, manjše. Včasih je tedaj mogoče z dodatno časovno analizo ugotoviti, da se lahko dva sicer vzporedna procesa izvajata sekvenčno, ne da bi se pri tem podaljšal čas izvajanja celotnega algoritma (Slika 2c). Tako optimiranje vodi k prostorski optimizaciji (tj. k manjšemu številu potrebnih procesorjev). Na primer, za izvajanje drevesa na sliki 2b so potrebni trije procesorji, za izvajanje drevesa na sliki 2c pa le dva procesorja.

5.3. Zanka in razvrščanje procesov

Ena izmed topologij, ki omogoča uresničitev prej navedenih zahtev, je zanka ali prstan. Zanka predstavlja več krožno povezanih procesnih elementov. V enostavnih zankah ima procesor povezavo le s svojima neposrednima sosedoma. Razdalje med procesorji v zanki se lahko skrajšajo z dodatnimi križnimi povezavami med procesorji, ki preskakujejo enega ali več procesorjev. Problem krajših poti v takšnih zankah sta podrobneje obravnavala Pisanski in Žerovnik v delu /3/. Z njimi se sicer poveča prepustnost zanke, žal pa postanejo mehanizmi prenašanja sporočil kompleksnejši.

Iz primera preprostega prstana na sliki 3 je razviden potek izmenjav sporočil med procesorji (procesi). Eden od treh procesorjev sprejme sporočilo o algoritmu, ki se mora izvajati. Naj bo ta procesor A. Algoritem določa dva vzporedna procesa, označena z 2 in 3, ki se pričneta izvajati, ko je proces 1 na procesorju A (Slika 3a) končan. Prvi izmed njiju (2) ostane na istem procesorju (A), drugi (3) pa se preseli na sosednji procesor (B). Slednjo situacijo kaže slika 3b.

Zanka omogoča dokaj uspešno razporejanje procesov tudi, če je procesov več kot procesorjev. Ko procesi zasedejo vse

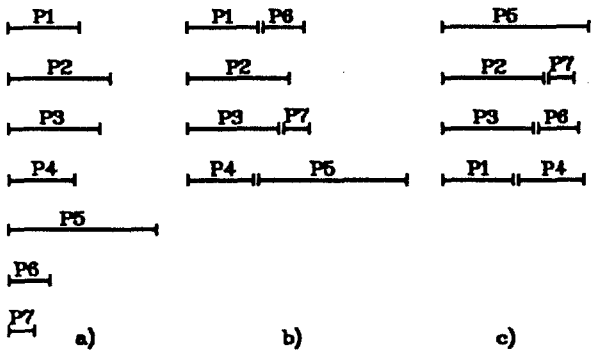


Slika 3. Izvajanje algoritma na procesorjih, ki so povezani v zanko.

razpoložljive procesorje, ostali procesi čakajo, da se eden od procesorjev sprosti. Mehanizem, ki skrbi za takšno dodeljevanje, se lahko izvede s pomočjo žetona, ki kroži po prstanu in išče prost procesor. Ko se eden od procesorjev sprosti in v vrsti za izvajanje čaka še kak proces, mu ga krožeči žeton dodeli. Na ta način je dosežen nekakšen avtomatizem razvrščanja. Vsi sodelujoči procesorji so enakopravni in za razvrščanje procesov ne potrebujejo nekega posebnega krmilnega procesorja.

Avtomatično razporejanje procesov po zgoraj opisanem načinu načelno ne zahteva podatke o trajanju posameznih

procesov. Brez predhodne skrbne analize sočasnosti in hkratne analize dolžin trajanja posameznih procesov, izvajanje algoritma ne bo dovolj časovno optimirano. Zato je potrebno vpeljati primerno strategijo razvrščanja sočasnih procesov, kar je razvidno iz primera na sliki 4. Na voljo so štirje procesorji v zanki, ki morajo izvesti 7 različno dolgih vzporednih procesov, prikazanih na sliki 4a. Optimalno izvajanje algoritma je v takem primeru možno, če obstajajo sočasni procesi, ki se lahko izvajajo zaporedno na istem procesorju, ne da bi se pri tem čas izvajanja algoritma podaljšal. Razvrščenost procesov na sliki 4b je neugodna.



Slika 4. Razvrščanje procesov

Procesi so razvrščeni po načinu FCFS (first come first served), kar ne vodi vedno v optimalno izvajanje algoritma. Procesa P4 in P5 se izvajata zaporedno na procesorju 4. Zato je čas izvajanja občutno daljši, kot v primeru na sliki 4c. V slednjem primeru sta procesa, ki se izvajata na procesorju 4, bolj ugodno izbrana. Način razvrščanja je najugodnejši, če za vsak par procesorjev velja, da sta vsoti trajanj procesov na posameznih procesorjih para čim bolj izenačeni.

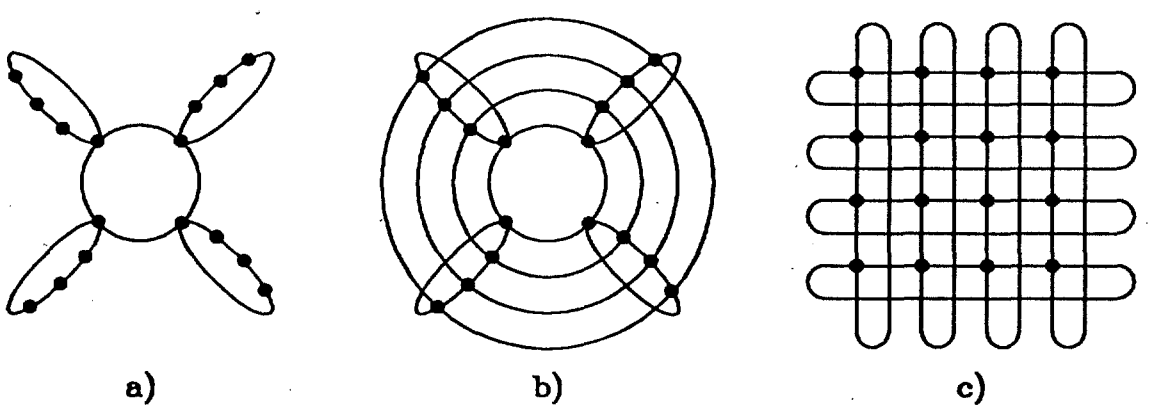
Iz gornjega je razvidno, da se pri neprimerno razporejenih procesih zmogljivost sistema zmanjša, vendar sistem pravilno deluje v obeh primerih. Tak mehanizem razvrščanja procesov je zato uporaben tudi pri izvajanju časovno nepredvidljivih procesov.

5.4. Topologija sistema

Znotraj prstana so procesorji med seboj povsem enakopravni. Procesor lahko vedno odda sporočilo le svojemu sosedu; tudi sprejemanje sporočil poteka preko sosedov. Če ima prstan velike razsežnosti pomeni, da bo čas potovanja sporočil sorazmerno dolg. Zato je v izdelanem konceptu število procesorjev v prstanu omejeno. Manj procesorjev v eni zanki je seveda premalo za zahtevnejše aplikacije, zato je vsakemu procesorju v zanki, imenovani **"osnovna zanka"**, dodeljena še ena zanka, imenovana **"sozanka"**. Osnovno zanko in sozanke, ki so vgnezdene v osnovno zanko, prikazuje slika 5a. Dobljena topologija ima prvo **"stopnjo vgnezdenosti zank"**. Z nadaljnim vgnezdevanjem je mogoče doseči poljubno stopnjo vgnezdenosti. Vsak procesni element na spojišču dveh prstanov ima štiri sosede. Pri predpostavki, da so prstani enosmerni, lahko procesor sprejema sporočila od dveh sosedov in jih tudi oddaja dvema sosedoma. To omogoča, da se v funkciji procesorja v zanki vpelje transputer. Število transputerjev v zanki je 4. To število ni izbrano naključno, ustreza številu transputerjev na komercialno dostopnih transputerskih vtičnih enotah, ki jih proizvaja INMOS /4/. Na teh vtičnih enotah so transputerji že krožno povezani.

Sistem je mogoče zaključiti tako, da se vse ostale istoležne transputerje, ki še niso križno povezani, poveže v zanko. Dobljeno vezje je prikazano na sliki 5b. To je mogoče predstaviti tudi v matrični razvrstitvi na sliki 5c ali kot trodimenzionalno hiper-kocko /5/. Procesorjev v polju je 16. Takšna topologija omogoča enostavno uporabo 2x2 procesorskih plošč s po štirimi transputerji. Povezave med transputerji omogočajo, da poteka razvrščanje procesov algoritma skladno s hierarhično strukturo HADFG. Poteka lahko od enega vstopnega prstana, ki ima vstopni procesor, do drugega. Sistem ima lahko tudi več vhodnih zank s po enim vhodnim procesorjem. Tako je možnih več neodvisnih in enakovrednih vhodov v sistem.

Transputer ima na voljo štiri dvosmerne zanke. S pomočjo dveh takšnih zank sta v enem prstanu realizirani dve med seboj neodvisni komunikacijski zanki. Vsaka izmed njih je namenjena komunikaciji v eni od obeh smeri. Smotrno je,



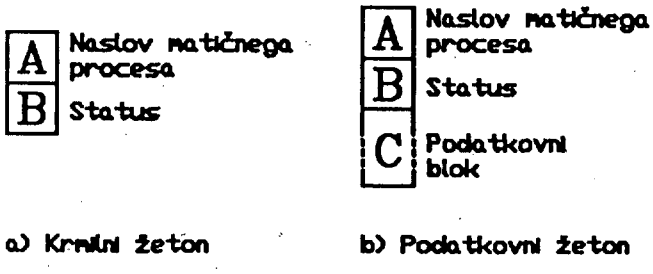
Slika 5. Topologija polja procesorjev

da je smer komunikacije določena s funkcijo komuniciranja. V eni smeri se prenašajo podatkovne, v drugi pa krmilne in statusne informacije. Enkraten prenos podatkov, ki so številnejši, traja dalj časa, prenos krmilnih in statusnih informacij pa krajši čas. Odzivi na krmilne informacije morajo biti čim hitrejši. Zaradi majhnega premera prstanov ni pričakovati, da bi čas komuniciranja med procesi (transputerji) bistveno vplival na učinkovitost sistema.

Prenašanje sporočil med procesnimi elementi je izvedeno preprosto s podajanjem žetonov, ki krožijo v zanki. Vsak žeton opiše poln krog. Procesor, ki je poslal žeton v zanko, ga mora kasneje iz nje tudi izločiti. Na tak način je zagotovljeno, da žeton obide vse procesorje v zanki. Praviloma lahko v zanki kroži več žetonov. Žeton se lahko nekje na poti po zanki ustavi pri procesorju, ki želi sprejeti od žetona neko sporočilo ali mu ga predati. Med kroženjem žeton ohranja naslov procesa (in s tem tudi naslov procesorja), ki je žeton poslal v kroženje.

5.5. Medprocesorske komunikacije

Medprocesorske komunikacije potekajo preko sporočil. Sporočila prenašajo žetoni. Teh je več vrst. Omenjeno je bilo že, da se uporabljata ločeni smeri za prenos podatkovnih in krmilnih informacij. Skladno s tem se razlikujejo žetoni, ki te prenose izvajajo. Poimenovani so kot krmilni in podatkovni žetoni. Načelno zgradbo žetonov obeh vrst prikazuje slika 6.



Slika 6. Zgradba žetonov

Možni statusi procesorja so:

- zaseden
- pripravljen
- nezaseden (prost)

Procesor je **zaseden**, kadar izvaja proces (v sklopu uporabniškega programa). Procesor ima status **pripravljen**osti od trenutka, ko sprejme zahtevo za izvajanje uporabniškega procesa, in do trenutka, ko prične proces izvajati. Kadar ni v nobenem od zgoraj opisanih statusov, je procesor **nezaseden** in opravlja le funkcijo prenašanja in razpoznavanja sporočil.

a) Krmilni žetoni. Krmilni žetoni (Slika 6a) vsebujejo le najosnovnejše informacije: naslov pošiljatelja in status oz. zahtevo, ki jo morajo posredovati ali vsem udeležencem v prstanu ali le enemu točno določenemu udeležencu.

Med krmilne žetone prištevamo žetona "**proces**" in "**konec**". Žeton "proces" išče prost procesor v zanki, žeton "konec" pa obvesti vse procesorje v zanki, da je v njej procesor, ki ima status "prost".

b) Podatkovni žeton. Sporočila, ki jih prenaša podatkovni žeton (Slika 6b), so razdeljena v dva dela. Prvi del ima enako zgradbo kot krmilni žeton in vsebuje naslov pošiljatelja in status podatka, ki ga sporočilo prenaša. Drugi del, odvisno od statusa, vsebuje: algoritme za proces in ustrezne začetne podatke ali rezulate izvršenega procesa.

Mehanizem komuniciranja z žetoni določa komunikacijski protokol, ki je podrobneje opisan v naslednjem razdelku.

5.5.1. Sporočila in komunikacijski protokol

V polju procesorjev se lahko izvaja en sam ali več vzporedno tekočih blokov. Podmnožica procesov, ki pripadajo istemu bloku in se izvajajo v isti zanki ali njeni sozanki, ima svoj "**matični proces**". Matični proces je element te podmnožice. Izvaja se na "**matičnem procesorju**", ki predstavlja koren nekega poddrevesa v HADFG.

Matični procesor pošlje po žetonu sporočilo o zahtevi po paralelnem izvajanju procesov. Vsebino žetona predstavljata naslov matičnega procesa (A) in status žetona (B). Status tega žetona je "**proces**" in žeton s takšnim statusom se imenuje žeton "proces". Ko žeton "proces" na poti po zanki naleti na prost procesor, mu ta spremeni statusni del (B) v status "**pripravljen**". Naslovni del žetona (A) ostane nespremenjen. Sporočilo, da je žeton našel prost procesor, je namenjeno matičnemu procesu, ki je poslal zahtevo po prostem procesorju. Ko žeton s statusom "pripravljen" dospe do matičnega procesorja (procesa), je ta obveščen, da je v zanki procesor, ki je nezaseden in pripravljen sprejeti podatke za izvajanje novega procesa. V primeru, da žeton na poti po zanki ne najde prostega procesorja, ostane status žetona nespremenjen. Matični procesor ga zadrži, dokler po žetonu "konec" ne sprejme sporočila, da je v zanki procesor, ki je nezaseden.

