Volume 16 Number 4 December 1992
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 (ISSN 0350 – 5596) is published four times a year in Winter, Spring, Summer and Autumn (4 issues).

The subscription price for 1992 (Volume 16) 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 Tomi Pretnar, Bratov Komel 52, Ljubljana.

Informacija za naročnike

Informatica (ISSN 0350 – 5596) izide štirikrat na leto, in sicer v začetku marca, junija, avgusta in novembra.

Letna naročnina v letu 1992 (letnik 16) 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 DEM 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 Tomi Pretnar, Bratov Komel 52, Ljubljana.

Na podlagi mnenja Ministrstva za informiranje št. 23/216 – 92, z dne 27.3.1992, šteje znanstveni časopis Informatica med proizvode informativnega značaja iz točke 13 tarifne številke 3, za katere se plačuje davek od prometa proizvodov po stopnji 5%.

Pri financiranju časopisa **Informatica** *sodeluje Ministrstvo za znanost in tehnologijo, Slovenska 50, 61000 Ljubljana, Slovenija*

Volume 16 Number 4 December 1992
ISSN 0350 – 5596

Informatica

A Journal of Computing and Informatics

EDITOR – IN – CHIEF
Anton P. Železnikar
Volaričeva ulica 8, 61111 Ljubljana

ASSOCIATE EDITOR
Rudolf Murn
Jožef Stefan Institute, Ljubljana

The Slovene Society INFORMATIKA
Ljubljana

Letnik 16 Številka 4 December 1992 ISSN 0350 – 5596

Informatica

Časopis za računalništvo in informatiko

V S E B I N A

Into the New Perspective

In its seventeenth year of publishing (Volume 17, 1993), the journal Informatica is entering in the following conceptual breakdown concerning its contents, typographical standards, and organization. The new image of Informatica will offer a heterogeneous international editing board with globally recognized personalities. On the other hand, several information-significant fields will be considered ranging from the technological field of computing to the philosophical realm of the extended notion of information. In this way, the boundary areas of contemporary philosophy, science, and technology will be conjoined representing the perplexed and disciplinarily interweaved realm of live and artificial informational entities.

As pointed out, in its contents, Informatica will follow several diversely oriented scientific fields in which informatics is placed into the focus of human investigation all over the world. In this sense, also the postmodernistic view of scientific mind pertaining to informatics, cybernetics, robotics, intelligence, informational theories, new formal symbolism—to stress only the most remarkable items—will be treated. Thus, Informatica will prepare a place of possibilities for a spontaneous and, simultaneously, variously open exchange of the informational-scientific thought and its researching within new landscapes of technological development.

In its typographical shape, Informatica will follow the so-called TEX standards which are, more or less, familiar to the scientific communities in the developed as well as developing countries. An internationally dispersed net of editors and referees will guarantee not only the highest possible professional (that is, objective and neutral) standards for submitted paper evaluation, but also the necessary objectivity to the possible diversity, significance, and contents innovation. Further, communication between editors and authors will be enhanced by using regular e-mail channels. In this way, the editorial service of Informatica will perform friendly and fluently for authors over the globe.

The new perspective of Informatica will contribute to the scientific alloy of global theoretical thought in the field of informatics, cybernetics, and artificial intelligence. It will become the friendly abroad for many authors, for Slovenia is a small, poor, and hospitable European developing country. This position of Informatica will offer a rhythmic and harmonic background for author' and editors' improving initiative and a loosened satisfaction to all involved parties.

Anton P. Železnikar
The Editor-in-Chief

AN ESSAY ON EPISTEMOLOGY AND ARTIFICIAL INTELLIGENCE

INFORMATICA 4/92

Keywords: cognition, mentalism, complexity, ratiocination, artifact, doxastic and epistemic notions, formalization, formal system, logic

Oliver B. Popov
Institute of Informatics
Faculty of Natural Sciences and Mathematics
University of Skopje, Skopje

In the selective constellation of pure philosophy epistemology or the study of knowledge has a long and distinguished tradition. However, in the last half of the twentieth century many epistemological concepts and paradigms have entered in a number of diverse disciplines such as linguistics, economics, computer science and artificial intelligence. Indeed, the study of knowledge has become a major area of research in the field of artificial intelligence. The interplay between epistemology and artificial intelligence and its reflections is in the focus of this essay.

The essay is comprised of a series of articles. The first one deals with the origin of relations between AI and epistemology by identifying some of the fundamental questions and problems in AI and the possibility for their solution through epistemology. The second one attempts to find a mode for modality or non-standard logical systems, as well as explores the classical theory of knowledge. The series proceeds with the study of the semantic and syntactic approach, and is finally exploring the hybrid paradigm. At the end, an extensive bibliography is provided for the concerned reader.

PART I: THE ORIGINS OF THE RELATIONSHIP

"All men by nature desire to know...: Aristoteles

1. Genesis

The contemporary echo of the idea of an intelligent machine originates with the invention of the stored program computer and the reflections by Alan Turing and Claude Shannon. Those reflections are certainly behind the emergence of Artificial Intelligence as a valid scientific discourse in the second half of the 20th century.

The possibility that an artifact could perform cognitive functions, ones exclusively in the domain of human competence, is intriguing in its implications. It appears that human nature has been deprived of the last marvels of existential uniqueness: the ability to reason and create knowledge which are intrinsic components of the phenomena termed as intelligence.

Indeed, a respectable number of researchers in Artificial intelligence

have explicitly equated the cogitative potential of humans with the one of artifacts. There are two plausible sources for the equivalence drawn between artifacts and humans: the first one is the complexity of artifacts and the second one is the rationalist tradition in the analysis of human cognitive behavior.

Complexity is reflected in the essential attributes of each artifact: relative independence, flexibility and multi-usefulness. All of the attributes are present both on a structural level (or hardware) and on a functional (or software) level.

The effect of unpredictability is a direct consequence of complexity. Regardless of the intended level of determinism that might be possibly achieved through simulation, the presence of unpredictability is inevitable. This might be due to the empirical notion that results of a process are at least one level higher in their complexity than the process that has produced them. Hence, there is a possibility for unequivocal opposition to the rationalist tradition which demands a closer examinations.

Intellectual disciplines usually evolve over many centuries. The 'success' of any intellectual discipline is often measured in how well its fundamental principles have been encapsulated within a formal system. The mathematical method that attempts to coincide with pure formality and rationality has been accepted as a metric of how 'serious' and 'hard' scientific endeavor is. Rationality requires clearly defined problems. However, most AI problems defy stable formulations which raises the second argument against the rationalist approach.

The rationalist method has also been adopted in the identification of the phenomena that create knowledge: experience and ratiocination. The notion of experience is reduced to that of a modification of a knowledge base. In this case, the initial set of facts represents an apriori knowledge built into a system which upon the application of valid inference becomes an experience for a reasoning agent. Informally, an argument as a result of inference is valid if and only if its premises could not be true and a conclusion false.

Ratiocination is to be understood as a form of analysis entirely dependent on the inherent power of the 'mind' to reason. The identification of processes of reasoning and computing leads to general (although not universal) acceptance of brute force capabilities by an artifacts as a form of ratiocination.

The characterization of experience and ratiocination is consistent with the axiomatic acceptance in AI of the dictum "cognition is a resolute computation". Pragmatics makes this quite necessary. After all, there will always be a difference between a form and a context. One must recognize that form is a prerequisite for computation. By adopting some of the principles of mathematical logic, the present body of work in epistemology and AI follows the rationalist tradition.

The diviation from rationalism is the requirement that epistemic reasoning is done within the system. It is acknowledged that cognition demands inner presence, or as

Heidegger argues "being-in-the world". The intention of an epistemic system is to capture a part of the world by empowering an artifact with its own mechanism for reasoning about the state of knowledge that encompasses it. Thus, it is important to distinguish between the method which is rationalistic and the intention which is mentalistic. The mentalism induced by the intention should not be confused with the phenomenon of anthropomorphic fallacy whose ramifications are addressed later.

2. The Limits of Formality

The word formal, as commonly understood, is synonymous with the word mechanical.The definition of Hilbert's Programme and the seminal works of Godel and Church put to rest the speculative arguments of unlimited expressive power of formal systems, hence on mechanization or automation [Chu56,God31]. These results are a vindication in the resolution of mentalistic problems with formal methods. Instead of confining, the results simply state that formality might prove to be a reasonable first step due to structural and functional constrains of an artifact and the incomplete knowledge of ourselves. The nature of discipline such as AI where Man is an object and a model of studying is inevitably a source of problems.

An adequate simulation of bona fide human properties subject to self-reflected interference which mislead some researchers to look for and recognize purely anthropomorphic features in artifacts. For example, Searle distinguishes between a weak AI that "can build a powerful tools to mimic and study the mind", and strong AI where "something can think in virtue of instantiating a computer program"[Tor84]. Accordingly, Searle claims that the former problem is solvable while the answer to the later one is a definitive no. Others have asserted that on an abstract level there is no difference between the mind and the computer. These arguments are so strong that can be easily dismissed on the grounds that the understanding of human cognitive properties and functions is still in its inception.

A prima facia value is the affirmation of the symbiotic relationship between AI and Man. Although there is "still to be learn more about machines form the study of man" as Von Neumann put it, hopefully there is something to be learned about man from the way some artifacts work. The artifacts are actually a possibility for an extension of human potential and never its replacement.

At this point, it might be prudent to propose a principle of 'rational commitment'. It is an informal recognition of the fact that the question of intellectual equality between Man and Machine is an exercise in impuissance and a statement against the reduction of intelligence to a single phenomenon. The principle does not imply either the acceptance of rationalist tradition or the exclusion of mentalism. As stated earlier, formalizations are based on abstractions. Abstractions, occasionally create deviant images of reality as it is reflected in efforts to capture the essence of common sense reasoning. By necessity models in AI start ad

hominem, by desire they end up ad machinum. Whether they coincide in extension is yet to be seen. Both should be acknowledged whenever a question is raised of how comprehensive, compelling and complete a certain theory is that leads to better understanding of human mental abilities. The principle is asserted since the emphases is on factual and positive rather then on speculative and negative.

3.Fundamental Concepts

Any serious intent by an artifact to encompass a level of cognitive generality postulates an integral ability on its part to reason with deductive, inductive and evidential rules [Chi66]. The evaluation of evidence is also subject to inductive and deductive criteria. A distinction is made in order to stress the meta-system significance of evidence. In essence, the rules of evidence govern the epistemic environment of an intelligent agent. They determine the course of proper reasoning since in extension the true evidence coincides with knowledge.

The assessment of knowledge within a reasoning system is usually a result of an internal or external query. The nature of the request might be either to revise the current knowledge of the system (in view of some new situation) or to answer whether some proposition is known or not. It is the later type of response that epistemic models are trying to capture. Inter alia, the solution of reasoning about what is known may also prove necessary to address the problem of knowledge revision.

The AI community shares the controversy concerning the ontological difference between declarative and procedural knowledge with so many paradigms in epistemology. Declarative knowledge is expressible through the mode 'knowing that' while procedural knowledge through the mode 'knowing how'. An additional mode, often used in the process of explanation, is expressed through 'knowing why'. There are authors who take the position that both procedural and explanatory modes are reducible to declarative mode. In the case of causal ordering (an instance of which is the notion of plan) the nature of ordering and its condition can also be expressed through a set of declarations (or propositions). If propositions are treated as objects of knowledge then the notion of knowing is examined as an empirical relation between a reasoning agent and an object.

Doxastic and epistemic inclinations such as knowing and believing are expressed through the verbs of propositional attitudes, know and believe [Rus61]. Propositional attitudes usually define or appear in an intensional context. A context is intensional if its co-referential expressions such as singular terms, predicates or sentences that have the same denotation are not substitutable without changing the truth value of the contexts as a whole. Otherwise the context is termed as extensional.

One can find the metaphysical difference between knowledge and belief rather interesting in understanding the nature of knowledge. Knowledge is a correct interpretation of reality; the imposition of correctness is distinguish knowledge

from mere ideas, opinions, perceptions, and beliefs. As defined, the notion of knowledge is identical with the notion of truth. De facto one of these notions would be redundant in the presence of the other.

Knowledge is intrinsically committed to representation and identification, The representation deals with the ontological properties of knowledge; the identification addresses the metaphysical properties.

It is commonly understood that: an agent 'a' knows Π if and only if
(1) The Truth conditions is satisfied: Π is true
(2) The Belief condition is satisfied: an agent 'a' believes Π .
The term 'agent' denotes an entity capable of reasoning while Π stands for an arbitrary proposition. The first condition is the least controversial one, since it is a matter of intuitive sanction and linguistic necessity.

Contrasting views exist with respect to the Belief condition. One view accepts the condition either in a strong form such as 'knowledge entails certainty' or in a weak form such as 'it is not the case that knowledge entails certainty'. The other view denies the Belief condition either in a weak form 'is not the case that knowledge entails not belief' or in a strong form 'knowledge entails not belief'. In summary, the study of an epistemic context requires that both conditions (1) and (2) are satisfied. Whence, in a doxastic context the Truth condition is neither necessary nor adequate.

Various paradigms for knowledge representation in AI such as semantic networks, frames and units, reach for mathematical logic whenever a need for justification and rigor arises. This is done by transformation of the object formal system into a modified system of logic.

It is not surprising that the need for formalization has been plagued with empirical adventures under the name of heuristic adequacy. One might agree with Ryle who posits that formal logic is a regimentation of the relevant sectors of the ordinary discourse. An extreme care should be taken in order not to defeat the aim and intent of logic which is the discrimination of invalid form valid argument and provision of rigorous but simple standards for the expression and evaluation of arguments.

The nature of extension is to be understood as follows: the basic system and the extend one share the same vocabulary, and have the same theorems and rules of inference a propo the shared vocabulary. The extension is done to enrich the expressive power of the underlying system via some extended vocabulary, additional axioms and rules of inference.

It is implicit that the arguments are subject to the criteria of rationality, material adequateness and intuitive admissibility. In the absence of consensus with respect to the standards for inclusion of arbitrary formal systems to the family of logics, the question would be ignored because it can lead to an unnecessary metaphysical digression.

The use of a logical system follows from the intention to put forward an explanatory model for understanding the use of epistemic notions in an ordinary discourse. Clearly the laws of logic and the laws of thought are not isomorphic [Hin62].

The former are consider to be an expression of ability rather then obligation.

Thus given an intellect and sufficient time (effort), logical and material information, inferences could be made about some state of affairs. It is in the nature of logic to induce new knowledge in an environment where the information content is restricted to the initial one. Hence, the intuitive understanding of logic that we have is probably best expressed by Witgenstain who wrote"...logic is not a theory of the word, but its reflection". It is this reflection, particularly in epistemology and AI, that is in the focus of our interest.

Bibliography

[Arm73] Armstrong, D. Belief, Truth, and Knowledge. Cambridge University Press, 1973.

[Chi66] Chisolm, M. J. Theory of Knowledge. Prentice Hall, 1966.

[Chu56] Church, A. An Introduction to Mathematical Logic. Princeton University Press, 1956.

[God31] Godel, K. "Uber Formal Unentscheidbare Satze der Principia Methematica und Verwandter Systeme", Mathematische Physik, Vol. 38, 1931.

[Hin62] Hintikka, J. Knowledge and Belief. Cornell University Press, 1962.

[Rus61] Russell, B. Basic Writings. A Clarion Book, 1961.

[Tor84] Torance, S.B. The Mind and the Machine. Ellis Horwood Ltd, 1984.

AN INTRODUCTION TO INFORMATIONAL MACHINE*

INFORMATICA 4/92

Keywords: architecture, cybernetical perquisites, data machine, foundations, informational machine, informational program, informational theory, intelligent machine, operating system, technological perquisites

Anton P. Železnikar
Volaričeva ulica 8
61111 Ljubljana
Slovenia

This essay is an introduction to informational machine (IM) as it might be implemented in the coming decades. IM is an informational phenomenon and informs in itself and to the user in an informationally phenomenal way (IM externalism, internalism, metaphysicalism, and phenomenalism). For this purpose it needs: a massively parallel structured interconnection net of high- performance microprocessors (e.g., 64-bit, three-dimensional RISC); specifically (informationally) designed processor interconnection network; an operating system offering informational, counterinformational, and embedding support to informing entities (operands and operators) and translating informational language; an informational dictionary of entities; and lastly, a powerful peripheral equipment capable to accept (memorize) and transmit all kinds of information (picture, sound, data, signal, control, linguistic information, etc.). The goal of IM design is to achieve the state of »intelligence«, by which IM is an understanding device operating (behaving) within the surrounding world.

Uvod v informacijski stroj.* Ta spis je uvod v informacijski stroj (IS), ki bi ga bilo mogoče uresničiti v naslednjih desetletjih. IS je informacijski fenomen in informira v sebi in k uporabniku na informacijsko fenomenalen način (eksternalizem, internalizem, metafizikalizem in fenomenalizem IS). V ta namen potrebuje: masivno paralelno strukturirano povezovalno mrežo visokozmogljivih mikroprocesorjev (npr. 64-bitni, trirazsežnostni RISC); operacijski sistem z informacijsko, protiinformacijsko in umeščevalno podporo informacijskim entitetam (operandom in operatorjem) in s prevajanjem informacijskega jezika; informacijski slovar entitet; in nazadnje zmogljive periferne naprave, ki spejemajo (pomnijo) in oddajajo vse vrste informacije (sliko, zvok, podatek, signalno, kontrolno, lingvistično informacijo itd.). Cilj načrtovanja IS je doseganje stanja »inteligence«, s katero postane IS razumevajoča naprava, ki deluje (se obnaša) v obdajajočem svetu.

0. Introduction

The phenomenological account of how scientific facts are arrived at by leaving out significance shows why, once we have stripped away all meaningful context to get the elements of theory, theory cannot give back meaning. Science cannot reconstruct what has been left out in arriving at theory; it cannot explain significance. For this reason, even though natural science can explain the *causal basis* of the referential whole, »'Nature' ... can never make *wordliness* intelligible«.

— Hubert L. Dreyfus [BIW] 121

One of the aims of informational theory [TIL, FIP, BIA] is to lay the foundations of the informational machine (IM). These foundations pertain greatly to the concept of an intelligent informational machine (IIM) and their laying is a process in which several new components have to be settled in theoretical, symbolic, cognitive, technological, architectural and other, yet not completely recognized, informational ways.

IM is a consequence of that what one understands as the computer, computing system, or data processing device. The computer is a data machine (DM) where data is a particular, informationally specific entity, a bounded form of information, which concerns facts, factical information, which as a fact does not change during an informational process. DM is an informationally reduced device within the concept of the arising IM and, as such, can never reach the level of, for instance, an intelligent informational machine. Data intelligence remains within the scope of the artificial, modeled, simulated intelligence which, in comparison to natural intelligence, roots in the structure and organization of the algorithmic and data programming on (and by means of) a computer system. Along these lines, the result is a data-stable, algorithmic, well-determined system designed by both mathematical and programming philosophy and conceptual means.

The difference between data and informational processing lies in algorithm vs. informational program, determined foreseeability vs. possibilities of spontaneous informational arising, strict splitting between data and programs vs. operand and operator duality of informational entities, recursively determined program and processing cycles vs. metaphysical informational circularity, the determined closeness of data and programs vs. the phenomenal informational openness of informational operands and operators. IM is an essential extension of DM, from the rigid, algorithmic, mathematical machining to the circularly spontaneous possibilities of informing within the goal-directness, intention, orientation, significance and, not lastly, possible intelligence of arising informational entities in the world. Only in case of an informational machine it becomes possible to speak about the informational understanding, which is a synonym for the intelligent informational behavior of entities within an informational realm, that is, informational entities' surrounding world. Intelligent entities have to behave informationally (in this case understandingly) in the real world, which concerns them in an essential, intentional, developmental, and significant way.

By this essay, the author communicates the concept of an informational machine in the form of a foundational, researching, and technological discourse. Is it already possible to presume and preview the architecture and operating system of the future IM? Both informational theory and philosophy offer concepts and mechanisms which could dramatically impact not only the structure and organization of the IM involving components in the field of electronic, photonic, and biological technology, but also the realms of sufficiently clear comprehension, conceptualism, design, and implementation of machines.

1. The Informational Theory and Informational Machine

Das Miteinandersein ist — ihm selbst verborgen — von der Sorge um diesen Abstand beunruhigt. Existenzial ausgedrückt, es hat den Charakter der *Abständigkeit.* Je unauffälliger diese Seinsart dem alltäglichen Dasein selbst ist, um so hartnäckiger und ursprünglicher wirkt sie sich aus.

Martin Heidegger [SZ] 126

Informational theory (IT) offers formalized means which pertain to informing and informedness of entities (things, phenomena, processes). Informational operands, informational operators, parentheses, commas, and semicolons constitute the syntactic background of formulas and informational formula systems of the theory. In IT, operands and operators are united entities which can be distinguished from each other in instantaneous situations of a formula or formula system. Thus, it is possible to determine informational theories for different purposes. The most remarkable theories of this sort concern understanding, computing, inference, cognition, etc. and also signal processing on a physical level, process control, living phenomenality in biological systems, informational theories of societies, informational mathematical theories, scientific languages, and system theories. IT's enable new approaches to various theoretical and practical problems using specific, informationally conceptualized languages, in which formulas by themselves inform and are informed, that is, arise and are arisen informationally. These formalisms (informational programs) are on the way to be implemented by the use of informational machines.

An IM will perform on the basis of informational formulas, which are programs, expressed in

informational language and behave as logically arising entities, possessing their own externalism, internalism, metaphysics, and phenomenalism. In this point, informational programs differ essentially from computer programs, which can be comprehended as texts which contents (semantics) does not change in any way. IP's (informational programs) surpass the computer programming structure and philosophy, programming bounds, and strict, mathematically conceptualized notions.

The way to the proposed informational machine goes back to the basic axiomatic concepts of informational theory. In a concrete case, IT by itself is a textual informational entity, possessing informational phenomenality, which in accordance with its intention, significance, understanding, etc. informs and is informed by means, mechanisms, and informational support of an informational machine. Thus, the informing of a theory's text can be achieved, realized, or implemented by an IM. In a similar way as computer programs are executed by computers, IP's will govern the informing of an IM. So, the question arises, how an IM has to be conceptualized and constructed to come close to the requirements of informational phenomenality of entities in general and in each particular (thinkable) case.

It is to understand that IT and IM support each other. First, a concept and realization of IM becomes possible by the (strict) application of IT's principles, derivation of theorems, and machining informational systems. Then, the IM is supported by the so-called system informational programs, which constitute an informational operating system. When an IM functions as a machining informational entity, the machine can take an IT and let it arise informationally within a given informational realm. Thus, an IT can develop the IM in an informational way and vice versa, simultaneously, in parallel, intentionally, significantly, and intelligently. To bring this possibility of informing to the foreground, we have to develop the sufficiently clear and straightforward detailed concept of informational machine. And this way of development will be shown stepwise in the sections which follow.

2. The Computer as a Data Machine

What is the difference between a data entity and an informational entity? What is the difference between a data machine (= computer) and an informational machine? A DM has to follow the data phenomenality in an architectural and programming construction, while an IM follows the informational phenomenality which broadens the conceptualism of its architecture and programming in an informationally arising manner. The difference between these phenomenalities is essential: data is well-defined information in a particular way; it is specifically bounded by an algorithmic structure and in this way not flexible as information at all.

As a fact or factical information, data informs externally in an unbounded way; it simply informs external observers as any other informational entity. Regardless of the nature of its facticity, the informing of data is spread (broadcasted) in space and time, where different observing entities can become sensitive for informing of data, that is, informed by data. Intelligent entities can, for instance, observe the steadiness of data informing, its facticity, and its informational unchangeableness, non-arising in contrary to a regularly informing entity. Thus, data appears to the observer as a steady, reliable, and predictable information, which can vary only in some known value, degree, truth, etc., but not in its, for instance, semantic character, mood, property, relation, intention, and significance. Data possesses a firm characteristics (property, relation) of something which it concerns, for instance, number, value, degree, structure, facticity of something. In this respect, its characteristics must never change and its changeable components are only factical parts of its characteristics.

The first question relating computers is how does data inform? As a specific informational phenomenon, data informs, but it is informed merely in the sense to preserve (memorize) its facticity, characteristics, or well-determined meaning. The meaning of a certain data, its semantic scope, must never change. It means that the informedness of data is a steady phenomenon, meaningly stable, and semantically unchangeable. How can this phenomenality of data be expressed abstractly, on a formal and symbolic level?

Let δ mark a data entity as an operand and let $\models$ be the symbol for the most general operator of informing. Informing of data is a data property (its own predicate) and the proposition *Data Informs* is the most primitive phenomenological fact of data itself. It belongs to data as an entity, is a part of data possible (entire) informational phenomenality. It means that data as an entity implies the informing of data. Thus the *informatio prima* pertaining to data is postulated by

(1) $\delta \Rightarrow (\delta \models)$

The sense of this implication (symbol $\Rightarrow$) is to show how the informing of data, that is, $\delta \models$, is an integral part of data itself and not a separate entity. The consequence of axiom (1) is

(2) $(\delta \models) \subset \delta$

where $\subset$ is the operator of informational inclusion which says that process $\delta \models$ is a part of δ itself.

It is to say that formulas (1) and (2) do not differ for any general case when data entity δ is replaced by an informational entity. Where does the essential difference between data and information occur?

Data has to keep its identity, which is the facticity of its phenomenality. With other words, it has to memorize its informational identity by preventing its informational structure to be changed. What does the keeping or memorizing of data identity mean at all? Data informs factitiously, which has the meaning of non-spontaneously and artificially. Thus, formula (1) can be particularized into

(1') $\delta \Rightarrow (\delta \models_{\text{factitious}})$

where operator $\models_{\text{factitious}}$ is a general informational operator of factitious informing, probably constructed (imagined) by the data observer in an artificial and informationally not spontaneous way. The question is: can data, in the described sense, exist as a natural situation (entity) at all or is it the consequence of an observer's attitude?

The next basic question is how data is informed? As a fact, data must keep its identity. So, the possible answer to the question is

(3) $\delta \Rightarrow (= \delta)$

The sense of this implication is to show how the informedness of data, that is, $= \delta$, is an integral part (property) of data itself or

(4) $(= \delta) \subset \delta$

and that this informedness is identical. Instead of operator $=$ operators $\models_{\text{identic}}$, $\models_{\text{steady}}$, or $\models_{\text{factitious}}$ could be used.

The consequence of identical informing of data δ is the so-called metaphysical identity which presupposes the steadiness of data, that is

(5) $\delta \Rightarrow (\delta = \delta)$

and, certainly,

(6) $(\delta = \delta) \subset \delta$

Formula (5) is an axiomatic consequence of both axioms (1) and (3). Thus, also the last fundamental axiom of informing of data can be given by

(7) $\delta \Rightarrow (\delta \models; = \delta)$

where

(8) $(\delta \models; = \delta) \subset \delta$

is the logical consequence.

We can unite the four basic axioms into the following logical scheme of informational formulas:

(9) $\delta \Rightarrow (\delta \models)$; [*data externalism*]
$(\delta \models) \subset \delta$;
$\delta \Rightarrow (= \delta)$; [*data internalism*]
$(= \delta) \subset \delta$;
$\delta \Rightarrow (\delta = \delta)$; [*data metaphysicalism*]
$(\delta = \delta) \subset \delta$;
$\delta \Rightarrow (\delta \models; = \delta)$; [*data phenomenalism*]
$(\delta \models; = \delta) \subset \delta$

Data externalism, internalism, metaphysicalism, and phenomenalism are basic properties of data δ where operators of informing $\models$ and $=$ are integral parts of data δ itself!

This axiomatic introduction of the nature of data was necessary to get the possibility for characterizing the components of data machine $\mathfrak{D}$ (computer). The straightforward formula of informing of a data machine $\mathfrak{D}$ is

(10) $(\delta_{input} \dashv_{receive} \mathfrak{D}) \models_{send} \delta_{output}$

where data input δ_{input} pertains to computer system programs $\mathfrak{P}_{sys}$ (operating system), computer user programs $\mathfrak{P}_{use}$ (applications, tools), and user data δ_{use} (user data bases). Informational operator $\dashv_{receive}$ says that δ_{input} is received by machine $\mathfrak{D}$. Informational operators of the reversed type, that is, $\dashv$, express the passive (participle) mode of informing when read from the left side to the right of informational formula. Implicitly, data machine $\mathfrak{D}$ as an data informing entity (device) operates and is operated by means of the received programs and data in an open, however, normally, well-determined way, that is,

(11) $\mathfrak{D} \Rightarrow (\mathfrak{D} =; = \mathfrak{D})$

Formula (11) shows another bounded situation of data machine $\mathfrak{D}$, where machine phenomenalism roots in an identically structured (architectural) machine externalism (informing $\mathfrak{D} =$) as well as in an identically structured (architectural) machine internalism (informedness $= \mathfrak{D}$). In this sense, machine $\mathfrak{D}$ receives ($\dashv_{receive}$) input data and sends ($\models_{send}$) results ρ_{result} of its action to its peripherals $\mathfrak{P}_{peripheral}$. Data machine $\mathfrak{D}$ is informationally open ($\mathfrak{D} =; = \mathfrak{D}$) only for programs, data, and results which fit its architecture, and system described by formula (11) implies the machine openness of the form

(12) $\mathfrak{D} \models_{arc}; \models_{arc} \mathfrak{D}$

where $\models_{arc}$ is the architectural (computer hardware) defined property of DM functioning. Certainly, the informing ($\mathfrak{D} \models_{arc}$ as externalism or output) and informedness ($\models_{arc} \mathfrak{D}$ as internalism or input) case can be particularized in different possible ways.

What is a more detailed structure of data machine (computer)? We already listed (marked) some components in the previous paragraph. That which is, in fact, called a machine is the data machine architecture (structure of machine's hardware), marked by $\mathfrak{A}_{arc}$. What does the machine architecture produce and what does it sense? The general answer is

(13) $\mathfrak{A}_{arc} \Rightarrow (\mathfrak{A}_{arc} \models_{arc\_prod}; \models_{arc\_sense} \mathfrak{A}_{arc})$;
$\mathfrak{P}_{sys}, \mathfrak{P}_{use}, \delta_{use} \models_{arc\_sense} \mathfrak{A}_{arc}$;
$\mathfrak{A}_{arc} \models_{arc\_prod} \rho_{result}$

The last system constitutes the informationally open phenomenal and metaphysical structure of machine architecture, where $\mathfrak{A}_{arc}$ is a well-defined and informationally non-arising physical structure, operator $\models_{arc\_prod}$ means *produce(s) architecturally* and operator $\models_{arc\_sense}$ has the meaning *sense(s) architecturally*. Thus, the phenomenalism of DM is data-informational. Functioning of a DM $\mathfrak{D}$ can be described by the formula

(14) $((\mathfrak{P}_{sys}, \mathfrak{P}_{use}, \delta_{use} \subset \mathfrak{P}_{peripheral}) \dashv_{receive}$
$(\mathfrak{A}_{arc} \models_{execute} \mathfrak{P}_{sys}, \mathfrak{P}_{use}, \delta_{use})) \models_{send}$
$(\rho_{result} \subset \mathfrak{P}_{peripheral})$

It is to stress that all occurring operands in the last formula belong to data entities which are well-determined, algorithmic, mathematical, and data structured items (alphabetically, δ_{use}, ρ_{result}, $\mathfrak{A}_{arc}$, $\mathfrak{P}_{peripheral}$, $\mathfrak{P}_{sys}$, and $\mathfrak{P}_{use}$). A data machine $\mathfrak{D}$ functions according to formula (14), where the architectural bounds of formula (13) have to be considered. If by $\varphi_{(14)}$ formula (14) is marked, the data machine open informing can be expressed by

(15) $\mathfrak{D} \Rightarrow (\varphi_{(14)} =; = \varphi_{(14)})$

or

(16) $\mathfrak{D} \Rightarrow (\varphi_{(14)} \models; = \varphi_{(14)})$

in case where $\mathfrak{D}$ is not necessarily recognized by its observers (users) as a data device, so, they may believe that $\mathfrak{D}$ informs informationally (for instance, in situations and attitudes of artificial intelligence).

3. Architecture of IM

We recognized in which sense the architecture of a data machine is merely data-informational (informationally non-arising). What is the architecture of an informational machine and what could it be in the present technological circumstances? The first request would be that the architecture of an IM, marked by $\mathfrak{A}_{arc\_IM}$, must be informational without any limits in advance. The request is

(17) $\quad \mathfrak{A}_{arc\_IM} \Rightarrow (\mathfrak{A}_{arc\_IM} \models; \models \mathfrak{A}_{arc\_IM})$

How could such a machine architecture be implemented conceptually and then technologically?

Let us examine some general architectural criteria for an IM in the following manner:

(1) $\mathfrak{A}_{arc\_IM}$ is a massive parallel processor system where the number of processors is not limited in advance.

(2) Massive parallelism of $\mathfrak{A}_{arc\_IM}$ means an extremely flexible, changeable, and arising processor network in which sufficiently complex processors and their interconnections arise in an informational sense.

(3) At the present state of technology, processors π_1, π_2, ... of $\mathfrak{A}_{arc\_IM}$ are data processors (e.g., 64-bit RISC) capable to fit into the massively parallel network by three (and not only two) dimensions.

(4) Informational entities (operands, formulas of operands and operators) represented by processors in the massively parallel network are supported by mechanisms for sensing relevant (entities concerning) information in the interconnection network.

(5) The architectural interconnection network ($\mathfrak{S}_{network} \subset \mathfrak{A}_{arc\_IM}$) possesses methods for the building-up of connections between processors representing operands and formulas (also subformulas) which interact informationally according to particular informational formulas.

(6) Each informational operand has its own processing informational system (basic informational determination of the entity by the basic entity formula in an informational dictionary of entities) to which processors are allocated. Processors are allocated not only to operands but also to informational formulas in which operands occur. Thus, to a complex formula, processors are allocated to each operand, subformula and to the formula itself. The last two behave as additional operands with additional processors allocated.

(7) An informational arising of IM architecture can be simulated by a dynamically changing interconnection network (interconnection distributed processing device), $\mathfrak{S}_{network}$, in which a sufficient number of operable processor components is available. On the other side, the functional power (operational complexity, speed) of processors and their interconnection can reach performances necessary for informational systems for the most pretentious applications.

(8) The listed performances of IM architecture can grow by the advancing of computer and telecommunication technology. For instance, the number of microelectronic components may reach several billions per chip. The next frontier may be based on single-electron devices and ballistic transistors where the philosophy and perspective of quantum physics is coming into scientific and technological game. Photonics will take over many electronic functions in transmission of information, logic functions, communication switching (interconnection networks), etc. [FOT].

Let us determine the IM architecture $\mathfrak{A}_{arc\_IM}$ in more detail by the informational architecture system. The following operands and operators are introduced:

π_1, π_2, ... are high-performance microprocessors inserted in the nodes of massive parallel interconnection network $\mathfrak{S}_{network}$;

σ_{state} represents the state of architectural informing $\mathfrak{S}_i$ (where i marks processors' subscripts) of processors within the interconnection network $\mathfrak{S}_{network}$;

$\sigma_{state}(\mathfrak{A}_{arc\_IM})$ is the state of architecture, expressed as informational function pertaining to the IM architecture itself;

$\mathfrak{A}_{arc\_IM}$ is the IM architecture as informational entity;

$\mathfrak{S}_1$, $\mathfrak{S}_2$, ... are the so-called informings belonging to particular processors π_1, π_2, ...;

$\mathfrak{S}_1(\pi_1)$, $\mathfrak{S}_2(\pi_2)$, ... are informational functions, where particular informing $\mathfrak{S}_i$ depends on

	the state of processor π_i;
$\mathfrak{S}_{network}$	is informational entity representing the processor interconnection network in all imaginable complexity which can be decomposed to the arbitrary details;
$\models_{by}$	means inform(s)_by or inform(s)_by_means_of;
$\models_{connect}$	means connect(s);
$\Rightarrow$	means implies/imply; and
$\subset$	means inform(s)_in or simply is/are_in.

The initial formula system (prior to the detailed informational decomposition) is

(18) $(\pi_1, \pi_2, \ldots \subset \mathfrak{S}_{network}) \subset \mathfrak{A}_{arc\_IM}$;
$\pi_1, \pi_2, \ldots \Rightarrow$
$(\pi_1 =; = \pi_1; \pi_2 =; = \pi_2; \ldots)$;
$(\mathfrak{S}_{network} \models_{connect} (\pi_1, \pi_2, \ldots)) \models_{by}$
$\sigma_{state}(\mathfrak{A}_{arc\_IM}))$;
$\sigma_{state} \Rightarrow (\mathfrak{S}_1(\pi_1), \mathfrak{S}_2(\pi_2), \ldots \subset \mathfrak{S}_{network})$

The first formula is an outset of the position of (the arising) IM architecture which functions as an array of dynamically connected high-performance microprocessors (RISC). In this sense, the IM architecture remains informationally open on the level of its interconnection network and, so, is informationally phenomenal. In the second formula the data nature of processors is explicated as a consequence of nowadays electronic and future photonic technology. The only request is that processors have a unique and »sufficiently« complex and effective structure (e.g., high-performance RISC with at least 64-bit). The third formula expresses the informational nature of processor connection in a particular situation by means of the present functional state of IM architecture. The last formula says that the functional state of IM architecture implies the informing of particular (active) processors within the interconnection network.

Formula (18) is an initial architectural position. This formula may now be decomposed in more and more details according to the concepts of the IM architecture designers. It may happen that a future microprocessor generation, at least in some respects, will reach the step of informational phenomenalism.

4. Informational Programs

So far, informational programs belong to the most promising concepts of informational phenomenality [IBU]. In a massively parallel IM $\mathfrak{M}$, the role of informational programs (in short, IP) becomes informational in all imaginable entirety. An IP, marked by $\mathfrak{P}$, is a regular informational entity in the IM environment. What could this property under machine circumstances mean at all?

An informational program $\mathfrak{P}$ has its externalism, internalism, metaphysicalism, and phenomenalism, that is, in a symbolic summarizing form,

(19) $\mathfrak{P} \Rightarrow (\mathfrak{P} \models; \models \mathfrak{P}; \mathfrak{P} \models \mathfrak{P})$

Then, IP $\mathfrak{P}$ is informationally open to the IM environment in several ways. Firstly, it informs the IM system and is informed by it or, symbolically,

(20) $(\mathfrak{P} \subset \mathfrak{M}) \Rightarrow (\mathfrak{P} \models \mathfrak{M}; \mathfrak{M} \models \mathfrak{P})$

and, secondly, it is informationally open to other active, that is, informing informational programs $\mathfrak{P}_1, \mathfrak{P}_2, \ldots$ within the informational machine $\mathfrak{M}$, thus,

(21) $(\mathfrak{P}, \mathfrak{P}_1, \mathfrak{P}_2, \ldots \subset \mathfrak{M}) \Rightarrow$
$(\mathfrak{P} \models \mathfrak{P}_1, \mathfrak{P}_2, \ldots ; \mathfrak{P}_1, \mathfrak{P}_2, \ldots \models \mathfrak{P})$

Thirdly, within IM $\mathfrak{M}$, informational program $\mathfrak{P}$ informs IM architecture $\mathfrak{A}_{arc\_IM}$ and IM operating system $\mathfrak{P}_{sys\_IM}$ and is informed by them. This fact yields

(22) $(\mathfrak{P} \models \mathfrak{A}_{arc\_IM}, \mathfrak{P}_{sys\_IM})$;
$(\mathfrak{A}_{arc\_IM}, \mathfrak{P}_{sys\_IM}) \models \mathfrak{P}$

Machine architecture and machine operating system support the informing of IP $\mathfrak{P}$ in an informational way, enabling its informing, counterinforming, and embedding of information and communicating in an informing way with other programmed informational entities. To an IP $\mathfrak{P}$ pertaining informational entities may be variously distributed through the interconnection network $\mathfrak{S}_{network}$ of machine architecture $\mathfrak{A}_{arc\_IM}$. System mechanisms for sensing certain IP involving and concerning information must be available for ac-

tive informational programs. On the other side, IM system must enable the arising of informational programs through suitable counterinforming and information embedding mechanisms. In the first step of development, these mechanisms could be algorithmic, using methods of deterministic chaos, fuzziness, probability, randomness, etc., but in the second step, they may become creative, unforeseeable, spontaneous, natural, etc. in the sense of Being-in-the-world.