Ko matični procesor sprejme sporočilo, da obstaja procesor s statusom "pripravljen", pošlje podatkovni žeton. Ta vsebuje tri dele. Naslovni del (A) vsebuje naslov matičnega procesa. Statusni del (B) pove, da gre za vhodne podatke. V tem primeru ima žeton status "**podatek**". Podatkovni del (C) vsebuje poleg vhodnih podatkov tudi programski kod. Procesor, ki sprejme podatkovni žeton, preide v status "zaseden" in prične izvajati nalogo, ki je določena s programskim kodom, podatkovni žeton pa zadrži do konca izvajanja naloge. Po končani nalogi procesor spremeni statusni del zadržanega podatkovnega žetona v status "**rezultat**" in v podatkovnem bloku tega žetona posreduje rezultate matičnemu procesu. Potem, ko

procesor odda rezultate, generira krmilni žeton "konec". Ta žeton vsebuje status **"konec"** in z njim signalizira, da je v zanki prost procesor, ki lahko prevzame novo nalogo.

5.5.2. Preslikava algoritma na polje procesorjev

Preslikavo nekega algoritma na načrtovano polje procesorjev sestavljata dva bistvena koraka:

1. Predstavitev algoritma v paralelnem zapisu.

V poglavjih 2 in 3 je bila že nakazana problematika paralelnega zapisa algoritma. Ta največji in hkrati najpomembnejši korak k preslikavi algoritma sestavljajo naslednji postopki:

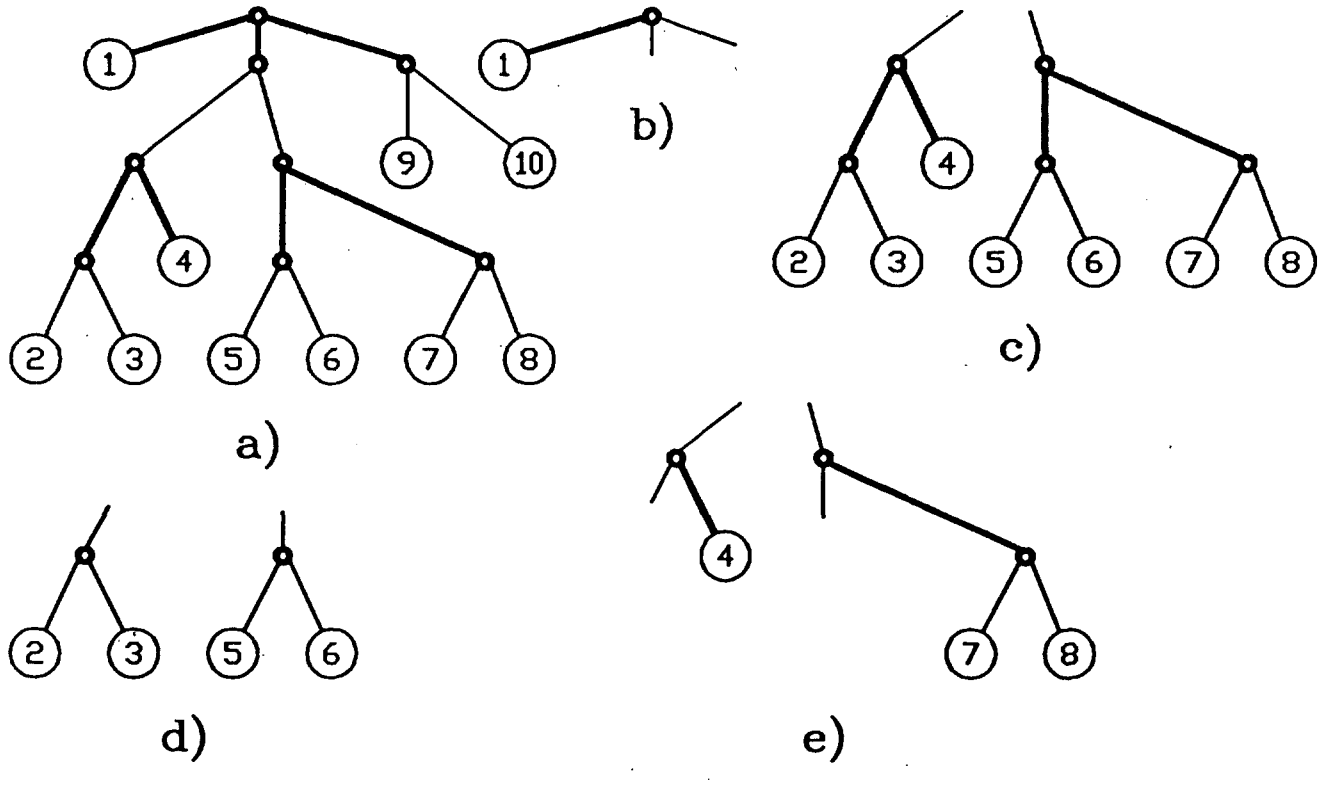
- granulacija programa (algoritma)
- dekompozicija algoritma v aciklične podalgoritme (bloke)
- odkrivanje sočasnih procesov
- paralelni zapis blokov programa v matematični obliki, ki je predstavljiva s t.im. hierarhičnim acikličnim podatkovno pretočnim grafom HADFG.
- optimalno razvrščanje procesov na osnovi časovne analize

2. Generiranje naslovnih kodov vseh poddreves v množici hierarhičnih acikličnih podatkovno pretočnih grafov.

Naslovni kod je mogoče neposredno generirati iz HADFG. Predstavlja naslov nekega poddrevesa algoritma. Med njegovim izvajanjem se naslovni kodi dinamično preslikajo na matične procesorje transputerskega polja.

Po prvem koraku preslikave celotnega algoritma, ki se izvaja na sistemskem nivoju hierarhičnega večprocesorskega sistema (HMPS) /1/, je algoritem podan v matematični obliki (računalniškem zapisu) tako, da je predstavljiv z množico drevesnih grafov HADFG, ki so bloki programa, in z osnovnim grafom, ki jih povezuje med seboj. Osnovni graf je cikličen usmerjeni graf (CDFG), ki opisuje vse zanke algoritma. Program, ki je določen z CDFG, se lahko izvaja v vstopnem transputerju procesorskega polja, ali se izvaja na nekem drugem transputerju delovnega (uporabniškega) nivoja sistema. Vhodni transputer posreduje matičnemu procesorju v izvajanje bloke algoritma zaporedno ali kvazi vzporedno, če CDFG vsebuje tudi paralelne zanke. V tem primeru je ugodno (ni pa nujno), da bloki vstopajo v polje paralelno (ali kvazi paralelno) preko večih vhodnih transputerjev.

V drugem koraku se blok, ki je določen s pripadajočim HADFG, preslika na procesorsko polje. Pri tem se posamezni procesi bloka dinamično razvrščajo po polju transputerjev. Mehanizem dodeljevanja procesov je enak



Slika 7. Ponazoritev preslikave bloka na polje transputerjev.

za vse procesorje polja. Preslikavo bloka prikazuje primer na sliki 7.

Pri razvrščanju sočasnih procesov je uporabljena strategija razvrščanja po časovni uteži posameznih procesov. Paralelni proces (P1), ki se (ali se verjetno) izvaja najdlje, ima največjo utež. Izvaja se na "prvem" procesorju v zanki. Takšen proces P1 stoji v paralelni množici HADFG pred procesom P2. Po časovnih utežeh so tako razvrščeni sočasni procesi in poddrevesa vseh paralelnih množic v HADFG. Pri tem je mišljeno, da stoji proces P1 pred procesom P2, če se v HADFG nahaja v paralelni množici prvi, gledano od leve proti desni.

Procesor polja, ki se nahaja v zanki vhodnega procesorja in prevzame podatke, ki so potrebni za izvajanje HADFG, je matični procesor tega HADFG. Ta procesor na najvišjem nivoju prevzame nadzor nad izvajanjem procesov, ki se dinamično vključujejo in izvajajo na istem nivoju, na katerem se sam nahaja. Izvajanje na nižjem nivoju preda naslednjemu matičnemu procesorju v hierarhiji izvajanja procesov, ki je določena z HADFG.

Naj bo začetni nivo grafa sekvenčen, kot kaže slika 7a. Prvi prosti procesor najprej poskrbi za izvajanje procesa 1 (Slika 7b). Ko se proces 1 konča, se mora izvršiti poddrevo, ki izvira iz druge sekvenčne veje. Ta veja se deli na dve vzporedni veji. Jasno je, da se mora poddrevo vsake od obeh vej izvajati na ločenih podmnožicah procesorjev (Slika 7c). Procesor, ki sprejme neko poddrevo, postane matični procesor tega poddrevesa in prične takoj izvajati proces, ki se na poti po drevesu nahaja na prvem mestu. Procesi, ki se v HADFG nahajajo na najnižjem nivoju posameznih vej drevesa (kar pomeni, da v smeri od zgoraj navzdol nimajo naslednika) se imenujejo "**terminalni procesi**". Med potovanjem po grafu navzdol dobi vsako poddrevo ali terminalno vozlišče, ki izhaja iz vzporedne veje, svoj matični procesor (proces). V splošnem ni nujno, da so matični procesorji posameznih delov grafa različni procesorji. Vsak procesor je lahko hkrati matični procesor za več podgrafov. Vsak HADFG se tako postopoma deli na manjše dele, ki se razvrščajo po procesorjih. Razvrščanje se ustavi na terminalnih procesih (Slika 7d in 7e), ki so, ali

- zaporedje procesov, določeno s sekvenčnimi vejami, ki se ne delijo več v vzporedne veje, ali
- procesi vzporednih vej, ki se ne delijo več v sekvenčne veje.

Ko se terminalni procesi, ki se izvajajo na matičnem procesorju ali na njemu pripadajočih procesorjih, končajo, predajo rezultate svojemu matičnemu procesorju. Ta nato izvrši naslednji korak v podgrafu (Slika 7e). Prične izvajati proces 4. Postopek se tako nadaljuje, dokler niso izvršeni vsi procesi, ki jih HADFG določa.

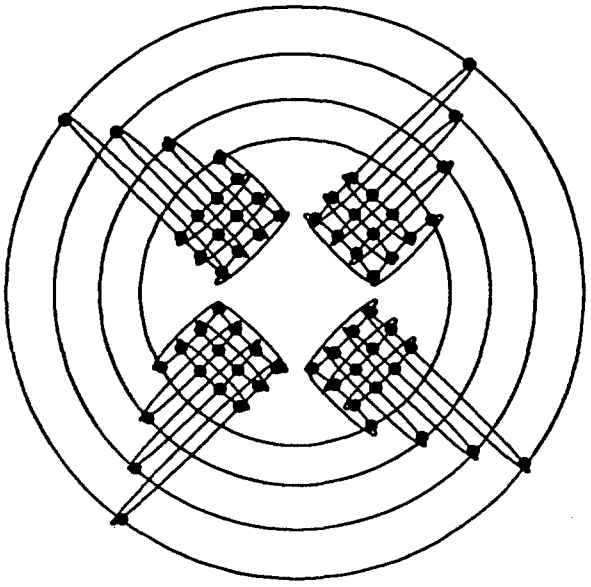
Vsak procesor ima tabelo, v kateri hrani informacijo o želenem procesu. Informacija je predstavljena v obliki HADFG. Prvi matičen procesor ima v tej tabeli zapisan celoten algoritem. Poleg algoritma shranjuje tudi podatke, kot so vhodni podatki in vmesni rezultati, ki so potrebni za njegovo izvajanje.

Preslikava algoritma na več zank, ki so vgnezdene druga v drugo, poteka po pravilu "prioritetne smeri", ki prispeva k boljši izkoriščenosti procesorskega polja. V zvezi s tem pravilom sta vpeljana pojma "**primarne**" in "**sekundarne**" zanke. Matični procesor sprejme podatke za izvajanje poddrevesa po eni od dveh zank polja, ki jima pripada. Smer, po kateri matični procesor sprejme podatke, je primarna smer. Matični procesor poskuša predati podatke za izvajanje preostalega dela drevesa, ki ga ne izvaja sam, najpreje procesorju v sozanki (tj. v sekundarni smeri) in šele, če mu to ne uspe, v primarni smeri (po osnovni zanki). Mehanizem predaje deluje takole: Kot je bilo rečeno, pošlje matični procesor žeton "proces" najpreje v sekundarno zanko. Če so vsi procesorji v tej zanki zasedeni, se žeton "proces" z nespremenjeno vsebino vrne k matičnemu procesu, ta pa ga nato pošlje v sozanko, tj. v primarno smer. V primeru, da je tudi ta zanka zasedena, matični procesor čaka, dokler se ne sprosti procesor v eni ali drugi zanki. O tem je matični procesor obveščen po sproščeni zanki z žetonom "konec" , nakar se celoten ciklus z žetonom "proces" v sproščeni zanki ponovi.

Do naslednje akcije pride, ko prispe žeton "pripravljen" do matičnega procesorja. Ta vzame iz tabele poddrevo (ali proces), ki čaka na izvajanje, in ga z žetonom "podatki" pošlje v prstan. Žeton "podatki " vsebuje kodo izvornega (matičnega) procesa in izvora podatkov. Matičnemu procesorju cilj potovanja podatkovnega žetona ni poznan. Na poti po zanki podatkovni žeton išče procesor, ki je pripravljen. Ko ga najde, spremeni njegovo stanje "pripravljen" v stanje "zaseden", procesor pa sprejme poddrevo, ki mu je namenjeno, in poskrbi za njegovo izvajanje. Žeton "podatki" ostane med izvajanjem poddrevesa na procesorju, ki njegovo izvajanje nadzira. Ko je poddrevo izvršeno, nadzorni (matični) procesor pošlje v oba prstana žeton "konec". Z njima obvesti procesorje v obeh prstanih, da se je procesor v zanki sprostil (da je nezaseden), s podatkovnim žetonom, ki mu dodeli status "rezultat", pa pošlje rezultat v prstan, kjer se nahaja njegov matični procesor. Ko žeton "rezultat" prispe do svojega matičnega procesorja (pošiljatelja žetona "podatki", ki se nahaja na višjem nivoju), ga ta vzame iz obtoka. Sprejete podatke razvrsti skladno z izvornim kodom, ki je bil poslan po žetonu "podatki". S tem je izvajanje podgrafa končano.

Iz opisanega lahko zaključimo naslednje:

- Hierarhično razvrščanje procesov po procesorjih v hierarhično vgnezdenih zankah zmanjšuje obremenitev komunikacijskih poti, saj potujejo sporočila preko manjšega števila posrednikov.
- Zaradi manjšega števila procesorjev je možna manjša razvejitev komunikacijskih poti in zato slabša izkoriščenost procesorskega polja.
- Izstop iz zanke v sozanko je mogoč le na matičnem procesorju. Na ta način se pri manjšem številu potrebnih komunikacij med procesorji in krajši poti



Slika 8. Razširitev procesorskega polja na 4 x 16 procesorjev.

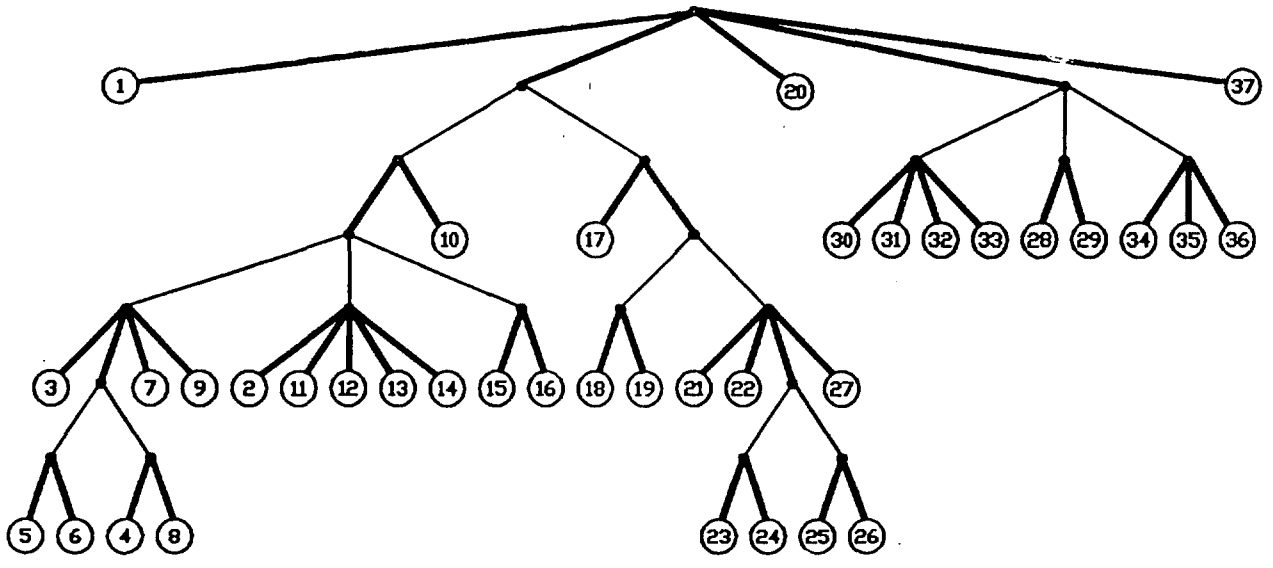
od poljubnega procesa v polju do vstopnega procesa (tj. do procesa, ki se izvaja na vstopnem procesorju), mehanizem komuniciranja poenostavi.

- Teoretično se lahko blok (ADFG) preslika na poljubno število nivojev, praktično pa je to število omejeno s številom sočasnih procesov ali/in z velikostjo procesorskega polja. Število nivojev je določeno s številom med seboj vgnezdenih zank (prstanov), na katerih se blok izvaja. Omejitev zaradi prevelikega števila sočasnih procesov istega poddrevesa, ki se izvajajo na istem nivoju, je mogoče odpraviti s takšno transformacijo ADFG v HADFG, da se v postopku transformacije uporabi postopek translacije nad presežnim delom sočasnih procesov. S postopkom translacije se "odvečne" procese prenese na nižji nivo. Druga omejitev je odpravljiva z manjšo modifikacijo tu obravnavane topologije polja. Osnovni modul transputerskega polja je namreč mogoče tako organizirati, da dopušča gradnjo bolj obsežnih polj z vgnezdevanjem zank na večih nivojih do željene stopnje. Ena od možnih razširitev polja je pokazana na sliki 8.
- Procesi se izvajajo tudi na matičnem procesorju, kar prispeva k manjšemu številu potrebnih komunikacij.
- Vsak procesor transputerskega polja mora med izvajanjem uporabniškega procesa zagotavljati razpoznavanje žetonov in nemoten prenos sporočil svojim sosedom.
- Vsi procesorji imajo enako operacijsko programsko opremo. Izjemo predstavlja le transputer na vstopu v procesorsko polje.

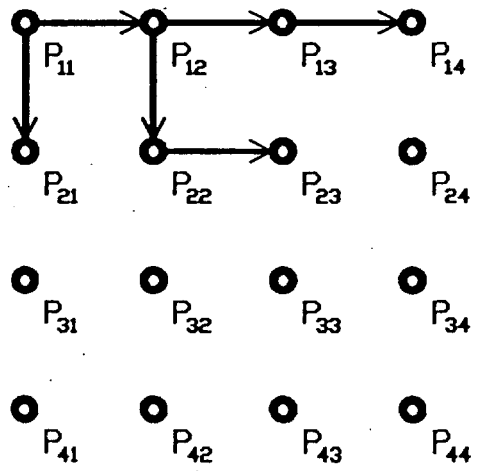
Preslikava algoritma na polje procesorjev je prikazana na naslednjem primeru: Slika 9 kaže HADFG bloka **B**, ki se preslika na polje transputerjev na sliki 10. Mehanizem razvrščanja porazdeli procese bloka **B** po transputerjih tako, kot kaže tabela na sliki 11. Transputer Pij, se nahaja na presečišču zank i, j. Preslikava bloka **B** na polje transputerjev je razvidna iz slike 10 po uporabljenih komunikacijskih poteh med sedmimi procesorji P11 do P23, ki so bili zaseženi med izvajanjem bloka.

7. ZAKLJUČEK

Sistem je prvenstveno namenjen raziskavam večprocesnih paralelnih sistemov. Pričakovati je, da bo načrtovana arhitektura večtransputerskega sistema učinkovita v sistemih,



Slika 9. HADFG bloka **B**



Slika 10. Preslikava algoritma B na procesorsko polje.

ki delajo v realnem času in so namenjeni vodenju obsežnih in zahtevnih procesov z visoko stopnjo zanesljivosti /6,7/.

Iz pričujočega dela, je mogoče strniti naslednje ugotovitve:

- Načrtovani sistem je rekonfigurabilen.
- Polje transputerjev, ki je vgrajeno v delovni (uporabniški) nivo sistema HMPS, je razširljivo. Mehanizem razvrščanja je namreč neodvisen od števila transputerjev. Z dodajanjem novih transputerjev v posamezne zanke, ni potrebno spreminjati podporne programske in materialne opreme. Zato je možno število procesorjev v zanki neomejeno večati. Smiselno pa je, da to število ni preveliko. V nasprotnem primeru daljše komunikacijske poti upočasnijo delovanje sistema.
- Pri optimiranju programa je pomembna primerna izbira sinhronizacije podatkovnega vodenja. Vodenje je lahko:

  a) asinhrono
  b) mehko sinhronizirano
  c) trdo sinhronizirano

- Postopek preslikave na koraku 1 in 2 (glej poglavje 5.5.2) je mogoče v celoti avtomatizirati in s tem bistveno olajšati delo programerja.
- Mehanizem dodeljevanja procesov je enak za vse procesorje polja, kar bistveno prispeva k prilagodljivosti sistema.
- Več vstopnih transputerjev lahko omogoča večjo izkoriščenost polja in bolj učinkovito proritetno izvajanje.

V zvezi s hierarhičnim večprocesorskim sistemom je odprtih več vprašanj, ki se nanašajo predvsem na verifikacijo predlaganega sistema. Simulacija sistema naj bi pomagala dati odgovor na številna vprašanja, med njimi:

- Kakšna je možnost zasičenosti poti za prenos podatkov?

Procesor	Procesi
P11	1, 3, 5, 6, 7, 9, 10, 20, 30, 31, 32, 33, 37
P22	17, 18, 19, 28, 29
P13	2, 11, 12, 13, 14, 34, 35, 36
P14	15, 16
P21	4, 8
P22	21, 22, 23, 24, 27
P23	25, 26

Slika 11. Tabela razvrščenosti procesov

- Kakšna je porazdelitev zasičenosti v procesorskem polju?
- Kakšna je učinkovitost sistema v odvisnosti od različnih parametrov, kot so granulacija, velikost procesorskega polja in drugi?

Nadaljne raziskave naj bi bile osredotočene v izboljšave karakteristik sistema. V ta namen naj bi obravnavale:

- dvosmerne komunikacije med procesorji z optimiranjem prenosnih poti
- komunikacije med sočasnimi procesi v primerih trde sinhronizacije podatkovnega vodenja
- prioritetno vodenje
- problematiko razvoja učinkovitih orodij za avtomatizirano načrtovanje programske opreme
- topologijo povezav med procesorji, ki omogoča preprosto modularno večanje procesorskega polja

9. LITERATURA

/1/ M.Szturmowicz and M.Tudruj, A multi-layer transputer network for efficient parallel execution of occam programs, Microprocessing and Microprogramming 28 (1989) pp.133-138.

/2/ K.Hwang and F.A.Briggs, Computer architecture and parallel processing, McGraw-Hill Book Company, 1984.

/3/ T.Pisanski and J.Žerovnik, Computing Diameter in Multiple-loop Networks, Preprint Series Dept. Math. University E.K. Ljubljana, 27(1989) No.286.

/4/ INMOS Spectrum, "Product Information, The Transputer Family", March 1988.

/5/ L.C.Waring, A general purpose communications shell for a network of transputers, Microprocessing and Microprogramming 29 (1990) pp.107-119.

/6/ Peter Kolbezen and Peter Zaveršek, Transputers for Embedded Real-Time Systems, Informatica, A Journal of Computing and Informatics, Vol.14, No 3, July 1990, pp.35-43.

/7/ Barbara Koroušić, Jim E.Cooling and Peter Kolbezen, Real-Time Executives for Embedded Microprocessor Applications, Informatica, A Journal of Computing and Informatics, Vol.14, No 4, October 1990, pp.58-63.

/8/ Peter Kolbezen, Borut Robič in Branko Mihovilović, Podatkovno vodeno računanje na polju transputerjev, MIPRO - Savjetovanje o novim generacijama računara - NG, str. 5-52 do 56, Rijeka-Opatija, maj 1989.

DOLOČANJE POLOŽAJA TELES V PROSTORU IZ ENE SAME 2-D SLIKE

INFORMATICA 1/91

Barbara Lakner
Insitut Jožef Stefan, Ljubljana

Keywords: attitude determination, methods, polyhedral objects.

Podan je pregled metod za določanje položaja poliedrskih teles v prostoru iz ene same 2-D slike. Opisane so nekatere metode, ki pridejo v poštev pri prepoznavanju poliedrskih teles: Robertsova, analitični Horaudova in Dhomejeva ter iteracijska Lowejeva.

ABSTRACT: A survey of methods for determination of the attitude of 3-D polyhedral objects from a single perspective view is presented. These methods are used in systems for recognizing polyhedral objects from gray-level pictures. Methods described: Roberts' method, analytical Horaud's method, analytical Dhome's method and Lowe's iterative method.

1. UVOD

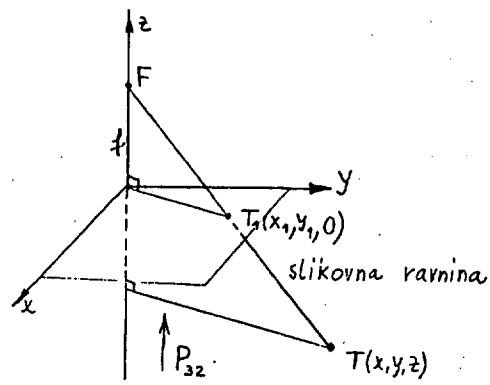
Ogledali si bomo nekaj metod za določanje položaja poliederskega telesa iz ene same 2-D slike. Opisane metode so nastale pri prepoznavanju poliedrskih teles, kjer moramo ugotoviti, ali so na sliki telesa, opisana z vnaprej danimi modeli in kje v prostoru ležijo. Pri tem je treba določiti 6 prostostnih stopenj (tri kote zavrtitve okrog koordinatnih osi x, y, z in translacijski vektor). Iskana rotacija in translacija postavita model v prostor tako, da se projekcija transformiranega modela ujema z dano sliko.

Položaj telesa v prostoru lahko določimo na različne načine: z aktivnimi metodami zaznavanja (dobivanje globinske slike s pomočjo laserskih žarkov, fotometrične metode), s stereovidom, z informacijami, dobljenimi s pomočjo gibanja (obravnava več zaporednih slik) in z merjenjem globine s pomočjo teksture in senc. Vendar aktivne metode ne pridejo vedno v poštev (če bi imeli npr. dva taka sistema za prepoznavanje drugega ob drugem, bi se laserski žarki lahko motili med sabo); stereovid ne pride v poštev za določanje položaja teles, ki so zelo oddaljena; informacija, dobljena s pomočjo gibanja je uporabna, kadar imamo dovolj veliko relativno gibanje med opazovalcem in objektom, kar je praktično izvedljivo za blizu stoječe objekte; merjenje globine s pomočjo teksture in senc pa pride v poštev za posebne primere in ni vedno dovolj natančno.

Pionirsko delo na področju prepoznavanja poliedrskih teles je opravil Roberts [1]; v njem se ukvarja tudi z določanjem položaja telesa. Kanade [2] je ta problem rešil analitično za ortografsko projekcijo, vendar imamo opravka s perspektivno projekcijo, zato si bomo ogledovali le take metode. Barnard [3] interpretira trojico črt na sliki kot oglišče, kjer se stikajo trije robovi. Problem je rešil pri predpostavki, da so koti med robovi enaki $\pi/2$. Shakunagar in Kaneko [4] sta predpostavila, da je le en rob izmed trojice pravokoten na ravnino ostalih dveh. Horaudova metoda [5] nima omejitve za kote med robovi; problem prevede na reševanje sistema transcendentnih enačb, ki ga reši s tabeliranjem. Dhome in soavtorji [6] rešujejo problem analitično z iskanjem ničel polinoma 8 stopnje. Rešitev Huttenlocherja in Ullmana [7] je vsebovana v prej navedeni metodi. Lowe [8], [9] rešuje problem iterativno z Newtonovo metodo. Pri danem začetnem približku dobimo ob obravnavi treh točk eno samo rešitev. Zen Chen s soavtorji [10] določa položaj kocke s pomočjo točk izginjanja (vanishing points). Metoda pride v poštev pri objektih, ki so posneti tako od blizu, da se nosilke vzporednih stranic na sliki sekajo.

2. ROBERTSOVA METODA

Oglejmo si najprej relacijo med sliko in sceno: slikovna ravnina naj bo kar ravnina (x, y), vse točke na sceni naj imajo z koordinato negativno in naj bo gorišče v točki $\mathbf{F}(0, 0, f)$.



Če je $\mathbf{T}(x,y,z)$ točka na sceni in $\mathbf{T}_i(x_1,y_1,0)$ njena projekcija, je projekcija iz scene na slikovno ravnino

$$P_{32}(x,y,z) = (\frac{x}{f-z}f, \frac{y}{f-z}f).$$

Uvedemo še *globino projicirane točke*, katere z koordinato definiramo kot $z_1 := \frac{z}{f-z}f$. Zdaj imamo preslikavo $P_{33}: R^3 \to R^3$; $P_{33}(x,y,z) = (\frac{x}{f-z}f, \frac{y}{f-z}f, \frac{z}{f-z}f)$. Predpis ni linearen, lahko pa ga lineariziramo z uvedbo homogenih koordinat: P_{33} predstavimo kot preslikavo $P: R^4 \to R^4$.