In its initial state, an informational program can be comprehended as a text. In Latin, *textum* or *textus* (from the verb *texo* with the meaning to weave, plait; to frame, construct, fabricate, make, compose) means web, texture, stuff, structure, construction, connection, which is a product of artificial weaving, plaiting, composing, etc. A text is not only a connection of words, but also various informational interweavement. The meaning of a text is a result of the text understanding and pertains to the arising semantic interpretation of the informationally involved understanding entity. Thus, a text is in no way a torpid (rigid) depiction of information. It would be true only in case of a data form which stays for itself and does not concern informing entities. In this case, a text would not be perceived or even observed by a text understanding entity. Thus, a text is an informational realm of its context and its contents, a landscape of words, word groups and their distributed meaning in which the informational spirit finds its arising dwelling, understanding, informational phenomenality. In a later state, a text develops informationally and this state can be compared to the understanding of text by a text observer (reader, comprehender). Thus, in this stage of development, a text becomes a parallel structure of its different interpretations, which all form a complex and an arising structure of text where parallel meaning, semantics, contents, intention, significance, etc. are coming to the informational surface.

Discussing informational properties of a text expressed in a natural language, we can determine the difference existing between a computer program and a text on one side and between a text and an informational program on the other side. There exists an informational hierarchy concerning the triplet program—text—informational program in the following sense:

(1) A computer program or mathematical formula (or formula system) is a rationalistically bounded (artificial, abstract) informational entity with a unique, unambiguous, fixed meaning for programmers, mathematicians, and machines, respectively. Understanding of programs or mathematical formulas is not a subject of these entities themselves; it is a process within human behavior (mathematician or programmer community) or machine's semantic routine. Programs and mathematical formulas belong to executively automatic, semantically unique, logically uncontroversial, or rationally understandable structures, delivering safe, predictable, and anticipated results. The informing among entities (operands, operators, relations, functions) inside a program or formula is well-defined by abstract data declarations, definition equivalences, algorithms, and other unique rules.

(2) A text in a natural language is in principle metaphoric, ununiquely structured in the meaning of its components (words and word groups) and in the semantic interconnection possibilities among its components. However, in its initial state, a text is the written, also recorded (stored) or transmitted (broadcasted) entity which as the text does not change its structure. The understanding entity of a text is, for instance, its reader, hearer, or seer who gives the text additional metaphoricalness, ambiguous meaning, oblique discursiveness, that is, arising understanding. Text is a higher informational structure in comparison to a program or informational formula; from the standpoint of the observer, it carries, causes, and informs creative (intelligent) information. This type of informing is not allowed in case of rationalistic information which is artificially bounded, logically and meaningly closed to a certain tautological structure.

(3) An informational program (formula or formula system) is simultaneously (in parallel) rationalistically structured in its symbolic expressiveness, keeps the informational metaphoricalness and a degree of informational indefiniteness for its constituting entities, possesses its own possibilities of arising (text changing, text developing) component and its own understanding (consciousness, intelligence). While programs and texts do not change as entities in the sense of informational phenomenality, infor-

mational programs arise, develop, change, and vanish through the course of their informing. We may say that programs and texts inform passively, while informational programs inform actively. Within this reality, the informing power of informational programs informing in an IM becomes superior to computer programs and texts because of its own informational arising and, maybe, intelligent development.

We are able to give the following additional comparisons:

(i) A program or mathematical formula is like a discrete black-white landscape (deserted, unique, once for all determined dot structure), where black dots play the role of logical significance on the white background.

(ii) A text is a written piece of paper, a stable picture (photo, depiction, well-known melody, aviso, a fixed sequence of words, sounds, pictures, etc.) which is laid down (written, recorded, fixed) as a foreseeable, predictable, repeatable structure.

(iii) An informational program is by itself a dynamic structure which is similar to a view to the live landscape in which entities exist, to an arising program which runs from one into another situation. It has its own understanding, interweavement of its items, and it changes lively, dynamically, but intentionally in the world of information. Within the triplet program—text—informational program, informational program stands at the highest possible place in the hierarchy of informational structure.

5. Operating System of IM

Operating system (OS) is a technological synonym for a system interaction of system programs and machine architecture within an IM. One of the basic, global questions pertaining to an IM is: What is the system support to informing entities being represented by informational operands, basic and composed formulas of operands and operators, and informational formula systems? Which are basic features of this kind offered by the OS of an IM? It is clear that some common properties of informing of entities can be »delivered« by OS to satisfy the request of efficient informing (counterinforming, embedding) of entities. Otherwise, a sophisticated mechanism of informing, counterinforming and embedding of information would be necessary for each occurring entity.

Let us join the OS of an IM in the following principles:

(1) OS of IM supports the informing of operands (including subformulas, formulas, and formula systems) in their own metaphysical nature, by which an entity representing operand informs, counterinforms, and embeds information which concerns it informationally.

(2) OS performs the searching of information pertaining to a particular operand.

(3) Operands marking original entities (simple or proper operands) have their informational definitions; their initial formulas are stored in the operand dictionary. In this dictionary also those operands which occur in the definition of proper operands are defined. In this way the operand dictionary is informationally closed (tautological).

(4) OS processes informational formulas and formula systems by means of understanding (translating, compiling, interpreting, etc.) informational language.

(5) OS enables the modification and the arising of initial informational formulas during the process of their informing and, within this process, supports their understanding after they have been informationally reshaped (arisen, modified, vanished).

(6) On the IM level, OS performs as an understanding entity of informational language. In this respect, an IM is the informational language machine (ILM).

(7) Of course, OS is adapted to the IM architecture and supports the informational massive processor parallelism together with the specific machine language pertaining to architecture processors (for instance, RISC) and also the specific control language of the IM architecture interconnection network.

(8) OS of IM is the informational interface between user's informational language and machine system language (processor and interconnection language). Processor interconnections on the machine level have their origin in operators of informational formulas, which connect particular operands.

(9) Formulas of a formula system are treated as parallel processes. This rule is extended to op-

erands and subformulas. OS manages the arising parallel situation of processing.

Operating system is a synonym for a system of system informational programs of IM, marked by $\mathfrak{P}_{sys\_IM}$, where

(23) $\mathfrak{P}_{sys,1}, \mathfrak{P}_{sys,2}, \ldots \subset \mathfrak{P}_{sys\_IM}$

OS $\mathfrak{P}_{sys\_IM}$ performs as an open informational entity possessing its own externalism, internalism, metaphysicalism, and phenomenalism. Operating system of an IM, $\mathfrak{P}_{sys\_IM}$, performs together with the IM architecture $\mathfrak{A}_{arc\_IM}$ in managing and informing of user informational programs.

How do IM architecture $\mathfrak{A}_{arc\_IM}$ and IM operating system $\mathfrak{P}_{sys\_IM}$ support an informing entity α informationally within an IM? Operand α as an informational phenomenon is confronted with the triplet of its informing, counterinforming, and embedding of information. In a narrower sense, informing of an entity embraces, in general, its informational broadening, modification, shortening, etc. which are a consequence of the entity's interpretive scope. On the linguistic level, a typical scope of informing represent the so-called synonyms of words or word groups. However, there are many other informational equivalents to occurring informational items, for instance, of parallel interpretive character. The consequence of this informational possibilities concerning an informing operand is its informational broadening which includes modification, informational shortening, vanishing and of course arising. If by $\mathfrak{Z}(\alpha)$ the informing of entity α is marked, the following initial system of IM architecture and operating system support to informing of operand α can be given:

(24) *System support to informing of operand α:*
$(\mathfrak{A}_{arc\_IM}, \mathfrak{P}_{sys\_IM} \models_{synonymize} \alpha, \mathfrak{Z}(\alpha))$
$\models_{produce} \alpha, \mathfrak{Z}(\alpha)$;
$(\mathfrak{A}_{arc\_IM}, \mathfrak{P}_{sys\_IM} \models_{interpret} \alpha, \mathfrak{Z}(\alpha))$
$\models_{produce} \alpha, \mathfrak{Z}(\alpha)$;
$(\mathfrak{A}_{arc\_IM}, \mathfrak{P}_{sys\_IM} \models_{broaden} \alpha, \mathfrak{Z}(\alpha))$
$\models_{produce} \alpha, \mathfrak{Z}(\alpha)$

The occurring informational operators have the following meaning:

$\models_{broaden}$ broaden(s)_the_scope_of,
inform(s)_in_an_informationally_
_broadened_way_to;
$\models_{interpret}$ interpret(s),
inform(s)_interpretatively_to,
deliver(s)_interpretation(s)_of;
$\models_{produce}$ produce(s), embed(s)_in;
$\models_{synonymize}$
synonymize(s),
give(s)_synonyms_for,
inform(s)_synonymously_in_
_respect_to,
inform(s)_by_synonyms_of,
inform)s)_in_a_synonymous_way_to

In formula (24), a possibility for shortening its formal expression arises when operational alternatives come to the surface. We can introduce the so-called alternative operator structure with the meaning operator1 or operator2 or operator3 or .... The Or is marked by symbol |. Thus,

(24') $(\mathfrak{A}_{arc\_IM}, \mathfrak{P}_{sys\_IM}$
$\models_{synonymize} | \models_{interpret} | \models_{broaden}$
$\alpha, \mathfrak{Z}(\alpha))$
$\models_{produce} \alpha, \mathfrak{Z}(\alpha)$

The second mode of system support to an informing entity is counterinformational. We can investigate principles of counterinforming from a characteristically pragmatical point of view. The adverb *counter* pertaining to conterinformation and counterinforming has several meanings: for instance, in the opposite informational way, contrary the course of information or informing, in the reverse informational direction, contrary in an informational way, in opposition to the ruling information or informing, in the sense to inform counter or against to the informing rules. The adjective *counterinformational* means informationally opposite, opposed, and contrary. Counterinformation informs opposite or contrary to information from which it arises. Counterinforming means to act in opposition to informing, also to frustrate informing by counterinforming. Thus, counterinforming could mean to produce antonymous information (to »antonymize«) in comparison to informing by which synonymous, meaningly broadened information is produced.

If by $\gamma(\alpha)$ and $\mathfrak{C}(\alpha)$ the counterinformation and counterinforming of entity α is marked, respectively, the following initial system of IM architecture and operating system support to counterinforming of operand α can be expressed:

(25) *System support to counterinforming of operand α:*

$(\mathfrak{A}_{arc\_IM}, \mathfrak{P}_{sys\_IM} \models_{antonymize} \gamma(\alpha), \mathfrak{C}(\alpha)) \models_{produce} \gamma(\alpha), \mathfrak{C}(\alpha);$
$(\mathfrak{A}_{arc\_IM}, \mathfrak{P}_{sys\_IM} \models_{counterinterpret} \gamma(\alpha), \mathfrak{C}(\alpha)) \models_{produce} \gamma(\alpha), \mathfrak{C}(\alpha);$
$(\mathfrak{A}_{arc\_IM}, \mathfrak{P}_{sys\_IM} \models_{counterbroaden} \gamma(\alpha), \mathfrak{C}(\alpha)) \models_{produce} \gamma(\alpha), \mathfrak{C}(\alpha)$

The occurring counterinformational operators have the following meaning:

$\models_{antonymize}$
antonymize(s),
give(s)_antonyms_for,
inform(s)_antonymously_in_
_respect_to,
inform(s)_by_antonyms_of,
inform(s)_in_a_antonymous_way_to;

$\models_{counterbroaden}$
broaden(s)_the_scope_contrary_to,
counterinform(s)_in_an_
_informationally_broadened_way_to;

$\models_{counterinterpret}$
counterinterpret(s),
counterinform(s)_interpretatively_to,
deliver(s)_counterinterpretation(s)_of;

$\models_{produce}$ produce(s), embed(s)_in

In formula (25), the possibility for shortening its formal expression is given:

(25') $(\mathfrak{A}_{arc\_IM}, \mathfrak{P}_{sys\_IM} \models_{antonymize} | \models_{counterinterpret} | \models_{counterbroaden} \gamma(\alpha), \mathfrak{C}(\alpha)) \models_{produce} \gamma(\alpha), \mathfrak{C}(\alpha)$

The third mode of system support to an informing entity is the embedding of new (arisen and arrived) information. New information is informationally broadened (informed), counterinformational and that which arrives to the informing entity as something informationally observed. How is the embedding of information taking place? A piece of information can be embedded by means of the embedding information by which the autonomous informed (broadened) information, counterinformation or input information is informationally connected to the informing entity. Embedding information is to the informing entity belonging connection information and its informing is called embedding.

We can distinguish three kinds of embedding information in respect to the informing entity (operand α), namely, informed information embedding information $\varepsilon(\alpha_{informed})$, counterinformation embedding information $\varepsilon(\gamma(\alpha))$ and input (arriving) information embedding information $\varepsilon(\iota_{input})$ and the corresponding embeddings, that is, informed embedding $\mathfrak{E}(\alpha_{informed})$, counterinformational embedding $\mathfrak{E}(\gamma(\alpha))$, and input (arriving) information embedding $\mathfrak{E}(\iota_{input})$. So, the following initial system of IM architecture and operating system support to the embedding phenomenality of operand α can be expressed:

(26) *System support to informational embedding within operand α:*

$\mathfrak{A}_{arc\_IM}, \mathfrak{P}_{sys\_IM} \models_{produce} (\varepsilon(\alpha_{informed}), \mathfrak{E}(\alpha_{informed}), \varepsilon(\gamma(\alpha)), \mathfrak{E}(\gamma(\alpha)), \varepsilon(\iota_{input}), \mathfrak{E}(\iota_{input}));$
$(\varepsilon \models_{embed} \alpha_{informed}, \gamma(\alpha), \iota_{input}) \models_{into} \alpha$

Informed information is that delivered by informing of α, that is, $\alpha_{synonymous}$, $\alpha_{interpretive}$, and $\alpha_{broadened}$, produced by formula (24).

Except formulas (24), (25), and (26), other scenarios of system support to informing of informational entities can be studied. However, the general concept of informing, counterinforming, and embedding information pertaining to informing of informational operands, seems to be the most general and can be particularized in each specific case. To the three modes of informing another very important informational entity must be available, called the dictionary of informational operands, marked by $\delta_{dictionary}$.

In the framework of IM operating system support $\mathfrak{P}_{sys\_IM}$, the dictionary of operands is a data base in which initial informational formulas of operands, a kind of basic informational definitions, are stored. In a formula, the informational operand is only the marker of an informational

entity, which is informationally connected with other operands in different formulas of a formula system. Thus, to each in formulas occurring operand, its definition from the dictionary is associated. In this operand definition formula or system of formulas, other operand occur, which have to be considered in the same way: they must be found in the dictionary too.

Informing of an entity, that is, informational operand, must consider the entire available informational realm. This realm concerns, for instance, the entire natural language, if an IM operates in the domain of language informing. In this way, for each informational operand, for instance, a word or word group, a complex formula system comes to existence, until all occurring operands become metaphysically closed, that is, until no informational operand is hanging non-circularly closed into itself. On this level, we are able to recognize that every informational concept of a language is tautological in principle. There is no way out of informational tautology, language tautology or system tautology: information remains circled into information.

The longer the metaphysical circle of an operand formula, the greater the number of operand concerning parallel interpretations, the more natural appears the informational meaning of the operand. For instance, there are words, for which meaning we never ask, because their explanation, interpretation, and understanding through sentences in which other words appear would be to complex, unclear, sophisticated, and understood intuitively, as we hope, only in the context with other words.

The future IM will include a complete informational dictionary of possible operands $\delta_{dictionary}$ as a standard part of IM operating system. In this way, the system support to informing operands, their informational arising, will be always possible. New operands will be added to the dictionary after their informational definition. Thus, the time will come when the so- called electronic dictionaries will be used in their informational form as standard parts of IM operating systems. One of the conditions to offer the informing support to an informational operand α will be

(27) $\alpha \subset \delta_{dictionary}$

Strategies for informing, counterinforming, and embedding concerning α will be developed as principles of informing of entities based on informational formulas (24), (25), and (26). Besides these system support strategies, an informing entity α can have its own, for instance, intentional, goal-directed formulas for informing, counterinforming, and embedding of information, that is, its internal possibilities of informational arising. In this sense, to formulas (24), (25), and (26) similar formulas for α's own informing, counterinforming, and embedding can come into existence.

6. Informational Machine as Informational Entity

Through the conceptualism of the previous sections, we gradually approach to the structure of an informational machine which performs as a regular informational entity. In future, the evolvable architecture (software configurable hardware) of IM will become quite possible. On the other hand, the so-called genetic programming technique (for instance, evolution of Lisp programs) is on the way to the question how to evolve self-reproducing systems. The other question is how to evolve self-reproducing systems which can also do something useful. These are the promising perspectives of nanotechnology in space (matter) and time. For instance, L. Buss and W. Fontana presented a mathematical theory of the development of systems towards life like behavior [ACR]. They represented levels of O, 1, and 2, with increasing sophistication, e.g. self replication, then genomes, etc.

In the previous sections, we identified several informational entities which constitute the informational functioning of an IM $\mathfrak{M}$. At the first look, such a machine is characterized by an informational structure of the global, two level informational entities

(28) $\mathfrak{M}$: $\mathfrak{A}_{arc\_IM}$, $\mathfrak{P}_{sys\_IM}$, $\delta_{dictionary}$,
$\mathfrak{P}_{use\_IM}$, ι_{use_IM}

where in the first line are the so-called machine operational entities meaning IM architecture, IM operating system, and operand dictionary, respectively, while in the second line are IM user systems

of informational programs and IM user information, respectively. Within machine $\mathfrak{M}$, these entities inform mutually or interact in different informational ways. Further, we have slightly pointed out the IM architecture as

(29) $\mathfrak{A}_{arc\_IM}$: $\pi_1, \pi_2, \ldots \subset \mathfrak{S}_{network}$;
$\mathfrak{S}_{network} \subset \mathfrak{A}_{arc\_IM}$

where sequence $\pi_1, \pi_2, \ldots$ marks the uniformly functionally and in an interconnecting way designed microprocessors. The IM interconnection network is in principle evolvable, at least program or information configurable, however in principle informationally arising.

The next essential part of IM belongs to the realm of system informational programs $\mathfrak{P}_{sys\_IM}$, called the IM operating system. This system controls the functioning of IM and offers the so-called basic informational support to informational operands, which occur in the formulas of user informational programs, marked by $\mathfrak{P}_{use,1}$, $\mathfrak{P}_{use,2}$, ..., shaping the user informational system $\mathfrak{P}_{use\_IM}$ of machine $\mathfrak{M}$. Thus, the system and user informational program systems together with the system of other user information items are

(30) $\mathfrak{P}_{sys,1}, \mathfrak{P}_{sys,2}, \ldots \subset \mathfrak{P}_{sys\_IM}$;
$\mathfrak{P}_{use,1}, \mathfrak{P}_{use,2} \ldots \subset \mathfrak{P}_{use\_IM}$
$\iota_{use,1}, \iota_{use,2}, \ldots \subset \iota_{use\_IM}$

An essential part of IM is the structure of system and user open informational operand dictionary $\delta_{dictionary}$ in which the initial informational definitions of system and informational user program operands are stored. This dictionary represents a semantic net of entities for the entire realm of an informational field, discipline, application, science, etc. An arbitrary net for a distinct operand must be informationally circled in a complete way. It means that for each occurring operand a complete or informationally well-circled informational system of formulas exists. If these entity operands of metaphysically structured parallel formulas are marked by α_i, the dictionary structure is

(31) $\alpha_1, \alpha_2, \ldots \subset \delta_{dictionary}$

At last the peripherals have to be considered in which input and output information is stored, that is, system and user informational programs, other informational entities, and results of IM informing. Thus, a general peripheral situation is

(32) $\mathfrak{P}_{sys\_IM}, \mathfrak{P}_{use\_IM}, \iota_{use\_IM}, \delta_{dictionary}, \rho_{result\_IM} \subset \mathfrak{P}_{peripheral\_IM}$

Under these conditions the metaphysicalism of an IM $\mathfrak{M} \models \mathfrak{M}$ can be observed as a general informational scheme in the form

(33) $\mathfrak{P}_{peripheral} \models$
$((((\mathfrak{P}_{use\_IM}, \iota_{use\_IM} \models \mathfrak{P}_{sys\_IM},$
$\delta_{dictionary}) \models$
$((\pi_1, \pi_2, \ldots \subset \mathfrak{S}_{network})$
$\subset \mathfrak{A}_{arc\_IM})) \models$
$\mathfrak{P}_{use\_IM}, \iota_{use\_IM}) \models$
$(\rho_{result\_IM} \subset \mathfrak{P}_{peripheral}))$

The last formula can now be decomposed, parallelized, etc. in more details. The outmost cycle is peripheral in which the initial informational programs and information is stored. Results of informing of IM are transferred to peripherals, so they can be observed by the IM user.

7. Informing and Informedness of IM

Informing and informedness of an IM constitute its phenomenalism. We say that an observer of IM as its user is impressed by IM informing and concludes what an informational phenomenalism of the machine could be, for instance, informationally efficient, comprehensive and, last but not least, intelligent. The last could be true especially because of the informational character of the machine when significant realms of information are taken into machines informing, that is, informationally arising processing. On the other hand, an IM can be informed by an observer as its user, and this informedness is in no way limited in advance, so the entire user informational creativity comes into the game of the machine informedness.

Because IM $\mathfrak{M}$ is no more as a machine, the questions concerning its externalism, internalism, metaphysicalism, and phenomenalism could be of particular importance. Usually, such questions are not customary for machines as rational, artificial entities which have their unique, determined purpose, functional ability, and applicability. How-

ever, this is only a common belief and the question of, for instance, machine's metaphysics remains entirely righteous.

The function of an IM hides the machine informing, counterinforming, and embedding of information. This function concerns as well the informing of operands occurring in informational programs which informing is partly supported by the IM system. In this sense, an IM becomes an informational phenomenon by itself.

7.1. IM Externalism

IM $\mathfrak{M}$ informs its exterior. Machine externalism ($\mathfrak{M} \models$) is important for the informedness of machine observers, users, and other machines. How does a machine $\mathfrak{M}$ inform according to its architecture, operating system, dictionary, user informational programs and user information?

An observer or user of IM is able to recognize several general machine properties which can be expressed verbally and by informational language. Let us see some of them.

(1) *External informing of* $\mathfrak{M}$. IM $\mathfrak{M}$ informs in parallel ways. One of the basic implication system of machine externalism is

$$(34) \quad (\mathfrak{M} \models) \Rightarrow (\mathfrak{M} \models\!\!\models); \quad (\mathfrak{M} \models\!\!\models) \Rightarrow (\mathfrak{M} \models; \mathfrak{M} \models; \ldots)$$

where operator $\models\!\!\models$ marks parallel informing. According to formula system (34), external parallelism of $\mathfrak{M}$ is circular and the consequence of it is

$$(34') \quad (\mathfrak{M} \models) \Rightarrow (\mathfrak{M} \models; \mathfrak{M} \models; \ldots)$$

This implication shows the parallel possibilities of informing of $\mathfrak{M}$ to external entities, where $\mathfrak{M}$'s external informational openness is manifested as parallel informing.

Parallelism of $\mathfrak{M}$ belongs to the most powerful, but theoretically unmasterable realm of informing under spontaneous and circular circumstances. For instance, problems of understanding and through it produced meaning of something may remain unmastered in several informational ways, that is, philosophically, theoretically, formally, linguistically, designingly, technologically, communicatively, connectively, etc. But, this is also true for any other informational parallelism. The phenomenon of parallel externalism belongs to the most unrevealed and simultaneously theoretically pretentious problems. On the other hand, the parallelism of mind concerning associative processes at the arising of thought seems to be natural. In case of $\mathfrak{M} \models\!\!\models$, machine $\mathfrak{M}$ can produce intentional, spontaneous, circular, etc. parallelism, possessing its own mechanisms of informing, creating of operand dictionary, user informational programs and user information. This kind of parallelism may look to be unforeseeable and not well-predictable, that is, informationally spontaneous for the IM observer or user.

(2) *External metaphysicalism of* $\mathfrak{M}$. IM $\mathfrak{M}$ informs in its own circular way. This kind of circularity is called *external metaphysicalism* (also metaphysics) of $\mathfrak{M}$. The machine observer or user copes with the metaphysical implication of the form

$$(35) \quad (\mathfrak{M} \models) \Rightarrow ((\mathfrak{M} \models \mathfrak{M}) \models)$$

concerning not only the general machine externalism $\mathfrak{M} \models$, but also the machine metaphysical externalism $(\mathfrak{M} \models \mathfrak{M}) \models$. The phenomenon of observing the machine metaphysicalism by machine users is a parallel informational phenomenon, where

$$(35') \quad ((\mathfrak{M} \models \mathfrak{M}) \models) \Rightarrow ((\mathfrak{M} \models \mathfrak{M}) \models\!\!\models); \quad ((\mathfrak{M} \models \mathfrak{M}) \models\!\!\models) \Rightarrow ((\mathfrak{M} \models \mathfrak{M}) \models; \quad (\mathfrak{M} \models \mathfrak{M}) \models; \ldots)$$

In this way, the observed metaphysicalism of $\mathfrak{M}$ is in no way unique and unambiguous since the arising metaphysicalism of $\mathfrak{M}$ shows the arisen phenomena to machine observers.

Within the machine metaphysicalism, informational formulas are circularly decomposed, parallelized, composed, etc. and through this phenomenality the observing of machine metaphysics $\mathfrak{M} \models \mathfrak{M}$ through $(\mathfrak{M} \models \mathfrak{M}) \models$ becomes possible.

(3) *External resulting of* $\mathfrak{M}$. Machine $\mathfrak{M}$ delivers results which may be expected, but also unforeseeable, unpredictable, and arising from a complex informational realm. The question of result ρ_{result_IM} is senseful because IM informs and produces essential changes of informational entit-

ies through their informational arising. Under such circumstances, there are dedicated results in the form of informational meanings concerning various kinds of information, for instance, linguistic, numeric, logical, formula-informational, signaling, controlling, etc. Results can be stored, displayed, or otherwise communicated, but they can also remain unobserved and lost in the process of informing. Thus, the user or observer of IM must determine, at least roughly, for which kind of results there exists an interest in distinct informational programs and how the informing machine $\mathfrak{M}$ may transfer its results to the peripherals, that is,

(36) $(\mathfrak{M} \models \rho_{result\_IM}) \models_{to} \mathfrak{P}_{peripheral}$

Because of the possibilities of intelligent informational programs, these programs (informational formula systems) can decide by themselves how to inform results, store them, or present to the exterior of IM. Thus,

(37) $(\mathfrak{M} \models) \Rightarrow ((\mathfrak{M} \models \rho_{result\_IM}) \models)$

is conditio sine qua non that the informing of $\mathfrak{M}$ is (becomes) informationally senseful. Otherwise, $\mathfrak{M}$ would perform as a closed system, which does not concern the machine exterior.

(4) From the pragmatical point of view, machine externalism $\mathfrak{M} \models$ is able to explicate not only the discussed forms of parallelism, metaphysicalism, or informational resulting (productivity, conclusiveness), but other, very specific phenomenal forms, exporting informationally dependent phenomena of understanding, managing, controlling, signaling, modeling, etc. in cooperation with other systems, machines, observers, users, and machine environment. In this way, an IM may be informationally embedded in other systems not only via its externalism, but also internalism, building up a composed, metaphysically structured informational system.

7.2. IM Internalism

Machine internalism $\models \mathfrak{M}$ characterizes the machine informational sensibility (informedness) for informing of informational programs, requests, and information coming from machine users, observers, and other informational systems, machines, environment, etc. How is machine $\mathfrak{M}$ informed according to its architecture, operating system, dictionary, user informational programs, and user information? Let us show some informational phenomena concerning the IM internalism.

(1) *Internal informedness of* $\mathfrak{M}$. IM $\mathfrak{M}$ is informed in parallel ways. One of the basic implication system of machine internalism (informedness) is

(38) $(\models \mathfrak{M}) \Rightarrow (\Vdash \mathfrak{M})$;
$(\Vdash \mathfrak{M}) \Rightarrow (\models \mathfrak{M}; \models \mathfrak{M}; \ldots)$

where operator $\Vdash$ marks parallel informing. According to formula system (38), internal parallelism of $\mathfrak{M}$ is circular and the consequence of it is

(38') $(\models \mathfrak{M}) \Rightarrow (\models \mathfrak{M}; \models \mathfrak{M}; \ldots)$

This implication shows the parallel possibilities of informedness of $\mathfrak{M}$ by external entities, where $\mathfrak{M}$'s internal informational openness is manifested as parallel informedness.

Parallel informedness of $\mathfrak{M}$ belongs to the most powerful, yet theoretically unmasterable realm of informedness under spontaneous and circular conditions. For instance, problems of machine understanding and through it produced meaning of something informational may remain unmastered in several informational ways, that is, philosophically, theoretically, formally, linguistically, designingly, technologically, communicatively, connectively, etc.

(2) *Internal metaphysicalism of* $\mathfrak{M}$. IM $\mathfrak{M}$ is informed in its own circular way. This kind of circularity is called *internal metaphysicalism* of $\mathfrak{M}$. The machine as an informing entity is confronted with the metaphysical implication of the form

(39) $(\models \mathfrak{M}) \Rightarrow (\models (\mathfrak{M} \models \mathfrak{M}))$

concerning not only the general machine internalism $\models \mathfrak{M}$, but also the machine metaphysical internalism $\models (\mathfrak{M} \models \mathfrak{M})$. In this case, machine $\mathfrak{M}$ becomes the observer of parallel possibilities in the form of its metaphysicalism, where

(39') $(\models (\mathfrak{M} \models \mathfrak{M})) \Rightarrow (\Vdash (\mathfrak{M} \models \mathfrak{M}))$;
$(\Vdash (\mathfrak{M} \models \mathfrak{M})) \Rightarrow$
$(\models (\mathfrak{M} \models \mathfrak{M}); \models (\mathfrak{M} \models \mathfrak{M}); \ldots)$

As we see, the observing metaphysicalism of $\mathfrak{M}$ is in no way unique and unambiguous since machine metaphysicalism is able to observe the the arisen phenomena in its environment and within itself.

Within the machine internal metaphysicalism, informational formulas are decomposed, parallelized, composed, etc. and through this phenomenality the observing machine metaphysics $\mathfrak{M} \models \mathfrak{M}$ through $\models (\mathfrak{M} \models \mathfrak{M})$ becomes possible.

(3) *Internal resulting (resultedness) of* $\mathfrak{M}$. Machine $\mathfrak{M}$ creates results internally according to informational programs and by its own informational characteristics (states, processes). Results may be expected, but also unforeseeable, unpredictable, controversial, and arising from a complex informational realm. The question of result ρ_{result_IM} is senseful because IM informs and produces essential changes of informational entities through their informational arising. How does the IM internalism, $\models \mathfrak{M}$, concern the IM resulting, that is, results ρ_{result_IM}? As IM informs and is simultaneously informed, the result of machine informedness may be any informational change within occurring informational operands. Various informational meanings may arise pertaining to linguistic, numeric, video, signal, and other information. The arisen potential results ρ_{result_IM} inform $\mathfrak{M}$ and within $\mathfrak{M}$'s informedness the decision takes place which information will be considered, transferred, and informed as result significant. One has

$$(40) \qquad (\rho_{result\_IM} \models \mathfrak{M}) \models \mathfrak{P}_{peripheral}$$

Within IM processed informational programs can decide by themselves (in an intelligent way) how and when to form interior results. Thus,

$$(41) \qquad (\models \mathfrak{M}) \Rightarrow (\models (\rho_{result\_IM} \models \mathfrak{M}))$$

The openness of $\mathfrak{M}$'s informedness guarantees the possibilities of choice, to obtain information which pertains to ρ_{result_IM} also from machine peripherals, so,

$$(42) \qquad \mathfrak{P}_{peripheral} \models (\rho_{result\_IM} \models \mathfrak{M})$$

By this discussion, the mutually reversing processes of machine resulting informing and resulting informedness can be understood and decomposed informationally to the necessary details.

7.3. IM Metaphysicalism

Machine metaphysicalism $\mathfrak{M} \models \mathfrak{M}$ is a particular case of machine phenomenalism constituted as machine externalism and machine internalism, that is, as a system $(\mathfrak{M} \models; \models \mathfrak{M})$. In this sense, IM metaphysicalism is hidden in machine phenomenalism or $(\mathfrak{M} \models \mathfrak{M}) \subset (\mathfrak{M} \models; \models \mathfrak{M})$. In general, machine metaphysicalism is an unrevealed phenomenon for the machine observer or user ω and can be observed only through the machine informing $\mathfrak{M} \models$ and the observer's informedness $\models \omega$, that is, $\mathfrak{M} \models \omega$.

(1) *General metaphysicalism of* $\mathfrak{M}$. Let us remind of the informational structure of $\mathfrak{M}$ which marks informationally interweaved entities, that is,

$$(43) \qquad \mathfrak{M} \models_{mark} (\mathfrak{A}_{arc\_IM}, \mathfrak{P}_{sys\_IM}, \delta_{dictionary}, \mathfrak{P}_{peripheral})$$

where

$$(44) \qquad \begin{array}{l} \pi_1, \pi_2, \dots, \mathfrak{S}_{network} \subset \mathfrak{A}_{arc\_IM}; \\ \mathfrak{P}_{sys,1}, \mathfrak{P}_{sys,2}, \dots, \iota_{sys} \subset \mathfrak{P}_{sys\_IM}; \\ \varepsilon_1, \varepsilon_2, \dots \subset \delta_{dictionary}; \\ \mathfrak{P}_{use,1}, \mathfrak{P}_{use,2}, \dots \subset \mathfrak{P}_{use\_IM}; \\ \iota_{use,1}, \iota_{use,2}, \dots \subset \iota_{use\_IM}; \\ \mathfrak{P}_{sys\_IM}, \mathfrak{P}_{use\_IM}, \iota_{use\_IM} \subset \mathfrak{P}_{peripheral} \end{array}$$

Within machine metaphysicalism $\mathfrak{M} \models \mathfrak{M}$, all metaphysical cycles of $\mathfrak{M}$'s constituting entities (components) are possible and can be deduced from the basic machine metaphysical cycle, that is,

$$(45) \qquad \begin{array}{l} (((\mathfrak{M} \models \mathfrak{P}_{peripheral})) \models \\ \quad (\mathfrak{A}_{arc\_IM}, \mathfrak{P}_{sys\_IM}, \delta_{dictionary})) \models \\ \quad (\mathfrak{P}_{use\_IM}, \iota_{use\_IM})) \models \mathfrak{M} \end{array}$$

where $\mathfrak{M}$ is represented by the informational structure in formula (43).

(2) *Metaphysical parallelism of* $\mathfrak{M}$. IM $\mathfrak{M}$ informs in metaphysically parallel ways. For, instance, machine metaphysicalism $\mathfrak{M} \models \mathfrak{M}$ is able to inform by keeping and arising of metaphysical (interiorly circular) processes in unforeseeable parallel ways. So,

(46) $(\mathfrak{M} \models \mathfrak{M}) \Rightarrow (\mathfrak{M} \Vdash \mathfrak{M})$;
$(\mathfrak{M} \Vdash \mathfrak{M}) \Rightarrow ((\mathfrak{M} \models \mathfrak{M}); (\mathfrak{M} \models \mathfrak{M}); \ldots)$

where operator $\Vdash$ is the shortcut for parallel informing. In this situation, as the consequence of the last formula, machine $\mathfrak{M}$ copes with the interior circular parallelism in the form of different metaphysical processes, that is,

(46') $(\mathfrak{M} \models \mathfrak{M}) \Rightarrow ((\mathfrak{M} \models \mathfrak{M}); (\mathfrak{M} \models \mathfrak{M}); \ldots)$

This implication shows how machine metaphysics $\mathfrak{M} \models \mathfrak{M}$ develops in different (parallel) ways. Within machine $\mathfrak{M}$, parallel informational processes arise and inform various, interweaved information simultaneously.

(3) *Metaphysical externalism of* $\mathfrak{M}$. We have to draw a clear distinction between the external metaphysicalism (35) and metaphysical externalism. The adjective *external* in external metaphysicalism is understood as a predicate (property) of $\mathfrak{M} \models \mathfrak{M}$, for instance, as

$$(\mathfrak{M} \models \mathfrak{M}) \models_{\text{external}}$$

What does the metaphysical externalism mean after that? We have the following basic implication concerning the machine metaphysics $\mathfrak{M} \models \mathfrak{M}$:

(47) $(\mathfrak{M} \models \mathfrak{M}) \Rightarrow (((\mathfrak{M} \models) \models \mathfrak{M});$
$(\mathfrak{M} \models (\mathfrak{M} \models)); ((\mathfrak{M} \models) \models (\mathfrak{M} \models)))$

There is a control of $\mathfrak{M}$'s externalism by $\mathfrak{M}$ itself (metaphysical control). The shortcut of this situation would be, for instance,

$$(\mathfrak{M} \models) \models_{\text{metaphysical}}$$

where the adjective *metaphysical* is understood as an interior predicate of machine externalism $\mathfrak{M} \models$.

(4) *Metaphysical internalism of* $\mathfrak{M}$. Similarly, one can determine the metaphysical internalism in contrary to the internal metaphysicalism discussed in formula (39). The adjective *internal* in internal metaphysicalism is understood as a predicate of entity $\mathfrak{M} \models \mathfrak{M}$, that is,

$$\models_{\text{internal}} (\mathfrak{M} \models \mathfrak{M})$$

We have the following implication concerning the metaphysical internalism:

(48) $(\mathfrak{M} \models \mathfrak{M}) \Rightarrow (((\models \mathfrak{M}) \models \mathfrak{M});$
$(\mathfrak{M} \models (\models \mathfrak{M})); ((\models \mathfrak{M}) \models (\models \mathfrak{M})))$

There is a control of $\mathfrak{M}$'s internalism by $\mathfrak{M}$ itself (metaphysical control). The shortcut of this situation would be, for instance,

$$\models_{\text{metaphysical}} (\models \mathfrak{M})$$

where the adjective *metaphysical* is understood as an interior predicate of machine internalism $\models \mathfrak{M}$.

(5) *Metaphysical metaphysicalism of* $\mathfrak{M}$. Among other informational phenomenalities, metaphysicalism aims at consciousness as informational entity; and metaphysicalism of metaphysicalism alludes at self-consciousness. An informationally sophisticated IM should possess the informational power which through time would succeed or even essentially surpass the informational power of a living actor because of possibilities to use extensive, informationally unlimited realm of referencing.

In the informing and self-observing of metaphysicalism $\mathfrak{M} \models \mathfrak{M}$, machine $\mathfrak{M}$ decomposes $\mathfrak{M} \models \mathfrak{M}$ in different (parallel) ways. Thus,

(49) $(\mathfrak{M} \models \mathfrak{M}) \Rightarrow$
$(((\mathfrak{M} \models \mathfrak{M}) \models \mathfrak{M}); (\mathfrak{M} \models (\mathfrak{M} \models \mathfrak{M}));$
$((\mathfrak{M} \models \mathfrak{M}) \models (\mathfrak{M} \models \mathfrak{M})))$

In $(\mathfrak{M} \models \mathfrak{M}) \models \mathfrak{M}$, machine $\mathfrak{M}$ is the observer of its metaphysicalism and, simultaneously, a circularly perplexed, informationally arising mechanism of informing, counterinforming, and embedding of information. This mode of informing may be called *conscious self-observing*. In case $\mathfrak{M} \models (\mathfrak{M} \models \mathfrak{M})$, metaphysicalism of $\mathfrak{M}$ observes the behavior of machine $\mathfrak{M}$, where $\mathfrak{M}$ informs $\mathfrak{M} \models \mathfrak{M}$ on its informational states (necessities and possibilities). This mode of informing is called *conscious self-informing*. The last case $(\mathfrak{M} \models \mathfrak{M}) \models (\mathfrak{M} \models \mathfrak{M})$ is the case of *recursive machine self-consciousness*. Each metaphysical machine cycle $\mathfrak{M} \models \mathfrak{M}$ can be decomposed by $\mathfrak{M}$ in different ways, offering a real realm of informational possibilities, unforeseeableness, uncertainty, etc. and, through this, a spontaneous

informational arising of entities (operands) within IM $\mathfrak{M}$.