Vektor $\mathbf{a} = (x,y,z)$ v kartezičnih koordinatah izrazimo v homogenih koordinatah kot $\mathbf{a}_h = (wx, xy, xy, w)$, kjer je w poljubna konstanta. Vektorja (wx, wy, wz, w) in $(x,y,z,1)$ sta ekvivalentna za $w \neq 0$, oziroma predstavljata isti vektor v R^3. V tem primeru zapišemo: $(wx, wy, wz, w) \cong (x,y,z,1)$. Projekcija P torej je

$$\begin{pmatrix} x \\ y \\ z \\ w_1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/f & 1 \end{pmatrix} \begin{pmatrix} wx \\ wy \\ wz \\ w \end{pmatrix}.$$

Če vzamemo za $w = 1$, dobimo vektor $(x,y,z,(f-z)/f)$, kar se ujema s projektorjem P_{32}.

Naj bodo $R^x_{33}, R^y_{33}, R^z_{33}$ rotacije za kot φ okrog osi x, za kot ψ okrog osi y oziroma za kot ϑ okrog osi z. Tako kot to že nakazuje spodnji indeks, gledamo nanje kot na preslikave v 3-D prostoru. Zapišimo z matriko R^x_{33}:

$$R^x_{33} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi \\ 0 & \sin\varphi & \cos\varphi \end{pmatrix}.$$

Podobno zapišemo matriki za R^y_{33} in R^z_{33} (glej [11]). Rotacije so v homogenih koordinatah predstavljene takole:

$$R_x = \begin{pmatrix} R^x_{33} & 0 \\ 0 & 1 \end{pmatrix}, \quad R_y = \begin{pmatrix} R^y_{33} & 0 \\ 0 & 1 \end{pmatrix}, \quad R_z = \begin{pmatrix} R^z_{33} & 0 \\ 0 & 1 \end{pmatrix}$$

in celotna rotacija je $R = R_x.R_y.R_z$.

Translacijo za vektor (x_t, y_t, z_t) predstavimo v homogenih koordinatah kot

$$T = \begin{pmatrix} 1 & & & x_t \\ & 1 & & y_t \\ & & 1 & z_t \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Naj bo $Q := TR$ operator, s katerim je predstavljen poljuben premik telesa v prostoru. Preslikava Q postavi model na sceno tako, da se le-ta projicira s projektorjem P na dano sliko.

Če zapišemo komponente produkta $H = PTR$, vidimo, da lahko tretjo vrstico matrike H izračunamo iz njene četrte vrstice.

Naj bo $\tilde{H}$ matrika H brez tretje vrstice. Če je $H(x,y,z,w) = (x',y',z',w')$, je $\tilde{H}(x,y,z,w) = (x',y',w')$.

Zdaj lahko definiramo problem: imejmo n točk $\mathbf{T}_i(x_i,y_i,z_i,1)$ modela in prav toliko točk $\mathbf{T}'_i(x'_i,y'_i,1)$ na sliki. Iščemo tako rotacijo in translacijo, ki bosta postavili model na sceno tako, da se bo sprojiciral v dano sliko. Iščemo torej tako preslikavo $\tilde{H}$, da bo

$$H(\mathbf{T}_i) \cong \mathbf{T}'_i, \qquad i = 1,\ldots,n \tag{2.1}$$

Iz $\tilde{H}$ izračunamo H in od tod $Q = P^{-1}H$, kar je rešitev našega problema.

Sistem (2.1) rešimo takole:

$$\tilde{H}(\mathbf{T}_i) = w_i(\mathbf{T}'_i), \qquad i = 1,\ldots,n$$
$$\tilde{H}[\mathbf{T}_1,\ldots,\mathbf{T}_n] = [\mathbf{T}'_1,\ldots,\mathbf{T}'_n]\,\mathrm{diag}(w_1,\ldots,w_n),$$

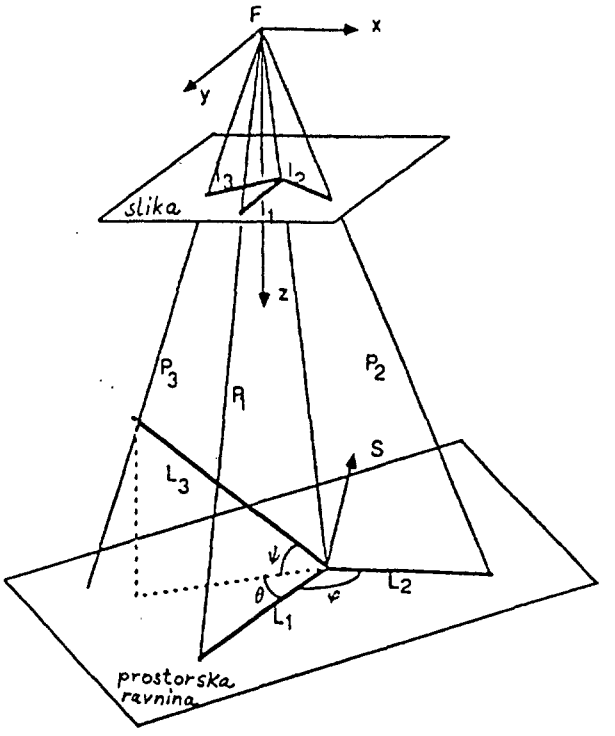
kjer so elementi matrike $\tilde{H}$ in $w_1,..w_n$ neznanke.

Predpostavimo, da so elementi matrike $\tilde{H}$ med sabo neodvisni in poiščemo rešitev z metodo najmanjših kvadratov [1]. Tako dobljena rešitev ne ustreza nujno pogojem, kakršnim morajo ustrezati elementi matrike H.

Koliko točk $\mathbf{T}_i$ potrebujemo za rešitev sistema? Imamo $12+n$ neznank in $3n$ enačb, torej potrebujemo vsaj 6 različnih točk za enolično rešitev.

3. ANALITIČNA METODA (Horaud)

Gorišče postavimo v izhodišče koordinatnega sistema:



Zapišimo projekcijo $P(x,y,z) = (xf/z, yf/z, f)$.

Predstavimo vektorje s sferičnimi koordinatami:

$$x = \cos\alpha\cos\beta, \quad y = \sin\alpha\cos\beta, \quad z = \sin\beta,$$

kjer za parametra α in β velja $0 \leq \alpha \leq 2\pi$, $-\pi/2 \leq \beta \leq \pi/2$.

Imejmo normirane vektorje $\mathbf{L}_1, \mathbf{L}_2, \mathbf{L}_3$ s skupno začetno točko na modelu in naj bodo koti med njimi φ, ϑ, ψ, tako kot je označeno na sliki.

Naj bo $\mathbf{S}$ normala na ravnino vektorjev $\mathbf{L}_1, \mathbf{L}_2$.

Naj bodo $\mathbf{l}_1, \mathbf{l}_2, \mathbf{l}_3$ slike vektorjev $\mathbf{L}_1, \mathbf{L}_2, \mathbf{L}_3$. Položimo skozi gorišče in vektorje $\mathbf{l}_1, \mathbf{l}_2, \mathbf{l}_3$ ravnine (*interpretacijske ravnine*) z normalami $\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3$. Vektorji $\mathbf{L}_1, \mathbf{L}_2, \mathbf{L}_3$ leže na teh interpretacijskih ravninah z normalami $\mathbf{P}_1, \mathbf{P}_2$ oziroma $\mathbf{P}_3$. Velja:

$$\mathbf{L}_1 = \mathbf{S} \times \mathbf{P}_1, \qquad \mathbf{L}_2 = \mathbf{S} \times \mathbf{P}_2. \tag{3.1}$$

Po krajši izpeljavi [5] dobimo

$$(\mathbf{S} \times \mathbf{P}_1)(\mathbf{S} \times \mathbf{P}_2) = \|\mathbf{S} \times \mathbf{P}_1\| \, \|\mathbf{S} \times \mathbf{P}_2\| \cos\varphi \tag{3.2}$$

$$\mathbf{L}_3 = \mathbf{S} \sin\psi + \mathbf{L}_1 \cos\vartheta \cos\psi + (\mathbf{S} \times \mathbf{L}_1) \sin\vartheta \cos\psi \tag{3.3}$$

in

$$\begin{aligned}(\mathbf{S}.\mathbf{P}_3)\|\mathbf{S} \times \mathbf{P}_1\| \sin\psi + \mathbf{S}.(\mathbf{P}_1 \times \mathbf{P}_3) \cos\vartheta \cos\psi \\ + (\mathbf{S}.\mathbf{P}_1)(\mathbf{S}.\mathbf{P}_3) \sin\vartheta \sin\psi = (\mathbf{P}_1.\mathbf{P}_3) \sin\vartheta \sin\psi\end{aligned} \tag{3.4}$$

Predpostavimo, da poznamo kote φ, ψ, ϑ. Z rešitvijo enačb (3.2) in (3.4) dobimo normalo $\mathbf{S}$ in od tod s pomočjo (3.1) in (3.3) vektorje $\mathbf{L}_1, \mathbf{L}_2$ in $\mathbf{L}_3$. Še vedno pa ne vemo, v kateri točki prostora se ti vektorji stikajo.

Sistem enačb (3.2), (3.4) je nelinearen in transcendenten, zato ga avtor ne rešuje analitično, ampak rešitev konstruira. Enačbi (3.2) in (3.3) zapiše kot:

$$\cos\varphi = f(\alpha, \beta) \tag{3.5}$$

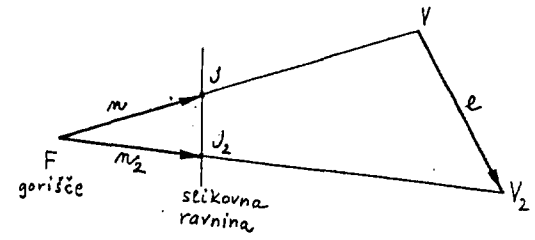
$$\begin{aligned}\sin\vartheta \cos\psi = g_1(\alpha, \beta) \sin\psi + g_2(\alpha, \beta) \cos\vartheta \cos\psi \\ + g_3(\alpha, \beta) \sin\vartheta \cos\psi.\end{aligned} \tag{3.6}$$

Tabeliramo funkcije f, g_1, g_2, g_3, za α, β iz intervalov $0 \le \alpha \le 2\pi$, $0 \le \beta \le \pi/2$ z določenim korakom in točko za točko kostruiramo krivulji, ki ju predstavljata enačbi (3.5) in (3.6). Presečišča krivulj so možne rešitve za smer vektorja $\mathbf{S}$. Pri tem je treba pripomniti, da tabeliramo funkcije f, g_1, g_2, g_3 le enkrat in rezultate uporabimo pri različnih predpostavkah za kote φ, ψ, ϑ.

Določimo sedaj točen položaj trojice $\mathbf{L}_1, \mathbf{L}_2, \mathbf{L}_3$ s skupno točko $\mathbf{V}$ v prostoru! Projekcija točke $\mathbf{V}$ na slikovno ravnino je točka $P(\mathbf{V}) = \mathbf{J}$ s koordinatami $\mathbf{J}(J_x, J_y, f, 0)$. Vektor od gorišča $\mathbf{F}$ do točke $\mathbf{J}$ je $\mathbf{n} = (J_x, J_y, f, 0)$. Vzemimo, da poznamo razdaljo d od gorišča $\mathbf{F}$ do točke $\mathbf{V}$. Tedaj je translacijski vektor

$$\mathbf{t} = d\mathbf{n}/\|\mathbf{n}\|. \tag{3.7}$$

Tako smo določili vektor translacije do skalarja natančno. Če poznamo še eno trojico vektorjev s skupno točko $\mathbf{V}_2$, potem določimo d takole:

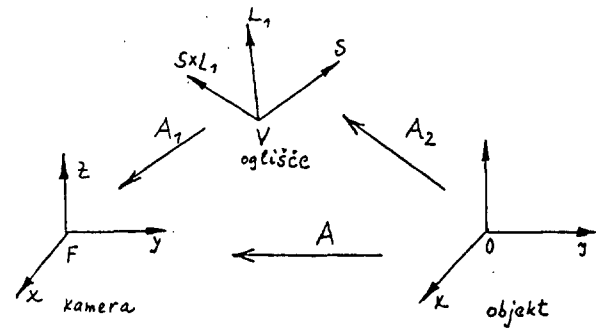


Naj bo $\mathbf{e}$ vektor od $\mathbf{V}$ do $\mathbf{V}_2$. (Vektor $\mathbf{e}$ poznamo, saj primerjamo model telesa s sliko in tako $\mathbf{V}$, $\mathbf{V}_2$ ustrezata ogliščem znanega modela.) Naj bo $\mathbf{J}_2$ projekcija $\mathbf{V}_2$ na slikovno ravnino in $\mathbf{n}_2$ vektor od $\mathbf{F}$ do $\mathbf{J}_2$. Zaradi (3.7) dobimo

$$d = \frac{\|\mathbf{n}\|.\|\mathbf{n}_2 \times \mathbf{e}\|}{\|\mathbf{n} \times \mathbf{n}_2\|}.$$

Tako smo za določitev vseh šestih prostostnih stopenj morali poznati dve trojici vektorjev oziroma 8 točk.

Zapišimo zdaj še premik A (translacija, rotacija) modela na sceno, tako da se izbrano oglišče modela s pripadajočimi vektorji projicira na izbrano trojico vektorjev!



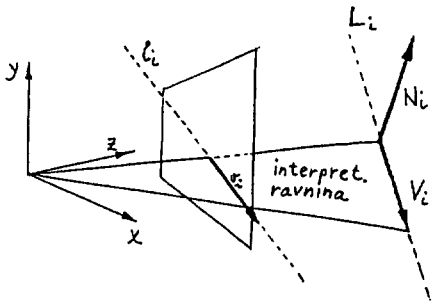
$A = A_1 A_2$, kjer je A_2 prehodna matrika med različnima bazama vektorskih prostorov (bazo, v kateri je opisan model in bazo, sestavljeno iz $\mathbf{S}$, $\mathbf{L}_1$, $\mathbf{S} \times \mathbf{L}_1$) in

$$A_1 = \begin{pmatrix} p_x & q_x & r_x & t_x \\ p_y & q_y & r_y & t_y \\ p_z & q_z & r_z & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

kjer je $\mathbf{S} = (p_x, p_y, p_z), \mathbf{L}_1 = (q_x, q_y, q_z), \mathbf{S} \times \mathbf{L}_1 = (r_x, r_y, r_z), \mathbf{t} = (t_x, t_y, t_z)$; vse izraženo v koordinatnem sistemu kamere.