(6) *Metaphysical production of results of* $\mathfrak{M}$. Few words have to be said in concern about possibilities of metaphysical production of results. Results is always significant information in a certain sense. Metaphysicalism of machine $\mathfrak{M}$ or of an informing entity in question decides which dedicated item of an informational phenomenon becomes a result because of its possible meaning, sense, significance, etc. Thus, there is not only the user who decides on the occurrence of a result, but metaphysical judgement of the informing entity (operand, formula) suddenly produces results and put them into a result index where they are available for other entities and users. Resulting entities are unforeseeable, unpredictable, informationally arising to a certain extent in their nature, kind, significance, occurrence, value, etc.

7.4. IM Phenomenalism

Machine phenomenalism $(\mathfrak{M} \models; \models \mathfrak{M})$ is the most general case of machine external informing and internal informedness. It would correspond to the term »machine as such« (in German, »die Maschine an sich«). Informational phenomenalism of machine $\mathfrak{M}$ says that machine as an informational entity cannot be revealed in its phenomenal entirety: because it is not possible to foresee completely its externalism which depends on machine observers; because an essential part of its internalism may be unrevealed to exterior observers and »known« only interiorly to the machine metaphysicalism; because its metaphysicalism or its interior circular informing is not directly informationally open to the machine exterior; because its phenomenalism is only a resultant phenomenon of machine externalism, internalism, and metaphysicalism. Certainly, these informational assumptions hold for any informational entity (operand, formula, phenomenon, thing, formation, etc.).

We have to remind the reader that machine phenomenalism is defined formally as a system $(\mathfrak{M} \models; \models \mathfrak{M})$ of informationally parallel, tied, and mutually impacting and impacted formulas, that is, $\mathfrak{M}$'s externalism $\mathfrak{M} \models$ and $\mathfrak{M}$'s internalism $\models \mathfrak{M}$. In this system, machine externalism and machine internalism are informationally interweaved, depended, metaphysically supported, etc. in a phenomenal (spontaneous, circular), entirely open way. As a system, machine phenomenalism is more than any of its components (externalism, internalism, metaphysicalism) and embraces these components, for instance, in the form

$$(50) \quad \begin{array}{l} (\mathfrak{M} \models) \subset (\mathfrak{M} \models; \models \mathfrak{M}); \\ (\models \mathfrak{M}) \subset (\mathfrak{M} \models; \models \mathfrak{M}); \\ (\mathfrak{M} \models \mathfrak{M}) \subset (\mathfrak{M} \models; \models \mathfrak{M}) \end{array}$$

After this introduction, let us overview some particular phenomenal cases of machine $\mathfrak{M}$.

(1) *Phenomenal externalism of* $\mathfrak{M}$. Several cases of phenomenal externalism of machine $\mathfrak{M}$ can be observed. How does a machine phenomenalism inform the machine exterior? The parallel external case is the following:

$$(51) \quad \begin{array}{l} ((\mathfrak{M} \models; \models \mathfrak{M}) \models) \Rightarrow ((\mathfrak{M} \models; \models \mathfrak{M}) \Vdash); \\ ((\mathfrak{M} \models; \models \mathfrak{M}) \Vdash) \Rightarrow (((\mathfrak{M} \models; \models \mathfrak{M}) \models)); \\ \quad ((\mathfrak{M} \models; \models \mathfrak{M}) \models); \ldots) \end{array}$$

where operator $\Vdash$ marks parallel informing. According to formula system (51), parallel phenomenal externalism can be expressed as

$$(51') \quad \begin{array}{l} ((\mathfrak{M} \models; \models \mathfrak{M}) \models) \Rightarrow (((\mathfrak{M} \models; \models \mathfrak{M}) \models); \\ \quad ((\mathfrak{M} \models; \models \mathfrak{M}) \models); \ldots) \end{array}$$

This implication shows the phenomenal parallel possibilities of external informing of $\mathfrak{M}$. The other methodically parallel case could be

$$(52) \quad ((\mathfrak{M} \models; \models \mathfrak{M}) \models) \Rightarrow ((\mathfrak{M} \Vdash; \Vdash \mathfrak{M}) \models)$$

or, in an extremely parallel form,

$$(53) \quad ((\mathfrak{M} \models; \models \mathfrak{M}) \models) \Rightarrow ((\mathfrak{M} \Vdash; \Vdash \mathfrak{M}) \Vdash)$$

The consequences of these implications can be derived easily.

(2) *Phenomenal internalism of* $\mathfrak{M}$. We may now repeat the questions and formulas pertaining to phenomenal externalism for phenomenal internalism. Thus, according to formulas (51) to (53), there is,

$$(54) \quad \begin{array}{l} (\models (\mathfrak{M} \models; \models \mathfrak{M})) \Rightarrow (\Vdash (\mathfrak{M} \models; \models \mathfrak{M})); \\ (\Vdash (\mathfrak{M} \models; \models \mathfrak{M})) \Rightarrow ((\models (\mathfrak{M} \models; \models \mathfrak{M})); \end{array}$$

$$(\models (\mathfrak{M} \models; \models \mathfrak{M})); \ldots)$$

where operator $\|\models$ marks parallel informing. According to formula system (54), parallel phenomenal internalism can be expressed as

$$(54') \quad (\models (\mathfrak{M} \models; \models \mathfrak{M})) \Rightarrow ((\models (\mathfrak{M} \models; \models \mathfrak{M})); (\models (\mathfrak{M} \models; \models \mathfrak{M})); \ldots)$$

This implication shows the phenomenal parallel possibilities of internal informing of $\mathfrak{M}$. The other methodically parallel case could be

$$(55) \quad (\models (\mathfrak{M} \models; \models \mathfrak{M})) \Rightarrow (\models (\mathfrak{M} \|\models; \|\models \mathfrak{M}))$$

or, in an extremely parallel case,

$$(56) \quad (\models (\mathfrak{M} \models; \models \mathfrak{M})) \Rightarrow (\|\models (\mathfrak{M} \|\models; \|\models \mathfrak{M}))$$

The consequences of these implications can be derived in a systematic way.

(3) *Phenomenal metaphysicalism of* $\mathfrak{M}$. Machine phenomenal metaphysicalism is at the background of machine's »consciousness« in the form of machine metaphysicalism and its phenomenalism. In this mode of informing, machine $\mathfrak{M}$ reaches the most complex phenomena of its externalism and internalism interweavement and interconnection, projecting the exterior and interior world into a global informational depiction of a distinguished informational realm. Through this spontaneous and circular phenomenology, machine $\mathfrak{M}$ becomes a part of the worldliness pertaining to its, instantaneously most significant informational realm. In this respect, according to informational terms, machine $\mathfrak{M}$ is able to behave as something in the world, with its own reflection, consciousness, and external and internal openness to the world. One may say that through metaphysically structured phenomenalism, IM $\mathfrak{M}$ reaches the state of the profound Being-in-the-World.

The metaphysical closing of machine phenomenalism, that is,

$$(57) \quad (\mathfrak{M} \models; \models \mathfrak{M}) \Rightarrow ((\mathfrak{M} \models; \models \mathfrak{M}) \models (\mathfrak{M} \models; \models \mathfrak{M}))$$

preserves the external and internal openness to the machine exterior and interior, that is, possibilities of machine functioning in the exterior and interior world.

(4) *Phenomenal phenomenalism of* $\mathfrak{M}$. Phenomenal phenomenalism means formally

$$(58) \quad (\mathfrak{M} \models; \models \mathfrak{M}) \Rightarrow (((\mathfrak{M} \models; \models \mathfrak{M}) \models); (\models (\mathfrak{M} \models; \models \mathfrak{M})))$$

which is nothing else than machine phenomenalism of machine phenomenalism. In short, as any other informational phenomenon, phenomenalism is recursive in the sense of phenomenal self-(re)production.

(5) *Phenomenal production of results of* $\mathfrak{M}$. Phenomenal production of results depends on machine particular modes of informing (externalism, internalism, metaphysicalism) and, in the last, the most complex informational consequence, on phenomenalism, which not only unites particular cases, but can also produce results outside of these particularities. Phenomenal production of results concerns the most general phenomenality of information (informing, counterinforming, embedding) in the framework of entities which govern and decide on the nature of results and their arising. This is the realm of IM productiveness as a specific view ('seeing') of the most significant informational entities.

8. Technological Perquisites of IM

IM requires a fundamentally new conceptualization and development in functional design and technology. It is essentially different from that what, by nowadays terms, is notionally understood as computer. For instance, processors in its interconnection network are new RISC type, at least 64-bit, high-performance, high-speed, electronic, and/or photonic microprocessors. The processor interconnection network requires informational elements concerning intention, significance, and other properties and predicates of informationally extensive, prompt, and sensitive criteria of information exchange. All this calls for a radically new architecture of IM. Operating system of IM must support the informing, counterinforming and embedding of information of machine and user program operands and its system programs must perform in informationally regular way. It must compile user informational programs written in

informational language. As a basic informational operand support a sufficiently extensive dictionary (similar to Japanese term of electronic dictionary, for instance in [OED, JWD, EWD, CD, BD, IMD]) must be available for free informing of informational operand entities, including the possibilities of sentence (formula, program) translation in other natural and formal languages. Japan & Co. may be the best suited background for the future IM development and production.

9. Cybernetic Perquisites of IM

Informing is certainly a cybernetic activity in which all kinds of known and arising creativity play their informationally phenomenal roles. The cybernetic is synonymous for the live, emerging, understanding, etc. as well as for the informational. The cybernetic comes from the conceptualism of managing and controlling machines, living beings, individuals and societies. In this traditional approach, not a specific accent is given to information as a phenomenon: information is simple data necessary for managing and controlling systems in an optimal, goal-directed, surviving, or other way. Consciousness of the informational is not present at the beginning of the cybernetic epoch, at least not in an informationally arising sense. But, only the consciousness of the informational triggers the request of rethinking of cybernetic phenomenality in a philosophic and symbolic theoretical, informational way. In this point of rethinking, informational and cybernetic phenomenalism come close to each other and can be examined from one direction or the other.

Informing and informedness of entities is cybernetic, but not only cybernetic. Cybernetics is a study of human control functions (one might say, more generally, informational or communicational functions) and of mechanical, electrical, and electronic systems designed to replace them. This determination was set at the very beginning of modern cybernetics (N. Wiener). Later on, cybernetics encountered the philosophical discourse as a scientific discipline studying dynamically self-regulating and self-organizing systems. Its boundary fields are system, control, information, communication, game theory, etc. Already Plato gave the word a general meaning (proceeding from the art of boat steering) in the sense of managing a state. As it has developed over the years, cybernetics has become multidisciplinary, involving engineering, computer science, artificial intelligence, psychology, biology, neural science, sociology, philosophy, etc.

10. Intelligent Informational Machine

When I assert my own belief that true intelligence requires consciousness I am implicitly suggesting (since I do not believe the *strong*-AI contention that the mere enaction of an algorithm would evoke consciousness) that intelligence cannot be properly simulated by algorithmic means, i.e. by a computer, in the sense that we use that term today. ... For I shall shortly argue strongly ... that there must be an essentially *non-algorithmic* ingredient in the action of consciousness.

—Roger Penrose [ENM] 407

In the full sense of the word, IIM is the goal of future machine development. Through times, the meaning of the adjective *intelligent* will more and more approach to the meaning of the »sufficiently complex and specific in informational way«. The main task of IIM will be the production of specific informational entities like meaning, sense, significance, intention, self-observation, conceiving, perceiving, consciousness, etc. which all can be characterized by the happening of machine understanding in the surrounding world. Understanding as an informational activity will remain on the top of intelligent informing of living beings and machines. Before this happens, the entire phenomenality of the universe has to be understood in informational terms, that is, by the ascent from the arbitrary formational (physical, biologic, chemical, etc.) to the adequate informational, throughout the scientific and everyday thinking and acting of mankind. In general, IIM as a future tool is a parallel understanding informational machine behaving informationally in the surrounding world in which it is informationally embedded.

In the very last consequence, the *informational* (and in this sense, the *intelligent*) means *to be* in a formation (the Being of Formation) with itself and with other existing (factical and possible) formations (entities, things, phenomena). Thus, information (Being in Formation) as it is, is always in context with information as formation of some-

Table of informational machine axioms with Latin, informational, formal, Kantian, informationally interpretive, and cognitional terminology

Latin terms	Informational terms concerning informational machine	Basic informational formula	Kantian terminology for informational machine	Informational interpretation	Cognitional terminology
informatio prima of informational machine $\mathfrak{M}$	informational machine *externalism;* external informing of machine $\mathfrak{M}$; *informing* of machine $\mathfrak{M}$	$\mathfrak{M} \models$	Informationsmaschine *für die anderen*	informational machine *for others*	informational machine *conceptualism*
informatio secunda of informational machine $\mathfrak{M}$	informational machine *internalism;* internal informing of machine $\mathfrak{M}$; *informedness* of machine $\mathfrak{M}$	$\models \mathfrak{M}$	Informationsmaschine *für sich*	informational machine *for itself*	informational machine *perceptiveness*
informatio tertia of informational machine $\mathfrak{M}$	informational machine *metaphysicalism;* metaphysical informing of machine $\mathfrak{M}$; *circular informing* and *informedness* of machine $\mathfrak{M}$	$\mathfrak{M} \models \mathfrak{M}$	Informationsmaschine *in (bei) sich selbst*	informational machine *in itself*	informational machine *consciousness*
informatio quarta of informational machine $\mathfrak{M}$	informational machine *phenomenalism; phenomenal informing* and *informedness* of machine $\mathfrak{M}$	$\mathfrak{M} \models; \models \mathfrak{M}$	Informationsmaschine *an sich*	informational machine *as such*	informational machine *phenomenology*

thing. Formations as observable phenomena are impacting and are impacted in formational ways and are in certain formations to themselves and to other formations. In information there is no vacuum in the sense of apartness, of contextless situation in concern to the surrounding system of formations. Information is like a text in a context and in this function it is able to approach something which demonstrates intelligence, consciousness, understanding.

11. Conclusion

The clue of IM will lead through formalization (symbolization) of informational phenomenality. A solid axiomatic method, that is, a language has to be found to enable technological implementation of informational conceptualism. In the attached table the reader will find four basic axioms of IM informing and their Latin, informational, formal, Kantian, informationally Kantian, and cognitional interpretation. Informational language (philosophy, theory, design) has to be developed in verbal and symbolic form with the aim to master not only the arising problems of informational machines but also informational problems of the living phenomenality. This article is on the way to symbolization and realization (machine implementation) of the proposed informational conceptualism.

IM is on the way to surpass essentially human performance. Imagine a written text and its understanding by man and IM. IM can »understand« all sentences of the text in a parallel manner, that is, simultaneously and, additionally, can inform and be informed in parallel by all in text occurring informational entities. Man, at least, on the level of consciousness, can never perform such a parallel task of text recognition and interpretation. And IM will be able to perform by such method of understanding in any case, for any kind of information (picture, sound, data, signal, »phenomenon«, etc.). It seems that informational parallelism will reach its processing and meaning zenith in various IM applications and, right through parallel operation, will extensively surpass certain possibilities of individual mind.

References

[BIW] Dreyfus, H.L., Being-in-the-World (*A Commentary on Heidegger*'s Being and Time, *Division I*), The MIT Press, Cambridge, MA (1991).

[ACR] de Garis, H., ALife III Conference Report, Email: bingvmb.bitnet, Aug 4, 1992.

[SZ] Heidegger, M., Sein und Zeit, Max Niemeyer Verlag, Tübingen (1986).

[FOT] Mayo, J.S., The Future of Telecommunication, AT&T Technology **7** (1992) *1*, 2-7.

[ENM] Penrose, R., The Emperor's New Mind (Concerning Computers, Minds, and the Laws of Physics), Punguin Books, New York (1989).

[IMD] Železnikar, A.P., Informational Models of Dictionaries I, Informatica **15** (1991) *1*, 11-21.

[TIL] Železnikar, A.P., Towards an Informational Language, Cybernetica **35** (1992) *2*, 139-158.

[FIP] Železnikar, A.P., Formal Informational Principles, Cybernetica **35** (1992) *?*, (in press).

[BIA] Železnikar, A.P., Basic Informational Axioms, Informatica **16** (1992) *3*, 1-16.

[IBU] Železnikar, A.P., An Informational Approach of Being-there as Understanding (in three parts), Informatica **16** (1992) *1*, 9-26; *2*, 29-58; *3*, 64-75.

[OED] An Overview of the EDR Electronic Dictionaries, TR-024, Japan Electronic Research Institute (April 1990).

[JWD] Japanese Word Dictionary, TR-025, Japan Electronic Dictionary Research Institute (April 1990).

[EWD] English Word Dictionary, TR-026, Japan Electronic Dictionary Research Institute (April 1990).

[CD] Concept Dictionary, TR-027, Japan Electronic Dictionary Research Institute (April 1990).

[BD] Bilingual Dictionary, TR-029, Japan Electronic Dictionary Research Institute (April 1990).

LEARNING QUALITATIVE MODELS WITH INDUCTIVE LOGIC PROGRAMMING

INFORMATICA 4/92

Keywords: machine learning, qualitative modelling, qualitative simulation, inductive logic programming

Sašo Džeroski
Institut Jožef Stefan

Qualitative models can be used instead of traditional numerical models in a wide range of tasks. These tasks include diagnosis, generating explanations of the system's behaviour and designing novel devices from first principles. Also, qualitative models are in some cases sufficient for the synthesis of control rules for dynamic systems. An important task in the theory of dynamic systems is the problem of identification of a model that explains given examples of system behaviour. This task can be formulated as a machine learning task of inducing a hypothesis that explains given examples. As the induced hypothesis (model) has to capture relations among the parameters of the observed system, we have to use an inductive tool for learning relations, i.e., an inductive logic programming system. In this paper we describe the application of the inductive logic programming system mFOIL to the problem of learning a qualitative model of the connected-containers dynamic system.

Učenje kvalitativnih modelov z induktivnim logičnim programiranjem

Kvalitativne modele lahko uporabimo za reševanje različnih nalog, npr. za diagnostiko, generiranje razlage obnašanja dinamičnega sistema ter načrtovanje naprav iz osnovnih načel delovanja. V nekaterih primerih zadošča kvalitativni model tudi za sintezo pravil vodenja dinamičnega sistema. Pomemben problem v teoriji dinamičnih sistemov je problem identifikacije modela, ki razloži znane primere obnašanja sistema. Omenjeni problem lahko formuliramo kot problem avtomatskega učenja kjer je treba generirati hipotezo, ki razloži podane primere. Glede na to, da sestoji model iz relacij med parametri sistema, uporabimo za reševanje problema sistem za avtomatsko učenje relacij oz. induktivno logično programiranje. V članku je opisana uporaba sistema za induktivno logično programiranje mFOIL pri problemu učenja kvalitativnega modela sistema povezanih posod.

1 Introduction

Qualitative models can be used instead of traditional numerical models in a wide range of tasks [Bratko 1991]. These tasks include diagnosis (e.g., [Bratko et al. 1989]), generating explanations of the system's behaviour (e.g., [Falkenheiner and Forbus 1990]) and designing novel devices from first principles (e.g., [Williams 1990]). Bratko [1991] conjectures that qualitative models are sufficient for the synthesis of control rules for dynamic systems, and supports this conjecture with an example.

Among several established formalisms for defining qualitative models of dynamic systems, the most widely known are qualitative differential equations called confluences [De Kleer and Brown 1984], Qualitative Process Theory [Forbus 1984] and QSIM [Kuipers 1986]. In this paper, we will adopt the QSIM (Qualitative SIMulation) formalism, as it has been already used for learning qualitative models.

A fundamental problem in the theory of dynamic systems is the identification problem, defined as follows [Bratko 1991]: given examples of the behaviour of a dynamic system, find a model that explains these examples. Motivated by the hypothesis that it should be easier to learn qualitative than quantitative models, [Bratko et al. 1992] have recently formulated the identification problem for QSIM models as a machine learning problem. Formulated in this framework, the task of learning QSIM-type qualitative models is as follows: given $QSIMtheory$ and $ExamplesOfBehaviour$, find a $QualitativeModel$, such that $QSIMtheory$ and $QualitativeModel$ explain the $ExamplesOfBehaviour$, or formally,

$$QSIMtheory \wedge QualitativeModel \models ExamplesOfBehaviour.$$

The identification task can be formulated as a machine learning task. Namely, the task of inductive machine learning is to find a hypothesis that explains a set of given examples. In some cases the learner can also make use of existing background knowledge about the given examples and the domain at hand. So, the learning task can be formulated as follows: given background knowledge $\mathcal{B}$ and examples $\mathcal{E}$, find a hypothesis $\mathcal{H}$, such that $\mathcal{B}$ and $\mathcal{H}$ explain $\mathcal{E}$, i.e., $\mathcal{B} \wedge \mathcal{H} \models \mathcal{E}$. We can see that $ExamplesOfBehaviour$ correspond to $\mathcal{E}$, $QSIMtheory$ corresponds to $\mathcal{B}$ and the target $QualitativeModel$ to $\mathcal{H}$.

As a qualitative model consists of relations among the parameters of the modelled system, we have to use an inductive system for learning relations. Systems that learn relations from examples and relational background knowledge, represented as a logic program, have been recently called inductive logic programming (ILP) systems [Muggleton 1992]. Bratko et al. [1992] describe the application of the inductive logic programming system GOLEM [Muggleton and Feng 1990] to the problem of learning a qualitative model of the dynamic system of connected containers, usually referred to as the U-tube system. There have been, however, several problems with the application of GOLEM to this task, stemming from the inability of GOLEM to use non-ground and non-determinate background knowledge.

In the paper, we describe the application of the inductive logic programming system mFOIL [Džeroski 1991], which can use non-ground background knowledge, to the same task [Džeroski and Bratko 1992]. A brief introduction to inductive logic programming is first given, followed by an outline of the main features of mFOIL. We proceed with an overview of the QSIM formalism and illustrate its use on the connected-containers (U-tube) system. The experiments and results of learning a qualitative model of the U-tube system with mFOIL are next presented, followed by a discussion of related work. Finally, we conclude with some directions for further work.

2 Inductive logic programming

In this section we introduce the field of machine learning of relations, or, as it has been recently called, inductive logic programming (ILP). We first mention some systems for learning relations, define the task of empirical inductive logic programming and illustrate it on a simple example. We then briefly outline some features of the ILP system mFOIL, which was used in our experiments in learning qualitative models.

Various logical formalisms have been used in inductive learning systems to represent examples and concept descriptions. These formalisms are similar to the formalisms for representing knowledge in general. Several widely known inductive learning systems, such as ID3 [Quinlan 1986] and AQ [Michalski 1983] use propositional languages to represent examples (objects) and concepts. In both cases objects are represented as tuples of attribute values, i.e, in terms of their global features. To represent concepts, decision trees are used in ID3 and if-then rules in AQ.

Another class of learning systems induce descriptions of relations (definitions of predicates). In these systems, objects are described structurally, i.e., in terms of their components and the relations between them. Training examples are represented by tuples of their components, while the relations between components belong to background knowledge. The languages used to represent examples, background knowledge and concept descriptions are typically subsets of first-order logic (logic programs). In this case, learning is in

fact logic program [1] synthesis and has recently been named *inductive logic programming (ILP)* [Muggleton 1991, Muggleton 1992].

Two different approaches can be distinguished in the ILP paradigm [De Raedt 1992]: the interactive and the empirical ILP approach. *Interactive ILP* systems include MIS [Shapiro 1983], MARVIN [Sammut and Banerji 1986] and CLINT [De Raedt 1992] as well as CIGOL [Muggleton and Buntine 1988] and other approaches based on inverting resolution [Rouveirol 1991, Wirth 1989]. These systems typically learn definitions of multiple predicates from a small set of examples and queries to the user.

Empirical ILP, on the other hand, is typically concerned with learning a definition of a single predicate from a large collection of examples. This class of ILP systems includes FOIL [Quinlan 1990], mFOIL [Džeroski 1991], GOLEM [Muggleton and Feng 1990] and LINUS [Lavrač et al. 1991]. LINUS, FOIL and mFOIL upgrade attribute-value learners from the ID3 and AQ family towards a first-order logic framework. A different approach is used in GOLEM which is based on Plotkin's notion of relative least general generalization (*rlgg*) [Plotkin 1969].

Empirical ILP systems are more likely to be applied in practice for two reasons. First, there is more experience with learning single concepts from large collections of data than with deriving knowledge bases from a small number of examples. Second, empirical ILP systems are much more efficient because of the use of heuristics, because there is no need to take into account dependencies among different concepts, and because no examples are generated [De Raedt and Bruynooghe 1992]. In fact, they are already efficient enough to be applied to real-life domains [Bratko 1992]. Several applications have been reported, including learning qualitative models from example behaviours [Bratko et al. 1992] [Džeroski and Bratko 1992], inducing temporal rules for satellite fault diagnosis [Feng 1991], learning to predict protein secondary structure [Muggleton et al. 1992] and learning rules for finite element mesh design [Dolšak and Muggleton 1992, Džeroski and Dolšak 1991].

[1] For an introduction to logic programming we refer the reader to [Bratko 1990]. A detailed theoretical treatment of the subject is given in [Lloyd 1987].

Empirical ILP

The task of empirical ILP, which is concerned with learning a single predicate, can be formulated as follows.

Given:

- a set of training examples $\mathcal{E}$, consisting of true $\mathcal{E}^+$ and false $\mathcal{E}^-$ facts of an unknown predicate p,
- a description language $\mathcal{L}$, specifying syntactic restrictions on the definition of predicate p,
- background knowledge $\mathcal{B}$, defining predicates q_i (other than p) which may be used in the definition of p and which provide additional information about the arguments of the examples of predicate p,

Find:

- a definition $\mathcal{H}$ for p, expressed in $\mathcal{L}$, such that $\mathcal{H}$ is complete, i.e., $\forall e \in \mathcal{E}^+ : \mathcal{B} \wedge \mathcal{H} \models e$, and consistent with respect to the examples, i.e., $\forall e \in \mathcal{E}^- : \mathcal{B} \wedge \mathcal{H} \not\models e$.

The true facts $\mathcal{E}^+$ are called *positive examples*, the false facts $\mathcal{E}^-$ are called *negative examples* and the hypothesis $\mathcal{H}$, i.e., the definition of p, is usually called the definition of the *target* predicate. When learning from noisy examples, the completeness and consistency criteria need to be relaxed in order to avoid overly specific hypotheses.

The ILP system mFOIL

The ILP system mFOIL [Džeroski 1991] is largely based on the FOIL [Quinlan 1990] approach. We thus briefly describe FOIL first and then outline some of the key features of mFOIL.

FOIL extends some ideas from attribute-value learning algorithms to the ILP paradigm. In particular, it uses a covering approach similar to AQ's [Michalski 1983] and an information based search heuristic similar to ID3's [Quinlan 1986]. The hypothesis language $\mathcal{L}$ in FOIL is the language of function-free program clauses, which means that no constants or terms other than variables may appear in the induced clauses. Function-free ground facts (relational tuples) are used to represent both training examples and background knowledge.

After the pre-processing of the training set, which consists of generating negative examples if none are given, the outermost loop of the FOIL algorithm repeats the following two steps untill all positive facts are covered:

- find a clause that covers some positive and no negative facts,
- remove the facts covered by this clause from the training set.

Finding a clause consists of a number of refinement steps. The search starts with the clause with empty body. At each step, the clause c built so far is refined by adding a literal to its body. These literals are positive or negative atoms of the form $X_i = X_j$ or $q_k(Y_1, Y_2, ..., Y_{n_k})$, where the X's appear in c, the Y's either appear in c or may be new variables and q_k is a relation (predicate) from the background knowledge or the target predicate p itself.

To stop the search for literals to be added to a clause, FOIL employs the *encoding length restriction*, which limits the number of bits used to encode a clause to the number of bits needed to explicitly indicate the positive examples covered by it. The construction of a clause is stopped when it covers only positive examples (is consistent) or when no more bits are available for adding literals to its body. The search for clauses stops when no new clause can be constructed under the encoding length restriction, or alternatively, when all positive examples are covered. One should be aware, however, that there are several problems with the encoding length restriction that actually degrade FOIL's performance on both noisy and non-noisy data as shown in [Džeroski and Lavrač 1991].

The most important differences between mFOIL and FOIL are related to the noise-handling mechanism used. mFOIL uses Bayesian probability estimates, namely the Laplace and the m-estimate [Cestnik 1990], of expected clause accuracy as search heuristics. These estimates have been successfully used in a similar way in propositional learning systems [Clark and Boswell 1991, Džeroski et al. 1992]. mFOIL also uses significance tests, similar to the ones in [Clark and Boswell 1991]. It achieved better results than FOIL on a test domain with artificially added noise and on a real-life domain of learning rules for finite element mesh design [Džeroski 1991, Džeroski and Bratko 1992].

Another key difference is the capability of mFOIL to use background knowledge which may contain rules and not ground facts only. This feature is especially important for learning qualitative models, since the *QSIMtheory* consists of rules and not ground facts only.

Other differences between FOIL and mFOIL are related to the search strategy and the space of possible hypotheses. As opposed to the hill-climbing search in FOIL, where only the best partially built clause is kept, mFOIL uses beam search and keeps several most promising clauses (the beam), which are refined gradually. mFOIL can also use information about the background knowledge, such as symmetry of predicates and types of predicate arguments, to reduce the space of possible hypotheses.

3 Qualitative modelling

In this section, we first introduce the QSIM [Kuipers 1986] formalism and then illustrate it on the U-tube system. We also describe how the QSIM theory can be formulated in logic [Bratko et al. 1992], so that it can be used as background knowledge in the process of learning qualitative models from examples.

The QSIM formalism

In the theory of dynamic systems, a physical system is represented by a set of continuous variables, which may change over time. Sets of differential equations, relating the system variables, are typically used to model dynamic systems numerically. Given a model (a set of differential equations) of the system and its initial state, the behaviour of the system can be predicted by applying a numerical solver to the set of differential equations.

A similar approach is taken in qualitative simulation [Kuipers 1986]. In QSIM, a physical system is described by a set of variables representing the *physical parameters* of the system (continuously differentiable real-valued functions) and a set of *constraint equations* describing how those parameters are related to each other. In this case, a (qualitative) model is a set of constraint equations. Given a qualitative model and a qualitative initial state of the system, the QSIM simula-

tion algorithm [Kuipers 1986] produces a directed graph consisting of possible future states and the immediate successor relation between the states. Paths in this graph starting from the initial state correspond to behaviours of the system.

The value of a physical parameter is specified qualitatively in terms of its relationship with a totally ordered set of *landmark values*. The *qualitative state* of a parameter consists of its value and direction of change. The direction of change can be *inc* (increasing), *std* (steady) and *dec* (decreasing). Time is represented as a totally ordered set of symbolic distinguished time points. The current time is either at or between distinguished time-points. At a distinguished time-point, if several physical parameters linked by a single constraint are equal to landmark values, they are said to have *corresponding values.*

The constraints used in QSIM are designed to permit a large class of differential equations to be mapped straightforwardly into qualitative constraint equations. They include mathematical relationships, such as $deriv(Velocity, Acceleration)$ and $mult(Mass, Acceleration, Force)$. In addition, constraints like $M^+(Price, Power)$ and $M^-(Current, Resistance)$ state that there is a monotonically increasing/decreasing functional relationship between two physical parameters, but do not specify the relationship completely.

The U-tube system

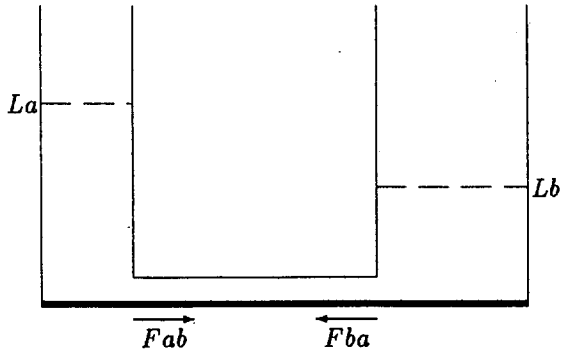


Figure 1: The U-tube system.

Let us illustrate the above notions on the connected-containers (U-tube) example, adapted from [Bratko et al. 1992]. The U-tube system (illustrated in Figure 1) consists of two containers, A and B, connected with a pipe and filled with water to the corresponding levels La and Lb. Let the flow from A to B be denoted by Fab, the flow from B to A by Fba. The variables La,Lb,Fab and Fba are the parameters of the system.

The flows Fab and Fba are the time derivatives of the water levels Lb and La, respectively, and run in opposite directions. Let the difference in the levels of the containers A and B be $Diff = La - Lb$. The pressure $Press$ along the pipe influences the flow Fab: the higher the pressure, the greater the flow. A similar dependence exists between the level difference and the pressure. The above constraints can be formulated in QSIM as follows:

$$\frac{d}{dt}La = Fba$$

$$\frac{d}{dt}Lb = Fab$$

$$Fab = -Fba$$

$$Diff = La - Lb$$

$$Press = M^+(Diff)$$

$$Fab = M^+(Press)$$

If we are not explicitly interested in the pressure, the last two qualitative constraint equations can be simplified into one:

$$Fab = M^+(Diff)$$

For comparison, in a numerical model, the last two equations might have the form

$$Press = c_1 \cdot Diff$$

$$Fab = c_2 \cdot Press$$

or, when simplified

$$Fab = c \cdot Diff$$

where c, c_1 and c_2 are positive constants. In this case, the relationship between the variables Fab, $Press$ and $Diff$ is completely, and not only qualitatively, specified given the values of c, c_1 and c_2.

The landmark values for the variables of this model for the U-tube, ordered left to right, are as follows:

$$La : minf, 0, la0, inf$$
$$Lb : minf, 0, lb0, inf$$
$$Fab : minf, 0, fab0, inf$$
$$Fba : minf, fba0, 0, inf$$

$Time$	La	Lb	Fab	Fba
$t0$	$la0/dec$	$lb0/inc$	$fab0/dec$	$fba0/inc$
$(t0,t1)$	$0..la0/dec$	$lb0..inf/inc$	$0..fab0/dec$	$fba0..0/inc$
$t1$	$0..la0/std$	$lb0..inf/std$	$0/std$	$0/std$
$(t1,inf)$	$0..la0/std$	$lb0..inf/std$	$0/std$	$0/std$

Table 1: Qualitative behaviour of the U-tube system.

These values are symbolic names corresponding to minus infinity, zero, infinity and the initial values of the four variables. The left-to-right ordering corresponds to the *less than* relation between the corresponding numerical values.

The QSIM simulation of the U-tube system produces the trace given in Table 1. From the trace we can see, for example, that in the initial state the value of the level La is equal to $la0$ and is decreasing (dec). This is represented as $La = la0/dec$. In the time interval that follows, La is between 0 and $la0$ and decreasing, which is written as $La = 0..la0/dec$.

Formulating QSIM in logic

Bratko et al. [1992] translate the QSIM approach to qualitative simulation into a logic programming formalism (pure Prolog). A sketch in Prolog of the QSIM qualitative simulation algorithm is given below.

$simulate(State) \leftarrow$
    $transition(State, NextState)$,
    $simulate(NextState)$.

$transition(state(V1,\ldots), state(NewV1,\ldots)) \leftarrow$
    $trans(V1, NewV1)$, $\%Model-independent$
    $\ldots$,
    $legalstate(NewV1,\ldots)$. $\%Model-dependent$

The simulation starts from the initial qualitative *State*, consisting of the qualitative values and directions of change of the system parameters. The simulator first finds a possible transition to a $NewState$ and then continues the simulation from the new state. The relation *trans* is a non-deterministic relation that generates possible transitions of the system parameters, i.e., possible new values for them. It is defined as part of the QSIM theory [Kuipers 1986]. The model of a particular system is defined by the predicate *legalstate* which imposes constraints on the values of the system parameters. The definition of this predicate is of the following form:

$legalstate(\ldots) \leftarrow$
    $constraint1(\ldots)$,
    $constraint2(\ldots)$,
    $\ldots$

where the constraints are part of the QSIM theory. Under continuity assumptions, the problem of learning the legality of states is equivalent to the problem of learning the dynamics of the system.

The following Prolog predicates correspond to the QSIM constraints:

$add(F1,F2,F3,Corr)$	$\%F1 + F2 = F3$
$mult(F1,F2,F3,Corr)$	$\%F1 * F2 = F3$
$minus(F1,F2,Corr)$	$\%F1 = -F2$
$m\_plus(F1,F2,Corr)$	$\%F2 = M^+(F1)$
$m\_minus(F1,F2,Corr)$	$\%F2 = M^-(F1)$
$deriv(F1,F2)$	$\%F2 = dF1/dt$

In the above $F1, F2$ and $F3$ stand for system parameters and $Corr$ stands for a list of corresponding values.

The qualitative model for the U-tube system can be written in Prolog notation as follows:

$legalstate(La, Lb, Fab, Fba) \leftarrow$
    $add(Lb, Diff, La, [c(lb0, d0, la0)])$,
    $m\_plus(Diff, Fab, [c(0,0), c(d0, fab0)])$,
    $minus(Fab, Fba, [c(fab0, fba0)])$,
    $deriv(La, Fba)$,
    $deriv(Lb, Fab)$.

where $c(x,y,z)$ means that x, y and z are corresponding values for the constraint. For example, in the *add* constraint, $c(lb0, d0, la0)$ means that $lb0 + d0 = la0$.

If we are not interested in the $Diff$ parameter, the first two constraints can be replaced by the constraint $add(Lb, Fab, La, [c(lb0, fab0, la0)])$ or the symmetrical constraint $add(La, Fba, Lb, [c(la0, fba0, lb0)])$.

4 An experiment in learning qualitative models

In this section, we describe the application of the ILP system mFOIL to the problem of learning a qualitative model of the U-tube system [Džeroski and Bratko 1992]. We first describe in detail the experimental setup and then present the results generated by mFOIL, followed by a comparison with the results obtained by GOLEM and FOIL on the same problem.

Experimental setup

As mentioned earlier, when formulating the task of identification of models as a machine learning task, $ExamplesOfBehaviour$ become training examples and $QSIMtheory$ constitutes the background knowledge. The $QualitativeModel$ to be learned corresponds to the hypothesis to be induced by the machine learning system. In the following we describe in more detail the training examples and background knowledge used in our experiment. We also give the parameter settings for the inductive logic programming system mFOIL used in the experiment.

As the model of a system is defined by the predicate *legalstate*, the learning task is to induce a definition of this predicate. The behaviour trace of the U-tube system (Table 1) provides three positive training examples for the predicate *legalstate* (the last two states of the behaviour trace are equal). In addition, a positive example which corresponds to the case where there is no water in the containers is considered. The set of positive examples considered is given in Table 4.

Negative examples (illegal states) represent 'impossible' states which cannot appear in any behaviour of the system. For instance, the state $(la : la0/inc, lb : lb0/inc, fab : f0/dec, fba : mf0/inc)$ is a negative example, because it cannot happen that the water levels in both containers increase. Negative examples can be either hand-generated by an expert, or can be generated under the closed-world assumption, when all positive examples are known.

Bratko et al. [1992] used six hand-crafted negative examples, which only slightly differ from the positive ones. Such examples are called near misses. In a preliminary experiment, from the four positive examples in Table 4 and the six near misses mFOIL generated an overly general, i.e., underconstrained model. We thus used a larger set of 543 negative examples, randomly chosen by Žitnik [1991] from the complete set of negative examples generated under the closed-world assumption.

The QSIM theory, formulated in logic, serves as background knowledge. To reduce the complexity of the learning problem, the corresponding values argument $Corr$ is omitted from the constraints. The background knowledge thus consists of the predicates $add(F1, F2, F3)$, $mult(F1, F2, F3)$, $minus(F1, F2)$, $m\_plus(F1, F2)$, $m\_minus(F1, F2)$ and $deriv(F1, F2)$, which correspond to the QSIM constraint primitives with empty lists of corresponding values. For comparison with GOLEM [Bratko et al. 1992], the *mult* relation was excluded from the background knowledge. All arguments of the background predicates are of the same type; they are compound terms of the form $FuncName : QualValue/DirOfChange$.

As mFOIL allows for the use of non-ground background knowledge, we used directly the Prolog definitions of the background predicates and did not tabulate them as ground facts. All of the background predicates, except *deriv*, are symmetric. For example, $add(X, Y, Z)$ is equivalent to $add(Y, X, Z)$. The same holds for the predicate *mult*. Similarly, $minus(X, Y)$ is equivalent to $minus(Y, X)$. This reduces the space of models to be considered.

All arguments of the background knowledge predicates were considered input, i.e., only relations between the given system parameters (La, Lb, Fab, Fba) were considered. This is reasonable, as the correct qualitative model can be formulated in terms of these parameters and without introducing new variables. In some cases, however, the introduction of new variables is necessary. For example, if the U-tube system were described only in terms of La and Lb, the new variables Fab and Fba (or at least one of them) would be necessary for the construction of a qualitative model of the system. As mFOIL allows for the introduction of new variables, such a case could be, in principle, handled if necessary.

Finally, let us mention that the default search heuristic and stopping criteria were used in mFOIL. The Laplace estimate was used as a

$legalstate(la : la0/dec, lb : lb0/inc, fab : fab0/dec, fba : fba0/inc).$
$legalstate(la : 0..la0/dec, lb : lb0..inf/inc, fab : 0..fab0/dec, fba : fba0..0/inc).$
$legalstate(la : 0..la0/std, lb : lb0..inf/std, fab : 0/std, fba : 0/std).$
$legalstate(la : 0/std, lb : 0/std, fab : 0/std, fba : 0/std).$

Table 2: Positive examples for learning a qualitative model of the U-tube.

search heuristic and a significance level of 99 % was employed in the significance tests. The default beam width of 5 was increased to 20 in order to avoid getting stuck in local optima. Namely, if the beam width is one, beam search is actually hill-climbing search and is prone to getting stuck in local optima during the search for good models.

Results

Given the 4 positive, the 543 negative examples and the background knowledge as described above, mFOIL generated 20 different models, shown in Table 4. They are all evaluated as equally good by mFOIL as they correctly distinguish between the given positive and negative examples. In addition, they are of the same length, i.e., consist of four constraints each. However, not all of them are equivalent to the correct model.

Model # 6 is equivalent to the correct model, shown in Section 3, provided that corresponding values are ignored in the latter. The same holds for model # 16 [Žitnik, personal communication]. Out of the 194481 possible states, these models cover 130 states (among which 32 are physically possible, i.e., are positive examples). When all 32 positive examples and the same 543 negative examples were given to mFOIL, it was able to generate, among other models, the two models from Table 4 which are equivalent to the correct model, even with a beam of size 10 and the *mult* relation in the background knowledge.

An important issue arises from the above results, namely, the need for a criterion of quality for qualitative models other than the standard criteria used in machine learning. Considering the models from Table 4, all of which cover all positive and no negative examples, and all of which are of same length, this need is obvious. This problem could be reduced by imposing additional constraints on the models considered in the search process. However, additional semantic criteria may still be needed.

Comparison with other ILP systems

Bratko et al. [1992] applied GOLEM to the problem of learning of qualitative models in the QSIM formalism. They used the four positive examples from Table 4 and six hand-crafted negative examples (near misses). The model induced by GOLEM was shown to be dynamically equivalent to the correct model.

However, several modifications had to be done in order to apply GOLEM. As GOLEM accepts only ground facts as background knowledge, the Prolog definitions mentioned above had to be compiled into tables of ground facts. To reduce the complexity of the learning problem, the predicates were tabulated with empty lists of corresponding values (argument $Corr$). The *mult* constraint was not compiled at all. Finally, the *add* constraint had to be decomposed into several subconstraints in order to avoid the explosion of the number of ground facts generated.

Žitnik [1991] conducted further experiments in learning a qualitative model of the U-tube system, using both GOLEM and FOIL. She used several sets of examples, including the set of 4 positive examples and the set of 543 randomly generated negative examples used by mFOIL. Among the conclusions of her work, we would like to mention the following:

- The need to compile the background knowledge into ground facts for FOIL and GOLEM causes an explosion in the complexity of the learning problem, which has to be handled by decomposing predicates into simpler primitives.
- The absence of type information causes problems in interpreting the generalizations produced by FOIL and GOLEM.
- Top-down systems, such as FOIL, need a larger number of negative examples in order to prevent over-generalization.

#	Model: $legalstate(La, Lb, Fab, Fba) \leftarrow$
1	minus(Fab,Fba), add(Lb,Fab,La), m_minus(La,Fba), deriv(Fab,Fba)
2	minus(Fab,Fba), add(Lb,Fab,La), m_minus(La,Fba), deriv(Lb,Fab)
3	minus(Fab,Fba), add(Lb,Fab,La), m_minus(La,Fba), deriv(La,Fba)
4	minus(Fab,Fba), add(La,Fba,Lb), deriv(La,Fba), m_minus(Lb,Fab)
5	minus(Fab,Fba), add(La,Fba,Lb), deriv(La,Fba), m_plus(Lb,Fba)
6	**minus(Fab,Fba), add(La,Fba,Lb), deriv(Lb,Fab), deriv(La,Fba)**
7	minus(Fab,Fba), add(La,Fba,Lb), deriv(Fab,Fba), deriv(La,Fba)
8	minus(Fab,Fba), add(La,Fba,Lb), deriv(Fba,Fab), deriv(La,Fba)
9	minus(Fab,Fba), add(La,Fba,Lb), m_plus(La,Fab), deriv(Fba,Fab)
10	minus(Fab,Fba), add(La,Fba,Lb), m_plus(La,Fab), deriv(Fab,Fba)
11	minus(Fab,Fba), add(La,Fba,Lb), m_plus(La,Fab), deriv(Lb,Fab)
12	minus(Fab,Fba), add(La,Fba,Lb), m_plus(La,Fab), deriv(La,Fba)
13	minus(Fab,Fba), add(La,Fba,Lb), m_minus(La,Lb), deriv(Fba,Fab)
14	minus(Fab,Fba), add(La,Fba,Lb), m_minus(La,Lb), deriv(Fab,Fba)
15	minus(Fab,Fba), add(La,Fba,Lb), m_minus(La,Lb), deriv(Lb,Fab)
16	**minus(Fab,Fba), add(La,Fba,Lb), m_minus(La,Lb), deriv(La,Fba)**
17	minus(Fab,Fba), add(La,Fba,Lb), m_minus(La,Fba), deriv(Fba,Fab)
18	minus(Fab,Fba), add(La,Fba,Lb), m_minus(La,Fba), deriv(Fab,Fba)
19	minus(Fab,Fba), add(La,Fba,Lb), m_minus(La,Fba), deriv(Lb,Fab)
20	minus(Fab,Fba), add(La,Fba,Lb), m_minus(La,Fba), deriv(La,Fba)

Table 3: Qualitative models for the U-tube generated by mFOIL.

The following model [Žitnik 1991] was induced by GOLEM from the same examples as used by mFOIL and background knowledge as in [Bratko et al. 1992].

$$legalstate(la : A/B, lb : C/D, fab : E/B, fba : F/D) \leftarrow$$
$$deriv\_simplified(D, E),$$
$$legalstate(la : A/G, lb : C/H, fab : I/G, fba : J/H).$$

The condition $deriv\_simplified(D, E)$ actually means $deriv(Lb, Fab)$. This example illustrates the problem that a model induced by GOLEM can have several interpretations. This is due to the fact that GOLEM may introduce new variables, such as the ones in the term $fba : J/H$ in the model below, the meaning of which may be difficult to grasp.

Similar problems appear when using FOIL [Žitnik 1991]. Some of these problems are absent in LINUS, as it has typed variables, and can use non-ground background knowledge. However, LINUS cannot introduce new variables, which can prevent it from learning an appropriate model if such variables are needed.

None of the above problems appear in mFOIL, as it can use non-ground background knowledge and the typing of variables prevents unclear generalizations, while still having the possibility to introduce new variables. It should be noted, however, that new variables that may be introduced by the background knowledge predicates are likely to be non-discriminating and thus some kind of lookahead would be needed to treat them properly.

5 Related work

An early system for learning qualitative models named QuMAS, using a restricted form of logic, is described in [Mozetič 1987] and also in [Bratko et al. 1989]. The idea of QuMAS to transform the problem of learning in logic to propositional form was further developed in LINUS. QuMAS was used, however, to learn about a *static* system. It used a completely different set of primitives (background knowledge predicates) and is thus incomparable to the work on learning qualitative models of *dynamic* systems [Bratko et al. 1992].

GENMODEL [Coiera 1989] also generates a QSIM model in logic, using a special kind of least general generalization. Similarly to mFOIL, it uses a strong typing of variables, so that the value and direction of change always appear together and cannot be mixed (unlike FOIL and GOLEM where they can get mixed). It shares the limitation of LINUS, namely no new variables may be introduced. However, it takes into account the corresponding values in the constraints.

The approach taken in MISQ [Kraan et al. 1991] is essentially identical to GENMODEL, sharing most of GENMODEL's limitations, including the inability to introduce new variables. The useful additions include a facility of extracting qualitative behaviours out of quantitative data and generation of alternative maximally consistent models when incomplete information is given about the example behaviours. Furthermore, dimensional analysis is used to reduce the set of constraints generated.

Varšek [1991] applied a genetic algorithm QME (Qualitative Model Evolution) to the problem of learning qualitative models from examples. The corresponding values are not taken into account. Models are represented as trees in the genetic algorithm. The genetic operator of *crossover* exchanges subtrees between trees, while the *mutation* genetic operator generates random subtrees on single trees. Using a different training set, QME obtained five models equivalent to the correct one. In addition, QME was apllied to several other small domains, including a RC-circuit (resistor-capacitor-circuit). Similarly to GENMODEL, QME can not introduce new variables.

PAL [Morales 1992], originally designed to learn chess patterns, is also based on the idea of least general generalization. It can also use non-ground background knowledge. Using only the four positive examples and the same background knowledge as mFOIL, PAL induced a model equivalent to the correct one. The model contains several redundant constraints, as PAL uses no negative examples to reduce it. [Morales 1992] also successfully induced models of the U-tube system described with three (La, Lb, Fab) and five $(La, Lb, Fab, Fba, Diff)$ parameters. However, adding new system parameters requires additional effort and has to be handled in a special way in PAL. The *rlgg*-based systems PAL, GENMODEL and MISQ do not need negative examples.

6 Conclusion

We have successfully applied the inductive logic programming system mFOIL to the problem of learning a qualitative model of the connected-containers dynamic system. The ability of mFOIL to use non-ground background knowledge proved useful in this respect and advantageous as compared to GOLEM and FOIL. mFOIL produced many models which correctly distinguish between the given positive and negative examples. Two of these proved to be equivalent to the correct model. At the same time, however, this reveals the necessity for criteria other than completeness, consistency and length (complexity) to distinguish among different qualitative models. Although the number of different models may be reduced by using dimensional analysis, this does not avoid the need for suitable criteria (bias).

Another problem with learning qualitative models is the problem of introducing new variables. Although GOLEM, FOIL and mFOIL can introduce new variables, the literals that introduce these variables are typically both non-determinate (the new variables can have more than one value) and non-discriminating (the literals do not distinguish between positive and negative examples). For example, if variable X has a positive qualitative value and variable Y has a negative value, the literal $add(X, Y, Z)$, where Z is a new variable will be non-determinate, as Z can be either positive, zero or negative. However, if X, Y and Z are described numerically, then Z is uniquely determined given X and Y. This suggests that a LINUS-like approach [Lavrač and Džeroski 1992] operating on real-valued variables, where new variables are introduced before learning, coupled with the approach of learning qualitative models from real-valued data [Kraan et al. 1991], might be effective. It might also be possible to generate qualitative models directly from numerical data, without extracting qualitative behaviours.

Acknowledgements

This research was funded by the Slovenian Ministry of Science and Technology. mFOIL was developed as a part of my MSc thesis under the supervision of professor Ivan Bratko, who also commented on earlier versions of several parts of the paper. Many thanks to Nada Lavrač and Tanja Urbančič for their comments on the paper.

References

[Bratko 1990] Bratko, I. (1990). *Prolog Programming for Artificial Intelligence.* Addison-Wesley, Wokingham, 2 edition.

[Bratko 1991] Bratko, I. (1991). Qualitative modelling: learning and control. In *Proc. Inter-*

national Conference on Artificial Intelligence. Prague.

[Bratko 1992] Bratko, I. (1992). Applications of machine learning: Towards knowledge synthesis. In *Proc. International Conference on Fifth Generation Computer Systems*, pages 1207–1218. Tokyo.

[Bratko et al. 1989] Bratko, I., Mozetič, I., and Lavrač, N. (1989). *KARDIO: A Study in Deep and Qualitative Knowledge for Expert Systems.* MIT Press, Cambridge, MA.

[Bratko et al. 1992] Bratko, I., Muggleton, S., and Varšek, A. (1992). Learning qualitative models of dynamic systems. In Muggleton, S., editor, *Inductive Logic Programming*, pages 437–452. Academic Press, London.

[Cestnik 1990] Cestnik, B. (1990). Estimating probabilities: A crucial task in machine learning. In *Proc. Ninth European Conference on Artificial Intelligence*, pages 147–149. Pitman, London.

[Clark and Boswell 1991] Clark, P. and Boswell, R. (1991). Rule induction with CN2: Some recent improvements. In *Proc. Fifth European Working Session on Learning*, pages 151–163. Springer, Berlin.

[Coiera 1989] Coiera, E. (1989). Learning qualitative models from example behaviours. In *Proc. Third International Workshop on Qualitative Physics.* Stanford, California.

[De Kleer and Brown 1984] De Kleer, J. and Brown, J. (1984). A qualitative physics based on confluences. *Artificial Intelligence*, 24:7–83.

[De Raedt 1992] De Raedt, L. (1992). *Interactive Theory Revision: An Inductive Logic Programming Approach.* Academic Press, London.

[De Raedt and Bruynooghe 1992] De Raedt, L. and Bruynooghe, M. (1992). Interactive concept learning and constructive induction by analogy. *Machine Learning*, 8(2):107–150.

[Dolšak and Muggleton 1992] Dolšak, B. and Muggleton, S. (1992). The application of inductive logic programming to finite element mesh design. In Muggleton, S., editor, *Inductive Logic Programming*, pages 453–472. Academic Press, London.

[Džeroski 1991] Džeroski, S. (1991). Handling noise in inductive logic programming. Master's thesis, Faculty of Electrical Engineering and Computer Science, University of Ljubljana, Ljubljana, Slovenia.

[Džeroski and Bratko 1992] Džeroski, S. and Bratko, I. (1992). Handling noise in inductive logic programming. In *Proc. Second International Workshop on Inductive Logic Programming.* Tokyo, Japan. ICOT TM-1182.

[Džeroski et al. 1992] Džeroski, S., Cestnik, B., and Petrovski, I. (1992). The use of Bayesian probability estimates in rule induction. Technical Report TIRM-92-051, The Turing Institute, Glasgow, Scotland.

[Džeroski and Dolšak 1991] Džeroski, S. and Dolšak, B. (1991). A comparison of relation learning algorithms on the problem of finite element mesh design. In *Proc. XXVI Yugoslav Conference of the Society for ETAN.* Ohrid, Yugoslavia. In Slovenian.

[Džeroski and Lavrač 1991] Džeroski, S. and Lavrač, N. (1991). Learning relations from noisy examples: An empirical comparison of LINUS and FOIL. In *Proc. Eighth International Workshop on Machine Learning*, pages 399–402. Morgan Kaufmann, San Mateo, CA.

[Falkenheiner and Forbus 1990] Falkenheiner, B. and Forbus, K. (1990). Self-explanatory simulations: an integration of quantitative and qualitative knowledge. In *Proc. Fourth International Workshop on Qualitative Physics.* Lugano, Switzerland.

[Feng 1991] Feng, C. (1991). Inducing temporal fault diagnostic rules from a qualitative model. In *Proc. Eighth International Workshop on Machine Learning*, pages 403–406. Morgan Kaufmann, San Mateo, CA.

[Forbus 1984] Forbus, K. (1984). Qualitative process theory. *Artificial Intelligence*, 24:85–168.

[Kraan et al. 1991] Kraan, I., Richards, B., and Kuipers, B. (1991). Automatic abduction of qualitative models. In *Proc. Fifth International Workshop on Qualitative Physics.* Austin, Texas.

[Kuipers 1986] Kuipers, B. (1986). Qualitative simulation. *Artificial Intelligence*, 29(3):289–338.

[Lavrač and Džeroski 1992] Lavrač, N. and Džeroski, S. (1992). Background knowledge and declarative bias in inductive concept learning. In Jantke, K., editor, *Proc. Third International Workshop on Analogical and Inductive Inference*. Springer, Berlin.

[Lavrač et al. 1991] Lavrač, N., Džeroski, S., and Grobelnik, M. (1991). Learning nonrecursive definitions of relations with LINUS. In *Proc. Fifth European Working Session on Learning*, pages 265–281. Springer, Berlin.

[Lloyd 1987] Lloyd, J. (1987). *Foundations of Logic Programming*. Springer, Berlin, 2 edition.

[Michalski 1983] Michalski, R. (1983). A theory and methodology of inductive learning. In R.S. Michalski, J. C. and Mitchel, T., editors, *Machine Learning: An Artificial Intelligence Approach*, volume I, pages 83–134. Tioga, Palo Alto, CA.

[Morales 1992] Morales, E. (1992). *First-order induction of patterns in chess*. PhD thesis, Department of Computer Science, University of Strathclyde, Glasgow, Scotland.

[Mozetič 1987] Mozetič, I. (1987). Learning of qualitative models. In Bratko, I. and c, N. L., editors, *Progress in Machine Learning*, pages 201–217. Sigma Press, Wilmslow, UK.

[Muggleton 1991] Muggleton, S. (1991). Inductive logic programming. *New Generation Computing*, 8(4):295–318.

[Muggleton 1992] Muggleton, S., editor (1992). *Inductive Logic Programming*. Academic Press, London.

[Muggleton and Buntine 1988] Muggleton, S. and Buntine, W. (1988). Machine invention of first-order predicates by inverting resolution. In *Proc. Fifth International Conference on Machine Learning*, pages 339–352. Morgan Kaufmann, San Mateo, CA.

[Muggleton and Feng 1990] Muggleton, S. and Feng, C. (1990). Efficient induction of logic programs. In *Proc. First Conference on Algorithmic Learning Theory*, pages 368–381. Ohmsha, Tokyo.

[Muggleton et al. 1992] Muggleton, S., King, R., and Sternberg, M. (1992). Protein secondary structure prediction using logic. In *Proc. Second International Workshop on Inductive Logic Programming*. Tokyo, Japan. ICOT TM-1182.

[Plotkin 1969] Plotkin, G. (1969). A note on inductive generalization. In Meltzer, B. and Michie, D., editors, *Machine Intelligence 5*, pages 153–163. Edinburgh University Press, Edinburgh.

[Quinlan 1986] Quinlan, J. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–106.

[Quinlan 1990] Quinlan, J. (1990). Learning logical definitions from relations. *Machine Learning*, 5(3):239–266.

[Rouveirol 1991] Rouveirol, C. (1991). Completeness for inductive procedures. In *Proc. Eighth International Workshop on Machine Learning*, pages 452–456. Morgan Kaufmann, San Mateo, CA.

[Sammut and Banerji 1986] Sammut, C. and Banerji, R. (1986). Learning concepts by asking questions. In R.S. Michalski, J. C. and Mitchel, T., editors, *Machine Learning: An Artificial Intelligence Approach*, volume 2, pages 167–191. Morgan Kaufmann, San Mateo, CA.

[Shapiro 1983] Shapiro, E. (1983). *Algorithmic Program Debugging*. MIT Press, Cambridge, MA.

[Varšek 1991] Varšek, A. (1991). Qualitative model evolution. In *Proc. Twelfth International Joint Conference on Artificial Intelligence*, pages 1311–1316. Morgan Kaufman, San Mateo, CA.

[Williams 1990] Williams, B. (1990). Interaction-based invention: designing devices from first principles. In *Proc. Fourth International Workshop on Qualitative Physics*. Lugano, Switzerland.

[Wirth 1989] Wirth, R. (1989). Completing logic programs by inverse resolution. In *Proc. Fourth European Working Session on Learning*, pages 239–250. Pitman, London.

[Žitnik 1991] Žitnik, K. (1991). Machine learning of qualitative models. Technical Report IJS-DP-6239, Jožef Stefan Institute, Ljubljana, Slovenia. In Slovenian.

EXPLANATION OF NEURAL NETWORK CLASSIFICATION

INFORMATICA 4/92

Matija Drobnič,
Viljem Križman
and Borut Korenjak
Institut Jožef Stefan

Keywords: machine learning, neural networks, decision trees

We introduced explanation in human-readable form into a neural network classiffier. The neural network was upgraded by an inductive learning system, which generated the decision tree to explain the way neural network classified new examples. The decision tree learned was compared to the neural network itself and to the inductive learning system regarding both transparrency and classification accuracy.

Razlaga klasifikacije z nevronsko mrežo

V delu predstavljamo metodo, ki pri klasifikaciji z nevronsko mrežo omogoča razlago klasifikacije v človeku razumljivi obliki. Nevronsko mrežo smo nadgradili s sistemom za induktivno učenje, ki generira odločitveno drevo kot razlago delovanja le-te. Tako dobljena odločitvena drevesa smo primerjali z izvirnimi nevronskimi mrežami in s sistemom za induktivno učenje tako glede razumljivosti kot tudi s stališča klasifikacijske točnosti.

1. Introduction

Artificial neural network models were introduced as an attempt to describe the way human brain copes with data, especially in cases of pattern recognition [1]. They are, in principle, based on our understanding of the human brain structure. Their computational power is based on the massive parallelism of simple elements and their dense interconnection. Many different types of neural networks (NN) were introduced during last years. In the field of digital pattern recognition, single-layer networks are mostly used [2], whereas three-layer feed-forward networks can be used as general classifying systems for the data described in an attribute-value laguage [3]. In this field, their adaptability and classification accuracy makes them a very useful tool. Their main drawback is the lack of transparrency to the human user, who cannot figure much out from the values of the NN weights.

Induction learning (IL) is another approach to the classification task (e.g. [4], [5]). Given the examples, the IL system tries to generate a classification function in the form of DT or in the form of **IF - THEN** rules. The main advantage is, the acquisition of knowledge in the form suitable for expert systems, where the transparrency of the results is strictly required. Introduction of statistical methods into the knowledge acquisition process also provides classification accuracy comparable to the one of the classification methods of classical statistics.

In this paper, we try to combine the advantages of both approaches. The main idea is to use IL methods to extract the information hidden in NN weights. The DT obtained in this way, provides an insight into the process of the classification of new examples.

2. Explanation in the neural network

As an example of NN classifier, the growing neural network (GNN) has been chosen. It is a single-layer NN, where neurons are vectors of weights with attached classes, belonging to the same space as the learning examples. Its classification is based on the nearest-neighbour method. To generate the GNN classifier, slightly modified unsupervised learning algorithm proposed by Kohonen [6] was used. It can be put as follows

```
normalise_vectors;
net = first_example_vector;
repeat
   x = next_example_vector;
   y = nearest_vector_from_network(x);
   if class(x) == class(y) then begin
      y = y + α(x — y);
      update_vector(y, net);
   end
   else
      add_vector(x, net);
until no_more_examples;
```

For each new learning example x, we find the nearest vector y from the network according to the $\| \ \|_2$ norm. If their classes match, y is slightly rotated into direction of x (in our experiments, we set $\alpha = 0.2$). Otherwise, x is added to the network as a new vector. When the network is built, the classification process is simple: given an example, find the nearest vector in network according to the $\| \ \|_2$ norm and use its class to classify the example. The implementation of the upper algorithm in C language is given in [7].

As an IL system, ASSISTANT Professional [8] was chosen. It is a tool for the induction of decision trees (DTs) from examples in the attribute-value language, based on ID3 algorithm [4], improved by the binarisation of the attributes, the mechanism for dealing with incomplete data and the tree pruning features.

Several ways of combining the NN and IL methods have been proposed recently [9]. One can classify all the learning examples with the NN learned from them, obtaining their new classes, and then feed them to the IL system as an input. Another possibility is, to generate artificial examples, classify them with NN, and use them as an IL system input again. We have chosen another way: we took the original weight vectors from the GNN and use them as learning examples. In the case of GNN, this simple schema makes sense, since the GNN uses its weight vectors as examples for the nearest-neighbour classification.

3. Experimental results

3.1. Experimental Setup

In our experiments, the medical domain, describing the condition of coronary arteries after the bypass operation has been used. Domain contains 112 examples. Each of them belongs to one of the following classes: deteriorated, unchanged or improved condition. The data is described with 30 parameters, 14 numerical and 16 logical. The numerical attribute values were normalised using the $\| \ \|_\infty$ norm. The logical ones were coded as 0 and 1. Before loaded into ASSISTANT Professional, the numerical values were discretized into 5 equal intervals.

For the cross-validation, the examples were 10 times randomly divided into a training set (70% or 80 examples) and a testing set (30% or 32 examples). For every distribution, GNN was built on learning examples (GNN). Then, DT was learned from the neural network (IL_GNN). As a reference, another DT was learned from the original learning examples (IL). All three methods were compared regarding the transparrency of the classification process.

In the next step, we have used the trees, learned from neural networks as standalone classifiers. All three methods were then compared also regarding the classification accuracy.

3.2. Transparrency of the classification

In this section, we compare three different methods with respect to their transparrency of the classification process to the human user. First, let us examine the GNN classifiers. The algorithm described in Section 2 generates networks containing about 15 neurons. The results are shown in Table 1. Every neuron contains 30 real-valued weights and attached class. A typical network (distribution 2) is presented in Figure 1.

	$Class_1$	$Class_2$	$Class_3$	$\sum$
0	1	0	12	13
1	3	0	13	16
2	2	1	13	16
3	0	1	8	9
4	1	1	13	15
5	2	0	12	14
6	3	1	13	17
7	3	0	16	19
8	2	1	13	16
9	4	1	13	18
$\langle x \rangle$	2.1	0.6	12.6	15.3
σ_x	1.1	0.5	1.9	2.7

Table 1: Number of neurons in GNNs.

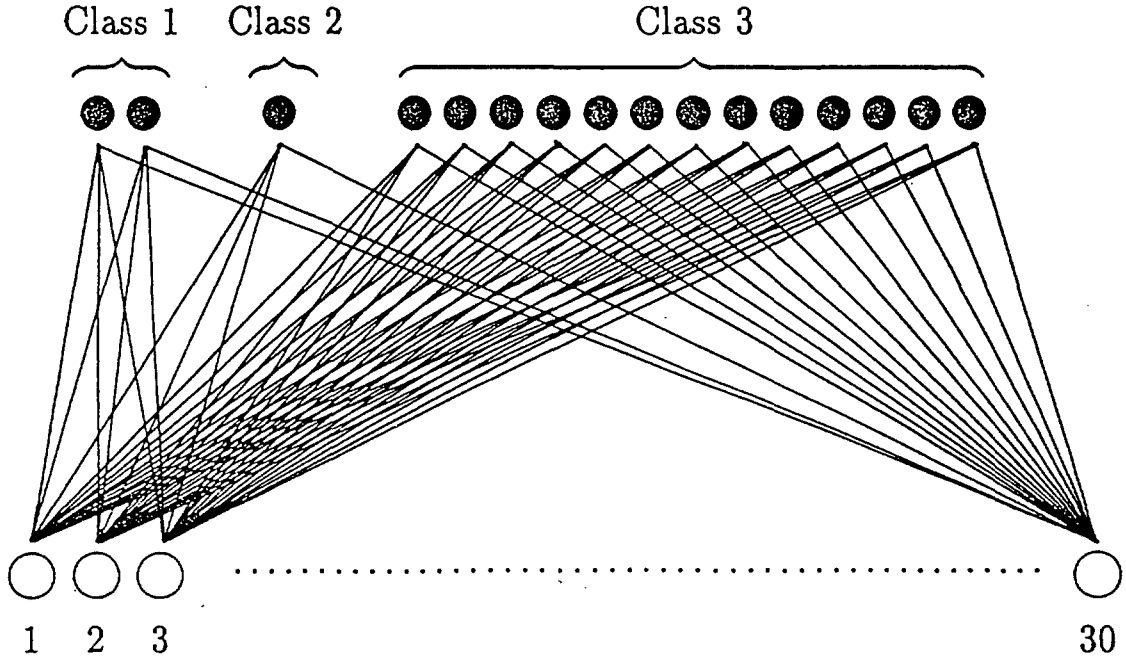


Figure 1: An example of neural network.

Every classification of the new example requires $16 \times 30 = 480$ subtractions, multiplications and additions to calculate Euclidean distance from all the vectors in neural network. Additionally, it also requires 16 comparisons. Even if a human user is capable of using GNN, it is a black box, returning the result without any explanation.

In the next stage, the neurons of GNNs were used as learning examples for the IL system. The DTs learned from the GNNs contained about 4–5 nodes and about 2–3 leaves. An example of such decision tree (distribution 2) is shown in Figure 2.

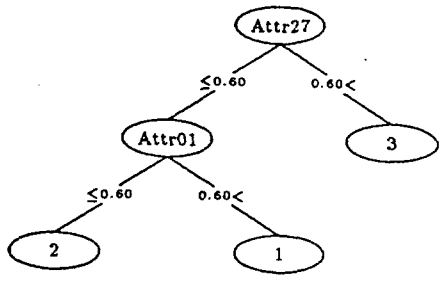


Figure 2: An example of DT, learned from network.

The difference between the classifiers from Figures 1 and 2 is obvious: in the second case, only two comparisons are required during the classification in the worst case. Furthermore, the DT is simple enough to be understood by humans, and can be easily used even without a computer. This is certainly not true with the neural network, where knowledge is hidden into 480 real-valued weights.

For further comparison, we also ran the ASSISTANT Professional with the original examples as an input. The DTs, learned in this way were much bigger than the ones, learned from the neural networks: they typically contained about 18 nodes and about 9 leaves. For comparison with the DTs learned from GNN neurons, the number of nodes and leaves for both methods are presented in Table 2. An example of DT, learned from the learning examples (distribution 2), is shown in Figure 3.

	DT learned from examples			DT learned from GNN		
	nodes	leaves	NULL	nodes	leaves	NULL
0	21	11	2	3	2	0
1	21	11	4	3	2	0
2	17	9	2	5	3	0
3	19	10	3	3	2	1
4	17	9	4	5	3	0
5	15	8	2	3	2	0
6	13	7	3	5	3	1
7	19	10	3	5	3	0
8	19	10	3	5	3	0
9	17	9	3	5	3	0
$\langle x \rangle$	17.8	9.4	2.9	4.2	2.6	0.2
σ_x	2.4	1.2	0.7	1.0	0.5	0.4

Table 2: The sizes of DTs, learned from the GNN neurons and from examples.

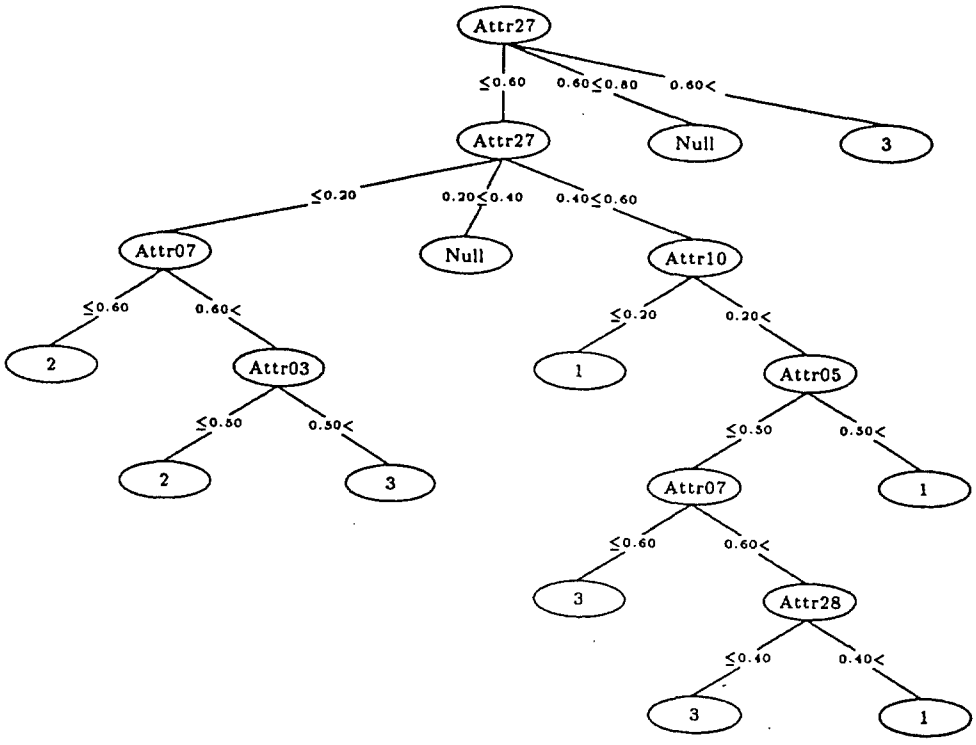


Figure 3: An example of DT, learned from examples.

The transparrency of the upper DT to a human user is also much better than the one of the neural network. Comparing to the trees, learned from neural network, it is slightly less handy for use. However, as it will be shown in the next section, its classification accuracy is much better.

3.3. Classification accuracy

In the next step of our experiments, the DTs learned from neural networks were used as standalone classifiers to validate their quality. their classification accurracy (IL_GNN) was then compared to the one of the GNNs alone (GNN) and to the classification accuracy of DTs learned from the original learning examples (IL). The classification accuracy of all three classifiers was estimated using testing examples. The results are shown in Table 3.

First, the inferiority of the GNN classifiers is obvious. Also the standard deviation of the classification accuracy is very high. The GNN learning algorithm is very sensitive to the ordering of the examples.

	GNN	IL_GNN	IL
0	0.688	0.844	0.906
1	0.813	0.875	0.875
2	0.875	0.875	0.906
3	0.688	0.813	0.906
4	0.813	0.781	0.813
5	0.781	0.844	0.813
6	0.844	0.813	0.844
7	0.844	0.844	0.938
8	0.906	0.938	0.906
9	0.719	0.813	0.906
$\langle x \rangle$	0.797	0.844	0.881
σ_x	0.073	0.042	0.041

Table 3: The classification accuracy of GNN, IL_GNN, and IL systems

However, with some improvements (randomly chosen learning examples, dissmission of weak neurons), the classification accuracy of the GNNs is improved and reaches about 82% [10]. Surprisingly, the classification accuracy of the DTs learned from the GNNs, is greater than the one of the networks themselves.

It seems to us that this happens due to the nature of the IL mechanism, which tries to extract the useful information and to suppress noise in data. In this way, it can use the knowledge hidden in GNN weights, which cannot be used by the nearest-neighbour mechanism of GNN. The DTs, learned directly from learning examples, were significantly more accurate than the ones learned from GNNs This was expected, since their learning sets were larger (the number of learning examples was typically much greater than the number of neurons in the corresponding GNNs).

4. Conclusions

In our attempt to introduce the explanation into a NN classifier, the latter was upgraded by the IL system ASSISTANT Professional. We showed that this system successfully extracted knowledge from the network and presented it in the form of decision tree, so that it could be directly used by human users.

The classification accuracy of the decision trees learned from the neural networks was significantly better than the one of the NNs themselves. This might be caused by badly chosen algorithm for the construction of neural networks, but it seems to us, that IL systems together with their incorporated statistical methods can significantly improve not only the transparrency of the neural network classifiers, but also their classification accuracy.

Acknowledgements

We would like to thank prof. dr. Andrej Dobnikar, who supervised our work on this project and gave us many useful advices. We would also like to thank dr. Matjaž Gams, who also gave us many suggestions during our work. Uroš Rezar implemented algorithms for the construction of the neural network clasifiers, calculated the network weights and provided us domain data. Dr. Bojan Cestnik kindly allowed us to use ASSISTANT 86. Our work was supported by the Ministry of Science, Research and Technology of Republic of Slovenia.

References

[1] T. Kohonen, An Introduction to Neural Computing, "Neural Networks", Vol.1, pp. 3 - 16, 1988.

[2] R. O. Duda and P. E. Hart, "Pattern Classification and Scene Analysis", John Wiley & Sons, New York, 1973.

[3] D. E. Rumelhart, G. E. Hinton and R. J. Williams, Learning Internal Representation by Error Propagation. In D. E. Rumelhart and J. L. McClelland, editors, "Parallel Distributed Processing Explorations in the Microstructure of Cognition. Vol. 1 Foundations", MIT Press, 1986.

[4] J. R. Quinlan, Discovering Rules by Induction from Large Collections of Examples. In D. Michie, editor, "Expert Systems in the Microelectronic Age", Edinbourgh University Press, 1979.

[5] L. Breiman, J. H. Friedman, R. A. Olsen and C. J. Stone, "Classification and Regression Trees", Belmont, California Wadsworth Int. Group, 1984.

[6] T. Kohonen, "Self-Organisation and Associative Memory", Springer-Verlag, Berlin, 1984.

[7] Y. Pao, "Adaptive Pattern Recognition and Neural Network", pp. 291 - 299, Addison Wesley Publishing, 1989.

[8] B. Cestnik, I. Kononenko and I. Bratko, ASSISTANT 86 A Knowledge-Elicitation Tool for Sophisticated Users. In I. Bratko and N. Lavrač, editors, "Proc. 2nd European Working Session on Learning", pp. 31 - 45, Sigma Press, Wilmslow, 1987.

[9] J. L. Shavlik, Constructive Induction in Knowledge-based Neural Networks. In "Proc. 8th International Workshop on Machine Learning", pp. 213 - 217, Evanston, Ilinois Morgan Kaufman, 1991.

[10] U. Rezar and A. Dobnikar, "Prediction of Coronary Disease Expiration", Proc. ERK, Portorož, 1992.

IMPLEMENTACIJA TEX-a NA RAČUNALU ATARI-ST*

INFORMATICA 4/92

Keywords: TEX user-interface, Desktop publishing, public-domain software

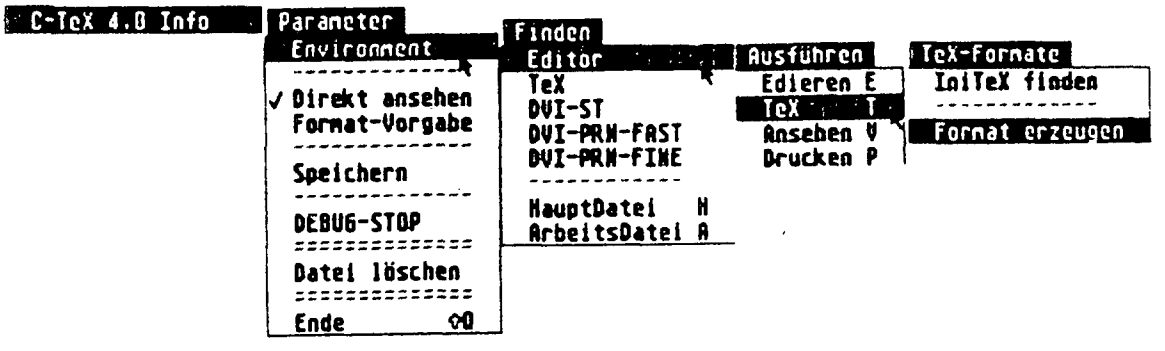
Mirko Varga
Fakultet organizacije i informatike
Varaždin, Hrvatska

SAŽETAK:

U ovom radu dan je prikaz integriranog okruženja namijenjenog korisnicima TEX-a na računalu ATARI-ST. Posebno se obradjuje način instalacije i korištenje TEX-a od strane korisnika koji imaju na raspolaganju minimalnu konfiguraciju, a žele koristiti kompleksan programski sustav TEX bez nadogradnje vlastitog radnog okruženja.