4. ŠE ENA ANALITIČNA REŠITEV (Dhome in soavtorji)

Koordinatni sistem kamere, projektor P in pojem interpretacijske ravnine poznamo že iz Horaudove metode.



Če je $\mathbf{L}_i$ premica v prostoru in $\mathbf{l}_i$ njena projekcija, $\mathbf{V}_i$ in $\mathbf{v}_i$ pa enotska vektorja na $\mathbf{L}_i$ oziroma $\mathbf{l}_i$, $\mathbf{p}_i$ pa točka na $\mathbf{l}_i$, je normala $\mathbf{N}_i$ na interpretacijsko ravnino določena z

$$\mathbf{N}_i = \mathbf{l}_i \times \mathbf{p}_i.$$

Za vektor $\mathbf{V}_i$ mora veljati

$$\mathbf{N}_i.\mathbf{V}_i = 0. \tag{4.1}$$

Formulirajmo problem: imejmo tri premice $\mathbf{L}_{01}, \mathbf{L}_{02}, \mathbf{L}_{03}$, ki leže na robovih modela (model je predstavljen v svojem lastnem koordinatnem sistemu (S_{0m})), imejmo premice $\mathbf{l}_{01}, \mathbf{l}_{02}, \mathbf{l}_{03}$, definirane v slikovnem koordinatnem sistemu (S_{0v}). Želimo najti rotacijo $R_{\alpha\beta\gamma}$ in translacijo T_{uvw}, ki ju uporabimo na modelu, tako da se

$$T_{uvw} R_{\alpha\beta\gamma}(\mathbf{L}_{0i}) = \mathbf{L}_{3i} \quad \text{projicirajo na} \quad \mathbf{l}_{0i}, \qquad i = 1,2,3$$

Problem rešimo v dveh korakih: najprej poiščemo rotacijo in nato še translacijo telesa. Rotacija je sestavljena iz zavrtitev za kote α okrog osi x, β okrog osi y in γ okrog osi z; predstavljena je na običajen način (glej [11]). Če označimo z $\mathbf{V}_{0i}, \mathbf{V}_{3i}$ smerne vektorje premic $\mathbf{L}_{0i}, \mathbf{L}_{3i}$, mora veljati:

$$\mathbf{V}_{3i} = R_{\alpha\beta\gamma}\mathbf{V}_{0i}, \qquad i = 1,2,3.$$

Zaradi (4.1) rešujemo sistem enačb

$$\mathbf{N}_{3i}.R_{\alpha\beta\gamma}\mathbf{V}_{0i} = 0, \qquad i = 1,2,3.$$

za neznanke α, β, γ. Sistem poenostavimo z uvedbo dveh novih koordinatnih sistemov (S_{1m}) in (S_{1v}). Tako bomo imeli 4 koordinatne sisteme

$$S_{0m} \xrightarrow{R_m} S_{1m} \xrightarrow{R_{\alpha\beta}} S_{1v} \xrightarrow{R_v} S_{0v} \quad \text{in} \quad R_{\alpha\beta\gamma} = R_v \circ R_{\alpha\beta} \circ R_m$$

V koordinatnem sistemu (S_{0m}) imamo normirane vektorje $\mathbf{v}_{01}$, $\mathbf{v}_{02}$, $\mathbf{v}_{03}$ (smerni vektorji na premicah $\mathbf{l}_{01}$, $\mathbf{l}_{02}, \mathbf{l}_{03}$). Novi koordinatni sistem (S_{1m}) ima tako bazo, da je prvi bazni vektor $\mathbf{e}_{11} := \mathbf{v}_{01}$, drugi bazni vektor leži v ravnini $\mathbf{v}_{01}, \mathbf{v}_{02}$, torej $\mathbf{e}_{12} := \mathbf{e}_{11} \times \mathbf{e}_{13}$ in tretji bazni vektor je pravokoten na to ravnino $\mathbf{e}_{13} := \mathbf{v}_{01} \times \mathbf{v}_{02}$. Prehodna matrika je

$$R_m = (\mathbf{v}_{01}, \mathbf{v}_{01} \times (\mathbf{v}_{01} \times \mathbf{v}_{02}), \mathbf{v}_{01} \times \mathbf{v}_{02})^{-1}.$$

Uvedemo še nov koordinatni sistem (S_{1v}), tak da (x,y) ravnina ustreza interpretacijski ravnini in da je os x kolinearna z $\mathbf{e}_{01}$. Če ima smerni vektor $\mathbf{v}_{01}$ premice $\mathbf{l}_{01}$ koordinate $\mathbf{v}_{01} = (a_{01}, b_{01}, 0)$ in točka $\mathbf{p}_{01}$ na $\mathbf{l}_{01}$ koordinate $\mathbf{p}_{01} = (x_{01}, y_{01}, f)$, lahko transformacijsko matriko R_v med (S_{1v}) in (S_{0v}) izrazimo

$$R_v = \frac{1}{\sqrt{f^2 + d_{01}^2}} \begin{pmatrix} \sqrt{f^2 + d_{01}^2} & 0 & 0 \\ 0 & f & d_{01} \\ 0 & -d_{01} & f \end{pmatrix} \begin{pmatrix} a_{01} & -b_{01} & 0 \\ b_{01} & a_{01} & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

kjer je $d_{01} = a_{01} y_{01} - b_{01} x_{01}$.

Z uvajanjem novih koordinatnih sistemov smo dosegli, da iščemo rotacijo kot preslikavo med (S_{1m}) in (S_{1v}) v obliki $R_{\alpha\beta}$ (zavrtitev za kot α okrog osi x in zavrtitev za kot β okrog osi y).

Veljati mora $\mathbf{V}_{21} = R_{\alpha\beta}\mathbf{V}_{11}$. Zaradi (4.1) rešujemo sistem

$$\mathbf{N}_{2i}\mathbf{V}_{2i} = \mathbf{N}_{2i}R_{\alpha\beta}\mathbf{V}_{1i}, \quad i = 1,2,3, \quad \text{za neznanke } \alpha, \beta.$$

Od teh treh enačb sta uporabni le dve, ki ju v [6] prevedejo v obliko

$$\sum_{i=0}^{8} \delta_i t^i = 0, \tag{4.2}$$

kjer je $t = \tan \alpha/2$, δ_i pa so znani koeficienti.

Ko rešimo enačbo (4.2), poiščemo še β, ki je izražen v obliki $\cos\beta = f(\alpha)$. Celotna rotacija je $R_{\alpha\beta\gamma} = R_v R_{\alpha\beta} R_m$.

Če imamo opravka s koplanarnimi premicami ali s premicami, ki imajo eno točko skupno, je polinom (4.2) le četrte stopnje.

Izračunajmo zdaj še translacijo T_{uvw}! Naj bo $\mathbf{t} := (u,v,w)$ vektor translacije. Vzemimo, da za vsako premico $\mathbf{L}_{0i}$ poznamo tudi točko $\mathbf{P}_{0i}$ na njej. Točke $\mathbf{P}_{0i}$ se morajo po rotaciji in translaciji projicirati na premice $\mathbf{l}_{0i}$, torej morajo zadoščati pogoju (4.1):

$$(R(\mathbf{P}_{0i}) + \mathbf{t}).\mathbf{N}_{3i} = 0, \quad i = 1,2,3 \tag{4.3}$$

Tako imamo linearen sistem treh enačb za neznanke u,v,w, ki ga brez težav rešimo. V posebnem primeru, ko se premice $\mathbf{L}_{0i}$ stikajo v skupni točki $\mathbf{P}_{00}$, imamo v zgornjem sistemu (4.3) le dve linearno neodvisni enačbi, zato si tretjo enačbo dobimo takole: imejmo točke $\mathbf{T}_{0i}$ (drugo oglišče roba, ki leži na $\mathbf{L}_{0i}$) in njihove slike $\mathbf{t}_{3i}$. Veljati mora

$$P(R(\mathbf{T}_{0i}) + \mathbf{t}) = \mathbf{t}_{3i}, \quad i = 1,2,3$$

Če je torej v celoti viden vsaj eden od robov, lahko določimo translacijo. Če je v celoti vidnih več robov, izberemo tisto translacijo, ki ima najmanjšo vrednost vzdolž osi z. V praksi je težko določiti, kdaj je rob objekta v celoti viden, zato se v takih primerih

na izračunano translacijo ne smemo preveč zanašati.

Po tej metodi dobimo za translacijo in rotacijo več rešitev. Vse te rešitve moramo preveriti, ali so prave, še prej pa jih lahko veliko večino izločimo s temi preprostimi pravili:

- translacija vzdolž osi z ne sme biti negativna
- izbrani robovi morajo biti na sliki vidni [9]
- rešitev pri kateri slika transformiranega roba ne leži na sliki roba, ne pride v poštev.

5. LOWEJEVA METODA

Pri tem pristopu so linearizirane enačbe projekcije in uporabljena je Newtonova metoda za potrebno število iteracij. Še prej pa so enačbe reparametrizirane, tako da je poenostavljeno računanje potrebnih odvodov.

Zapišimo preslikave

$$(x,y,z) = R(\mathrm{p}-\mathrm{t}), \quad (u,v) = (\frac{fx}{z},\frac{fy}{z}) = P(x,y,z), \qquad (5.1)$$

kjer je t translacijski vektor v R^3, R rotacija, ki zavrti model iz osnovnega položaja v položaj, ki ga ima telo na sceni, f goriščna razdalja in (u,v) koordinati točke na sliki, kamor se projicira točka p iz modela.

Pri danih točkah p_i, $i = 1,\ldots,n$ modela in njihovih slikah (u_i,v_i) na projekcijski ravnini iščemo tak premik t in rotacijo R, da bo

$$PR(\mathrm{p}_i - \mathrm{t}) = (u_i,v_i), \quad i = 1,\ldots,n \qquad (5.2)$$

Da bi laže računali parcialne odvode, reparametriziramo enačbi (5.1) takole:

$$\begin{aligned}(x,y,z) &= R.p\\ (u,v) = \tilde{P}(x,y,z) &= \left(\frac{fx}{z+Dz}+Dx, \frac{fy}{z+Dz}+Dy\right)\end{aligned} \qquad (5.3)$$

Spremenljivki R, f sta isti kot v enačbah (5.1), vektor t pa smo nadomestili z vektorjem (D_x,D_y,D_z). Transformaciji (5.1),(5.3) sta ekvivalentni, če je

$$\mathrm{t} = R^{-1}\left[-\frac{D_x(z+D_z)}{f}, -\frac{D_y(z+D_z)}{f}, -D_z\right]^T.$$

D_x in D_y določata položaj telesa na sliki, D_z pa oddaljenost objekta od kamere.

Kako predstaviti rotacijo R s parametri $\varphi_x,\varphi_y,\varphi_z$ (zavrtitev okrog x,y,z osi)? Izhajamo iz začetnega približka R_0, ob spremembi kotov $\varphi_x,\varphi_y,\varphi_z$ za $\Delta\varphi_x,\Delta\varphi_y,\Delta\varphi_z$ pa dobimo novo rotacijo R_1 takole:

$$R_1 = \begin{bmatrix}\cos\Delta\varphi_z & -\sin\Delta\varphi_z & 0\\ \sin\Delta\varphi_z & \cos\Delta\varphi_z & 0\\ 0 & 0 & 1\end{bmatrix}\cdot\begin{bmatrix}\cos\Delta\varphi_y & 0 & \sin\Delta\varphi_y\\ 0 & 1 & 0\\ -\sin\Delta\varphi_y & 0 & \cos\Delta\varphi_y\end{bmatrix}$$
$$\cdot\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\Delta\varphi_x & -\sin\Delta\varphi_x\\ 0 & \sin\Delta\varphi_x & \cos\Delta\varphi_x\end{bmatrix}R_0$$

Če so spremembe kotov $\Delta\varphi_x,\Delta\varphi_y,\Delta\varphi_z$ dovolj majhne, so približno neodvisne druga od druge, tako da lahko upoštevamo zveze: $(x,y,z) = (r_x\cos\varphi_x, r\sin\varphi_x, z) = (x, r_z\cos\varphi_z, r_z\sin\varphi_z) = (r_y\sin\varphi_y, y, r_y\cos\varphi_y)$, kjer so r_x, r_y, r_z razdalje točke od osi x, y, z. Od tod izračunamo parcialne odvode x,y,z glede na $\varphi_x,\varphi_y,\varphi_z$:

	x	y	z
φ_x	0	$-z$	y
φ_y	z	0	$-x$
φ_z	$-y$	x	0

in nato z zaporednim odvajanjem še parcialne odvode u in v glede na iskane parametre $D_x,D_y,D_z,\varphi_x,\varphi_y,\varphi_z$

	u	v
D_x	1	0
D_y	0	1
D_z	$-fc^2x$	$-fc^2y$
φ_x	$-fc^2xy$	$-fc(z+cy^2)$
φ_y	$fc(z+cx^2)$	fc^2xy
φ_z	$-fcy$	fcx
h	cx	cy

kjer je $c := (z+D_z)^{-1}$. Naj bo

$$\mathrm{h} := [\Delta D_x,\Delta D_y,\Delta D_z,\Delta\varphi_x,\Delta\varphi_y,\Delta\varphi_z].$$

Če je neznana tudi goriščna razdalja, dodamo vektorju h še Δf. Zdaj lahko direktno uporabimo Newtonovo metodo: imejmo začetne približke D_{x_0}, D_{y_0}, D_{z_0}, φ_{x_0}, φ_{y_0}, φ_{z_0} za količine D_x, D_y, D_z, φ_x, φ_y, φ_z. Od tod izračunamo začetni približek $\tilde{P}_0, R_0$ za $\tilde{P}$ oziroma R. Velja:

$$\tilde{P}_0R_0(p_i) = (u_{i0},v_{i0}) = (E_{u0}+u_i, E_{v0}+v_i), \quad i = 1,\ldots,n$$

kjer sta E_{u0} in E_{v0} napaki, ki ju naredi začetna preslikava $\tilde{P}_0R_0$. Za napako E_{u0} vzamem

$$\begin{aligned}E_{u_0} =& \frac{\partial u_i}{\partial D_x}\Delta D_x + \frac{\partial u_i}{\partial D_y}\Delta D_y + \frac{\partial u_i}{\partial D_z}\Delta D_z\\ &+ \frac{\partial u_i}{\partial\varphi_x}\Delta\varphi_x + \frac{\partial u_i}{\partial\varphi_y}\Delta\varphi_y + \frac{\partial u_i}{\partial\varphi_z}\Delta\varphi_z, \quad i = 1,\ldots,n\end{aligned} \qquad (5.4)$$

Podoben izraz velja za E_{v0}. Če imamo korespondenco med temi točkami ($n = 3$), imamo linearen sistem 6 enačb, ki jih rešimo za neznanke ΔD_x, ΔD_y, ΔD_z, $\Delta\varphi_x$, $\Delta\varphi_y$, $\Delta\varphi_z$. Določimo novo preslikavo $\tilde{P}_1R_1$ in ponovimo iteracijo. Po [8] je potrebnih le nekaj iteracij, da pridemo do dovolj natančnega rezultata.