ABSTRACT:

The paper presents an integrated environment for the TEX program on a computer ATARI-ST. The art of installation and use of TEX are described, their characteristics and possible errors in work are given and compared, for users which have minimum system configuration. The minimum system configuration for use of TEX-a includes an Atari ST with at least 2 megabytes of RAM, and an optional hard disk.



Sl. 1 Prikaz Shell-a za TEX

††Slog je autorov(public-domain software LATEX, laserski printer ATARI SLM605)

1.UVOD

TEX sustav najbolje se može opisati kao programski jezik visokog nivoa za slaganje teksta, koji je razvio **Donald E. Knuth** sa suradnicima na Stanford univerzitetu. Rezultat njegovog višegodišnjeg rada prihvaćen je kao standard u Sjedinjenim Američkim Državama i zapadnoj Evropi tako da velike izdavačke kuće i udruženja kao što je npr: AMS ("American Mathematical Society") izdaju svoje knjige i časopise uz pomoć TEX sustava.

Primjenom toga omogućava se da se znatno skrati vrijeme od primitka znanstvenog članka do njegovog izdavanja, jer se članci primaju kao TEX datoteke, koje se onda jednostavno uključuju u cjelokupni TEX sustav.

TEX je sustav za slaganje rečenica, tj. program koji zauzima vrlo visoko mjesto u području obrade teksta, koji čak graniči s umjetnošću. Naziv TEX izveden je od grčkih slova Tau, Epsilon, Chi; korektan izgovor glasi "Tech". TEX raspolaže širokim spektrom vlastitih naredbi, odnosno proširenih naredbi definiranih od strane korisnika TEX-a uz pomoć makro definicija. Sustav je vrlo otvoren i fleksibilan a jedna od njegovih značajnih prednosti leži u mogućnostima dogradnje sustava prema korisnikovim potrebama sredstvima koja su nj ugradjena.

TEX jest u pravilu "public-domain" software. Dostupan je gotovo na svim poznatijim računalima. Naravno, implementiran je najprije na velikim računalima ("mainframe") zbog njihovih karakteristika, kao što je npr: VAX. Prebacivanje na mikroračunala trebalo je pričekati izvjesno vrijeme. Povećanjem učinkovitosti mikroračunala otvorene su mogućnosti daljnjih implementacija TEX-a tako da se najprije razvija verzija za Apple Macintosh, slijedi verzija za IBM-PC i kompatibilna računala, zatim za ATARI-ST itd.

2. TEX na računalu ATARI-ST

TEX na računalu ATARI-ST isporučuje se u tzv. "arhiviranoj formi". Integralan paket nalazi se na 11 disketa i za instalaciju na harddisku zahtijeva raspoloživi memorijski prostor od otprilike 15 do 20 MB. Prilikom instalacije uglavnom nema nekih većih problema budući da nas kroz čitav proces instalacije vodi poseban "instalacijski program". No, budući da većina korisnika ATARI-sustava nema na raspolaganju hard-disk, postavlja se pitanje da li je moguće instalirati i organizirati tako kompleksan programski sustav na način, da ga mogu koristiti korisnici koji nemaju u vlastitoj konfiguraciji hard-disk, već samo disketnu jedinicu. Iskustvo autora ovih redaka pokazuje da je to zaista i moguće uz odgovarajuću organizaciju čitavog sustava po pojedinim disketama te uz dosta strpljenja u radu. Instalacija TEX-a s disketa je prilično naporan posao koji traži dosta umijeća te ću u nastavku opisati najznačajnije korake. Budući da se cjelokupan sustav isporučuje u "arhiviranom formatu" najprije je potrebno pronaći pravi put za "dearhiviranje" potrebnih datoteka. Najbolji način zato jest, otvoriti RAM disk "prave veličine", čime se ubrzava "dearhiviranje" potrebnih datoteka. To se izvodi vrlo lako preko za to predvidjenog programa "UNARC.TTP", koji se zajedno s datotekama za dearhiviranje smještava na RAM-disk. Nakon toga se odabiru datoteke koje će se "otpakirati".

2.1 Startanje INITEX-a

Integralan program za instalaciju i korištenje TEX-a pripada u grupu tzv. "public-domain" software-a, te se koristi za produkciju odgovarajućeg formata kojeg ćemo kasnije koristiti prilikom prevodjenja izvornih TEX programa. Možemo izvršiti izbor izmedju slijedećih formata: PLAIN, PLAING, LATEX i LATEXG. Neophodne pretpostavke i koraci za inicijalizaciju su slijedeći:

- Kopirajte programe CTEX.PRG, INITEX.TTP i TEX.POO na jednu praznu disketu, koju označimo s "A".
- Dearhivirajte datoteku INITEX.ARC te kopirajte pripadajuće datoteke na drugu disketu, koju označimo s "B". Takodjer dearhivirajte datoteku TFM.ARC i kopirajte pripadajuće datoteke u poddirektorij TFM.

- Startajte SHELL program za TEX CTEX.PRG i setirajte radno okruženje. Postavite slijedeće parametre:
  1. TEXINPUTS=b:
  2. TEXOUTPUTS=a:
  3. TEXFORMATS=a:
  4. TEXFONTS=b: \ tfm
- Pronadjite INITEX.TTP u meniju, pospremite setirane parametre i startajte "Format erzeugen" nakon izbora vašeg željenog formata (npr: PLAIN.INI). Generiranje odredjenog formata će potrajati odredjeno vrijeme, te ukoliko se obavlja samo uz pomoć disketnog pogona, zahtijevat će takodjer dosta izmjena disketa. Npr: generiranje formata LATEXG.FMT traži otprilike oko 20 minuta uz uvjet da ste dosta vješti u izmjeni disketa. Osim toga produkcija formata LATEXG.INI zahtijeva oko 500 Kb slobodnog prostora sekundarne memorije uz najmanje 1 Mb raspoložive radne memorije.

2.2 TEX na ATARI-ST

Nakon instalacije sustava i generiranja odgovarajućeg formata na način opisan u prethodnom poglavlju, potrebno je podijeliti kompleksan sustav u srodne cjeline, koje će pratiti pojedine faze rada s TEX-om. Ovo je posebno važno za najveći dio korisnika odnosno vlasnika ATARI računala koji u konfiguraciji nemaju hard-disk već samo disketnu jedinicu.

SHELL za TEX implementiran je kao integralan program koji prati slijedeće faze rada: (vidi sl. 1)

1. unos i ažuriranje izvornih TEX programa,
2. prevodjenje izvornih programa,
3. preusmjeravanje izlaza generiranog TEX-om na odgovarajući medij.

Zbog toga je potrebno organizirati programe tako da sinhronizirano prate navedene korake i omogućavaju nesmetan rad. Preporučljivo je to izvesti na slijedeći način:

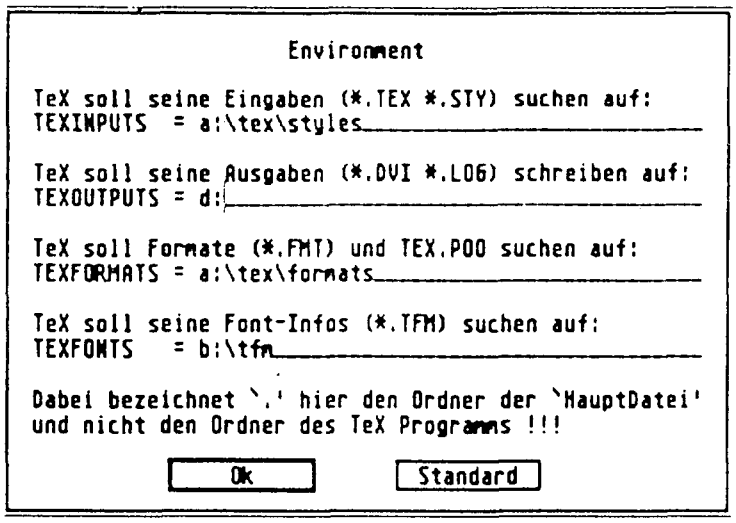
- Kopirajte CTEX.PRG, TEX.TTP, vaš omiljeni editor (TEMPUS, WORDPLUS i sl.) i kompletan poddirektorij fontova na jednu praznu disketu (po mogućnosti što većeg kapaciteta) koju označite kao Programska disketa A:
- Generirani format (PLAIN(G),LATEX(G); G - označava germansku opciju) kopirajte na drugu praznu disketu koju označite kao Programska disketa B: Ukoliko koristite samo PLAIN format datoteku PLAIN.FMT možete iskopirati i na poddirektorij \ TEX programske diskete A.
- Odgovarajuće programe za usmjeravanje izlaza na odredjene medije s pripadajućim fontovima za tu vrstu izlazne jedinice organizirajte takodjer po pojedinim disketama. Npr: za pregledavanje izlaznih datoteka na zaslonu snimite program DVI2ST.TTP ili DVIVIEW.TTP s fontovima za ekran na jednu disketu a za korištenje nekih drugih izlaznih medija nabavite potreban program ("driver") i pripadajuće fontove. Posebno se ovo odnosi na različite vrste printera i ostalih izlaznih jedinica čije vam korištenje omogućava TEX.
- Ulazne datoteke, korištene stilove (posebno ako koristite LATEX) smjestite na disketu podataka.

Prije pokretanja SHELL-a za TEX otvorite RAM-disk (ca. 40 - 200 kB) za redirekciju izlaza. Startajte CTEX.PRG i setirajte radno okruženje u kojem ćete raditi: (vidi sl. 2)

1. TEXINPUTS=b:
2. TEXOUTPUTS=d:
3. TEXFORMATS=a:
4. TEXFONTS=b: \ TFM

Time ste stvorili sve pretpostavke za korištenje TEX-a uz pomoć samo disketnog pogona. Morat ćete se pomiriti s činjenicom da će

rad biti znatno sporiji i da ćete u radu morati dosta često izmjenjivati diskete.



Sl. 2 Radno okruženje

3. POGREŠKE U RADU

Najznačajnije pogreške u radu s TEX-om mogu se svesti na slijedeće:

1. pogreške operacijskog sustava,
2. pogreške kod rada sa SHELL-om za TEX.

Rad s TEX-om zahtijeva otprilike oko 1 Mb slobodnog prostora (kod TOS-a u ROM-u) što odgovara oko 1.2 Mb memorije kada se operacijski sustav TOS učitava s diskete u radnu memoriju. To znači da je potrebno izvršiti proširenje memorije na 2 odnosno 2.5 Mb kako bi se osigurali od pogrešaka ovakve vrste.

Druga vrsta pogrešaka odnosi se na poznavanje koncepta TEX-a i SHELL-a za TEX. Ova vrsta pogrešaka rijetko se može pripisati programskoj realizaciji odnosno implementaciji SHELL-a, već se više odnosi na nedovoljno znanje i obučenost korisnika da koriste TEX. Poznavanje i savladavanje TEX-a traži dosta vremena i praktičnog iskustva te se ne može nikako usporedjivati s naporom koji je potrebno uložiti da se savlada neki drugi tekstprocesor ("wordprocessor"). Tome u prilog govori i izvrsna dokumentacija za TEX počevši od autora Knutha do drugih autora koji su TEX nastojali približiti sve većem broju korisnika.

Najznačajnije fatalne pogreške u radu s TEX-om su slijedeće:

- **"TEX capacity exceeded, sorry"** - nedostatak potrebnih kapaciteta za korištenje TEX-a,
- **"stack overruns heap"** - premala radna memorija,
- **"disk or directory full"** - prepunjen disk ili radni direktorij,
- **"bomben"** - ne smiju nikako nastupiti.

Općenita preporuka za izbjegavanje fatalnih pogrešaka svodi se na osiguravanje potrebnog memorijskog prostora kako na radnoj memoriji tako i na radnim direktorijima vaših disketa. To znači da sigurno podignete operacijski sustav, oslobodite ga od nepotrebnih programa koji mogu izazvati negativne implikacije na TEX (npr:"accessories"), te vodite uvijek računa o slobodnom prostoru. Ovdje SHELL takodjer pomaže korisnicima jer omogućava podešavanje radnog okruženja prema vlastitim potrebama odnosno specifičnostima rada svakog korisnika.

Druga vrsta pogrešaka predstavlja blažu vrstu tj. odnosi se na upozorenja korisniku TEX-a, koje uglavnom zadiru u samo poznavanje ovog programskog sustava i njegove implementacije. Najznačajnije pogreške ove vrste su:

- neodgovarajući format-file tj. pravi format ne postoji ili nije pronadjen na pravom mjestu,
- "Overfull hbox or vbox" - prepunjena horizontalna ili vertikalna kutija,
- "Error in Treibern" - pogreška kod poziva programa za usmjeravanje izlaza na odgovarajući medij,
- "Bad DVI file" - neprikladna izlazna datoteka,
- "Cannot write xxx file" - nemože zapisati datoteku odredjenog tipa (npr: DVI, LOG, AUX) na odredjeni medij.

Ova vrsta pogrešaka lakše se otklanja jer zahtijeva blage intervencije u sam SHELL odnosno u izvorne TEXprograme. Što se tiće samog korištenog formata, producirajte ga uz pomoć INITEX-a te ga smjestite na poddirektorij *\TEX\FORMAT*. Budući da odredjeni formati zauzimaju dosta memorije (npr: LATEXG) preporučljivo je organizirati odvojene diskete koje podržavaju rad s odredjenim formatima npr: PLAIN(G) te LATEX(G).

Upozorenja koja se odnose na prepunjenost vertikalnih odnosno horizontalnih kutija najlakše se otklanjaju ukoliko se naredbe hfil ili vfil postave na pravo mjesto. Naravno, to se može riješiti i instrukcijom hyphenation mada treba reći da je ona prvenstveno usmjerena na engleski jezik.

Slijedeća vrsta pogrešaka uglavnom se odnosi na nepostojanje odgovarajućih programa ili fontova za odredjene medije. Izbjegnite korištenje nepostojećih fontova ili ih producirajte uz pomoć METAFONT-a.

Neprikladna izlazna DVI datoteka uglavnom se javlja kod nedostatka memorijskog prostora što ima za direktnu posljedicu nekompletiranost te datoteke i nemogućnost da se ona dalje koristi za dobivanje izlaznih rezultata.

Nemogućnost zapisa odgovarajućih datoteka može biti posljedica više uzroka. Kao najčešće spominjemo:

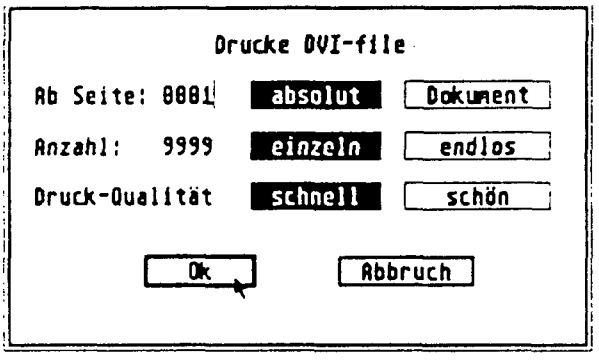
- zaštita od upisivanja (npr: disketa),
- nedostatak memorijskog prostora (npr: premali RAM - disk),
- loše podešeno radno okruženje.

4. ZAKLJUČAK

Integrirano okruženje za rad s programskim sustavom TEX na računalu ATARI-ST objedinjuje cjelokupan proces produciranja teksta koji se sastoji od slijedećih faza:

- unos izvornog TEX koda pomoću omiljenog tekstprocesora kojeg uključujemo u SHELL,
- prevodjenje TEX koda uz generiranje izlazne datoteke (**DVI**) nezavisne od izlaznih uredjaja na koji će ona biti usmjerena,
- pregledavanje izlaza na zaslonu,
- tiskanje izlaznih datoteka na različitim vrstama printera koje uključujemo u naše radno okruženje. (vidi sl. 3)

Navedene faze rada u idealnom slučaju moguće je provoditi u danom redoslijedu ili u iterativnom postupku. Fleksibilnost sustava sastoji se u mogućnostima izbora odgovarajućih faza rada i konfiguriranja vlastitog sustava prema specifičnim potrebama i radnom okruženju samog korisnika.



Sl. 3 Tiskanje izlaznih DVI datoteka

REFERENCES:

[1] Donald E. Knuth, Computers and Typesetting Vol. A-E. Addison-Wesley Co., Inc.,Reading, MA, 1984-1986.

[2] Norbert Schwartz, Einführung in TEX Addison-Wesley, 1987.

[3] Helmut Kopka, LATEX - Eine Einführung, Addison-Wesley, 1988.

VPLIV NAČINA ČASOVNE RAZVRSTITVE IZVAJANJA MODULOV ALGORITMA NA POSPEŠITEV PORAZDELJENEGA SISTEMA ZA PROCESNO VODENJE

INFORMATICA 4/92

Vladimir Jovan
Center za tehnologijo vodenja sistemov
Odsek za računalniško avtomatizacijo
in regulacije
Institut Jožef Stefan, Ljubljana

Keywords: industrial process control, distributed control system, scheduling, speedup

Pomembna značilnost računalniških sistemov procesnega vodenja je zahteva po delovanju sistema v realnem času. Načrtovalec lahko temu pogoju zadosti z določitvijo ustrezne arhitekture strojne opreme, v primeru uporabe večračunalniškega sistema pa tudi z izbiro pravilnega zaporedja izvajanja posameznih modulov implementiranega algoritma. V članku opisujemo postopek in rezultate raziskave vpliva načina časovne razvrstitve izvajanja modulov na trajanje cikla algoritma pri porazdeljenem sistemu vodenja poljubnega industrijskega procesa.

INFLUENCE OF TIME SCHEDULING ON SPEEDUP IN A DISTRIBUTED PROCESS CONTROL SYSTEM. The important characteristic of computer process control systems is the capability to run in real-time. This condition can be fulfilled by proper hardware and software design and in case of multicomputer system also by appropriate time scheduling of implemented control algorithm's modules execution. The article describes the procedure and the results of the research which was realized to determine the contribution of proper program module scheduling to the speedup in an industrial distributed control system.

1. UVOD

Naloga računalniškega sistema vodenja industrijskega procesa se je v zadnjih letih bistveno spremenila; iz pasivnega prikazovalca trenutnega stanja vodenega procesa in zapisovalca posameznih dogodkov v procesu se težišče nalog sistema vodenja usmerja predvsem k aktivnemu izvajanju različnih nalog vodenja, samostojnemu odločanju v primeru kritičnih situacij, prilagajanju načina vodenja trenutnim razmeram in doseganju optimalnega režima obratovanja vodenega procesa. V mnogih primerih nekateri razlogi, kakor so narava tehnološkega procesa, fizične razdalje med posameznimi objekti vodenja, zahteve po visoki stopnji zanesljivosti in avtonomnosti delovanja sistema za vodenje, narekujejo uporabo porazdeljenega računalniškega sistema. Zmogljivost enoračunalniškega sistema vodenja je določena z njegovimi tehničnimi značilnostmi (hitrost izvajanja, kapaciteta pomnilnika, itd) in sistem vodenja je operativen le, če implementirani algoritem ne presega časovnih in prostorskih omejitev. Pri porazdeljenem sistemu pa imamo možnost, da prostorske omejitve premostimo z dodajanjem novih procesnih enot v sistem. Časovnim omejitvam zadostimo s sočasnostjo izvajanja modulov algoritma, ki jih porazdelimo med posamezne procesne enote sistema vodenja. Sočasnost izvajanja posameznih modulov algoritma vodenja na porazdeljenem računalniškem sistemu pospeši (angl. *speedup*) izvajanje algoritma, kar daje načrtovalcu sistema vodenja možnost, da z izbiro ustreznega števila procesnih enot zagotovi izvajanje algoritma v realnem času.

Pospešitev izvajanja algoritma vodenja je (ob zanemarjanju tehnoloških značilnosti sistema) odvisna predvsem od števila uporabljenih procesnih enot in od strukture algoritma (od vsebovane vzporednosti).

Znano je, da na stopnjo pospešitve vplivata tudi zaporedje (časovna razvrstitev) izvajanja modulov algoritma in način razmestitve posameznih modulov med procesnimi enotami računalniškega sistema. Stopnjo vpliva razvrstitve na čas izvajanja algoritma kvantitativno definira Grahamov teorem /QUIN87/, ki zagotavlja, da je razmerje časov izvajanja v primeru najslabše oziroma najboljše razvrstitve ob upoštevanju načela kontinuitete izvajanja, manjše od 2 (*načelo kontinuitete izvajanja:* modul, ki je pripravljen za izvajanje, se začne izvajati takoj, ko se v sistemu pojavi prosta procesna enota /EAGE89/). Ker omenjeni teorem določa le zgornjo mejo vpliva razvrstitve in ker imamo z ustrezno razvrstitvijo možnost povečanja učinkovitosti porazdeljenega računalniškega sistema oziroma prihranka na številu uporabljenih procesnih enot v sistemu, smo se

odločili za analizo, ki naj bi podala dejanski vpliv razvrstitve na pospešitev pri porazdeljenih računalniških sistemih za vodenje (industrijskih) procesov.

V ta namen smo programsko opremo, ki predstavlja algoritem vodenja, predstavili z modelom v obliki usmerjenega acikličnega grafa. Na osnovi večjega števila grafov (testne množice) s strukturo, značilno za algoritme vodenja industrijskih procesov, smo določili dejanski vpliv načina razvrstitve na stopnjo pospešitve pri algoritmih za procesno vodenje in si s tem odgovorili na vprašanje o pomenu določitve dobrega načina razvrstitve izvajanja modulov pri zasnovi porazdeljenih računalniških sistemov za vodenje procesov.

2. VLOGA PORAZDELJENIH SISTEMOV PRI VODENJU INDUSTRIJSKIH PROCESOV

Kompleksnost problemov vodenja zahtevnih industrijskih procesov, pojavljanje manjših, cenenih mikroračunalnikov in obstoj zanesljivih komunikacijskih povezav so bili vzroki, ki so omogočili, da se je razvoj računalniško podprtih sistemov za vodenje industrijskih procesov usmeril v uporabo porazdeljenih sistemov. Lahko trdimo, da je prav vodenje procesov eno izmed področij uporabe računalniške tehnologije, kjer se je uporaba distribuiranih sistemov pokazala za najbolj upravičeno in učinkovito.

Lastnosti porazdeljenih sistemov /SLOM87, SCHO84/ omogočajo izpolnjevanje vseh specifičnih zahtev do sistema za vodenje procesa /BENN82/, predvsem v smislu delovanja v realnem času, zanesljivosti obratovanja, dopolnjevanja sistema in vzdrževanja. Namestitev računalnikov neposredno ob samih objektih vodenja operaterjem omogoča avtonomno vodenje in nadzor podsistemov objekta vodenja, zagotavlja hitrejše odzivne čase ter zmanjšuje stroške ožičenja in vpliv motenj. Prav tako je v porazdeljen sistem možna vključitev namenskih procesnih enot, kot tudi povezava z ostalimi računalniškimi sistemi, ki neposredno niso vključeni v proces vodenja proizvodnje.

Osnovna zahteva do sistema vodenja je zmožnost njegovega ***odziva na spremembe stanja vodenega objekta v vnaprej predpisanem času*** (delovanje v realnem času). Ustrezen način odziva določa algoritem vodenja. Ker se algoritem vodenja na posameznem (enoprocesorskem) računalniku lahko izvaja le sekvenčno, lahko pride do prekoračitve časa, znotraj katerega se morajo izvesti vse operacije sistema vodenja kot odgovor na spremembo stanja vodenega objekta in sistem vodenja tako postane s stališča zmožnosti vodenja neuporaben. Vzroka za prekoračitev časovnih omejitev sta predvsem v:

- kompleksnosti funkcij sistema vodenja, ki se morajo izvesti kot odgovor na spremembo stanja v vodenem sistemu,
- veliki dinamiki vodenega sistema, kar se odraža na velikem številu sprememb parametrov sistema v časovni enoti in ustreznem številu potrebnih reakcij sistema vodenja.

Ena izmed rešitev problema je uporaba hitrega (in dragega) računalnika, pri čemer se poleg cene pojavlja še vprašanje primernosti in dobavljivosti take naprave. Drugo rešitev predstavlja razporeditev funkcij med več manj zmogljivih procesnih enot (porazdeljen računalniški sistem), s katerimi skušamo izrabiti možnost sočasnega izvajanja delov algoritma vodenja in s tem zadostiti časovnim omejitvam. Pri načrtovanju sistemov vodenja industrijskih procesov ni poudarek na reševanju problema optimalne arhitekture računalniškega sistema ali problema čimbolj enakomerne zasedenosti posameznih enot, temveč je osnovni cilj doseči pri dani zasnovi strojne opreme in danem algoritmu delovanje sistema vodenja v realnem času, torej izvajanje vseh potrebnih modulov algoritma vodenja znotraj časovnega intervala, ki ga določa narava vodenega procesa.

3. MODEL PORAZDELJENEGA SISTEMA

Splošni model večračunalniškega sistema predstavljata množica procesnih enot in množica modulov razdrobljenega algoritma, ki skupaj zagotavljata izvajanje predpisanih funkcij sistema. Tak model štejemo kot deterministični, če so informacije, ki opisujejo značilnosti elementov sistema, znane vnaprej. Tipične informacije, ki jih potrebujemo, so npr. čas izvajanja posameznega modula, število modulov, število procesnih enot, omejitve v vrstnem redu izvajanja posameznih modulov, lastnosti procesnih enot, itd. Množico resursov (procesnih enot) lahko preprosto definiramo kot množico $P = (P_1, P_2, P_3, ..., P_n)$. Glede na specifične primere so lahko elementi množice P bodisi identični, identični v funkcionalnih zmogljivostih, a različni po hitrosti izvajanja, ali različni tako po funkcijah kot po hitrosti izvajanja.

Množico dekomponiranih modulov splošnega algoritma vodenja lahko v povezavi z množico procesnih enot formalno predstavimo kot peterko $(T, \prec, [\ t_{ij}\ ], R_j, \{\ w_j\ \})$, kjer:

1. $T = (T_1, T_2, ..., T_m)$ predstavlja množico modulov implementiranega algoritma, ki naj se izvaja,
2. " $\prec$ " predstavlja (irefleksivno) relacijo delne urejenosti, ki določa omejitve vrstnega reda izvajanja. Relacija med dvema elementoma T_i in T_j iz množice T, $T_i \prec T_j$ tako določa, da se mora izvajanje modula T_i končati pred začetkom izvajanja modula T_j,
3. $[\ t_{ij}\ ]$ predstavlja matriko časov izvajanja dimenzije n x m, kjer je $t_{ij}>0$ in pomeni čas, potreben za izvedbo modula T_j, $1 \le j \le m$ na procesni enoti P_i, $1 \le i \le n$. Če velja $t_{ij} = \infty$, to

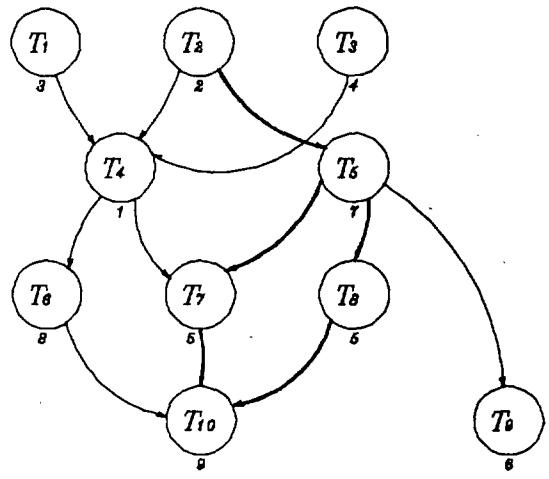
pomeni, da se modul T_j ne more izvajati na procesni enoti P_i. Za vsak j obstoja vsaj en i tak, da $t_{ij} < \infty$. V primeru identičnosti procesnih enot nam t_j predstavlja čas izvajanja modula T_j na eni od procesnih enot. Začetni čas izvajanja modula označimo z s_{ij} in čas, ko se konča izvajanje modula T_j na procesni enoti P_i z f_{ij}.

4. $R_j = [\ R_1(T_j), \ldots, R_s(T_j)\ ]$, $1 \le j \le m$, določa količino dodatnih resursov R, potrebnih za izvedbo modula T_j.
5. uteži $\{\ w_i\ \}$, $1 \le i \le m$, so poljubne funkcije razvrstitve Ψ in predstavljajo ceno, ki jo določa specifična razvrstitev Ψ glede na posamezen modul T_i. Navadno jih tretiramo kot konstante. Tako npr. velja, da je "cena" izvajanja modula T_i v času t preprosto $w_i.t$.

Opisana formulacija sistema modulov splošnega algoritma vsebuje precej več posplošitev, kot jih nameravamo uporabljati v naši analizi. Večina avtorjev uporablja za določitev splošnega algoritma le točke 1. do 3., to je število modulov, njihovo medsebojno odvisnost v smislu časovnega zaporedja izvajanja in čas, potreben za izvajanje posameznega modula.

Relacijo delne urejenosti $\prec$ je zelo primerno predstaviti v obliki ***usmerjenega acikličnega grafa***, kjer jo predstavimo z množico povezav med vozlišči grafa /TENE86, McHU90, QUIN87/. Z dekompozicijo algoritma na posamezne module razbijemo enovit algoritem v množico modulov $T=(T_1, T_2, \ldots, T_m)$. Če obstojajo medsebojne povezave med moduli (podatkovni pretoki, sinhronizacijski signali, itd), potem eksistira neka delna urejenost med moduli, ki jo definira binarna relacija imenovana "***precedenčnost***" (prevod: *predhodnost, prednost (v času)*), ki delno definira potrebni vrstni red izvajanja posameznih modulov. Množico modulov algoritma in precedenčne relacije med njimi lahko ponazorimo z grafom $G(V,A)$ (slika 1), kjer V določa množico uteženih vozlišč grafa (modulov algoritma) in A množico usmerjenih povezav v grafu (podatkovne, sinhronizacijske in druge povezave), ki ponazarjajo precedenčne relacije.

V ***usmerjenem uteženem acikličnem grafu*** lahko določimo najdaljšo pot iz začetnega vozlišča do končnega vozlišča in na ta način izračunamo minimalni možni čas, ki nam v splošnem zagotavlja izvršitev (enega cikla) algoritma. Minimalni možni čas algoritma je enak kritični poti v grafu /CONW67, McHU90/ in ni odvisen od števila uporabljenih računalnikov v sistemu, temveč samo od časa izvajanja posameznih modulov in njihovih medsebojnih povezav. Na osnovi analize kritične poti lahko analitično ugotovimo, če je v algoritmu zadostna stopnja paralelizma, ki pri določeni strojni opremi omogoča izvajanje algoritma v predpisanem času.



Slika 1: *Primer usmerjenega uteženega acikličnega grafa*

4. RAZVRSTITEV IZVAJANJA MODULOV

Časovno zaporedje izvajanja posameznih modulov je določeno s tem, da se posamezen modul lahko začne izvajati takrat, ko so se izvedli vsi njegovi predhodniki in ko obstoja prosta procesna enota, na kateri se modul lahko izvaja. Vrstni red izvajanja v splošnem ni enolično določen, saj lahko hkrati nastopa več modulov, ki nimajo (neizvedenih) predhodnikov in so tako kandidati za izvajanje. Potrebna je neka strategija ***določitve vrstnega reda izvajanja*** Ψ_T in ***razmestitve*** Ψ_P izvajanja modulov po posameznih procesnih enotah, ki omogoča pravilno delovanje algoritma ob upoštevanju danih omejitev in jo lahko imenujemo ***strategija razvrstitve*** Ψ. Problem razvrstitve izvajanja modulov algoritma v večračunalniškem sistemu vsebuje torej:

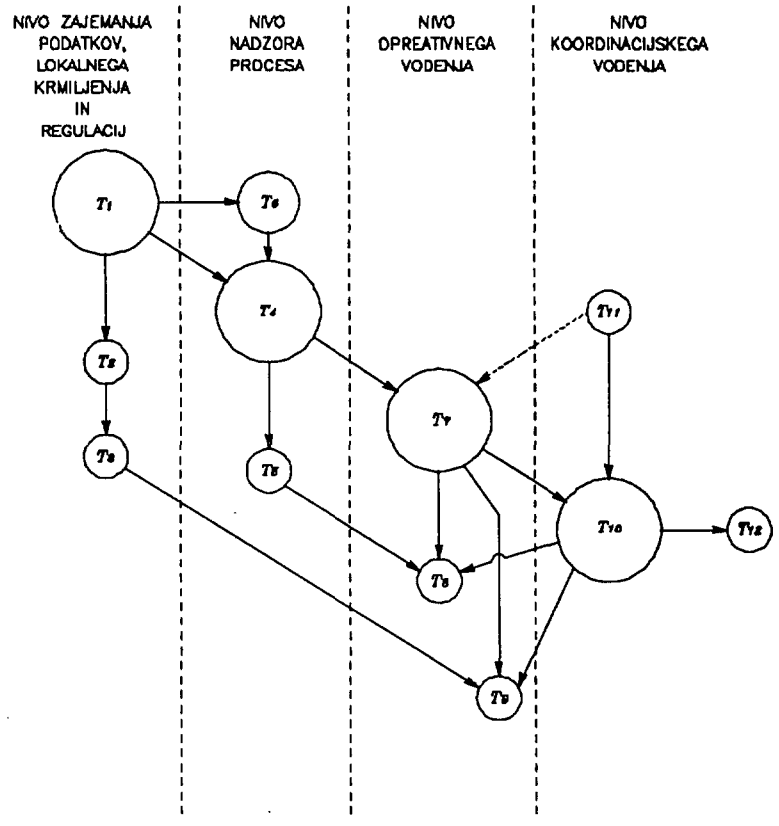
- časovno komponento problema, ki določa časovno razvrstitev izvajanja modulov,
- prostorsko komponento problema, ki rešuje vprašanje mesta izvajanja posameznega modula v celotnem sistemu,

torej $\Psi = f(\Psi_T, \Psi_P)$.

Pri načrtovanju sistemov vodenja industrijskih procesov so sistemske funkcije znane vnaprej; torej so znane tudi značilnosti programskih modulov kot preslikave sistemskih funkcij v realizirano programsko opremo. Zato razvrstitev modulov algoritma sistema procesnega vodenja po eni izmed uveljavljenih klasifikacij /CASA88/ opredelimo kot statično, globalno in suboptimalno, ker jo ob poznanih značilnostih modulov (čas trajanja, načini in vrste povezav) lahko za vse module izvedemo vnaprej, za rezultat razvrstitve pa ni nujno, da je optimalen, temveč je lahko suboptimalen v smislu zagotavljanja izvajanja algoritma znotraj predpisanih časovnih omejitev.

5. VPLIV NAČINA RAZVRSTITVE ALGORITMA VODENJA NA STOPNJO POSPEŠITVE

Pri računalniško vodenem procesu je računalniški sistem objekt, ki obvladuje informacijski tok in ga v skladu z zahtevami vodenja zbira, prilagaja, usmerja, transformira in shranjuje. Način izvajanja teh funkcij določa algoritem vodenja kot sestavni del računalniškega sistema vodenja. Algoritem vodenja je za vsak konkreten industrijski proces specifičen, vendar ima skupne osnovne značilnosti, ki izhajajo iz koncepta vodenja procesov oziroma nalog, ki jih izvaja sistem vodenja /MATK87/. Za izvedbo vodenja mora algoritem vodenja omogočati sprotno zajemanje procesnih podatkov, njihovo obdelavo, prikaz, izvedbo različnih računskih in logičnih operacij nad podatki z namenom delovanja krmilnih, regulacijskih, nadzornih, optimizacijskih in drugih funkcij sistema, pošiljanje krmilnih in regulacijskih signalov v vodeni proces, kratkoročno in dolgoročno shranjevanje podatkov in povezavo sistema vodenja z okoljem. Posamezne funkcije splošnega algoritma vodenja so realizirane s programskimi moduli, ki v iteraciji izvajajo operacije nad podatki in katerih začetek izvajanja je pogojen z dostopnostjo potrebnih vhodnih podatkov, ki so bodisi merjene procesne veličine, ročno vpisani podatki ali ukazi, rezultati, ki jih posredujejo drugi moduli, itd. Če module tretiramo kot vozlišča in podatkovne povezave med moduli opišemo z usmerjenimi povezavami med vozlišči, lahko splošno strukturo algoritma vodenja prikažemo z usmerjenim acikličnim grafom (slika 2), iz katerega je razvidna tudi hierarhična soodvisnost funkcij vodenja.



T1 – zajemanje podatkov iz procesa,predobdelava podatkov
T2 – enostavni algoritmi lokalnega krmiljenja in regulacije
T3 – pošiljanje izhodnih signalov lokalnega krmiljenja in regulacije v proces
T4 – ugotavljanje trenutnega stanja procesa
T5 – obravnavanje alarmov
T6 – prikaz trenutnega stanja procesa
T7 – operativno vodenje procesa,lokalna optimizacija
T8 – shranjevanje podatkov,protokoliranje
T9 – pošiljanje izhodnih signalov operativnega vodenja v proces
T10 – koordinacija,globalna optimizacija
T11 – sprejem podatkov iz drugega (višjega) sistema vodenja
T12 – oddaja podatkov drugemu (višjemu) sistemu vodenja

Slika 2: *Struktura splošnega algoritma vodenja v obliki splošnega usmerjenega acikličnega grafa*

Začetna vozlišča grafa algoritma vodenja predstavljajo moduli za zajemanje procesnih podatkov in sprejemanje podatkov oziroma navodil z drugega hierarhično višjega sistema vodenja. Končna vozlišča grafa so moduli za pošiljanje izhodnih signalov v proces, shranjevanje podatkov o stanju sistema, protokoliranje in povezavo z hierarhično višjim nivojem vodenja. Trajanje enega cikla je odvisno od konkretne izvedbe algoritma vodenja, v splošnem pa lahko rečemo, da kritična pot vodi od zajemanja procesnih podatkov, ugotavljanja trenutnega stanja sistema, operativnega vodenja do koordinacije vodenja procesa z okolico.

Obsežnost, vsebina in čas izvajanja posameznega modula so odvisni od konkretne realizacije vodenja na določenem procesu. V splošnem velja, da moduli na nižjem nivoju izvajajo vrsto enostavnejših in medsebojno neodvisnih operacij nad velikim številom potrebnih podatkov, moduli na višjih nivojih pa kompleksnejše algoritme nad relativno manjšim naborom podatkov. Zato je možna razdrobitev osnovnih modulov algoritma vodenja na več manjših neodvisnih ali šibko podatkovno povezanih modulov in njihova razvrstitev med več procesnih enot, kar pa spet nudi možnosti sočasnosti izvajanja določenih modulov in s tem povezanega hitrejšega izvajanja algoritma vodenja.

Iz teorema o razmerjih časov izvajanja pri različnih razvrstitvah /ONDA84/ lahko ugotovimo, da poljuben način razvrstitve ob upoštevanju načela kontinuitete izvajanja zagotavlja čas izvajanja cikla algoritma, ki je

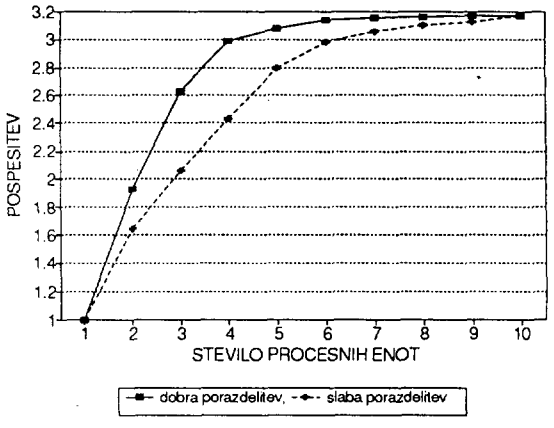
lahko do dvakrat daljši od časa izvajanja pri optimalni razvrstitvi. Zato je način razvrstitve pomemben dejavnik pri načrtovanju sistemov vodenja, posebno v mejnih primerih, ko primerna razvrstitev lahko odločilno skrajša čas izvajanja ali pa prihrani vgradnjo dodatne procesne enote v sistem. Prav tako primerna razvrstitev omogoča zmanjšanje pretoka podatkov med procesnimi enotami, kar razbremeni tako same procesne enote, predvsem pa zasedenost komunikacijske mreže.

V redkih primerih lahko dosežemo, da smiselna dekompozicija algoritma vodenja da kot rezultat množico modulov, ki so po času izvajanja enaki. Še redkeje so precedenčne relacije med dekomponiranimi moduli algoritma take, da relacija delne urejenosti predstavlja drevo. V obeh primerih imamo na voljo algoritma, ki dajeta optimalen rezultat /ONDA84/. V vseh ostalih različnih načinih dekompozicije algoritma, ki se odražajo na stopnji izkoriščenosti možnega paralelizma, velikosti posameznih modulov in različnih oblikah precedenčnih povezav, pa nimamo na voljo pravil, ki bi vnaprej zagotavljala uspešnost določenega načina razvrstitve. Nekateri, sicer uspešni postopki razvrstitve, včasih privedejo do vrste anomalij /COFF76/. Zato lahko uspešnost pristopa k problemu razvrstitve po določeni metodi lahko ocenimo le na osnovi uporabe metode na večjem številu primerov.

Ker je velika večina problemov razvrstitve NP-polnih /COFF76/, ni presenetljivo, da se teorija razvrstitve v zadnjem času usmerja na iskanje učinkovitih ***hevrističnih postopkov***. Ena zelo popularnih hevrističnih metod razvrstitve modulov je metoda imenovana LPT (angl. *Largest Processing Time*) ali HNF (angl. *Heavy Nodes First*), ki temelji na prednostnem izvajanju tistih modulov, ki so s stališča časa izvajanja največji /CONW67, SHIR89/. Metodo razvrščanja LPT smo uporabili tudi v našem primeru. Postopek časovnega razvrščanja po metodi LPT smo dopolnili z upoštevanjem nivoja vozlišča v grafu, ki v primeru, da imamo med moduli več kandidatov za izvajanje z enakimi časi, prednostno izbere za izvajanje modul z najvišjim nivojem, to je z največjo oddaljenostjo vozlišča od končnega vozlišča. Na testni množici 50-ih različnih grafov s strukturo, značilno za algoritme procesnega vodenja in s po 30-imi vozlišči, katerih uteži so bile naključno izbrane v velikostnem redu od 1 do 9, smo razvrstitev izvedli na dva načina:

- prvič po metodi LPT z upoštevanjem nivojev posameznih vozlišč,
- drugič po postopku, ki daje prednost izvajanju modulom z najmanjšim časom izvajanja in najnižjim nivojem.

Pri obeh metodah smo upoštevali načelo kontinuitete izvajanja. Na ta način smo določili za vse grafe v celotni testni množici stopnjo pospešitve v odvisnosti od načina razvrstitve zaporedja izvajanja posameznih modulov. Ker je znano, da je metoda LPT ena izmed hevrističnih metod razvrščanja, ki daje dobre rezultate glede na čas trajanja algoritma, in ker smo eksperimentalno ugotovili, da razvrščanje po drugi metodi daje slabe rezultate glede na ostale hevristične metode, smo na ta način dobili razmerje med povprečnimi časi izvajanja algoritma v primeru ***slabega*** in ***dobrega*** načina razvrščanja, ki ga navzgor omejuje že opisani Grahamov teorem na vrednost 2. Slika 3 prikazuje razliko med povprečjem doseženih vrednosti pospešitve v primeru uporabe dobre oziroma slabe strategije ravrstitve.



Slika 3: *Pospešitev v primeru dobre (HNF) in slabe porazdelitve pri številu modulov m=30 in različnem številu procesnih enot*

Povprečna razlika med doseženimi vrednostmi pospešitve je pri določenem številu uporabljenih procesnih enot več kot 20 odstotkov. Razlika stopnje pospešitve kot posledica dobrega in slabega načina razvrstitve je prisotna pri vseh grafih testne množice (npr. pri številu razpoložljivih procesnih enot n=2 je bila najmanjša razlika obeh pospešitev 7.5 odstotkov in največja razlika 30.3 odstotka). To v praksi potrjuje vpliv uporabljenega načina razvrstitve na pospešitev, po drugi strani pa potrjuje dejstvo, da vpliv porazdelitve ni tako velik, kot bi ga lahko pričakovali na osnovi Grahamovega teorema.

6. ZAKLJUČEK

Osnovna zahteva, ki jo mora načrtovalec računalniškega sistema procesnega vodenja v fazi načrtovanja upoštevati, je zmožnost delovanja sistema v realnem času. Predpisanim časovnim (in prostorskim) omejitvam lahko zadostimo z uporabo porazdeljenega računalniškega sistema, kar pa nam daje dodatno možnost, da z ustrezno razvrstitvijo vrstnega reda izvajanja posameznih modulov algoritma vodenja še zmanjšamo čas izvajanja (enega cikla) celotnega algoritma vodenja.

Namen pričujočega članka je bil praktično ovrednotiti vpliv razvrstitve časovnega izvajanja modulov na pospešitev izvajanja algoritma na porazdeljenem

računalniškem sistemu. Čeprav je stopnja pospešitve odvisna predvsem od števila uporabljenih procesnih enot, strukture samega algoritma vodenja in količine podatkov, ki se prenašajo med posameznimi enotami porazdeljenega sistema, smo ugotovili, da vsaj pri algoritmih s strukturo, ki je značilna za algoritme procesnega vodenja, vpliv načina razvrstitve izvajanja modulov na pospešitev ni zanemarljiv. Zato je v fazi načtovanja porazdeljenega sistema procesnega vodenja za konkreten algoritem vodenja potrebno določiti ustrezen vrstni red izvajanja posameznih modulov. S skrajšanjem časa izvajanja cikla algoritma dosežemo zanesljivejše delovanje sistema, v določenih primerih pa z ustrezno razvrstitvijo izvajanja modulov lahko zmanjšamo število uporabljenih procesnih enot v sistemu.

7. LITERATURA

/BENN82/: S. Bennet, D. A. Linkens: ***Computer Control of Industrial Processes,*** Peter Peregrinus Ltd, 1982

/CASA88/: T. L. Casavant, J. G. Kuhl: ***A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems***, IEEE Transactions on Software Engineering, Vol. 14, No. 2, 1988

/CONW67/: R. W. Conway, W. L. Maxwell, L. W. Miller: ***Theory of Scheduling***, Addison-Wesley Inc., 1967, pp. 132 - 140

/COFF76/: E. G. Coffman: ***Computer and Job-Shop Scheduling Theory***, John Wiley and sons, Inc., New York, 1976, pp. 2 - 50

/EAGE89/: D. L. Eager, J. Zahorjan, E. D. Lazowska: ***Speedup Versus Efficiency in Parallel Systems***, IEEE Transactions on Computers, Vol. 38, No. 3, marec 1989

/MATK87/: D. Matko, B. Zupančič: ***Računalniški sistemi v vodenju procesov***, 3. izdaja, Univerza v Ljubljani, Fakulteta za elektrotehniko, 1987

/McHU90/: J. A. McHugh: ***Algorithmic Graph Theory***, Prentice Hall Inc., 1990

/ONDA84/: J. Ondaš: ***Algorithms for Scheduling Homogeneus Multiprocessor Computers***, Algorithms,Software and Hardware of parallel Computers, uredila J. Mikloško in V. E. Kotov, VEDA, 1984

/QUIN87/: M. J. Quinn: ***Designing Efficient Algorithms for Parallel Computers***, McGraw-Hill Inc, 1987

/SCHO84/: J. D. Schoeffler: ***Distributed Computer Systems for Industrial Process Control***, IEEE Computer, February 1984

/SHIR89/: B. Shirazi, M. Wang: ***Heuristic Functions for Static Task Allocation***, Microprocessing and Microprogramming 26, 1989

/SLOM87/: M. Sloman, J. Kramer: ***Distributed Systems and Computer Networks***, Prentice-Hall Int., 1987

/TENE86/: A. M. Tenenbaum, M. J. Augenstein: ***Data Structures Using Pascal***, Prentice-Hall Inc., 1986, pp. 575-643

Keywords: evolutionary computation, genetic algorithms, adaptive search, stochastic optimization, machine learning

Bogdan Filipič
Odsek za računalništvo in informatiko
Institut Jožef Stefan
Jamova 39, Ljubljana

Genetski algoritmi so verjetnosti preiskovalni algoritmi, zasnovani na darvinističnih načelih evolucije bioloških sistemov. Njihovo bistvo je simulirana evolucija množice dopustnih rešitev obravnavanega problema. Genetske algoritme odlikujejo robustnost, učinkovitost in nizka računska kompleksnost. Uveljavili so se v problemih preiskovanja in optimizacije ter v avtomatskem učenju. Članek predstavlja njihove osnove na primeru t.i. enostavnega genetskega algoritma, povzema njegovo teoretično analizo, obravnava implementacijo in nekatere razširitve enostavnega modela ter področja uporabe genetskih algoritmov.

GENETIC ALGORITHMS — Genetic algorithms are stochastic search algorithms based on Darwinian principles of biological evolution. They key idea of genetic algorithms is to evolve a set of candidate solutions to a given problem. Genetic algorithms turned out to be robust and efficient in finding near-optimal solutions in complex problem spaces. They have been successfully applied to search, optimization and machine learning tasks. The paper summarizes the basic principles of genetic algorithms by presenting the so called simple genetic algorithm and its theoretical analysis, discusses the implementation and some extensions of the simple model, and reviews typical applications of genetic algorithms.

1. Uvod

V računalništvu zasledimo vrsto modelov in metod, zasnovanih po zgledih iz narave. Med njimi so tako pristopi, ki temeljijo na zakonitostih iz neživega sveta, kot tudi takšni, ki posnemajo dogajanje v bioloških sistemih. Znan primer prvih je verjetnostni optimizacijski algoritem simulirano ohlajanje (angl. *simulated annealing*) (van Laarhoven in Aarts, 1987), ki je analogen modelu procesa ohlajanja snovi iz statistične mehanike. V drugo skupino sodijo npr. celični avtomati, nevronske mreže in genetski algoritmi. Skupne značilnosti metod iz te skupine so sočasno obravnavanje množice enostavnih objektov, lokalnost, nehierarhična struktura in funkcionalnost, ki ni predpisana vnaprej, temveč je rezultat interakcij med obravnavanimi objekti (Langton, 1989).

Od omenjenih metod v tem delu predstavljamo *genetske algoritme* (Holland, 1975; Goldberg, 1989a; Rawlins, 1991). Ti temeljijo na načelih naravnega izbora in zakonih genetike. Razviti so bili z namenom pojasniti adaptivne mehanizme iz narave in jih uporabiti pri načrtovanju umetnih, računalniških sistemov. Pionirsko delo na tem področju je s svojimi sodelavci na Univerzi v Michiganu opravil John Holland, čigar monografija *Adaptation in Natural and Artificial Systems* (Holland, 1975) je temelj današnjih raziskav genetskih algoritmov.

Osnovni princip genetskih algoritmov je simulirana evolucija množice rešitev obravnavanega problema. Pristop lahko uporabljamo kot preiskovalno metodo, optimizacijsko metodo ali metodo avtomatskega učenja (Goldberg, 1989a). Genetske algoritme odlikujejo robustnost, učinkovitost in nizka računska kompleksnost. Pri reševanju problemov z njimi ne potrebujemo predznanja, temveč le ustrezno kodiranje rešitev in mero njihove uspešnosti. Na osnovi lokalne informacije o kvaliteti trenutnih rešitev algoritem z adaptivnim preiskovanjem sam odkrije globalne zakonitosti problemskega prostora. Zaradi tega je evolucijski pristop posebno primeren za reševanje zahtevnih nalog, pri katerih zaradi nelinearnosti,

nezveznosti, multimodalnosti in podobnih lastnosti mnoge druge metode odpovedo.

Od tradicionalnih optimizacijskih in preiskovalnih metod, kot so analitične in numerične metode, plezalni algoritmi (angl. *hill-climbing*), naštevne metode ipd., se genetski algoritmi razlikujejo po tem, da:

- operirajo s kodami parametrov problema in ne z njihovimi vrednostmi,
- sočasno obravnavajo množico rešitev in ne ene same rešitve,
- zahtevajo le vrednost kriterijske funkcije, ne pa tudi dodatnih informacij, kot so npr. vrednosti odvodov,
- so verjetnostni in ne deterministični algoritmi.

2. Enostavni genetski algoritem

Oglejmo si najprej preprosto varianto genetskega algoritma, iz literature znano pod imenom *enostavni genetski algoritem* (Goldberg, 1989a). Ta kljub svoji enostavnosti vsebuje vse osnovne mehanizme genetskih algoritmov, tako da je z njim mogoče reševati mnoge naloge. Zaradi svoje elementarnosti je hkrati zelo primeren za teoretično obravnavo.

2.1 Osnovni koncepti

Naj bodo točke v prostoru, ki ga preiskujemo z enostavnim genetskim algoritmom, predstavljene z nizi. Tako kodirane rešitve imenujemo *osebki*, kadar želimo poudariti njihovo strukturo, pa tudi *kromosomi*. Znake v nizu imenujemo *geni*. Včasih posebej omenjamo mesto (*lokus*) gena v kromosomu in njegovo konkretno vrednost (*alel*).

Vsaki točki v prostoru preiskovanja pripada vrednost kriterijske funkcije. Genetski algoritem ne operira s kriterijsko funkcijo neposredno, pač pa jo transformira v t.i. *funkcijo uspešnosti* (angl. *fitness function*). Z njo ocenjuje osebke. Za funkcijo uspešnosti zahtevamo, da je pozitivna in definirana tako, da osebkom, ki predstavljajo boljše rešitve, priredi višjo uspešnost. (Pri maksimizacijskih problemih so torej uspešnejše rešitve z višjo, pri minimizacijskih pa z nižjo vrednostjo kriterijske funkcije.) Tem zahtevam je moč zadostiti z linearno transformacijo kriterijske funkcije v funkcijo uspešnosti. Funkcija uspešnosti je ponavadi realna.

Množica osebkov sestavlja *populacijo*. Osebke začetne populacije največkrat generiramo naključno, lahko pa zanje uporabimo tudi kako drugače dobljene rezultate, ki jih želimo z genetskim algoritmom izboljšati. Populacijo procesiramo iterativno, v korakih, imenovanih *generacije*. Iz osebkov (*staršev*) iz trenutne populacije tvorimo naslednike (*potomce*), ki pripadajo novi populaciji. Velikost populacije se pri tem ne spreminja. Za generiranje naslednikov uporabljamo različne operatorje. V enostavnem genetskem algoritmu nastopajo *reprodukcija, križanje* in *mutacija*. Ti učinkujejo na osebke v vsaki generaciji.

Postopek traja, dokler ni izpolnjen ustavitveni pogoj. Ta je lahko podan na več načinov: s številom generacij, časom izvajanja, kvaliteto rešitev, njihovo konvergenco ali s kombinacijo teh kriterijev. Najpogosteje, posebno pri testiranju, predpišemo kar število generacij. Za rezultat štejemo najboljšo rešitev, ki jo je algoritem našel med izvajanjem. Osebek, ki ustreza tej rešitvi, ne pripada nujno končni populaciji, kajti potomci niso nujno uspešnejši od svojih staršev. Enostavni genetski algoritem lahko povzamemo takole:

```
Enostavni_genetski_algoritem(g, n, pc, pm);
begin
   Inicializiraj(populacija, n);
   repeat
      Reprodukcija(populacija,nova_populacija);
      Križanje(nova_populacija, pc);
      Mutacija(nova_populacija, pm);
      populacija := nova_populacija
   until ustavitveni_pogoj(g, populacija)
end;
```

V algoritmu nastopajoči parametri imajo naslednji pomen: g je predpisano število generacij, n število osebkov v populaciji, p_c verjetnost križanja in p_m verjetnost mutacije. Vlogo zadnjih dveh bomo pojasnili skupaj z opisom operatorjev.

2.2 Operatorji

V enostavnem genetskem algoritmu nastopajo trije operatorji: reprodukcija, križanje in mutacija. Z reprodukcijo se izvaja selekcija osebkov. Preživetje

uspešnih in odmiranje neuspešnih osebkov ima za posledico konvergenco od slabših k boljšim rešitvam obravnavanega problema. S križanjem in mutacijo variiramo osebke. Za razliko od reprodukcije ta dva operatorja povzročata spremembe v genetski strukturi osebkov in ju zato imenujemo tudi genetska operatorja. Opišimo delovanje operatorjev podrobneje.

2.2.1 Reprodukcija

Reprodukcija posnema načelo naravnega izbora, t.j. preživetja uspešnejših članov populacije. Operator multiplicira osebke glede na njihovo uspešnost. Večja kot je uspešnost osebka, večja je verjetnost, da bo ta osebek prispeval določeno število potomcev v naslednjo generacijo. Ker pa se velikost populacije skozi generacije ne spreminja, manj uspešni osebki ob tem odmirajo. Reprodukcija na ta način deluje selektivno.

Sposobnost preživetja osebka je definirana z razmerjem med njegovo uspešnostjo in povprečno uspešnostjo populacije. Naj bo velikost populacije n, vsota uspešnosti vseh osebkov v njej pa $\sum f$. Potem je pričakovano število potomcev, ki jih ob reprodukciji prispeva osebek i z uspešnostjo f_i, enako $n f_i / \sum f = f_i / \bar{f}$, kjer je $\bar{f}$ povprečna sposobnost osebkov v populaciji. Z drugimi besedami: nadpovprečni osebki v boju za obstoj preživijo, podpovprečni pa propadejo.

Implementacije reprodukcije so različne. Znana je izvedba z "ruleto", na kateri posamezni odseki predstavljajo osebke, širina odsekov pa je premosorazmerna uspešnosti osebkov. Potomce določamo z zadetki na tako uteženi ruleti. Slabost tega modela je možnost precejšnjega odstopanja dobljene porazdelitve potomcev od pričakovane zaradi verjetnostnega pristopa k izbiri potomcev. To slabost odpravlja t.i. deterministično vzorčenje, pri katerem izračunamo pričakovano števila naslednikov, vsak osebek reproduciramo v številu kopij, ki je enako celemu delu pričakovanega števila potomcev, ostanek nove populacije pa zapolnimo tako, da osebke prvotne populacije razvrstimo po padajočih vrednostih decimalnega dela pričakovanega števila potomcev, nakar ustrezno število začetnih osebkov s tako dobljenega seznama prispeva po enega potomca. Pogosto se uporablja tudi kombinacija determinističnega in verjetnostnega načina, pri katerem celi del pričakovanega števila potomcev obravnavamo deterministično, decimalni del pa verjetnostno (angl. *stochastic remainder selection*).

2.2.2 Križanje

Križanje (angl. *crossover*) je genetski operator, ki skupaj z reprodukcijo največ prispeva k moči genetskih algoritmov. Operator deluje nad pari osebkov in iz njih tvori pare potomcev. Potomci v populaciji nadomeščajo svoje starše.

Križanje poteka tako, da v populaciji naključno izberemo starševska niza. Zatem, prav tako naključno, določimo *mesto križanja* (angl. *crossing site*), t.j. celo število k z intervala $[1, l-1]$, kjer je l dolžina nizov. V zadnjem koraku med nizoma izmenjamo podniza med mestoma $k+1$ in l. Opisani način križanja imenujemo *enostavno* ali *enomestno* križanje. Primer enostavnega križanja dveh binarnih nizov vidimo na sliki 1.

Starša		Potomca	
1 0 1 1 0 1	1 1 1 0	1 0 1 1 0 1	1 0 0 1
0 0 0 1 1 0	1 0 0 1	0 0 0 1 1 0	1 1 1 0

Slika 1: Enostavno križanje.

Križanje zagotavlja izmenjavo genetskega materiala med osebki, skupaj z reprodukcijo pa širjenje tistih podnizov v populaciji, ki tvorijo dobre rešitve. Običajno ne križamo vseh osebkov v populaciji, pač pa operator uporabljamo z verjetnostjo p_c, ki je parameter genetskega algoritma. Pričakovano število osebkov, ki jih v populaciji nadomestimo z njihovimi potomci, dobljenimi s križanjem, je torej np_c.

2.2.3 Mutacija

Drugi genetski operator—mutacija—je v primerjavi s križanjem drugotnega pomena. Deluje tako, da naključno spreminja vrednosti znakov v nizih. Vloga mutacije je v preprečevanju nepovratnih izgub koristnih podnizov, ki jih lahko povzročita reprodukcija in križanje. Na mutacijo samo lahko gledamo kot na naključno preiskovanje problemskega prostora.

Mutacijo ponavadi uporabljamo z zelo majhno verjetnostjo, predpisano s parametrom algo-

ritma p_m. Za razliko od križanja, kjer verjetnost pomeni pričakovani delež osebkov v populaciji, ki se bodo križali, ta parameter pri mutaciji določa pričakovani delež mutiranih genov v populaciji.

3. Matematične osnove genetskih algoritmov

Ugotovimo lahko, da je genetski algoritem zelo preprost, saj vključuje le elementarne operacije nad nizi, ki predstavljajo rešitve danega problema. V praksi se izkaže, da tako simulirana evolucija pogosto vodi k zelo dobrim rešitvam. Poglejmo zato, kako delovanje genetskih algoritmov pojasnjuje teorija. Po knjigi (Goldberg, 1989a) bomo povzeli izpeljavo osnovnega izreka o genetskih algoritmih. V ta namen najprej definirajmo pripomoček za opisovanje podobnosti med nizi, t.i. *sheme.*

3.1 Sheme

Naj bodo rešitve problema kodirane z nizi dolžine l nad abecedo A. Dodajmo ji nov znak $*$, tako da dobimo razširjeno abecedo $A^+ = A \cup \{*\}$. Znak $*$ naj ima poseben pomen: označuje naj poljuben znak iz abecede A. Sheme (angl. *schemata*) tedaj definiramo kot nize dolžine l nad abecedo A^+.

Sheme so kompakten način zapisovanja podobnosti med nizi. Nanje lahko gledamo kot na vzorce, katerim ustrezajo konkretni nizi v populaciji. Ponazorimo to s primeri. Naj bo $A = \{0, 1\}$ in $l = 4$. Shemi $0{*}{*}1$ ustreza podmnožica nizov $\{0001, 0011, 0101, 0111\}$. Shemi 1100 ustreza le niz 1100, shemi $****$ pa ustrezajo vsi binarni nizi dolžine 4.

Za nize dolžine l nad abecedo A, kjer je $|A| = k$, obstaja $(k+1)^l$ možnih shem. Po drugi strani je posamezen niz dolžine l nad abecedo A predstavnik 2^l shem, t.j. vseh nizov z $0, 1, 2, \ldots, l$ znaki $*$ namesto originalnih znakov. V populaciji n nizov dolžine l nad abecedo A je tako zastopanih med 2^l in $n2^l$ shem. Natančno število je odvisno od raznolikosti populacije.

Definirajmo še dve lastnosti shem. Imenujmo število fiksnih, t.j. od $*$ različnih znakov v shemi, *red sheme* (angl. *schema order*) in ga označimo z o. Primera: $o(01*1) = 3$ in $o(***1) = 1$. Razdaljo med prvim in zadnjim fiksnim mestom v shemi pa imenujmo *aktivna dolžina* (angl. *defining length*) sheme in jo označimo z δ. Primera: $\delta(01*1) = 3$ in $\delta(***1) = 0$.

3.2 Osnovni izrek o genetskih algoritmih

Analizirajmo sedaj vpliv reprodukcije in genetskih operatorjev križanja in mutacije na število predstavnikov določene sheme v populaciji. Opazujmo neko konkretno shemo H. Število predstavnikov sheme H v populaciji ob času t označimo z $m(H, t)$. To število je seveda različno za različne sheme in se s časom spreminja. Čas merimo v generacijah.

Vpliv reprodukcije na pričakovano število predstavnikov sheme H v naslednji generaciji je enostavno ugotoviti. Če upoštevamo, da je pričakovano število potomcev, ki jih ob reprodukciji v času t prispeva niz z uspešnostjo f_i enako $f_i / \bar{f}(t)$, kjer je $\bar{f}$ povprečna sposobnost osebkov v populaciji, lahko zapišemo

$$m(H, t+1) = m(H, t) \frac{f(H, t)}{\bar{f}(t)}, \qquad (1)$$

pri čemer je $f(H, t)$ povprečna sposobnost nizov, ki ustrezajo shemi H v času t. S tem imamo tudi za sheme zapisano selekcijsko načelo o povečevanju števila nadpovprečnih in odmiranju podpovprečnih primerkov. Ali rast števila predstavnikov uspešnih shem lahko natančneje opišemo? Predpostavimo, da velja $f(H, t) = \bar{f}(t)(1 + c)$, kjer je c konstanta. Potem enačbo (1) lahko zapišemo tudi takole:

$$m(H, t+1) = m(H, t)(1 + c). \qquad (2)$$

Začenši s časom $t = 0$ pa ob privzeti stacionarni vrednosti c dobimo

$$m(H, t+1) = m(H, 0)(1 + c)^t. \qquad (3)$$

Enačba (3) definira geometrijsko zaporedje, s katerim je opisan učinek reprodukcije na število predstavnikov sheme H. Vidimo, da reprodukcija povzroči eksponentno rast števila predstavnikov nadpovprečnih in eksponentno upadanje števila predstavnikov podpovprečnih shem v populaciji.

Vpliv križanja proučimo preko verjetnosti, da shema H "preživi" križanje, t.j. da mesto križanja ne bo med prvim in zadnjim fiksnim mestom v shemi. Mesto križanja lahko izberemo na

$l-1$ načinov, kjer je l dolžina sheme. Od tega je $\delta(H)$ križanj destruktivnih, ker "prerežejo" shemo H. Če predpostavimo enakomerno izbiranje mest križanja, je verjetnost izgube sheme zaradi križanja enaka $\delta(H)/(l-1)$. Križanje, katerega verjetnost je p_c, bo torej shema H preživela z verjetnostjo

$$p_{sc}(H) = 1 - p_c \frac{\delta(H)}{l-1}, \qquad (4)$$

ali povedano drugače, križanje najmanj destruktivno učinkuje na sheme s kratko aktivno dolžino.

Analizirajmo podobno še vpliv mutacije. Verjetnost spremembe posameznega gena zaradi mutacije je p_m, verjetnost njegovega preživetja pa $1-p_m$. Shema H bo preživela mutacijo, če nobeno od $o(H)$ fiksnih mest v njej ne bo prizadeto. Zaradi neodvisnosti mutacij je verjetnost tega dogodka p_{sm} enaka $(1-p_m)^{o(H)}$. Ker je $p_m \ll 1$, lahko poenostavimo:

$$p_{sm}(H) = 1 - o(H)p_m. \qquad (5)$$

Mutacijo torej bolj verjetno preživijo sheme nizkega reda.

Z združitvijo (1), (4) in (5) lahko sedaj opišemo skupen učinek reprodukcije, križanja in mutacije na pričakovano število predstavnikov določene sheme H v naslednji generaciji:

$$m(H,t+1) \geq m(H,t)\frac{f(H,t)}{\bar{f}(t)} \left[1 - p_c \frac{\delta(H)}{l-1} - o(H)p_m\right]. \qquad (6)$$

Tako smo dobili izrek o shemah, imenovan tudi osnovni izrek o genetskih algoritmih (Holland, 1975). Povzamemo ga takole: med evolucijo populacije z genetskim algoritmom se z generacijami eksponentno povečuje število predstavnikov nadpovprečno uspešnih shem s kratko aktivno dolžino in nizkim redom.

3.3 Hipoteza o gradnikih in implicitni paralelizem

Izrek o shemah pojasnjuje delovanje genetskih algoritmov. Namesto direktnega konstruiranja rešitev le-ti z odkrivanjem in kombiniranjem predstavnikov uspešnih shem postopoma sestavljajo čedalje boljše rešitve. Pri tem igrajo posebno vlogo nadpovprečne sheme kratke aktivne dolžine in nizkega reda. Take sheme zato imenujemo *gradniki* (angl. *building blocks*).

Procesiranje shem v genetskih algoritmih je analiziral že Holland (1975). Izpeljal je oceno števila koristnih shem, t.j. takšnih, za katere število predstavnikov v populaciji zaradi prej omenjenih lastnosti eksponentno raste. Po tej oceni je pri procesiranju n nizov $O(n^3)$ takšnih shem. Tej lastnosti genetskih algoritmov pravimo *implicitni paralelizem.*

Omenimo v zvezi s shemami še en zanimiv teoretičen rezultat. To je *funkcija shem* (angl. *schema function*), s katero Goldberg (1989b) popiše število različnih shem v populaciji. Rezultat je splošen v toliko, da ne obravnava le binarnega pač pa k-vrednostno kodiranje in možnost upoštevanja sheme šele, če je v populaciji prisotnih določeno število nizov, ki tej shemi ustrezajo. S pomočjo te funkcije avtor za serijski in paralelni genetski algoritem izpelje optimalne velikosti populacij, pri katerih se v povprečju sprocesira največje število shem na generacijo.

4. Implementacija genetskega algoritma

4.1 Kako kodirati rešitve

Spoznali smo, da v delovanju genetskega algoritma igrajo osrednjo vlogo značilni vzorci—sheme, ki se pojavljajo v nizih. Z nagrajevanjem uspešnejših nizov in njihovim kombiniranjem algoritem gradi čedalje boljše rešitve. Kako torej zagotoviti, da bo opisan proces kar najbolj učinkovit? Če se zavedamo, da algoritem procesira sheme posredno preko nizov, je odgovor na dlani: poskrbeti je potrebno za ustrezno kodiranje rešitev. To dosežemo z upoštevanjem dveh principov kodiranja: *principa pomembnih gradnikov* in *principa najmanjših možnih abeced.* Po prvem rešitve problema predstavimo tako, da v predstavitvi nastopajo za reševanje konkretnega problema smiselni gradniki, ki vodijo k dobrim rešitvam. Po drugem uporabimo za kodiranje najmanjšo abecedo, ki omogoča smiselno predstavitev problema. Uporaba skromne abecede ima za posledico bogatejšo strukturo zapisov in s tem več možnosti za uspešnost genetskega preiskovanja.

Način kodiranja izberemo v vsakem primeru

posebej. Včasih se nam sama po sebi ponuja naravna rešitev, ki obenem zadošča omenjenim principom, v nekaterih primerih pa jo je teže določiti. Kljub enostavnosti zahtev za kodiranje zanj ni predpisanega postopka, pač pa je v veliki meri stvar intuicije in izkušenj.

4.2 Kako določiti vrednosti parametrov algoritma

V genetskih algoritmih nastopa več numeričnih parametrov. V enostavnem genetskem algoritmu smo spoznali naslednje:

- število generacij,
- velikost populacije,
- verjetnost križanja in
- verjetnost mutacije.

Z različnimi razširitvami algoritma nastopijo še dodatni parametri. V zvezi s parametri se seveda pojavlja vprašanje, kako določiti njihove vrednosti, da bo algoritem kar najbolj učinkovit in bo hkrati dal dobre rezultate. Za velikost populacije npr. velja, da premajhno število osebkov ne vsebuje zadosti shem, ki bi vodile k dobrim rešitvam, v prevelikih populacijah pa je njihovo procesiranje premalo učinkovito, zaradi česar rešitve konvergirajo prepočasi. Podobno je z verjetnostjo uporabe operatorja križanja. Prenizka povzroča stagniranje preiskovanja, pri previsoki pa uspešni osebki odmirajo hitreje, kot se utegnejo reproducirati.

Spet moramo ugotoviti, da tudi za določanje vrednosti parametrov genetskih algoritmov ni pravil ali formul. Te vrednosti pri reševanju konkretnega problema najpogosteje določimo empirično, na osnovi začetnih eksperimentov. "Optimalne" vrednosti parametrov so v precejšnji meri odvisne tudi od problema samega. Na srečo pa so genetski algoritmi dokaj neobčutljivi na vrednosti parametrov. V večini primerov se namreč izkaže, da genetski algoritem približno enako deluje pri vrednostih parametrov z določenih intervalov in ne le pri točno določenih vrednostih. Za orientacijo navedimo najpogostejše vrednosti, ki jih najdemo v literaturi. Verjetnost križanja je ponavadi med 0.5 in 1. Mutacija je dosti redkejši dogodek. Njena verjetnost je ponavadi nekaj odstotkov. Običajna velikost populacije je od nekaj deset do nekaj sto osebkov. Število generacij izmed vseh omenjenih parametrov še najbolj variira, od nekaj deset do več tisoč.

Podrobnejše študije učinka parametrov algoritma so bile narejene predvsem v optimiranju funkcij (npr. De Jong, 1975; Schaffer in sod., 1989). Znanih pa je tudi nekaj načrtnih načinov določanja vrednosti parametrov. Grefenstette (1986) je v ta namen uporabil evolucijski pristop. Definiral je t.i. *meta genetski algoritem*, to je genetski algoritem, ki kot osebke obravnava genetske algoritme za reševanje danega problema, ali natančneje, nabore vrednosti njihovih parametrov. Optimira jih glede na kvaliteto in konvergenco rezultatov. Meta genetski algoritem sicer poišče dobre vrednosti parametrov, njegova slabost pa je relativna neučinkovitost. Adaptivno, vendar bolj učinkovito problem rešuje Davis (1989). Njegov genetski algoritem vsebuje različne modifikacije genetskih operatorjev, med izvajanjem pa spremlja njihov prispevek k izboljšanju rešitev in skladno z njim spreminja verjetnosti uporabe operatorjev.

5. Razširitve enostavnega genetskega algoritma

Našteli bomo nekaj razširitev enostavnega genetskega algoritma, ki omogočajo reševanje zahtevnejših problemov. Nanašale se bodo na kodiranje rešitev, funkcijo uspešnosti, genetske operatorje in paralelizacijo genetskih algoritmov.

5.1 Kodiranje rešitev

V zvezi s kodiranjem omenimo, da probleme z več parametri oz. spremenljivkami obravnavamo analogno enoparametrskim problemom. Vsakega od parametrov kodiramo z nizom, dobljene nize pa združimo v niz—kromosom, ki predstavlja rešitev problema. Genetski operatorji ponavadi obravnavajo kromosome ne oziraje se na njihovo delitev na podnize, ki ustrezajo posameznim parametrom.

Dinamično kodiranje parametrov (Schraudolph in Belew, 1992) je dodaten mehanizem, ki omogoča usmerjanje genetskega preiskovanja v obetavnejša področja problemskega prostora in izboljševanje natančnosti rezultatov. To dosežemo s prilagajanjem interpretacije rešitev med izvajanjem algoritma. Preiskovanje pričnemo na širokih intervalih in z velikim diskretizacijskim korakom, postopama pa ožimo intervale preiskovanja in zmanjšujemo korak. Predstavitev pri tem ostaja sintaktično nespremenjena.

Za kodiranje rešitev so v uporabi tudi drugačne strukture, ne le nizi. Varšek (1991) se z genetskim algoritmom uči kvalitativnih modelov, predstavljenih z drevesi, in za njihovo evolucijo uporablja posebej prirejene operatorje. Antonisse in Keller (1987) obravnavata kodiranje in operatorje za visokonivojsko predstavitev znanja, ki vključuje objekte, relacije in pravila.

5.2 Funkcija uspešnosti

Funkcijo uspešnosti pogosto *skaliramo*, tako da omejimo razmerje med uspešnostjo najboljšega in najslabšega osebka v populaciji. Namen skaliranja je preprečiti prezgodnjo konvergenco populacije, ki je rezultat prevlade močno nadpovprečnega osebka, slučajno generiranega v zgodnji fazi evolucije. S skaliranjem uspešnosti je število potomcev, generiranih med reprodukcijo, enakomerneje porazdeljeno med osebke, kar ohranja raznolikost populacije. Skalirano uspešnost f' dobimo iz uspešnosti f bodisi z linearnim skaliranjem po predpisu $f' = af + b$ ali z eksponentnim skaliranjem $f' = f^c$.

Vzdrževanju raznolikosti populacije in s tem preprečevanju prezgodnje konvergence služi tudi *porazdeljevanje uspešnosti* (angl. *fitness sharing*) (Goldberg, 1987). Bistvo postopka je v medsebojnem zmanjševanju uspešnosti osebkov glede na njihovo podobnost, kar preprečuje pretirano množenje sorodnih osebkov. Rezultat je specializacija ali uvedba vrst. Pristop je koristen pri iskanju več ekstremov multimodalnih funkcij.

S pomočjo funkcije uspešnosti lahko obravnavamo tudi omejitve, ki veljajo za rešitve danega problema. Neizpolnjevanje omejitev ovrednotimo s "kaznovanjem", t.j. ustreznim zmanjševanjem uspešnosti osebkov. Genetskemu algoritmu samemu prepustimo, da med evolucijo kot neuspešne izloči osebke, ki ne predstavljajo dopustnih rešitev. Slabost tega pristopa je v tem, da je težko razlikovati "skoraj dopustne" rešitve, pri katerih je zadoščeno večini omejitev, od tistih, ki so sicer dopustne, a imajo nizko vrednost kriterijske funkcije.

5.3 Operatorji

Posplošitev enostavnega križanja je *večmestno križanje* (angl. *multiple-point crossover*). Pri njem za par starševskih nizov naključno določimo ne le eno temveč več mest križanja, nato pa med nizoma izmenjamo vsak drugi par istoležnih podnizov. Slika 2 kaže primer trimestnega križanja.

Starša	Potomca
1 0 | 1 1 0 1 | 1 1 1 | 0	1 0 | 0 1 1 0 | 1 1 1 | 1
0 0 | 0 1 1 0 | 1 0 0 | 1	0 0 | 1 1 0 1 | 1 0 0 | 0

Slika 2: Trimestno križanje.

Pri t.i. *uniformnem križanju* (angl. *uniform crossover*) (Syswerda, 1989) izmenjava genetskega materiala med osebkoma poteka še bolj stohastično. Uniformno križanje izvedemo tako, da se za vsak par istoležnih znakov v paru starševskih nizov posebej odločimo o morebitni izmenjavi, upoštevaje pri tem predpisano verjetnost, ki je dodaten parameter algoritma.

Inverzija je genetski operator, ki se v genetskih algoritmih redkeje uporablja, združuje pa lastnosti križanja in mutacije. Z inverzijo dobimo naslednika obstoječega niza tako, da v izbranem podnizu obrnemo vrstni red znakov, kot vidimo iz primera na sliki 3.

1 0 \| 1 1 0 1 \| 1 1 1 0

1 0 \| 1 0 1 1 \| 1 1 1 0

Slika 3: Inverzija.

Elitizem je dodaten korak v genetskih algoritmih. V elitističnem genetskem algoritmu določeno število osebkov v novo generirani populaciji nadomestimo z najboljšimi iz prejšnje generacije. S tem preprečimo morebitno izgubo najboljših osebkov zaradi genetske rekombinacije.

5.4 Paralelni genetski algoritmi

Za enostavni genetski algoritem lahko ugotovimo, da je njegova časovna zahtevnost linearno odvisna od števila generacij g, velikosti populacije n in dolžine nizov l, torej $\mathcal{O}(gnl)$. Ker je med izvajanjem algoritma v splošnem potrebno oceniti gn rešitev, na čas izvajanja seveda bistveno vpliva čas potreben za oceno (evaluacijo) posamezne rešitve. Zaradi svoje zasnove pa so genetski algoritmi zelo primerni za paralelno implementacijo, s katero

lahko dosežemo dodatno pohitritev oz. povečamo število rešitev obdelanih v enoti časa.

Ideja paralelizacije je v sočasnem obravnavanju populacije na večprocesorskem računalniku, kjer vsak procesor ustreza enemu osebku ali manjši skupini osebkov. Ozko grlo za tako izvedbo predstavlja faza reprodukcije, kjer je za izračun povprečne uspešnosti populacije potrebna uskladitev vzporednih procesov. Globalnemu nadzoru se razvijalci paralelnih genetskih algoritmov običajno izognejo z uvedbo subpopulacij ali t.i. otoškega modela evolucije. Subpopulacije vsebujejo majhno število osebkov, se razvijajo neodvisno druga od druge, le občasno si izmenjajo najboljše posameznike. Implementacija je takšna, da zahteva le lokalno komunikacijo: en procesor obravnava celotno subpopulacijo, ali pa en procesor ustreza enemu osebku, subpopulacijo pa tvori skupina osebkov na sosednjih procesorjih.