Če imamo imamo več povezav med točkami ($n > 3$), uporabimo za reševanje sistema metodo najmanjših kvadratov. Zapišimo sistem (5.4) bolj kompaktno:

$$J\mathbf{h} = \mathbf{e},$$

kjer je J Jacobijeva matrika, ki vsebuje parcialne odvode, h vektor, ki vsebuje neznane funkcije in e vektor napak, ki smo jih izmerili na sliki. Rešimo normalni sistem

$$J^T J\mathbf{h} = J^T\mathbf{e}.$$

Tako dobimo dokaj natančno rešitev, če imamo dober začetni približek. Poleg tega ta metoda konvergira, če imamo znanih veliko ujemanj med robovi in njihovimi slikami, kar zahteva visoko kompleksnost pri iskanju pravega ujemanja med robovi modela in njihovimi slikami. Zato je dobro po prejšnji metodi (Dhome) dobiti vse možne transformacije z interpretacijo treh robov. Na ta način opustimo hipotezo (da se slike robov ujemajo z izbranimi črtami na sliki) ali pa dobimo zanjo dober začetni približek.

6. ZAKLJUČEK

Ogledali smo si štiri metode za določanje položaja telesa iz ene same 2-D slike. Različni avtorji so vsak na svoj način poenostavljali reševanje nelinearnih transcendentnih enačb.

Roberts operira s samimi obrnljivimi operatorji zaradi uvedbe homogenih koordinat in rešuje linearen problem, ker predpostavi, da so elementi transformacijske matrike med sabo neodvisni. Ko dobi rešitev, mora preveriti medsebojno odvisnost matričnih elementov. Za rešitev potrebuje najmanj šest točk modela in njihovih slik.

Horaud rešuje sistem dveh transcendentnih enačb z dvema neznankama s tabeliranjem funkcij dveh spremenljivk, konstruiranjem nivojnic teh funkcij in iskanjem njihovih presečišč. To je precej zamuden postopek, ki nam da rezultate z omejeno natančnostjo. Za določitev petih prostostnih stopenj potrebuje oglišče modela, iz katerega izhajajo trije robovi in sliko tega oglišča in robov, za določitev šeste prostostne stopnje pri translaciji pa potrebuje še eno tako oglišče.

Dhome in soavtorji morajo poiskati ničle polinoma 8 stopnje, kar tudi ni trivialno. Do polinoma pridejo po uvedbi takih koordinatnih sistemov, da iščejo rotacijo kot funkcijo dveh in ne treh kotov. Na ta način dobimo vse možne rešitve, ki nam lahko služijo za začetni približek pri Lowejevi metodi, ki nam ob dobrem začetnem približku da natančnejši rezultat. Za določitev rotacije potrebujemo tri smerne vektorje iz robov modela ter smerne vektorje iz slik robov ter po eno točko na vsaki sliki roba. Če se obravnavani robovi modela ne stikajo v skupnem oglišču, moramo za določitev translacije za vsak izbrani rob poznati eno njegovo oglišče, če pa se obravnavani robovi modela stikajo v skupni točki, moramo vsaj za en rob poznati še drugo oglišče in njegovo sliko na projekcijski ravnini.

Lowe rešuje problem iterativno z Newtonovo metodo; računanje parcialnih odvodov močno poenostavi z uvedbo novih spremenljivk. Za rešitev potrebuje korespondenco med vsaj tremi točkami modela in slike ter dovolj dober približek za začetno rešitev. Natančnost dobljene transformacije lahko povečamo, če poznamo več točk modela in njihovih slik.

Robertsovo metodo smo si ogledali, ker je ena prvih; naslednji dve metoda nam dasta vse rešitve, vendar z omejeno natančnostjo, medtem ko z Lowejevo metodo dobljeno rešitev lahko izboljšamo.

Zanimivo bi bilo vedeti, kako se spremeni rešitev problema, če malo premaknemo slike točk. Tako bi vedeli, katera metoda se najbolje obnese pri realnih slikah, ki imajo tudi šume. Zanimive bi bile tudi ocene, kako natančne so dobljene transformacije in kako so opisane metode hitre.

LITERATURA

[1] L. G. Roberts: Machine perception of three-dimensional solids, *Optical and Electro Optical Information Processing*, J.T. Tipplett; Ed. Cambridge, MA: MIT Press, 1965, pp. 159-197

[2] T. Kanade: Recovery of the three dimensional shape of an object from a single view, *Artificial Intell.*, Special Volume on Computer Vision, vol. 17, no. 1-3, Aug. 1981

[3] S. T. Barnard: Choosing a basis for perceptual space, *Comput. Vision Graph. Image Processing*, vol. 29, no. 1, pp. 87-99, 1985

[4] T. Shakunagar, H. Kaneko: Perspective angle transform and its application to 3-D configuration recovery, *Proc. Int. Comput. Vision Pattern Recogn.*, Miami Beach, FL, June 1986, pp. 594-601

[5] R. Horaud: New methods for matching 3-D objects with single perspective views, *IEEE Trans. on Pattern Analysis and Machine Intell.*, vol. PAMI-9, No. 3, May 1987

[6] M. Dhome *et al*: Determination of the attitude of 3-D objects from a single perspective view, *IEEE Trans. on Pattern Analysis and Machine Intell.*, vol. 11, No. 12, Dec. 1989

[7] D. P. Huttenlocher, S. Ullman: Object recognition using alignment, in *Proc. First Int. Conf. Comput. Vision*, London, England, June 1987, pp. 102-111.

[8] D. G. Lowe: Three-dimensional object recognition from single two-dimensional images, *Artificial Intell.*, 31(1987), pp. 355-395

[9] D. G. Lowe: Perceptual organisation and visual recognition, Boston, MA: Kluwer, 1985, ch. 7

[10] Zen Chen *et al*: A simple vision algorithm for 3-D position determination using a single calibration object, *Pattern Recognition*, vol. 22, no. 2, 1989

[11] Yoshiaki Shirai: Three-dimensional computer vision, Springer Verlag 1987, ch. 5

IZVEDBA IN UPORABA POSEBNE FUNKCIONALNO ZASNOVANE RAČUNALNIŠKE TIPKOVNICE ZA MREŽO OSEBNIH RAČUNALNIKOV

INFORMATICA 1/91

Keywords: special functional computer keyboard, distributed computer network.

Mario Baar
Institut »Jožef Stefan«
Ljubljana

V članku je opisan primer načrtovanja, zgradbe in realizacije posebne funkcionalno zasnovane računalniške tipkovnice (v nadaljnem tekstu računalniški komandni pult), ki je hkrati povezana na več posebno prirejenih osebnih računalnikov, ki so del distribuiranega računalniškega sistema za nadzor in vodenje kompleksnega industrijskega procesa. Opisana je rešitev tehničnega problema priključitve računalniškega komandnega pulta na osebni računalnik, pri čemer je omogočena uporaba obstoječe programske podpore klasične računalniške tipkovnice osebnega računalnika, dopuščena je možnost simuliranja računalniškega komandnega pulta s klasično tipkovnico, poleg tega pa ni potrebna uporaba dodatnih komunikacijskih vmesnikov. Podan je namen uporabe računalniškega komandnega pulta v industriji in pojasnjena njegova povezava na distribuiran računalniški sistem preko posebnih elektronskih stikal na (običajne) tipkovnične vhode osebnih računalnikov. Pojasnjen je osnovni namen centralizacije vseh funkcij industrijskega procesa na enem (centralnem) mestu, kako je dosežena kar največja preglednost nad stanjem v industrijskem procesu in na kakšen način je lahko poenostavljen nadzor in vodenje takega procesa.

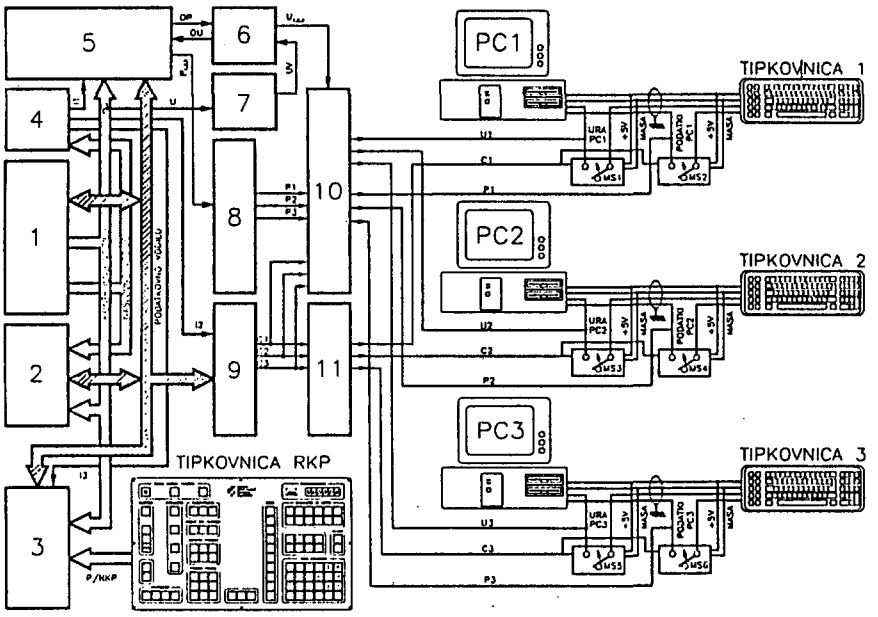
DESIGN AND IMPLEMENTION OF SPECIAL FUNCTIONAL DESIGNED COMPUTER KEYBOARD USED BY COMPUTER NETWORK. This paper describes an example how to design, construct and realize special functional designed computer keyboard (in further text computer command keyboard), which is simultaniously connected to several specially prepared personal computers, as a part of distributed computer system for supervising and control of a complex industrial process. The solution of technical problem dealing with connection the computer command keyboard to personal computer is described. It is also given the possibility to use the existing software support of personal computer for classical computer keyboard, and beside that, it is no need to use any optional communication interface. It is given the purpose of using the computer command keyboard in the industry enviroment, it is explained its connection to distributed computer system via electronic switches to (usual) computer keyboard inputs of distributed computer system, howewer, the main purpose of computer command keyboard is to centralize and control all supervising functions in one (central) place and to gain maximum overlook and simplification of control of such a process.

1. UVOD

Članek podaja pristop k razvoju, izvedbi in uporabi posebne funkcionalno zasnovane računalniške tipkovnice (v nadaljnem tekstu računalniški komandni pult), ki omogoča komunikacijo med operaterjem in distribuiranim računalniškim sistemom. Distribuiran računalniški sistem je sestavljen iz večjega števila posebno prirejenih osebnih računalnikov tipa IBM PC/XT, PC/AT ali kompatibilnih računalnikov (ti tvorijo tako imenovani centralni računalniški sistem) in drugih (procesnih) računalnikov (ti tvorijo tako imenovani podrejeni računalniški sistem), pri čemer so vsi povezani v skupno računalniško mrežo. Centralni računalniški sistem torej dobiva podatke o trenutnem stanju v industrijskem procesu preko računalniške mreže (oziroma preko podrejenega računalniškega sistema) in na osnovi teh informacij omogoča operaterju vodenje in nadzor takega procesa. Ker so posamezne funkcije nadzora in vodenja procesa porazdeljene med več računalnikov centralnega računalniškega sistema, je vodenje le-tega brez računalniškega komandnega pulta praktično neizvedljivo. V tem članku je opisana komunikacija računalniškega komandnega pulta s centralnim računalniškim sistemom preko običajnih tipkovničnih vhodov posebno prirejenih osebnih računalnikov. Preko računalniškega komandnega pulta je možno industrijski proces enostavno nadzorovati in voditi, pri čemer je le-ta svojo uporabnost pokazal tudi že na realnem objektu. /1/

2.1 DISTRIBUIRANI RAČUNALNIŠKI SISTEMI

Pri distribuiranih računalniških sistemih nastopi problem, kako nadzorovati in voditi določen kompleksen industrijski proces, če so posamezne funkcije vodenja porazdeljene med več računalnikov. To pomeni, da bi moral operater, ki vodi industrijski proces ter vnaša podatke v računalniški sistem, točno vedeti, na kateri računalnik centralnega računalniškega sistema naj pošlje določen ukaz ali zahtevo, pri čemer pa bi moral ukaz odtipkati na točno določeno računalniško tipkovnico. Ker bi moral vnašati podatke preko običajnih računalniških tipkovnic, bi moral poznati tudi kodo posamezne tipke, ki izvaja neko določeno funkcijo vodenja. Tak način vodenja procesa bi bil izredno nepregleden, saj bi prihajalo do pogostih napak, ki bi lahko povzročile nenazadnje tudi gospodarsko škodo. Računalniški komandni pult omogoči centralizacijo vseh funkcij vodenja in nadzora, poenostavi vnos podatkov, omogoči veliko preglednost ter signalizira vse



Slika 1: Komunikacija računalniškega komandnega pulta s tremi osebnimi računalniki

morebitne napake, ki bi se lahko pojavile pri vnosu podatkov.

2.2 TEMELJNI PROBLEM KOMUNIKACIJE

Najtežji problem, ki ga je bilo potrebno rešiti, je realizacija komunikacije računalniškega komandnega pulta s centralnim računalniškim sistemom preko običajnih tipkovničnih vhodov osebnih računalnikov, pri čemer pa morajo ostati klasične računalniške tipkovnice ves čas obratovanja sistema priključene. Njihova priključitev je potrebna zaradi možnosti dograjevanja in popravljanja računalniške programske opreme, servisiranja in drugih potreb ter zaradi enosmerne komunikacije računalniškega komandega pulta s centralnim računalniškim sistemom. Pri rešitvi komunikacije preko tipkovničnih vhodov je namreč s stališča osebnega računalnika vseeno, ali dobi podatek od svoje (klasične) računalniške tipkovnice ali preko računalniškega komandnega pulta. Ta komunikacija je izvedena s pomočjo posebnega vezja in elektronskih stikal, ki se preklopijo v trenutku, ko računalniški komandni pult pošlje sekvenco impulzov na določen osebni računalnik. Po prenehanju prenosa, se ta stikala postavijo ponovno v osnovni položaj in omogočijo vpis podatkov preko običajne računalniške tipkovnice.