Teoretično obravnavo paralelnega genetskega algoritma zasledimo v (Pettey in Leuze, 1989), v implementacijah in eksperimentalnih rezultatih pa prednjačita skupini raziskovalcev iz Bruslja (Manderick in Spiessens, 1989; Spiessens in Manderick, 1991) in St. Augustina v Nemčiji (Mühlenbein, 1989; Gorges-Schleuter, 1989).

6. Področja uporabe genetskih algoritmov

Genetski algoritmi so bili doslej že preizkušeni v optimiranju funkcij (De Jong, 1975; Bethke, 1980; Schaffer in sod., 1989) in na problemih kombinatorične optimizacije, kot sta problem trgovskega potnika in razporejanje opravil. Problema trgovskega potnika se z genetskimi algoritmi lotevajo mnogi avtorji, med njimi Goldberg in Lingle (1985), Grefenstette in sod. (1985), Oliver in sod. (1987), Fogel (1988). Za nekatere referenčne primere problema so bili prav z genetskimi algoritmi dobljeni najboljši znani rezultati, ali pa primerljivi z že znanimi, vendar ob manjši porabi računskega časa (Mühlenbein in sod., 1988; Peterson 1990).

Razporejanje opravil je v splošnem zahteven problem kombinatorične optimizacije, na katerega naletimo v proizvodnih enotah in večprocesorskih računalniških sistemih. Uporabo genetskih algoritmov pri reševanju realnih primerov tega problema najdemo v delih (Cleveland in Smith, 1989; Whitley in sod., 1989; Syswerda in Palmucci, 1991; Filipič, 1992a,b).

Drugo področje, na katerem se uveljavlja evolucijski pristop, pa je avtomatsko učenje. Tu nastopajo genetski algoritmi kot modul za generiranje pravil v t.i. *klasifikatorskih sistemih* (angl. *classifier systems*) (Booker in sod., 1989; De Jong, 1988, 1990). To so sistemi, ki se učijo sintaktično enostavnih pravil, imenovanih klasifikatorji, v stalni interakciji z okoljem. Načelo selekcije in adaptivnost, ki ju v delovanje takega sistema vnaša evolucijski pristop, omogočata sprotno prilagajanje klasifikatorjev na spremembe v okolju. Primer takega učenja je sistem za učenje opisov konceptov GABIL (De Jong in Spears, 1991). Poznani, čeprav manj pogosti, so tudi primeri učenja z genetskimi algoritmi, ki ne uporabljajo arhitekture klasifikatorskih sistemov. Med njimi omenimo učenje enostavnih računalniških programov (Cramer, 1985; Koza, 1989) in učenje kvalitativnih modelov (Varšek, 1991).

Uporabnost genetskih algoritmov je bila pokazana na nekaterih praktičnih, predvsem inženirskih optimizacijskih nalogah. V pregledih aplikacij (Goldberg, 1989a; Davis, 1991) najdemo med drugim primere vodenja dinamičnih sistemov, načrtovanja komunikacijskih omrežij in transportnih poti, razpoznavanja vzorcev ter procesiranja slik. Genetski algoritmi so se izkazali kot učinkovita optimizacijska metoda pri določanju vrednosti parametrov regulacijskega sistema (Filipič, 1992c). V delu (Davidor, 1991) je opisana genetska optimizacija trajektorij v robotiki. Končno omenimo še *genetsko programiranje*, t.j. uporabo genetskih algoritmov pri načrtovanju kompleksnih sistemov, kot so nevronske mreže, integrirana vezja in avtonomni sistemi, katerega ambiciozni cilj je umetna evolucija računalniških modulov in robotskih sistemov (de Garis, 1991).

7. Zaključek

Genetski algoritmi so interdisciplinarno področje, ki z modeliranjem evolucijskega procesa daje nov zamah raziskavam v biologiji, genetiki, antropologiji in sociologiji, hkrati pa so izsledki s tega področja uporabni v računalništvu. Uporabni so kot splošna preiskovalna metoda, ki jo odlikujejo

nizek red časovne kompleksnosti, robustnost glede vrste problemov, ki jih z njimi lahko rešujemo, in relativna neobčutljivost na vrednosti parametrov. Njihovo delovanje je presenetljivo enostavno, saj vključuje le sintaktične operacije nad kodami, ki predstavljajo rešitve obravnavanega problema. Mnogi koraki v njihovem delovanju so naključni, vendar vgrajeni mehanizmi selekcije in kombiniranja rešitev zagotavljajo konvergenco od slabših k boljšim rešitvam.

Matematična analiza delovanja genetskega algoritma se naslanja na sheme. Z njimi opišemo značilne vzorce, ki se pojavljajo v kodah rešitev, in ki jih genetski algoritem procesira posredno, ko izvaja operacije nad kodami. Po osnovnem izreku o genetskih algoritmih število predstavnikov nadpovprečnih shem v populaciji eksponentno raste, kar prispeva k generiranju čedalje boljših rešitev.

Genetski algoritmi so definirani zelo splošno, tako da načrtovanje genetskega algoritma za konkretno nalogo običajno zahteva individualni razvoj mnogih elementov algoritma. Ključni elementi, s katerimi vplivamo na učinkovitost genetskega algoritma in kvaliteto rezultatov, so kodiranje rešitev, funkcija uspešnosti, genetski operatorji in vrednosti parametrov algoritma.

Literatura

Antonisse, H. J. in Keller, K. S. (1987) Genetic operators for high-level knowledge representations. *Genetic Algorithms and Their Applications: Proc. 2nd Int. Conf. on Genetic Algorithms*, Hillsdale: Lawrence Erlbaum Associates, str. 69–76.

Bethke, A. D. (1980) Genetic algorithms as function optimizers. Doctoral dissertation, Univ. of Michigan.

Booker, L. B, Goldberg, D. E. in Holland, J. H. (1989) Classifier systems and genetic algorithms. *Artificial Intelligence 40*, 235–282.

Cleveland, G. A. in Smith, S. F. (1989) Using genetic algorithms to schedule flow shop releases. *Proc. 3rd Int. Conf. on Genetic Algorithms*, San Mateo: Morgan Kaufmann, str. 61–69.

Cramer, N. L. (1985) A representation for the adaptive generation of simple sequential programs. *Proc. Int. Conf. on Genetic Algorithms and Their Applications*, Hillsdale: Lawrence Erlbaum Associates, str. 183–187.

Davidor, Y. (1991) *Genetic Algorithms and Robotics—A Heuristic Strategy for Optimization*, London: World Scientific Publishing Co.

Davis, L. (1989) Adapting operator probabilities in genetic algorithms. *Proc. 3rd Int. Conf. on Genetic Algorithms*, San Mateo: Morgan Kaufmann, str. 61–69.

Davis, L. (ur.) (1991) *Handbook of Genetic Algorithms*, New York: Van Nostrand Reinhold.

de Garis, H. (1991) Genetic programming—GenNets, artificial nervous systems, artificial embryos. PhD Thesis, Faculty of Sciences, Free University of Brussels.

De Jong, K. A. (1975) An analysis of the behavior of a class of genetic adaptive systems. Doctoral dissertation, University of Michigan.

De Jong, K. (1988) Learning with genetic algorithms: An overview. *Machine Learning 3*, 121–138.

De Jong, K. (1990) Genetic-algorithm-based learning. V knjigi Kodratoff, Y. in Michalski, R. S. (ur.) *Machine Learning—An Arificial Intelligence Approach*, Vol. III, San Mateo: Morgan Kaufmann, str. 611–638.

De Jong, K. A. in Spears, W. M. (1991) Learning concept classification rules using genetic algorithms. *Proc. 12th Int. Joint Conf. on Artificial Intelligence*, San Mateo: Morgan Kaufmann, str. 651–656.

Filipič, B. (1992a) Enhancing genetic search to schedule a production unit. *Proc. 10th European Conf. on Artificial Intelligence ECAI-92*, John Wiley, str. 603-607.

Filipič, B. (1992b) Optimiranje proizvodnih procesov z genetskimi algoritmi. *Zbornik prve Elektrotehniške in računalniške konference ERK '92*, Portorož, zvezek B, str. 167-170.

Filipič, B. (1992c) Genetic optimization of controller parameters. *Proc. 14th Int. Conf. on Information Technology Interfaces ITI-92*, Pula, str. 173-178.

Fogel, D. B. (1988) An evolutionary approach to the traveling salesman problem. *Biological Cybernetics 60*, 139–144.

Goldberg, D. E. (1987) Genetic algorithms with sharing for multimodal function optimization. *Genetic Algorithms and Their Applications: Proc. 2nd Int. Conf. on Genetic Algorithms*, Hillsdale: Lawrence Erlbaum Associates, str. 41–49.

Goldberg, D. E. (1989a) *Genetic Algorithms in Search, Optimization and Machine Learning*, Reading: Addison-Wesley.

Goldberg, D. E. (1989b) Sizing populations for serial and parallel genetic algorithms. *Proc. 3rd Int. Conf. on Genetic Algorithms*, San Mateo: Morgan Kaufmann, str. 70–79.

Goldberg, D. E. in Lingle, R. (1985) Alleles, loci and the traveling salesman problem. *Proc. Int. Conf. on Genetic Algorithms and Their Applications*, Hillsdale: Lawrence Erlbaum Associates, str. 154–159.

Gorges-Schleuter, M. (1989) ASPARAGOS An asynchronous Parallel Genetic Optimization Strategy. *Proc. 3rd Int. Conf. on Genetic Algorithms*, San Mateo: Morgan Kaufmann, str. 422–427.

Grefenstette, J. J. (1986) Optimization of control parameters for genetic algorithms. *IEEE Trans. Syst., Man and Cybern. 16* (1), 122–128.

Grefenstette, J. J., Gopal, R., Rosmaita, B. J. in Van Gucht, D. (1985) Genetic algorithms for the traveling salesman problem. *Proc. Int. Conf. on Genetic Algorithms and Their Applications*, Hillsdale: Lawrence Erlbaum Associates, str. 160–168.

Holland, J. H. (1975) *Adaptation in Natural and Artificial Systems*, Ann Arbor: University of Michigan Press.

Koza, J. R. (1989) Hierarchical genetic algorithms operating on populations of computer programs. *Proc. 11th Joint Conf. on Artificial Intelligence*, San Mateo: Morgan Kaufmann, str. 768–774.

Langton, Ch. G. (ur.) (1989) *Artificial Life*. Reading: Addison-Wesley.

Manderick, B. in Spiessens, P. (1989) Fine-grained parallel genetic algorithms. *Proc. 3rd Int. Conf. on Genetic Algorithms*, San Mateo: Morgan Kaufmann, str. 428–433.

Mühlenbein, H. (1989) Parallel genetic algorithms, population genetics and combinatorial optimization. *Proc. 3rd Int. Conf. on Genetic Algorithms*, San Mateo: Morgan Kaufmann, str. 416–421.

Mühlenbein, H., Gorges-Schleuter, M. in Krämer, O. (1988) Evolution algorithms in combinatorial optimization. *Parallel Computing 7*, 65–85.

Oliver, I. M., Smith, D. J. in Holland, J. R. C. (1987) A study of permutation crossover operators on the traveling salesman problem. *Genetic Algorithms and Their Applications: Proc. 2nd Int. Conf. on Genetic Algorithms*, Hillsdale: Lawrence Erlbaum Associates, str. 224–230.

Peterson, C. (1990) Parallel distributed approaches to combinatorial optimization: Benchmark studies on traveling salesman problem. *Neural Computation 2*, 261–269.

Pettey, C. C. in Leuze, M. R. (1989) A theoretical investigation of a parallel genetic algorithm. *Proc. 3rd Int. Conf. on Genetic Algorithms*, San Mateo: Morgan Kaufmann, str. 398–405.

Rawlins, G. J. E. (1991) *Foundations of Genetic Algorithms*, San Mateo: Morgan Kaufmann.

Schaffer, J. D., Caruana, R. A., Eshelman, L. J. in Das, R. (1989) A study of control parameters affecting online performance of genetic algorithms for function optimization. *Proc. 3rd Int. Conf. on Genetic Algorithms*, San Mateo: Morgan Kaufmann, str. 51–60.

Schraudolph, N. N. in Belew, R. K. (1992) Dynamic parameter encoding for genetic algorithms. *Machine Learning 9*, 9–21. Tudi Technical Report LA-UR-90-275, Los Alamos National Laboratory, Los Alamos, New Mexico, 1991.

Spiessens, P. in Manderick, B. (1991) A masively parallel genetic algorithm—Implementation and first analysis. *Proc. 4th Int. Conf. on Genetic Algorithms*, San Mateo: Morgan Kaufmann, str. 279–286.

Syswerda, G. (1989) Uniform crossover in genetic algorithms. *Proc. 3rd Int. Conf. on Genetic Algorithms*, San Mateo: Morgan Kaufmann, str. 2–9.

Syswerda, G. in Palmucci, J. (1991) The application of genetic algorithms to resource scheduling. *Proc. 4th Int. Conf. on Genetic Algorithms*, San Mateo: Morgan Kaufmann, str. 502–508.

van Laarhoven, P. J. M. in Aarts, E. H. L. (1987) *Simulated Annealing: Theory and Applications*, Dordrecht: Kluwer Academic Publishers.

Varšek, A. (1991) Qualitative model evolution. *Proc. 12th Joint Conf. on Artificial Intelligence*, San Mateo: Morgan Kaufmann, str. 1311–1316.

Whitley, D., Starkweather, T. in Fuquay, D. (1989) Scheduling problems and traveling salesmen: The genetic edge recombination operator. *Proc. 3rd Int. Conf. on Genetic Algorithms*, San Mateo: Morgan Kaufmann, str. 61–69.

METODOLOGIJA NAČRTOVANJA NAMENSKIH RT SISTEMOV

INFORMATICA 4/92

Keywords: Real-time systems, embeeded systems,
software design, structured design, multi-tasking

Peter Kolbezen
Laboratorij za računalniške arhitekture
Institut Jožef Stefan, Ljubljana

V članku je opisan splošen pristop k načrtovanju namenskih računalniških sistemov za delo v realnem času. Vpeljane so tehnike načrtovanja programske opreme takšnih sistemov. Obravnavani so principi, ki so osnovani na dveh metodah: metodi MASCOT in metodi strukturnega načrtovanja RT sistemov. Poudarek je na delitvi programske opreme na module in na izvajanju več opravil hkrati.

DESIGN METHODOLOGY OF EMBEEDED REAL TIME SYSTEMS. In the paper the general approach to the design of embeeded real-time systems is described. Software design techniques for these systems are introduced. The principles uderlying two methods, MASCOT and real-time structured design, are covered. Emphasis is given to dividing the software into modules and to the use of multi- tasking.

1. UVOD

Namenski računalniški sistem imenujemo sistem za delo v realnem času, ker omogoča krmiljenje naprav v okolju v realnem času. Zato je sistem z napravami fizično povezan preko senzorjev, ki zaznavajo dogodke v okolju, ki ga nadzoruje in krmili preko prikazovalnikov oz. aktuatorjev. Računalnik, ki je tako vgrajen v sistem (embeeded system), zaznava signale, ki se sprožijo glede na zunanje dogodke, in se mora nanje pravočasno in pravilno odzivati. To pomeni, da mora računalnik zaznati vse dogodke, čeprav se lahko pojavljajo istočasno, in izvesti dogodkom ustrezna opravila v predvidenem času. V nasprotnem primeru se obnašanje namenskega sistema poruši in posledice so lahko katastrofalne. V primerih, ko igra odzivni čas sistema posebej pomembno vlogo, govorimo o namenskih sistemih, ki delajo v strogem realnem čau (hard real-time system). Takšen sistem, ki ga imenujemo tudi trdi, je na primer sistem vodenja in nadzora letala, medtem ko je nadzor parkirišča primer t.im. mehkega sistema (soft real-time system). V slednjem primeru so časovne zahteve bistveno milejše, tako v pogledu odzivne hitrosti kot v posledicah (izguba parkirnine).

Pospešen razvoj RT sistemov sega nazaj v leta 1970 /BIB74/ in še vedno narašča. Namenske sisteme dandanes srečujemo na vsakem koraku. Na komercialnem področju so namenjeni dinamični obdelavi velikih količin podatkov, kot na primer pri rezervacijah letalskih vozovnic ali v bančnem poslovanju. Tudi zunaj komercialnih področij so takšni sistemi učinkoviti, na primer v bolnicah, knjižnicah, šolah. Posebej pomembno vlogo pa igrajo v industriji, kjer računalniki upravljajo procese z elektronsko obdelavo podatkov. Računalniki omogočajo hitrejšo in natančnejšo analizo operacij in spremenljivk, kot so temperatura, pritisk, kvaliteta produkcije ipd. Hkrati so učinkovitejši od človeka pri zaznavanju nemotenega in nevarnega obnašanja procesa. Kmalu so si utrli pot tudi na druga področja, zlasti na področje vodenja plovil in najrazličnejših vozil, na področje raziskav vesolja ipd.

Računalniška materialna oprema je lahko preprost mikroprocesorski sistem, v katerem mikroprocesor zaznava signale, ki se sprožijo glede na zunanje dogodke. Ker je obnašanje namenskega sistema pogojeno z realnim časom, mora tudi računalniški sistem izvajati nadzor pravočasno. To pomeni, da mora zaznati vse dogodke, čeprav se pojavljajo tudi istočasno in izvesti vsa opravila za zaznane dogodke v predvidenem času. Sicer se obnašanje sistema poruši in posledice so lahko katastrofalne. Ker postajajo te zahteve bolj in bolj kompleksne, v namenskih sistemih vse pogosteje srečujemo večprocesorske strukture. Te največkrat predstavljajo porazdeljene večprocesorske sisteme

/Coo86/. Procesorji takega sistema izvajajo posamezne akcije (opravila) sočasno. Vendar često tudi v takšnih sistemih število akcij presega število možnih virov. Problemi te vrste se rešujejo s posebnimi tehnikami.

Pri načrtovanju namenskih (embedded) računalniških sistemov za delo v strogem realnem času uporabljamo splošne pristope k načrtovanju RT sistemov.

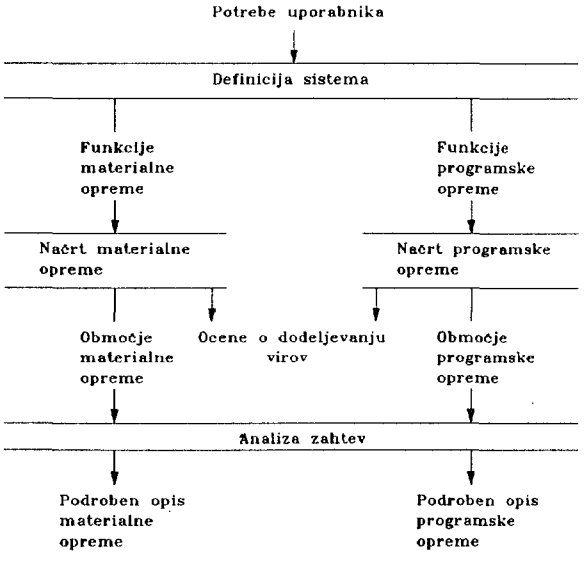
2. NAČRTOVANJE SISTEMA

Načrtovanju RT sistema se v splošnem ne razlikuje od načrtovanja namenskega sistema. V obeh primerih so faze načrtovanja razdeljene v dva dela:

- 1.faza: Faza planiranja sistema
- 2.faza: Faza razvoja

2.1. Faze načrtovanja

V fazi planiranja (slika 1) se raziščejo in natančno pojasnijo zahteve uporabnika, ki so potrebne za izdelavo podrobne specifikacije načrtovanega sistema in plana virov - ljudi, časa, naprav in razvojnih stroškov. V tej fazi (t.im. pripravljalni stopnji načrtovanja) se uvodno odločamo za porazdelitev funkcij med materialno in programsko opremo.



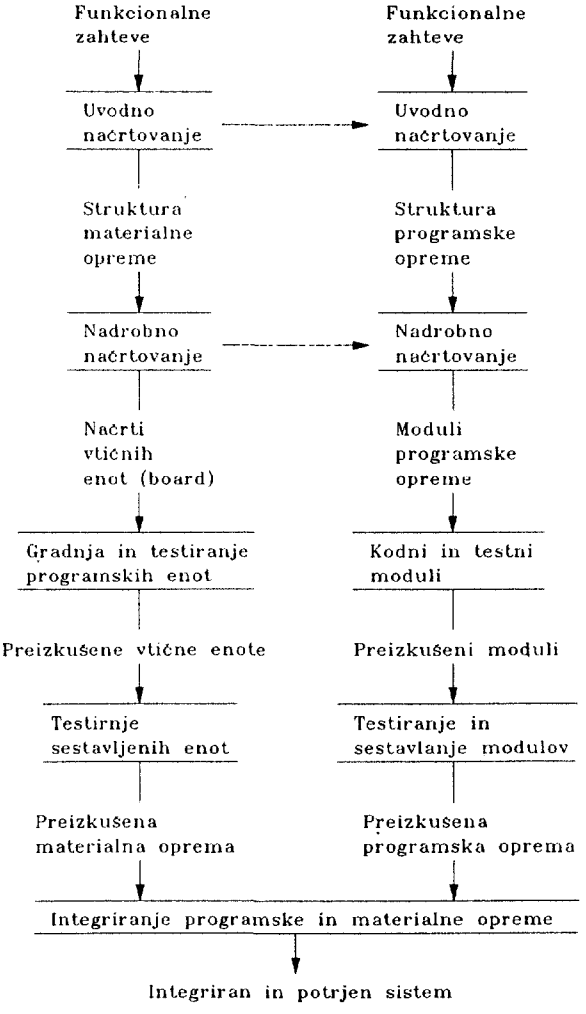
Slika 1. Faza planiranja.

V pripravljalni stopnji se najprej posvetimo izboru računalniške arhitekture. Odločamo se o eni od treh možnih variant:

- centralizirani sistem z enim samim centralnim računalnikom,
- hierarhičen sistem ali
- distribuirani sistem.

Rezultat te stopnje je dokument o specifikacijah sistema, ki jih izdela dobavitelj oziroma načrtovalec sistema. Izdela jih na osnovi dokumentov, ki vsebujejo zahteve, katerim naj bi sistem zadoščal. Zahteve poda uporabnik oziroma kupec sistema. V pripravljalni stopnji je bistveno, da so izdelani dokumenti popolni, detajlirani in povsem jasni. Izkušnje so namreč pokazale, da je pretežni del napak, ki se pojavljajo v končnem sistemu, posledica napačnih informacij v prvobitnih dokumentih, informacij z nejasnimi, nepopolnimi ali celo napačnimi specifikacijami.

Pogosto imamo skušnjavo, da rečemo "o tem vprašanju bo mogoče odločati kasneje". Posledica takšne odločitve lahko potegnejo za seboj tudi spremembe na drugih delih sistema, ki so že načrtani, pri naknadnem spreminjanju teh delov pa se pogosto vrinjajo napake. Priporoča se, da dokumentacija o specifikacijah sistema vsebuje tudi uporabniško dokumentacijo. Vsekakor pa mora biti taka dokumentacija natančno preverjena skupaj z



Slika 2. Stopnje razvojne faze.

uporabnikom (kupcem) že pred začetkom druge faze.

Razvojna faza je prikazana na sliki 2.

Cilj uvodne faze načrtovanja je dekompozicija sistema v množico specifičnih podopravil, ki se lahko obravnavajo ločeno. Načrtovanje v tej fazi poteka na visokem nivoju. To pomeni, da izhajamo iz visokonivojskih specifikacij. Rezultati te faze so globalne podatkovne strukture in visokonivojske programske arhitekture. Načrtovanje na tej stopnji zahteva, da obstaja tesna povezanost med načrtovalci materialne in programske opreme. To je še posebej pomembno za sisteme, pri katerih so možni popravki predhodnih odločitev o računalniški arhitekturi. Pri uporabi distribuiranega sistema so te vrste popravki potrebni v pogledu števila procesorjev, komunikacij (prepustnost, tip arhitekture), ipd. Na zaključku faze uvodnega načrtovanja mora biti izdelana kritična analiza celotne opreme, tako programske kot materialne.

Nadrobno načrtovanje sistema je razbito na dve stopnji:

- dekompozicija sistema na module, in
- načrtovanje posameznih modulov.

Na prvi stopnji faze načrtovanja materialne opreme (MO) se zastavlja vprašanje strukture sistema. Med takimi vprašanji je n.pr. vprašanje vtičnih enot (boards) sistema. Odločati se moramo: ali naj bodo posamezne vtične enote namenjene samo analognim ali samo digitalnim vhodom, ali pa naj bodo vtične enote take, da bodo vsi vhodi (analogni in digitalni) koncentrirani na eni enoti? Drugo takšno vprašanje je n.pr., kakšen tip vodila naj bo uporabljen? Na drugi stopnji že načrtujemo posamezne vtične enote.

Prva stopnja programerskega inženirstva vključuje razpoznavanje aktivnosti, ki so med seboj na nek način povezane. Obstajajo heuristična pravila, ki načrtovalcu pomagajo pri odločitvah, kako razdeliti sistem na module /Pres82/. Izdelane heuristike dajejo različne poudarke posameznim metodologijam načrtovanja. V nadaljevanju bomo podali kratek opis nekaterih splošnih metodologij načrtovanja RT sistema.

2.2. Metodologije načrtovanja

Na kratko si bomo ogledali najbolj uporabljene metodologije, med katere sodijo: funkcionalni razcep, prekrivanje informacij, sklapljanje in povezovanje, ter razbitje.

Funkcionalni razcep (functional decomposition) je pristop, imenovan tudi "od vrha navzdol". Zagovarjajo ga Wirth in še nekateri. Predstavlja delitev programske opreme na module po funkcijah. To pomeni, da vsak modul izvaja le svojo, zanj specifično funkcijo.

Prekrivanje informacij (information hiding). Parnas /1972/ je bil velik nasprotnik funkcionalne dekompozicije. Zavzemal se je za metodo, ki je osnovana na takšnem prekrivanju informacij med moduli, da so v posameznem modulu prisotne vse možne informacije, ki jih modul uporablja /Par72/.

Razlike med obema pristopoma si oglejmo na naslednjem primeru. Vzemimo program, ki mora iz naprave prebrati blok besed nekega teksta. Besede v bloku morajo biti urejene po abecednem redu in se ne smejo podvajati. Tako urejen seznam besed moramo izpisati.

V primeru, da pristopimo k preprosti "funkcionalni dekompoziciji", razdelimo sistem na tri module: vhodni, urejevalni in izpisni. Vsi moduli imajo dostop do dodeljene podatkovne strukture, v kateri se nahaja tekst, in vsak od njih ve, iz katere seznama (linked list), zapisa (record) ali datoteke (file).

Parnas zagovarja delitev, pri kateri obstaja modul, imenovan upravljalnik pomnilnika (Store-Manager). Ta se ukvarja s shranjevanjem podatkov. Drugi moduli imajo možen dostop do podatkov le preko funkcij tega upravljalnika, ki lahko opravlja obe funkciji: vpisuje in izpisuje (besede). Prva funkcija omogoča, da se informacija shrani v pomnilnik in druga, da se pokliče iz pomnilnika (je dosegljiva), pri tem pa ostalim modulom ni potrebno vedeti, kako je pomnilnik organiziran. Prednost takega pristopa je, da med postopkom načrtovanja sprememba na enem modulu ne zahteva spremembe na drugem modulu.

Sklapljanje in povezovanje (coupling and cohesion) sestavljata dve metodi. Pri prvi metodi je prisotna težnja po minimalni sklopljenosti, pri drugi metodi pa težnja po maksimalni povezanosti med moduli, obe pa uporabljata heuristične pristope načrtovanja podatkovnega vodenja (data flow). Heuristiko in metodologije zanje so razvili Constantine, Steven /StMC74/ in Myers /Mye78/, /You79/.

Razbitje (partition) za minimizacijo vmesnikov je zasnovano na heuristiki, ki jo je predlagal DeMarco (1973) /BenL84/, in jo je mogoče kombinirati z metodami pretoka podatkov. Pravilno je menil, da je mogoče s transformacijo, ki jo je pokazal na diagramu pretoka podatkov, diagram razbiti v poddiagrame (module) tako, da se med njimi minimizira število povezav.

Pri RT sistemih so potrebne še dodatne heuristike, ki so značilne za posamezne module naslednjih kategorij:

- RT-HC: realni čas, trda omejitev (real-time, hard constraint)
- RT-SC: realni čas, mehka omejitev (real-time, soft constraint)
- IACT: pogovoren (interactive)

Uveljavljeni tipi programa (sekvenčni, večopravilni, program realnega časa) kažejo na težnje, da se zmanjša obseg programske opreme modula, ki pripada RT-HC kategoriji, saj je le-tega najtežje načrtati in testirati.

V pogledu načrtovanja programske opreme obstajajo bistvene razlike med RT in standardnimi sistemi na stopnji uvodnega načrtovanja in v postopkih načrtovanja na stopnji dekompozicije na module. Zato se bomo v nadaljevanju osredotočili prav na tovrstno problematiko, saj je načrtovanje, kodiranje in testiranje posameznega modula več ali manj podobno gradnji programske opreme standardnega tipa (non-real-time software).

2.3. Dokumentiranje specifikacij

Splošen primer dokumentiranja za fazo planiranja obsega dokumentacijo o:

- vmesniški (interface) opremi (povezava z okoljem)
- nadzorni napravi
- komunikaciji z operaterjem, ki vsebuje dokumentacijo o
  - prikazovalniku
  - nastavitvi krmilne naprave
  - posegih operaterja (tudi v smislu nastavljanja novih parametrov)
  - upravljavski informaciji (management information)
  - splošni informaciji

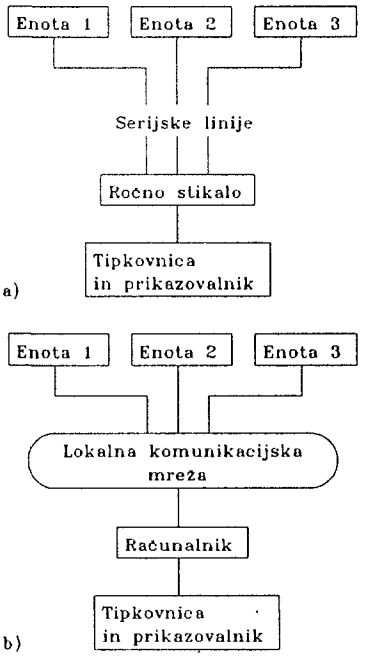
3. UVODNO NAČRTOVANJE

3.1. Načrtovanje materialne opreme

Ekonomski učinek materialne opreme računalnika v RT sistemu je odvisen od več činiteljev. Eden takšnih je gotovo tisti, ki bo odločal, ali naj bo računalnik na eni sami vtični enoti ali naj bo modularno grajen in naj uporablja standardno vodilo. Zahteve vplivajo tudi na izbiro 8-bitnega ali večbitnega mikroprocesorja, števila analognih in digitalnih signalov (kanalov), zahtevane hitrosti prenosov, konverzij ipd. Obstaja tudi ekonomski rizik pri izbiri števila potrebnih procesorjev: eno- ali več- procesorski sistem. Za primer si oglejmo dve možni konfiguraciji in ju poskusimo ovrednotiti.

Konfiguracija 1 (na sliki 3a) predstavlja preprosto rešitev materialne opreme z minmalnim številom procesorjev in preprostim mehaničnim stikalom, ki lahko poveže tastaturo z enim od 12-tih procesorjev. Vsak procesor bo moral vsebovati prikazovalnik ter vhodne programe in programe za procesiranje informacij.

Konfiguracija 2 (na sliki 3b) upošteva ločitev pravih kontrolnih funkcij od upravljalskih in operatorskih vhodno/izhodnih funkcij. V primerjavi s konfiguracijo 1 zahteva konfiguracija 2 dodatno materialno opremo, ki jo sestavljata poseben procesor in komunikacijska mreža. Pri tem je potrebno za vsakega krmilnega procesorja zagotoviti ustrezne pomnilniške kapacitete.



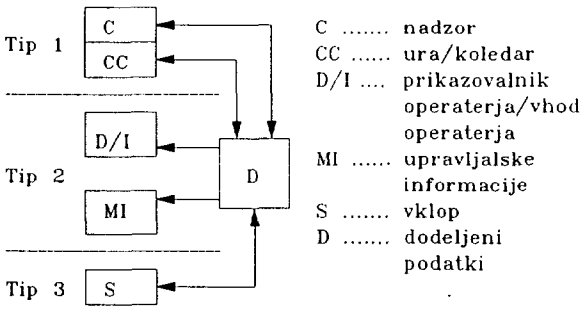
Slika 3. Mo'ne konfiguracije materialne opreme: a) Konfiguracija 1, b) Konfiguracija 2.

Del postopka načrtovanja MO je, da se oba možna pristopa k izbiri obeh konfiguracij na sliki 3 in drugi možni pristopi primerno ovrednotijo.

S podatkovnimi in časovnimi zahtevami (kot je n.pr. minimalni interval vzorčenjaja) je določena sistemska ura realnega časa. Potrebna je tudi odločitev, kakšni terminali naj bodo uporabljeni: specifični ali standardni terminali, koliko enih in koliko drugih ? To je odločitev, ki mora biti rezultat skrbne analize cen in prednosti, ki jih nudi posamezna rešitev.

3.2. Načrtovanje programske opreme

Programsko opremo RT-sistema iz prejšnega poglavja lahko razdelimo v segmente, ki so razvidni iz slike 4.



Slika 4. Konfiguracija programske opreme.

Krmilni modul (C modul) ima n.pr. **strogo omejitev** (hard constraint), saj mora vzorčevati najmanj vsakih 40 ms. V praksi je lahko ta omejitev nekoliko bolj ohlapna, recimo 40 ms ±1 ms s povprečno vrednostjo okrog 1 minute, kar zapišemo 40 ms ±0.5 ms. V splošnem čas vzorčevanja določimo z izrazom $T_s \pm e_a$. Zahteve lahko nekoliko omilimo, če n.pr. dopuščamo, da med 100 vzorci izpustimo en vzorec. Takšne omejitve so del testne specifikacije in jih moramo upoštevati tudi med testiranjem sistema.

Od CC modula (Clock/Calendar modul) se zahteva, da se ne izgubi noben urin impulz. Zato se mora aktivirati vsakih 20 ms (pri osnovni frekvenci 50 Hz). Takšno omejitev lahko spremenimo v **ohlapno** (soft constraint) z dodatno materialno opremo v obliki števnika, ki ga CC modul lahko čita in resetira. Omejitev je v tem primeru 1 sekundni odzivni čas s 5 sekundnim intervalom med dvema sosednjima odčitkoma števnika. Kolikšna je velikost števnika pri takšni omejitvi, si lahko izračunamo sami!.

D modul (operator display) ima strogo omejitev, tj. časovni interval odčitkov 5 sekund. Po občutku bi rekli, da takšen interval ni stroga omejitev, saj bi mora ustrezati povprečnemu času 5 sekund. Vendar je omejitev stroga, ker mora biti specificiran hkrati s povprečnim časom tudi maksimalni časovni interval, ki je n.pr. 10 sekund.

Mehke omejitve lahko pripišemo I modulu (operator input) in MI modulu (management information). O njih odločamo v soglasju s kupcem in oblikujejo del specifikacij v dokumentu zahtev.

Za S modul (start up) ni nujno, da dela v realnem času. Zato ga lahko obravnavamo kot standardni interaktivni modul.

Obstajajo različne aktivnosti, ki jih lahko delimo v več manjših. Kašna bo ta delitev in kako bodo potekale naslednje stopnje, je odvisno od splošnega pristopa k implementaciji. Obstajajo trije možni pristopi:

- enojen program
- sistem priviligiranih programov (foreground/background system - FG/BG system)
- večopravilni sistem

V nadaljevanju bomo obravnavali vsak pristop posebej.

3.3. Enojen program

Standardni pristop programiranja smatra posamezne module na sliki 4 kot procedure ali subrutine enojnega, glavnega programa:

```
begin Vklop;
  while not prekinitev do
    odčitavanje ure na vhodni liniji;
    if odčitek ure then begin
      ažuriranje ure in koledarja (CC);
      case true do
        čas za krmiljenje: krmiljenje (CO);
        čas za prikazovanje: ažuriranje
                    prikazovalnika (DU);.
        vhod operaterja: vnos podatkov (OI);
        zahteva za upravljanje: upravljanje
                              izhoda (MO);
      enddo
    end
  enddo
end.
```

Pri programiranju takšne strukture ni nikakršnih težav. Vendar se pri tem pojavlja več resnih časovnih omejitev, kot je n.pr. zahteva, da se mora CC modul aktivirati vsakih 20 ms, neglede na ostale module. Če so t_1,t_2,t_3,t_4 in t_5 maksimalni časi računanja, ki ustrezno pripadajo modulom CC, C, DU, OI in MO (management output), potem je za delujoč sistem postavljena zahteva, ki jo lahko izrazimo z izrazom

$t_1 + \max(t_1,t_2,t_3,t_4,t_5) \leq 20$ ms(Pomni: Vrednosti t_1,t_2,t_3,t_4 in t_5 morajo vključevati čas, ki je potreben za testiranje, in v t_1 mora biti vključen tudi čas za odčitavanje urinih impulzov).

Pristop k načrtovanju, ki je zasnovano na enojnem programu, lahko uporabljamo za preproste, majhne sisteme. Takšen pristop daje pregledno rešitev ob

minimalni materialni in programski opremi sistema, ki ga je lahko testirati. Z večanjem opreme pa narašča problem na stopnji nadrobnega načrtovanja. Pojavlja se težnja, da se program razbije na module. Problem ne narašča zaradi funkcionalnih razlik med moduli, ampak preprosto zato, da se lahko znotraj posameznih modulov programska oprema sčasoma tudi dopolnjuje. Glede na zahteve MO modula je pristop "enojnega programa" neprimeren. Uporabljiv je le, če zahteve spremenimo. Lahko pa tudi zahtevamo, da DU modul (display update module) razbijemo na tri module: prikazovanje datuma in časa, prikazovanje procesnih vrednosti in prikazovanje krmilnih parametrov.

3.4. Sistem priviligiranih programov

Vsekakor obstajajo prednosti, če so moduli s strogimi časovnimi omejitvami ločeni in neodvisno vodeni od modulov z ohlapnimi ali brez omejitev - manj je interakcij med moduli in manj tesnih časovnih situacij. Moduli s strogimi časovnimi omejitvami, ki so najbolj tesno povezani z zunanjimi procesi, tečejo v t.im. ospredju (foreground) in jih bomo imenovali **priviligirani moduli.** Moduli z ohlapnimi omejitvami ali brez njih tečejo v t.im. ozadju (background) in jih bomo imenovali **nepriviligirani moduli.** Priviligirani moduli, ali opravila, kot jih navadno imenujemo, imajo visoko prioriteto glede na opravila, ki tečejo v ozadju. Zato se uporabljajo posebni mehanizmi, ki omogočajo, da priviligirano opravilo prekine nepriviligirano opravilo.

Razdelitev opravil v FG in BG opravila navadno zahteva podporo RT operacijskega sistema, kakršno nudi že sistem RT/11. Tej zahtevi je mogoče prilagoditi tudi nekatere standardne operacijske sisteme. Tako CP/M omogoča preproste FG/BG operacije, če materialna oprema podpira prekinitve. FG opravilo je prekinitvena rutina in BG opravilo standardni program.

V primeru pristopa, ki je zasnovan na sistemu priviligiranih programov, lahko strukturo zgoraj opisanega enojnega programa modificiramo v strukturo, ki je deljena na dva sistemska dela: v ospredju, v ozadju.

v ospredju

```
if prekinitev then begin
  ažuriranje ure in koledarja (CC);
  if čas za krmiljenje (C) then krmiljenje (CO);
  RTI
```

Zahteva za delo v ospredju (FG področje) je strnjena v izrazu:

$$t_1, t_2 \leq 20 \text{ ms},$$

kjer je
t_1 = maksimalen čas za CC modul, in
t_2 = maksimalen čas za C modul.