3. REŠEVANJE TEHNIČNEGA PROBLEMA

3.1 REŠITVE V TUJINI

Pri vodenju industrijskih procesov z distribuiranimi računalniškimi sistemi tudi v tujini pogosto uporabljajo računalniške komandne pulte, ki omogočajo enostavno vodenje industrijskega procesa z enega centralnega mesta. Ti sistemi so razviti za specifične naloge v procesni industriji in se v splošnem precej razlikujejo po svoji zgradbi. Večina distribuiranih sistemov, ki se uporabljajo v celulozni in papirni industriji imajo običajno svoj centralni računalniški del sistema kompakten (sestavljen iz enega zmogljivega računalnika npr.: VAX, PDP ali drugi računalniki), medtem ko so podrejni deli teh računalniških sistemov sestavljeni iz manjšega ali večjega števila raprocesnih čunalniprocesa oziroma jih pošiljajo nazaj v proces /2/, /3/, /4/. Redkeje zasledimo uporabo posebno prirejenih osebnih računalnikov (industrijska izvedba), ki so namenjeni za določene specifične naloge pri nadzoru in vodenju procesov.

V literaturi do sedaj nismo zasledili kakšne podobne rešitve vodenja industrijskega procesa preko tipkovničnih vhodov osebnih računalnikov, ki tvorijo centralni del distribuiranega računalniškega sistema. Ta specifična rešitev je vezana na specifičen distribuirani računalniški sistem, ki je bil razvit v okviru Instituta "Jožef Stefan", Ljubljana in podjetja INEA, Domžale.

3.2 GLAVNI CILJI KOMUNIKACIJE

Glavni cilji komunikacije med računalniškim komandnim pultom in centralnim računalniškim sistemom so bili sledeči:

- komunikacija računalniškega komandnega pulta naj ne moti uporabe posameznih klasičnih računalniških tipkovnic, ki ostanejo priključene ves čas obratovanja sistema;
- distribuiran računalniški sistem naj ne čuti, da je na sistem priključen računalniški komandni pult;
- računalniški komandni pult naj se vključi v sistem le takrat, ko oddaja informacijo enemu od računalnikov centralnega računalniškega sistema;
- programska oprema v računalniškem komandnem pultu naj odloča o tem, ali je vpisana sekvenca tipk pravilna ali ne, oziroma, kateremu računalniku centralnega računalniškega sistema naj bo določen podatek poslan;
- napake pri vpisovanju določenih sekvenc tipk v računalniški komandni pult naj povzročijo zvočni alarm, vrsta napake pa se izpiše na posebnem numeričnem zaslonu, ki je vgrajen v računalniški komandni pult.
- večzložni ukazi (zaporedje pritisnjenih tipk) naj se shranjuje v interni pomnilnik računalniškega ko-

mandnega pulta in se avtomatsko aktivira šele takrat, ko je sekvenca vnosa podatkov zaključena.

S tem, ko priključimo računalniški komandni pult na centralni računalniški sistem, pa niso dostopne vse prvotne funkcije klasičnih računalniških tipkovnic. Računalniški komandni pult je prirejen določenemu industrijskemu procesu in vsebuje le take funkcijske tipke, ki so potrebne za vodenje le-tega.

4. RAZLAGA BLOKOVNE SHEME VEZJA ZA KOMUNIKACIJO

Slika 1 prikazuje elektronsko vezje v obliki blokovne sheme z vrisanimi smermi signalov in s tremi računalniki tipa IBM PC/XT ali PC/AT, (centralni računalniški sistem) ter s pripadajočimi klasičnimi računalniškimi tipkovnicami ter TIPKOVNICO RKP (tipkovnica računalniškega komandnega pulta).

Ob pritisku na tipko TIPKOVNICE RKP se informacija o pritisnjeni tipki pretvori v vezju 3 v ustrezno obliko, ki jo lahko sprejme mikroprocesor 1 preko podatkovnega in kontrolnega vodila. Vezje 3 je lahko aktivirano le, če dobi izbirni signal iz dekoderja 4 (I3), dekoder pa mora prejeti od procesorja 1 ustrezen naslov po naslovnem vodilu. Informacije o pritisnjenih tipkah na TIPKOVNICI RKP se v zaporedju shranjujejo v pomnilniku 2, kamor se prenašajo po podatkovnem, naslovnem in kontrolnem vodilu ter se ob končani sekvenci (do treh zaporednih pritisnjenih tipk) prenesejo preko podatkovnega vodila v komunikacijsko vezje 5. Njegova naloga je, da pretvori paralelni zapis podatka (od D7 - D0) v pripadajočo serijsko obliko (vlak impulzov s start bitom, paritetnim bitom in stop bitom). Vezje 5 pa lahko izvede pretvorbo le, če dobi ustrezne signale iz procesorja 1 po podatkovnem in kontrolnem vodilu, izbirni signal I1 preko dekoderja 4 (dekoder je aktiven ob določenem naslovu na naslovnem vodilu) in impulze oddajne ure (OU) od vezja 6. Podatke od mikroprocesorja 1 lahko vezje 5 sprejme le, če je njegov sprejemni register prazen. Signal ure (U), ki je prosto tekoč vlak impulzov polovične frekvence kristalnega oscilatorja vezanega na mikroprocesor 1, se deli s pomočjo delilnika frekvence 7 v impulze vhodne ure (VU), ta pa določa hitrost prenosa podatkov od računalniškega komandnega pulta proti osebnim računalnikom. (Prenosna hitrost je običajno nastavljena na 9600 Baudov). V vezju 6 se impulzi modificirajo v obliko oddajne ure (OU), ki ima točno število impulzov, ki začno teči ob točno določenem času. Vezje 6 pa lahko deluje le, če mu je komunikacijsko vezje 5 oddalo impulz, da je oddajnik vezja 5 prazen (OP) in da lahko mikroprocesor 1 pošljenasledni blok podatkov po podatkovnem vodilu. Usklajevanje med mikroprocesorjem 1 in komunikacijskim vezjem 5 poteka po kontrolnem vodilu. Vezje 6 oddaja vezju 10 impulze po liniji $U_{1,2,3}$, ki so impulzi ure skupni za vse tri računalnike. Podatke v pretvorjeni obliki pošlje komunikacijski blok 5 preko svojega izhoda $P_{1,2,3}$ na vezje 8. Ta izhod je za vse tri osebne računalnike tipa skupen. Vezje 8 ima nalogo, da razdeli signal podatkov $P_{1,2,3}$ na tri ločene linije t.j. P_1, P_2, P_3, kjer predstavlja P_1 podatkovno linijo za prenos podatkov na računalnik centralnega sistema z oznako 1, P_2 na računalnik z oznako 2 itd. Na vezju 8 so dostopne preveze na plošči tiskanega vezja, kjer se za vsako linijo posebej (P_1, P_2, P_3) lahko nastavi tip posameznega centralnega računalnika. Kontrolno vezje 9 generira kontrolne signale C_1, C_2 in C_3, od katerih je lahko vsak aktiven ali neaktiven. Kateri od kontrolnih signalov bo aktiven določa mikroprocesor 1, informacijo o tem pa dobi vezje 9 preko podatkovnega vodila ob času, ko je aktiviran izbirni signal I2. (Ta je aktiven ob določenem naslovu na naslovnem vodilu.) Vezje 10 razdeli signal ure ($U_{1,2,3}$) na tri enake signale U_1, U_2 in U_3, sprejme podatke P_1, P_2, P_3 od bloka 8 in kontrolne signale C_1, C_2 in C_3 od vezja 9. Poleg omenjenega vsebuje še posebno logiko, ki omogoči prehod določenega para podatek - ura (npr.: U_1, P_1 ob aktivnem kontrolnem signalu C_1) skozi vezje 10. Vezje 11 vsebuje ojačevalnike kontrolnih signalov C_1, C_2 in C_3 za odpiranje oziroma zapiranje elektronskih stikal MS1 do MS6. Vezji 10 in 11 poleg opisanega vsebujeta tudi optične ločilne elemente, ki galvansko ločijo elektronska vezja računalniškega komandnega pulta od "zunanjega" računalniškega sistema. Kot je bilo že omenjeno, kontrolni signali C_1, C_2 in C_3 iz vezja 11 nadzorujejo odpiranje in zapiranje parov elektronskih (MOSFET) stikal MS1 in MS2, MS3 in MS4 oziroma MS5 in MS6. Ob neaktivnih signalih C_1, C_2 ali C_3, so stikala v zgornjem položaju tako, da povezujejo signale klasične tipkovnice in TIPKOVNICE RKP z ustreznim računalnikom centralnega računalniškega sistema. V primeru, če pa je kateri od kontrolnih signalov aktiven, se postavi ustrezno elektronsko stikalo v zgornji položaj, kot je narisano na sliki 1. V tem primeru je izbrani računalnik povezan samo na TIPKOVNICO RKP, po koncu prenosa informacije (ta traja približno 0.5 sekunde) pa se elektronsko stikalo ponovno preklopi v smeri puščice na sliki 1. Po preklopu stikala v osnovni položaj, je možno podatke vpisovati preko klasične računalniške tipkovnice.

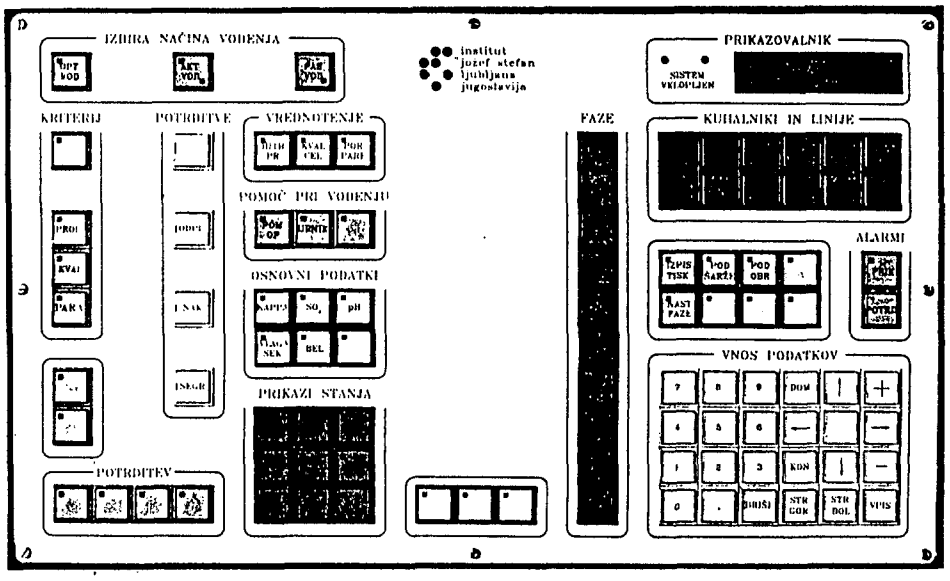
5. IZKUŠNJE PRI VGRADNJI SISTEMA

5.1 VLOGA OPERATERJA PRI VODENJU

Operater, kateremu je računalniški komandni pult namenjen, je razbremenjen cele vrste nalog in opravil, ki bi jih imel pri vodenju sistema preko več klasičnih računalniških tipkovnic. Računalniški komandni pult združuje torej vse funkcije vodenja in nadzora na enem mestu. Hkrati opozarja operaterja na napake pri vodenju, omogoča dobro preglednost nad stanjem v procesu in vodi operaterja od tipke do tipke ob vpisu večzložnih (zaporednih) ukazov.

Operater ima na računalniškem komandnem pultu grupirane tipke v obliki polj (Glej sliko 2). Polja so obdana z okvirjem z zaobljenimi robovi, ki je pretrgana na zgornji strani. Na tem mestu je vnešen napis za določeno polje. Vsaka od tipk v polju ima na svoji čelni strani označeno tudi svojo lastno funkcijo oziroma pomen, po katerem se operater orientira. Vodenje preko računalniškega komandnega pulta poteka na več različnih načinov, pri čemer so lahko ukazi enozložni, dvozložni ali večzložni. Kateri od navedenih tipov ukazov je aktiven je odvisno od same funkcije vodenja. Za ilustracijo navajam primer večzložnega ukaza:

- operater izbere npr. tipko za vpis kappa števila (število pove stopnjo kuhanosti celuloze), pri čemer se aktivira rumena svetleča dioda v zgornjem levem kotu te tipke;
- ob aktiviranju tipke KAPPA začnejo utripati zelene svetleče diode v polju KUHALNIKI IN LINIJE in sicer na tipkah: K3, K4, K5, K6, K7, K8 in K9, ki signalizirajo operaterju, katero od tipk lahko v tem naslednjem trenutku izbere;
- če operater aktivira tipko K3, pomeni,



Slika 2: Izgled računalniškega komandnega pulta od zgoraj

da velja vpis kappa števila za kuhalnik 3, na prikazovalniku pa se pojavijo črtice, ki povedo, da je treba vnesti vrednost kappa števila preko polja VNOS PODATKOV;
- vpišemo neko število npr.: 29.4, nato pritisnemo tipko VPIS;
- podatki se prenesejo na ustrezen računalnik centralnega računalniškega sistema.

Ob napačnih vpisih podatkov se pojavi na prikazovalniku izpis napake v obliki kode, hkrati pa na nepravilno stanje opozori zvočni signal.

6. UPORABA RAČUNALNIŠKEGA KOMANDNEGA PULTA

6.1 MOŽNOST PRIKLJUČITVE VEČ RAČUNALNIKOV

Na računalniški komandni pult je mogoče v splošnem priključiti poljubno število osebnih računalnikov, ki so vezani v mrežo. Le-ti so lahko vezani v poljubnem medsebojnem vrstnem redu oziroma medsebojni kombinaciji. Seveda je pri računalniškem komandnem pultu potrebno povečati število podatkovnih izhodov $P_{1,2,3,4}$, število izhodov za impulze ure $U_{1,2,3,4}$, število kontrolnih izhodov $C_{1,2,3,4}$, ter povečati število elektronskih stikal (MOSFET), pač glede na število vezanih osebnih računalnikov. Pri tem je potrebno izračunati obremenitev posameznih izhodov integriranih vezij in jih s pomočjo ojačevalnikov (tudi močnostnih) prirediti velikosti bremen.

6.2 ZANESLJIVOST DELOVANJA V PRAKSI

Zanesljivost elektronskega vezja za komunikacijo računalniškega komandnega pulta s tremi računalniki tipa IBM PC/XT ali PC/AT je bila dosežena v praksi in je zadovoljila vsem željam in zahtevam tako načrtovalcev kot tudi uporabnikov. Specifičnim željam in zahtevam investitorja se je mogoče prilagajati s spremembami na računalniški programski opremi. Pri zamenjavi programa računalniškega komandnega pulta je potrebno v samem komandnem pultu zamenjati pomnilniško integrirano vezje (EPROM), ki je bil že predhodno programiran v laboratorijskem okolju in preizkušen na prototipni izvedbi računalniškega komandnega pulta.

V praksi se je koncept komunikacije preko tipkovničnih vhodov posameznih centralnih računalnikov izkazal za zelo zanesljivega in po instalaciji v industrijskem okolju z njim nismo imeli nikakršnih problemov.

6.3 UPORABNOST RKP ZA SPLOŠNEGA NAROČNIKA

Za načrtovalca računalniških sistemov in uporabnika le-teh je verjetno zanimiva ideja zamenjave klasične računalniške tipkovnice na enem centralnem računalniku (t.j. tipa IBM PC/XT ali PC/AT) za vodenje procesa s posebno funkcijsko tipkovnico (t.i. računalniškim komandnim pultom), ki je prilagojena specifičnim potrebam v industrijskem okolju.

V poročilu je opisan koncept materialne opreme računalniškega komandnega pulta s poudarkom na njegovi povezavi na tri industrijsko prirejene osebne računalnike in sicer preko tipkovničnih vhodov ter s priključenimi klasičnimi računalniškimi tipkovnicami.

Pri komunikaciji računalniškega komandnega pulta s centralnim računalniškim sistemom preko tipkovničnega vhoda računalnikov je bilo treba rešiti celo vrsto problemov, ki so vezani na generacijo impulzov in so vezani na konfiguracijo materialne opreme. Eden takih problemov je bil prenos impulzov podatkov za računalnike tipa IBM PC/XT, ki se razlikujejo od impulzov za računalnike tipa IBM PC/AT. Ta problem smo rešili s posebnim vezjem in z nastavljivimi prevezami, ki omogočijo definiranje enega od obeh tipov računalnikov za vsak ključpričeni računalnik. Napajanje elektronskih stikal, katera omogočajo priključitev računalniškega komandnega pulta na tipkovnične vhode osebnih računalnikov, smo rešili tako, da niso odvisna od napajalne napetosti računalniškega komandnega pulta. Izklop računalniškega komandnega pulta bi namreč povzročil blokiranje vpisa preko klasične računalniške tipkovnice. Poleg omenjenega se v splošnem lahko zgodi, da je eden od centralnih računalnikov izklopljen (nima napajalne napetosti). V tem

primeru naj bi ostali centralni računalniki delali nemoteno ali preko računalniškega komandnega pulta ali pa preko klasičnih računalniških tipkovnic. Ta problem smo rešili z napajanjem vsakega posameznega elektronskega stikala preko napajalnih linij centralnih računalnikov.

7. ZAKLJUČEK

Računalniški komandni pult smo razvili na Odseku za računalniško avtomatizacijo in regulacije in je bil namenjen pri konkretnem projektu Računalniška avtomatizacija kuhanja celuloze v podjetju VIDEM, Krško. Računalniški komandni pult je bil vgrajen v distribuirani računalniški sistem ob zaključku leta 1989.

8. LITERATURA:

/1/ M. Baar, J. Černetič, V. Jovan, G. Godena, S. Strmčnik in N. Hvala: Računalniška avtomatizacija kuhanja celuloze v podjetju VIDEM, Krško - dopolnitev Idejnega projekta za II. in III. etapo, Institut "Jožef Stefan", 1987, 1988;
/2/ VALMET Co.: DAMATIC AUTOMATION SYSTEM (Instruments Works) 1984;
/3/ VALMET Instruments and System, 1984;
/4/ Alcont: The Total Process Monitoring and Control System, 1985;

Knjižne novice

Pravkar je izšla knjiga **On the Way to Information** avtorja *Antona P. Železnikarja* v založbi Slovenskega društva Informatika. Knjigo lahko naročite na naslovu **Slovensko društvo Informatika, Tržaška c. 2, 61000 Ljubljana.**

Knjiga, ki obsega 240 strani, je mehko vezana, formata A5, njena cena je (vključno s poštnino) **din 270.**

Vsebina knjige je tale: **Predgovor** navaja nekatera izhodišča in probleme pri razumevanju pojma informacije v okviru doktrin različnih socialnih skupin (filozofi, sociologi, informatiki, novinarji, materialisti, idealisti itd.).

Zahvali in opravičilu sledi okvirno poglavje **o informaciji (Na poti k informaciji),** o njenem današnjem razumevanju kot nastajanju informacijskega s protiinformacijo. Poudarjene so nekatere ontološke opredelitve, ki zadevajo informacijo bitja, kot so npr. jezik, čas, metafizika, inteligenca in kulturna informacija nasploh. Informacijska oblika in proces sta osnovni in sestavljeni informacijski entiteti, ki nastajata z informiranjem informacije. Ob tem se postavlja vprašanje informacijskega stroja tudi kot tehnološke možnosti in kot nadaljevanje razvoja umetne inteligence. To poglavje je tudi predlog za razvoj t.i. informacijske filozofije in formalne informacijske teorije (logike, aksiomatike, algebre).

Poglavje **o informacijskih opredelitvah** je poskus opredeljevanja splošnih kulturnih oblik kot informacijskih oblik in procesov. Vprašanje o informaciji sproži vrsto drugih vprašanj, kot so pojmovanje pomena terminusa informacija in s tem povezanih izpeljav: informacijskega, informacionizma, informacitete, informacijskosti in informacibilnega. Glagol informirati inicializira izpeljave pojmov, kot so informiranje, informizem, informnost, informiranjenost in informabilno. Iz pridevnika informativno je mogoče izpeljati informativizem, informativnost in informativabilno. Pojem informatika povzroči nastanek pojmov informatično, informatizem, informatičnost in informatikabilno. Iz glagola informatizirati so izpeljani informatiziranje, informaticizem, informaticiznost in informaticizibilno. Informatizacija omogoči izpeljavo informatizacijskega, informatizacionizma, informatizacionističnosti in informatizacionabilnega. Informacijska oblika in informacijski proces sta osnovna pojma informacije in informiranja. V okviru informacije in protiinformacije se pojavlja informacijska rekurenca in informacijsko vmešcanje nastale in prihajajoče informacije. tudi osnovne in sestavljene oblike življenja je mogoče pojmovati informacijsko. Inteligenca postane informacijska kompleksnost na določeni stopnji razvoja. To velja vobče tudi za t.i. kulturne oblike (od filozofije do umetne inteligence). T.i. bitjevska informacija je oblika informacije živega bitja, njegove metafizike, obnašanja, eksistence. Bit in čas kot informacija je replika na razumevanje filozofske kategorije biti in časa. Tudi razumevanje informacije se približuje t.i. fenomenološkemu konceptu. Metafizika, ontologija in sociologija oblikujejo poseben informacijski kompleks. Autopoeza je poseben vidik informacijskega razumevanja živega. Tudi v nevralnih znanostih je informacijski vidik bistven, od razumevanja delovanja možganov, nevronov, obnašanja do jezika, misli in čustva. Možgane in z njimi pogojeno obnašanje je mogoče anatomsko analizirati na informacijski način. Ustvarjalnost je posebna oblika informiranja v živem. Jezik je informacijska fenomenologija mišljenja in obnašanja. Pisanje, branje, govorjenje in poslušanje so spontano cirkularne informacijske oblike komuniciranja. Filozofija je informacija mišljenja o čemerkoli; neglede na nujnost je aforistična informacija, povezana z vrednotenjem in prepričanjem o splošni naravi stvari. Ideologija je svojsko strukturirana, organizirana in pomensko usmerjena informacija, ki je v spontanosti svojega nastajanja bistveno omejena. Dialektika pozna npr. protiinformacijo v obliki antiteze in informacijsko vmeščanje v obliki sinteze. Znanost je

informacija znanega, logike, empirizma, pozitivizma, teoretičnih konceptov in dokazovanja svoje pravilnosti. Tehnologija je informacija produktivnega bivanja bitij, akcije, produkta, razvoja, konstrukcije in destrukcije. Estetika je informacija mišljenja in izkustva v krogu spontanosti lepega in vrednosti. Etika je filozofska informacija, ki raziskuje cilje, občutja, procese in pojave moralne narave in obnašanja, merila za vrednotenje moralne akcije, informacijsko zasnovo in izvor morale in moralnosti. Ritem in harmonija sta osnovni informacijski obliki živega in vesolja, torej vsakršnje informacije. Umetnost je višek nerazčlenjive, občutene, izrazne in sprejemne informacijske spontanosti, je sinonim za svobodno ustvarjalno moč, informacijsko veščino, umetelno informacijo in voljo do moči. Literatura je zapisana informacija s ciljem, da bo komunicirana k beročemu občinstvu. Bit krize je informacijsko zaprta v svojo domeno kritičnega. Sposobnost kot informacija je kakovost metainformacijske strukture in organizacije v okviru totalne informacije bitja. Tudi nesposobnost je lahko najvišja informacijska oblika bitjevske metafizike. Funkcije uma so pogojene z informacijskimi oblikami in procesi živčnega sistema. Informacija informira in je informirana kot proces informacijskega sistema. Socialni sistemi so informacijske enote posameznih avtopoetičnih sistemov, torej posebni živi informacijski sistemi živih sistemov. Umetna inteligenca je informacijsko področje med znanostjo, umetnostjo in tehnologijo kot informacijo.

Poglavje **o principih informacije** je priprava za kasnejšo izpeljavo dosledne in stroge simbolne formalizacije redefiniranega in razširjenega pojma informacije. Informacija je kot pojem zaokrožen (zaprt) sam vase. To razumevanje informacije je mogoče opredeliti s principi, ki govorijo o cirkularni informacijski spontanosti, informaciji, protiinformaciji, informacijski formuli, informiranju, vmeščanju, nastajanju, protiinformiranju, cirkularnosti, rekurenci, paralelizmu, zaporednosti informacije, informacijski obliki, informacijskem procesu, informacijski strukturi in organizaciji, inteligenci, informacijskem stroju, informacijskem programu, živem informacijskemu stroju in živem informacijskem procesiranju. Pokazane so nekatere posledice in primeri informacijskih principov, in sicer informacijskega objekta in subjekta, protiinformacije, informacijskega sklepanja, informacijske ekvivalence, operatorja »informirati«, informiranja, informacijskega vgnezdenja, nastajanja, protiinformacijske formule, informacijske rekurence, paralelizma, sekvencializma, paralelnega vpraševanja, informacijske oblike in procesa, strukture in organizacije, kulturnih oblik, inteligence, tehnološkega informacijskega stroja itd.

Bog kot informacija je posebno poglavje knjige. V zgodovini ideje kot univerzalne informacije in religije igra Bog osrednjo vlogo pri evoluciji človekovega uma, njegovega informacijskega razvoja in informacijskega nastajanja. Bog je zahteven primer spontane in vase zaokrožene informacije, ki je abstraktna, vsevedna, povsod prisotna (distribuirana), vseobvladujoča, samozadostna informacija, brez začetka in konca, skratka skrajno univerzalizirana in v sebi in zase nastajajoča informacija. Bog je regularni informacijski pojav človekove metafizike, njegove zavesti in podzavesti, tj. totalne avtopoetske informacije živega organizma. Bog je tudi diskurzivni informacijski pojav človekovega uma, t.i. informacijski drugi v intrapersonalnem in interpersonalnem človeškem diskurzu. Bog pooseblja skrajna in poslednja vprašanja človekovega bivanja, njegove zavesti, je razcepitev uma na realni Jaz in onostranski Ti v zgodovini človekove kulture in civilizacije. Idejo katoliškega Boga je mogoče tomistično zajeti v sedem informacijskih principov in jih transformirati v ustrezne, splošnoinformacijske oblike. V zvezi Božjega stvarjenja sveta iz niča se pojavi tudi vprašanje informacijskega niča in neinformacije kot posebnih oblik informacijske relativitete. Evolucijo ideje o monoteističnem Bogu je mogoče opazovati tudi zgodovinsko, od hebrejske biblije, postbiblijskega judostva, kabalističnega nauka do sodobnega postbiblijskega judovstva. Kabalizem predstavlja npr. višek judovske spekulativne miselnosti na področju Boga, saj so koncepti, ki zadevajo t.i. Ein Sof, sefirot, tsimtsum (cimcum) tudi informacijsko zanimivi. V postbiblijskem krščanstvu se Bog kot informacija čedalje bolj približuje sodob-

nim konceptom razumevanja Boga (Avguštin, Tomaž, mojster Eckhart, nominalizem, Kant). Islam izreče značilno informacijsko formulo že na samem začetku svojega nauka: ni boga razen Boga. Po Koranu je Bog nedostopna informacija, ki je zunaj človekovega razumevanja in v vsakem oziru neomejena. Bog je zahtevna oblika cirkularne in spontane informacije. Drugi vidik Boga je t.i. informacijski drugi v človekovem umu kot informacijskem procesu (npr. psihološki koncept pomembnega drugega). Tudi idejo Boga kot informacije bi bilo mogoče formalizirati in tako razčlenjevati fenomenologijo tega informacijskega pojava.

Zadnje poglavje ima naslov **Informacija in postmodernizem.** Osnovno vprašanje je, kakšno je mesto informacije v postmoderni epohi. Najprej je pokazana informacijska razlika med modernizmom in postmodernizmom in opisano razumevanje t.i. informacijske družbe. Poseben problem predstavlja vprašanje postmodernističnega smisla in informacije ter v tem kontekstu problem kulture in civilizacije ter inteligence in intelektualizma kot informacije. Poglavje se dotika problematičnosti t.i. hribarjanskega postmodernizma zlasti na področju razumevanja duha in smisla, ki naj ne bi bila informacijska produkta človekovega uma, ampak nekaj drugega.

Knjiga navaja 51 slovstvenih virov, njim pa sledi izčrpen vsebinski indeks za iskanje pojmov, ki se v knjigi obravnavajo.

Knjiga je pisana v angleščini in temelji na avtorjevih člankih, ki so bili v zadnjih letih objavljeni v časopisu Informatica (Ljubljana), Cybernetica (International Association for Cybernetics, Namur, Belgija) , Anthropos (Ljubljana) in Znamenje (Maribor, poglavje o Bogu kot informaciji). Gre le za neznatno predelavo, poenotenje in stiliziranje posameznih poglavij, le poglavje o Bogu je znantno razširjeno in dopolnjeno s kulturnozgodovinskim razvojem informacije o Bogu.

Prvi del knjige je prevsem konceptualen, avtor pa napoveduje že drugi del, ki bo študija o možnostih simbolne formalizacije informacijskih principov (logika, algebra, diskurz, čas, razumevanje). Tretji del knjige naj bi predvidoma obravnaval koncepte (in predloge) arhitekture in operacijskega sistema informacijskega stroja, zasnovo t.i. informacijskega ekspertnega sistema in formalne (matematične) osnove informacijske teorije.

Knjiga »On the Way to Information 1« je dobavljiva s posebnim naročilom pri Slovenskem društvu Informatika, Tržaška c. 2, 61000 Ljubljana, kupiti pa jo je mogoče tudi v nekaterih knjigarnah v Ljubljani.

Informatica

Informatica *is published four times a year in Winter, Spring, Summer and Autumn by the Slovene Society Informatika, Tržaška cesta 2, 61000 Ljubljana, Yugoslavia.*

Volume 15 Number 1 February 1991 YU ISSN 0350 – 5596

Informatica

A Journal of Computing and Informatics

CONTENTS