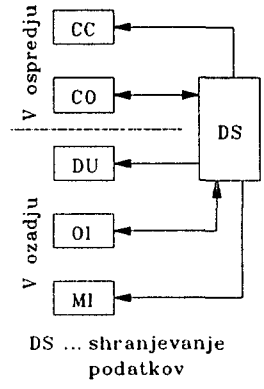
v ozadju

```
begin Vklop;
  while not prekinitev do
    case true do
      čas za prikazovanje: ažuriranje
                    prikazovalnika (DU);
      vhod operaterja: vnos podatkov (OI);
      zahteva za upravljanje: upravljanje
                              informacij (MI);
    enddo
  enddo
end.
```

Zahteve za delo v ozadju (BG področje) pa so:

1., max(t_3,t_4,t_5) 10 sek;
2., prikazovalni modul skanira povprečno vsakih 5 sek;
3., vhodni operator se odziva v času 10 sek.

Vidimo, čeprav so časovne omejitve bolj ohlapne, je ovrednotenje sistemskih zmogljivosti težje. Izvajati moramo meritve, ki so bolj zapletene, kot v primeru enojnega programa.



Slika 5. Povezanost programskih modulov FG/BG sistema.

Čeprav FG/BG pristop razbije **krmilno** strukturo v dva dela - FG in BG module, so posamezni moduli med seboj še povezani preko podatkovne strukture (slika 5).

Povezave med moduli skrbijo za dodeljevanje podatkovnih spremenljivk (krmilnih parametrov). Pri enojnem programu (imenovan tudi enoopravilni program) ni nobenih težav pri nadzorovanju dostopa do dodeljenih spremenljivk, ker je vedno aktivno samo eno opravilo (modul). V primeru FG/BG sistemskih opravil, pa so možne paralelne aktivnosti. Istočasno je lahko aktivno eno opravilo

iz FG in eno opravilo iz BG področja (oz. se opravili izvajata kvazi paralelno, če imamo enoprcesorski sistem).

V našem posebnem primeru lahko spremenljivke dodelimo krmilnim, prikazovalnim in vhodnim modulom brez posebnih težav, saj neko spremenljivko priredimo samo enemu modulu. Modulu operatorja vhodov priredimo krmilne parametre in množico kazalčnih spremenljivk, CC modulu podatkovne in časovne spremenljivke, ter krmilnemu modulu podatkovne spremenljivke procesa, ki ga sistem nadzira. Zaradi varnosti moramo vhodne podatke iz operatorja predhodno shraniti in jih prenesti dodeljenemu pomnilniku šele potem, ko so verificirani. V ta namen se uporablja posebna metoda shranjevanja vhodnih podatkov (parametrov). Če za take varnostne ukrepe ne poskrbimo, lahko npr. v telefoniji pri vzpostavljanju zveze pride do nezaželjenih prekinitev po drugi poti in s tem do nepravilnega ali nestabilnega nadzora zveze.

Prenos podatkov med FG in BG opravili je znan kot **kritična sekcija** programa, ki mora biti nedeljiva. To preprosto dosežemo s prepovedjo vseh prekinitev med prenosom podatkov.

Težava pri takem - sicer preprostem pristopu - je, da je pri večih ločenih modulih nezaželjeno, da ima vsak modul dostop do osnovne materialne opreme stroja in možnost spremiti status prekinitev. Praksa je pokazala, da je materialno opremo takšnih modulov teško načrtovati, kodirati in testirati, in da obstaja v povprečju večja možnost napak. Zato je priporočljivo, da je število takšnih modulov čim manjše.

Idealni prenosi morajo biti časovno prilagojeni krmilnemu modulu, kar pomeni, da morata biti operatorski in krmilni modul sinhronizirana ali se morata med seboj dogovarjati (rendezvous).

3.5. Večopravilen sistem

Oblikovanje programske opreme velikih RT sistemov in sistemov s pogostimi interakcijami je lahko bistveno lažje, če je mogoče delitev FG/BG še dalje deliti na več delov, kar omogoča rabo koncepta večih aktivnih opravil. Vsako aktivnost, ki se mora izvesti, smatramo že v začetni stopnji oblikovanja programske opreme za ločeno opravilo. Takšen (večopravilni) pristop se implicira v možnosti paralelnega izvajanja opravil, pri čemer število uporabljenih procesorjev v sistemu sploh ni bistveno.

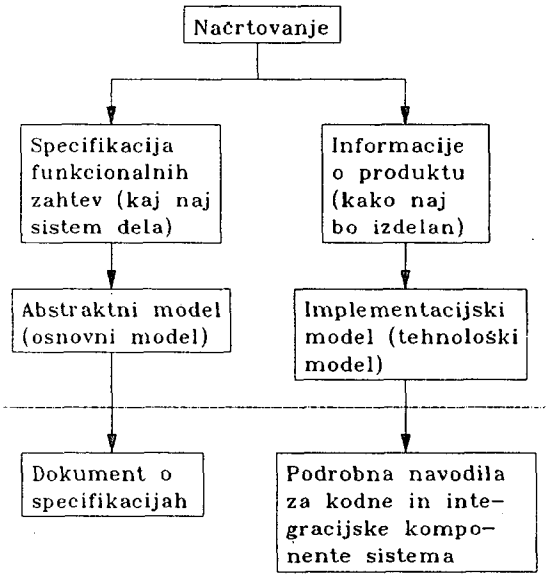
Za implementacijo večopravilnega sistema morajo biti dane naslednje možnosti:

- kreiranje ločenih opravil;
- razporejanje opravil med izvajanjem, navadno na prioritetni osnovi;
- dodeljevanje podatkov med opravili;
- sinhronizacija opravil med seboj in z zunanjimi dogodki;
- preprečevanje kvarnih vplivov enega opravila na drugega; in
- nadzor nad izvajanjem opravil (start/stop).

Naštete sposobnosti zagotavlja večina RT operacijskih sistemov. Predvsem jih zagotavljajo operacijski sistemi, ki so namenjeni večjim RT sistemom.

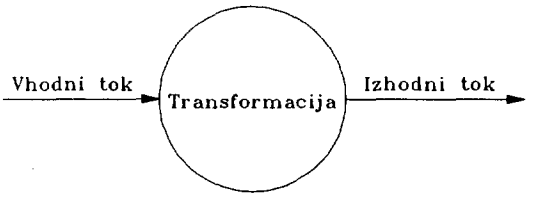
4. SPLOŠEN PRISTOP

Obravnava v zgornjih razdelkih je posvečena predvsem tistim tehničnim podrobnostim načrtovanja, ki imajo vpliv na oblikovanje arhitekture programske opreme. Učinkovitost načrtovanja pa postane zadovoljiva šele, če je možna abstrakcija sistema.



Slika 6. Abstraktni in izvedbeni model.

Oblikovanje velikih kompleksnih sistemov (za katere predpostavljamo, da so večopravilni) zahteva dva modela: abstraktnega in izvedbenega. Abstraktni model - ki se včasih nanaša na logični ali osnovni model - ne vsebuje nobene informacije o



Slika 7. Podatkovni pretok - osnovna notacija.

implementaciji; produkcijski ali izvedbeni model pa vsebuje tehnične detajle o delovanju sistema (slika 6).

Modela lahko razvijamo drug za drugim; tj. najprej razvijemo abstraktni in nato izvedbeni model. Bolj običajno pa je, da ju razvijamo sočasno, saj lahko odločitve o implementaciji na visokem nivoju vplivajo na odločive abstraktnega modela na nižjem nivoju.

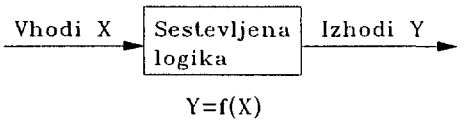
Razvoj abstraktnega modela, zasnovanega na specifikacijah, je odvisen od opisa sistema. Ta mora biti dovolj bogat idej, s katerimi je mogoče izraziti vse zahteve za načrtovanje sistema. Pri razvoju mora biti upoštevana metodologija, ki omogoča, da se abstraktni model prevede v izvedbeni model. Smernice razvoja se lahko opirajo na izsledke, dobljene s pomočjo heurističnih postopkov, ki jih bomo obravnavali kasneje.

Večino uporabnih zapisov in metodologij za načrtovanje RT sistemov je razvil Myers /1974/ na osnovi podatkovnega vodenja /May78/. Prvobitna temeljna predpostavka tehnike podatkovnega vodenja je, da lahko sistem ali podsistem predstavimo s podatkovno transformacijo (pretvorbo) kot osnovno notacijo (slika 7).

Podatkovni pretok je označen s puščico. Veljajo naslednje predpostavke:

1. Vhodni podatki za transformacijo in izhodni podatki so diskretne enote; vsaka enota predstavlja prevod (transakcijo);
2. Transformacija se proži na dospeli prevod;
3. Trajanje transformacije ne vpliva na natančnost operacije;
4. Izhod transformacije je odvisen le od tekoče transakcije. Na transformacijo ne vplivajo prejšni vhodni podatki. To pomeni, da transformacija nima nobenega notranjega zapisa o prejšni transakciji.

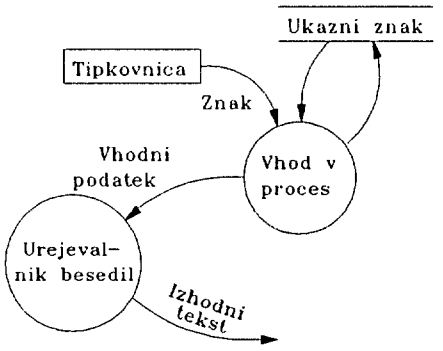
Model sistema je ekvivalenten sestavljenemu logičnemu vezju na sliki 8.



Slika 8. Sestavljen logični ekvivalent preprostega podatkovnega pretoka.

Preprost zapis, ki je zasnovan na podatkovnem vodenju, povzroča probleme pri predstavitvah številnih aplikacij, predvsem takšnih sistemov, ki so interaktivni. V veliki večini interaktivnih sistemov zahteva podatkovni vhod obravnavo različnih poti, ki so odvisne od prejšnih vhodnih podatkov. Pri modelu takšnih sistemov lahko uporabimo standardni zapis podatkovnega vodenja, če predhodne vhodne podatke, ki smo jih shranili, pred glavno sistemsko transformacijo dodamo novim vhodnim podatkom.

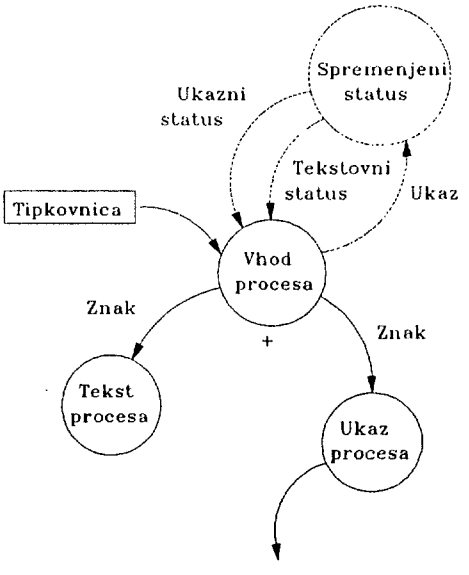
Primer takšnega pristopa je npr. sistem besednega urejevalnika. Iz tastature prihaja ali besedilo ali del ukaznega niza. Start ukaznega niza določa znak "ESC" in konec niza znak "CR". Rešitev kaže slika 9.



Slika 9. Ukazno procesiranje.

Vhodna transformacija prebere znak iz prikazovalnika in ga postavi pred znak, ki ga vzame iz pomnilnika ukaznih znakov. Ukazni startni znaka (ESC) in zadnji znak (CR) se z vhodno transformacijo shranita v pomnilnik. Podatek, ki ga sprejme glavna transformacija je tako ali ESC + znak, ali CR + znak, in transformacija procesiranja besed podatke ustrezno interpretira.

Alternativni pristop je takšen model sistema, da se vhodna transformacija zapomni, če je bil startni



Slika 10. Uporaba krmilne notacije

ukazni znak sprejet pri spremembi njenega notranjega stanja tako, da so kasneje sprejeti znaki obravnavani kot ukazi in prenešeni na ukazni procesor pred tekst procesorjem. Takšen model vidimo na sliki 10.

Poleg podatkovne transformacije je v modelu vpeljana dodatna notacija, ki označuje "ukazno" transformacijo. Ukazne akcije so označene s prekinjenimi črtami. Krmilni tokovi so dogodki, ki niso vezani na podatke. Krmilna transformacija je določena s tabelo prehajanja stanj.

V drugem pristopu je model sistema ekvivalenten sekvenčnemu logičnemu sistemu. Izhod je odvisen ne le od trenutnih vhodov, ampak tudi od notranjega stanja sistema, ki je funkcija prejšnih vhodov.

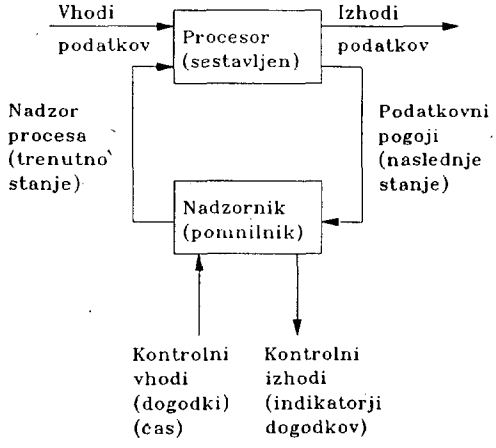
Študija RT sistemov kaže, da imajo le-ti dodatne značilnosti. Tipično ima RT sistem različne tipe vhodov in izhodov, katerih notacije predstavljajo:

1. diskretne dogodke, ki se lahko pojavljajo naključno;
2. zvezno spreminjajoče vrednosti podatkov; in
3. diskretne podatkovne vrednosti (ekvivalent transakcij)

Transformacije se lahko aktivirajo:

1., ob prihodu diskretne podatkovne vrednosti;
2., pri vnaprej določenih časovnih intervalih;
3., ko se pojavijo specifični notranji ali zunanji dogodki; ali
4., kontinuirano.

Spremembe notranjega stanja sistema so možne ne le kot posledica podatkovnih pogojev ampak tudi kot posledica zunanjih dogodkov. Hatley /Hat84/ je predlagal model, ki ga prikazuje slika 11. Krmilni vhodi in izhodi so tokovi dogodkov.



Slika 11. Splošni model RT sistema (po Hatley-u, 1984)

Upoštevajoč gornje značilnosti RT sistemov, so bile razvite številne metodologije načrtovanja RT sistemov. Glavne od njih so:

1. MASCOT (modularni pristop v konstrukciji programske opreme delovanja in testiranja); vpeljala sta ga Jackson in Simpson /JaS75/.
2. DARTS (pristop k načrtovanju RT sistemov); Gomaa /Gom84/.
3. PAISLey (procesno orientiran, aplikativen, interpretativen, specifikacijski jezik) /Zav84b/.
4. Metoda za struktuirano analizo velikih RT sistemov; Hatley /Hat84/.
5. Struktuiran razvoj za RT sisteme; Ward and Mellor /WaM86/.

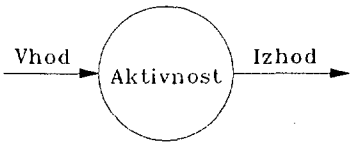
V nadaljevanju bomo obravnavali dve metodi: MASCOT in Ward/Mellor-jev strukturni razvoj.

5. MASCOT

Natančna (uradna) definicija metodologije MASCOT je podana v "The Official Handbook of MASCOT", MASCOT Suppliers Association. Poglede na različne aspekte metod najdemo v delih /JaS75/, /SiJ79/ in /Sim82/. Pri uporabi sistema MASCOT lahko preliminaro načrtovanje RT sistema interpretiramo kot konstruiranje virtualnega stroja, na katerem rešujemo dani problem. Virtualni stroj je zgrajen iz posebne množice abstraktnih enot /All81/. Konstruiranje takšnega stroja s pomočjo Module 2 je opisal /Bud85/. V naslednjih stopnjah načrtovanja se posvečamo implementaciji virtualnega stroja na realnem, ali na več realnih strojev, računalnikov. Ta metoda načrtovanja ne predpostavlja rabo določenega števila procesorjev.

Nadalje si oglejmo abstraktne entitete, kot so: aktivnost, komunikacija, kanal, nabiralnik in sinhronizacija.

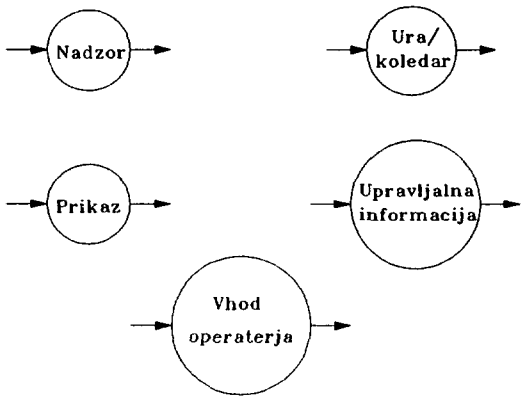
Aktivnost. Vsako aktivnost, ki se izvaja, označimo s krogom. Shematično je aktivnost prikazana na sliki 12, množica aktivnosti iz primera na sliki 4 pa na sliki 13.



Slika 12. Notacija aktivnosti.

Predpostavljamo, da virtualni stroj obravnava vsako aktivnost kot ločeno opravilo. Tako se le-te lahko nanašajo na opravila ali procese. Vendar je bolj zaželjeno, da jih do nadaljnega imenujemo aktivnosti, ker se na tej stopnji še ne odločamo o dejanski

strukturi opravil: več aktivnosti je lahko združenih v eno samo opravilo.



Slika 13. Aktivnosti iz primera na sliki 4.

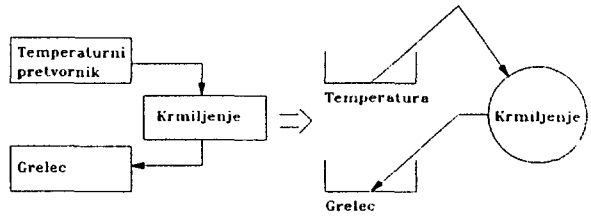
Vsekakor sistem na sliki 13 ni delujoč, saj manjkajo še posebne aktivnosti, ki urejajo odnose med posameznimi aktivnostmi (tj. komunikacije med aktivnostmi).

Komunikacije med aktivnostmi delimo na tri tipe:

- neposredni prenos med dvema akcijama;
- porazdeljevanje informacije večim akcijam; in
- sinhronizacijski signali.

Kanali. Učinkovit koncept za opisovanje direktnih komunikacij med aktivnostmi je kanal (podoben vodu - pipe). Poleg tega, da kanal povezuje dve aktivnosti, lahko združuje tudi neke pomnilniške lastnosti. Tako se lahko skozi kanal pretaka "hkrati" več posameznih (delov) informacij iz množice informacij. Posamezni deli informacije so navadno urejeni tako, da je prvi del informacije, ki vstopi v kanal, na drugem koncu kanala tudi prevzet. Na tej stopnji načrtovanja je skrb za implementacijo kanala nepotrebna. Obstaja pa več metod implementacije, ki zadevajo problematiko konkurenčnega programiranja. Zato jih v tem sestavku ne bomo obravnavali. Kanal označimo s posebnim simbolom: prečno črto med vhodom in izhodom (glej sliko 15!).

Nabiralniki. Fizikalne naprave (pomnilniki datotek, prikazovalniki ipd.) so naprave nadziranega okolja. Zamišljamo si jih kot nabiralnike. Nabiralnik (pool) imenujemo zbirko informacij, ki jo je mogoče čitati in/ali vanjo vpisovati iz večih aktivnosti sistema. Operacija čitanja informacije v nabiralniku ne briše ali spremeni. Tako je lahko vsaka informacija, ki je shranjena v nabiralniku, večkrat prečitana. Nabiralnik dela kot banka podatkov ali shramba informacij. Katerikoli del informacij v nabiralniku je enako lahko dosegljiv vsem aktivnostim, ki nabiralnik uporabljajo. Na stopnji načrtovanja tudi v tem primeru ni pomemben mehanizem ustvarjanja in delovanja nabiralnika - nabiralnik je preprosto abstraktna ideja. Pri nekaterih implementacijah so potrebni mehanizmi, ki ščitijo podatke pred okvarami in nadzorujejo uporabo informacij. Nabiralnik je simboliziran na sliki 14 kot nabiralnik temperaturnih vrednosti in krmilnih vrednosti grelnika.



Slika 14. Primer načrtovalne notacije (sistem nadzora tempera ture).

Preprost krmilni sistem na sliki 14 je sestavljen iz temperaturnega pretvornika, nadzorne naprave (krmilnika) in grelca. Krmilna naprava odčitava temperaturne podatke iz temperaturnega merilnega mostička, jih procesira in vpisuje v krmilnik grelca. Programski model obravnava temperaturni pretvornik in nadzirani grelec kot nabiralnik, krmilno napravo pa kot aktivnost.

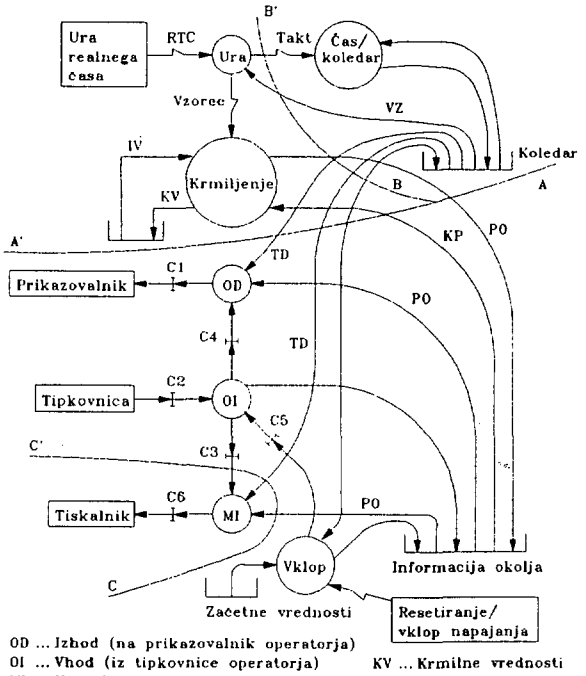
Sinhronizacija. V programih, ki zahtevajo potrditev, je potrebna sinhronizacija procedur. Ta zahteva je še posebej izrazita pri večopravilnem programu, v katerem se podatki razporejajo med več opravili. Poleg drugih aktivnosti so potrebne tudi aktivnosti za stop, start in zakasnitev (delay). Potrebne so tudi takšne aktivnosti, ki zahtevajo npr. vprašanje "Si pripravljen na sprejem podatka ?" ali vpračanje "Imaš kakšen podatek za mene ?". Aktivnostim ni dovoljeno, da se prekinejo, poženejo ali da zahtevajo vprašanja od drugih aktivnosti direktno. Pri razporejanju podakov bomo namreč videli, da je potrebna aktivnost, ki daje ukaz ali zahtevo za razpoznavanje stanja aktivnosti, kateri druga aktivnost nekaj ukazuje ali jo sprašuje. Direktna komunikacija signalov aktivnosti (ukaza ali zahteve) vodi k metodi sinhronizacije, ki ne omogoča verifikacije programa. To še ne pomeni, da direktne metode niso uporabljive: običajno se pogosto uporabljajo zastavice, ki so skupne, dosegljive večim aktivnostim, in označujejo, da se je izvršil nek poseben dogodek. Oglejmo si ta način na našem prejšnem primeru; za sinhronizacijo med CC in D aktivnostjo je potreben interval 5 sek; zastavico postavi CC vsakih 5 sekund, medtem ko D aktivnost v intervalih ugotavlja, če je zastavica že postavljena.

Uporaba te metode pri velikih sistemih prinaša več problemov. Predvsem je neugodno, da mora operator aktivnosti nepretrgoma pregledovati zastavico; in, če so prisotne še kake druge aktivnosti, mora operator aktivnosti poskrbeti, da D aktivnost čaka na ponovno testiranje zastavice. Potreben je nek mehanizem, ki omogoča, da D aktivnost čaka na dogodek (5 sekundni interval) in na CC aktivnost, ki signalizira, da se je dogodek pojavil.

Če zahtevamo, da imata proceduri WAIT(dogodek) in SIGNAL(dogodek) spodaj naštete lastnosti, potem lahko z njima opišemo želeno sinhronizacijo.

- WAIT(dogodek). Opravilo začasno ustavi aktivnost, čim se izvrši WAIT operacija. Neaktivnost traja do signalizacije dogodka, če se le-ta pojavi. Če pa je dogodek signaliziran že pred WAIT operacijo, se aktivnost takoj nadaljuje (tj. kontinuirano, brez čakanja).
- SIGNAL(dogodek). Operacija SIGNAL signalizira, da se je zgodil dogodek in tako omogoči, da se proces čakanja nadaljuje.

WAIT je procedura, ki kontinuirano odčitava dogodkovni status kanala ali nabiralnika, SIGNAL pa procedura, ki vanju vpisuje informacijo. V primeru ene same centralne procesne enote, ne more ena aktivnost stalno odčitavati neko spremenljivko in druga aktivnost spremeniti status



Slika 15. Uvodni (preliminarni) načrt programske opreme grelnega sistema.

spremenljivke za prekinitveno aktivnost. Zato je za izvajanje WAIT procedure, ki teče na realnem stroju, potrebna drugačna metoda; na uvodni stopnji načrtovanja se nam ni potrebno ukvarjati z vprašanjem, kako proceduro WAIT implementiramo.

Poudariti je treba, da informacija o posameznem dogodku ni nujno omejena na dve aktivnosti; to pomeni, da lahko več aktivnosti čaka na isti dogodek. Seveda lahko dogodek signalizira ena sama aktivnost.

Pri nadzoru opreme RT sistema je lahko dogajanje na opremi zelo pomembno. Takšni dogodki morajo biti SIGNALizirani s pomočjo zunanjih prekinitev, ki predstavljajo nekakšne sinhronizacijske signale. Dogodek, ki je SIGNALiziran z zunanjo prekinitvijo, mora aktivirati operacijo "čakanje" (WAIT). Sinhronizacijski signali so predstavljeni s simbolom impulza, vidni na sliki 15. Naprave (v materialni opremi) so označene s pravokotnimi bloki.

PRIMER

Implementacijo obravnavane notacije si lahko ogledamo na sliki 15. Predstavlja sistem ogrevanja s toplim zrakom. Osnovni načrt tega sistema je podan na sliki 4. Mogoče ga je načrtati na več načinov. V našem primeru je načrtan tako, da je vmesnik okolja predstavljen z nabiralnikom in prikazovalnik operaterja kot zunanja enota, ki je preko kanala povezana z aktivnostmi prikazovalnika.

6. NADROBNO OBLIKOVNAJE PO

Naslednja stopnja načrtovanja programske opreme (PO) je delitev sistema na segmente, kot kaže slika 15. Sistem razčlenjujemo postopoma. Delimo ga na vse manjše dele znotraj vsakega novega segmenta. Način, kako izvesti delitev, je odvisno predvsem od vaje in izkušenosti načrtovalca, ki lahko upošteva nekatere splošne smernice:

1. Grupiranje tistih aktivnosti, ki imajo podobne časovne karakteristike.
2. Grupiranje tistih aktivnosti, ki so tesno povezane z zunanjo opremo (okoljem).
3. Grupiranje aktivnosti tako, da je število kanalov, ki povezujejo posamezne grupe med seboj, čim manjše.

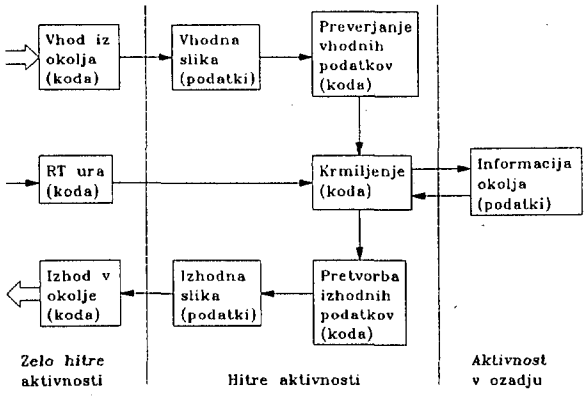
Z uporabo tretjega pravila dobimo razdelitev, ki je na sliki 15 označena s pikčastimi črtami in oznako AA'. Če sedaj uporabimo še pravilo 2, razdelimo gornji del diagrama tako, kot kaže pikčasta črta BB', s pravilom 1 pa še spodnjo polovico diagrama tako, kot kaže pikčasta črta CC'. V naslednjem razdelku bomo bolj na drobno obravnavali povezavo A'BB'

med segmentom, ki vsebuje urine aktivnosti, in segmentom, ki vsebuje aktivnosti krmiljenja.

Najpreje bomo bolj podrobno obravnavali funkcije, ki se morajo izvajati s pomočjo aktivnosti znotraj segmenta. Te aktivnosti so:

1. URA (CLOCK)
   (a) strežba urine prekinitve;
   (b) prekinitve števnika urinih taktov za generiranje verige urnega takta (TICK);
   (c) štetje verig za generiranje vzorcev (SAMPLEs); in
   (d) preverjanje, če se spremeni VZOREC_INTERVAL
2. KRMILJENJE (CONTROL)
   (a) čitanja in krmiljenje podatkov procesa v okolju;
   (b) izračuna krmilnih vrednosti;
   (c) izhodnih krmilnih vrednosti za okolje;
   (d) vnašanja podatkov OKOLJE_PODATKI v nabiralnik; in
   (e) preverjanje, če so NASTAVITVENE_TOČKE ali KRMILNI_PARAMETRI spremenjeni;

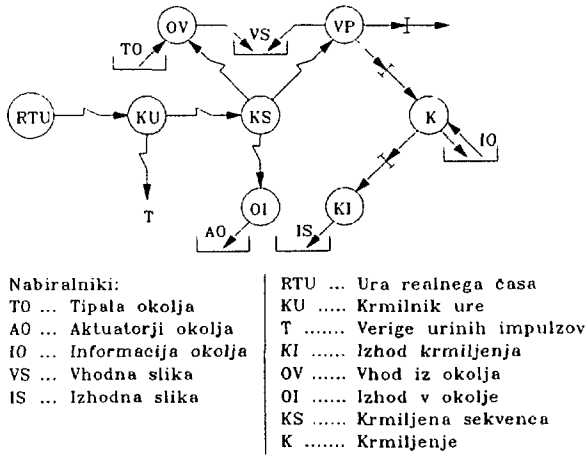
Nadaljne razbijanje programske opreme ure je nesmiselno. Iz funkcij ostalih aktivnosti pa je razvidno, da je le-te mogoče grupirati v tri grupe. Z okoljem sta tesno povezani le dve aktivnosti: zajemanje (t.j. sprejemanje) podatkov in oddajanje krmilnih vrednosti. Obe aktivnosti sta skupaj s prekinitveno rutino časovno najbolj kritični. Zato pripadata grupi z najvišjo prioriteto (t.j. grupi tistih aktivnosti, ki se morajo aktivirati zelo hitro). Glede na notacijo sistema (slika 17) je mogoče preostale aktivnosti krmiljenja nadalje grupirati tako, kot kaže slika 16.



Slika 16. Shema delitve krmilne programske opreme.

V vsakem intervalu vzorčevanja se ob taktu RT ure zajamejo podatki na vhodu iz okolja, t.im. "vhodna podatkovna slika". Nato se istočasno s krmilno informacijo shrani v izhodno podatkovno polje, t.im. "izhodna podatkovna slika", ki se pošlje nazaj v nadzorovano okolje. Informacija, shranjena v izhodni podatkovni sliki, predstavlja krmilne vrednosti, izračunane na osnovi predhodne vhodne podatkovne slike. To pomeni, da vsakemu intervalu vzorčenja sledijo nove izhodne podatkovne vrednosti. Prednost takšnega pristopa je, da sistem zagotavlja krmilne izhode v natančno določenih intervalih vzorčenja. Čas t_c, ki je določen s časom izračuna krmilnih vrednosti, mora zadoščati kriteriju t_c t_s in nič ne de, če t_c variira, le da je manjši od t_s. Krmilnik mora biti načrtovan tako, da kompenzira mrtvi čas, ki se pojavlja med vzorčenjem. Zato mora biti ta čas upoštevan že v modelu nadzorovanega okolja. Iz slike 16 je jasno, kako se lahko pri RT krmiljenju uporabi več procesorjev. Možne so paralelne operacije OKOLJE_VHOD (OV), OKOLJE_IZHOD (OI) in KRMILJENJE (K), ki lahko tečejo na ločenih procesorjih.

Nadaljno delitev aktivnosti krmiljenja predstavlja grupa, sestavljena iz operacij VHOD_PREVERJANJE (VP), KRMILJENJE_IZRAČUN (KI) in IZHOD_PRETVORBA (IP). V preprostih sistemih ta delitev ni nujna, v splošnem pa se mora vhod iz nadzorovanega okolja testirati na pogoje alarmiranja in na karakteristike tipal okolja (smer, napake, ipd. /Ise84/). Izhod na nadzorovano okolje mora biti prilagojen posebnim karakteristikam aktuatorjev. Načrtovalec se mora sam odločiti, ali smatra te operacije kot del aktivnosti OKOLJE_VHOD oz. OKOLJE_IZHOD ali kot ločene aktivnosti VHOD_PREVERJANJE in IZHOD_PRETVORBA, kot je prikazano na sliki 16. V primerih, da imamo samo eno krmilno zanko, je slika razčlenitve modula krmiljenja preprostejša (slika 17).



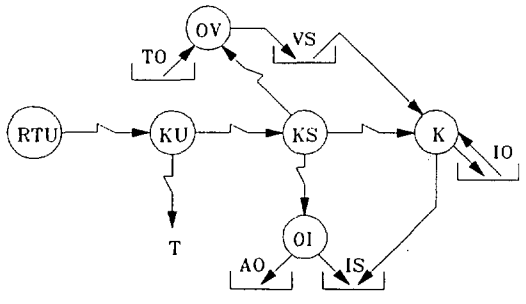
Slika 17. Razdelitev modula krmilne sekcije.

Vsako delitev sistema na sliki 15 moramo še naprej deliti v manjše module. Z vsakim novim diagramom

aktivnosti, ki ga na ta način dobimo, pa bo slovar podatkov bogatejši.

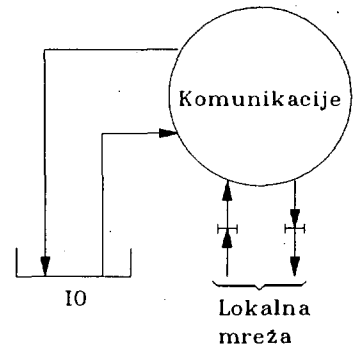
7. PREGLED NAD NAČRTOVANJEM

Nad načrtovanjem, ki vsebuje pogoste intervale v procesu načrtovanja, moramo imeti zelo dober pregled. Ta je sestavljen iz pregleda nad materialno in programsko opremo. Načrtovanje programske opreme ne more prehitevati odločitev v pogledu materialne opreme (npr. števila uporabljenih procesorjev). Programske strukture omogočajo že na uvodni stopnji načrtovanja identificirati aktivnosti, ki se lahko izvajajo paralelno (čeprav bo verjetno neka sinhronizacija med njimi potrebna). Cena računalniške materialne opreme sistema bo lahko v primeru na sliki 3 zaradi izbire procesorja za vsako aktivnost posebej prevelika. Po specifikacijah primera mora vsaki od 12-ih enot pripadati vsaj en procesor in enote morajo biti identične. Možni sta dve konfiguraciji materialne opreme, ki sta pokazani na sliki 4. V primeru konfiguracije 1 se morajo aktivnosti, podane na sliki 17, in aktivnosti, ki pokrivajo potrebe operaterja in upravljanja, izvajati na enem procesorju. V primeru konfiguracije 2, so potrebna sredstva komuniciranja z lokalno mrežo. Modul na sliki 17 se tedaj modificira tako, kot kaže slika 19. Odločitev bo vplivala na posebne komunikacijske mehanizme, uporabljene med aktivnostmi, in na dodeljevanje le-teh. Na primer, pri odločitvi, da uporabimo en procesor za aktivnosti na sliki 18 pomeni, da aktivnosti KRMILJENJE_SEKVENCA, OKOLJE_VHOD, KRMILJENJE_IZRAČUN in OKOLJE_IZHOD smatramo kot eno opravilo, ki ga lahko programiramo takole:



Slika 18. Poenostavljena delitev modula.

```
PROCEDURE KrmilnoOpravilo;
  REPEAT
    Wait(takt);
    OkoljeVhod;
    Okoljelzhod;
    Krmiljenjelzračun:
  UNTIL stop;
END KrmilnoOpravilo;
```



Slika 19. Modifikacija, ki omogoča komunikacijo z mrežo.

8. MASCOT SISTEM

MASCOT predstavlja prvi poskus formalizacije načrtovanja RT sistemov. Podpira jezik CORAL 66, ki je bil tedaj uporabljen v skladu z zahtevami del na projektih obrambe V.Britanije. Ker CORAL ne podpira večopravilne sisteme, naj bi MASCOT to pomnjklivost odpravil.

MASCOT ima dodatne prednosti, ki so bile že omenjene. Te se nananšajo na grupiranje aktivnosti v opravilih in na prilagajanje operacijskega sistema karakteristikam opravil.

Bistvene pomanjklivosti sistema MASCOT so:

- Slabo podpira velike sisteme ali sisteme, ki tečejo na večih procesorjih.
- MASCOT ni struktuiran ("flat" sistem). Vse informacije morajo biti prisotne na enem samem nivoju. Tako ne podpira t.im. metodologijo načrtovanja od zgoraj navzdol (top-down design methodology).
- Sinhronizacijski mehanizmi so omejeni: predvsem, opravil ni mogoče sinhronizirati tako, da se izvajajo v določenih urinih intervalih.

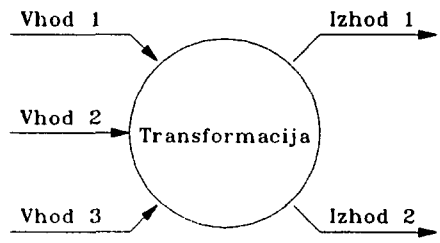
Nekatere od naštetih pomanjklivosti naj bi bile kasneje odpravljene z razvojem sistema MASCOT 3 /Sim84/. Tudi sistem DARTS, ki ga je razvil Gomaa v letih 1984 in 1986 je podoben MASCOTu. Slednji omogoča številne abstrakcije, s katerimi je mogoče zgraditi model načrtovanega RT sistema.

9. STRUKTURNI RAZVOJ

Struktuiran razvoj RT sistemov sta vpeljala Ward in Mellor leta 1986.

Dobro vtečena metoda za načrtovanje sistemov, ki ne delajo v realnem času, je t.im. "metoda pretoka podatkov" (dataflow method). Načrtovalec opazuje

program z vidika prenosa podatkov. Osnovni zapis, ki se uporablja pri tej metodi, je prikazan na sliki 20.



Slika 20. Notacija podatkovnega pretoka.

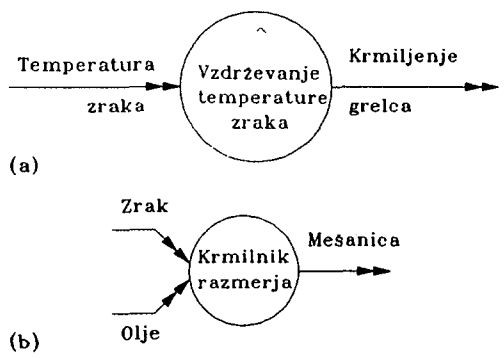
Tu se vhodni podatkovni tokovi transformirajo v določene izhodne podatkovne tokove. Transformacija je ekvivalentna akciji, ki jo poznamo v sistemu MASCOT. Pri tem se predpostavlja, da je podatek, ki se obdeluje, časovno diskreten. To pomeni, da je določen samo v specifičnem trenutku, v ostalem času pa je nedoločen ali ima vrednost 0.

Metoda je široko uporabljena pri načrtovanju transakcij, ki se nanašajo na obdelavo podatkov. Takšen tip sistema je na primer program, ki izvrši preklic bančnega računa. Vsak preklic je ločen diskreten dogodek (transakcija), med katerim podatek ni določen.

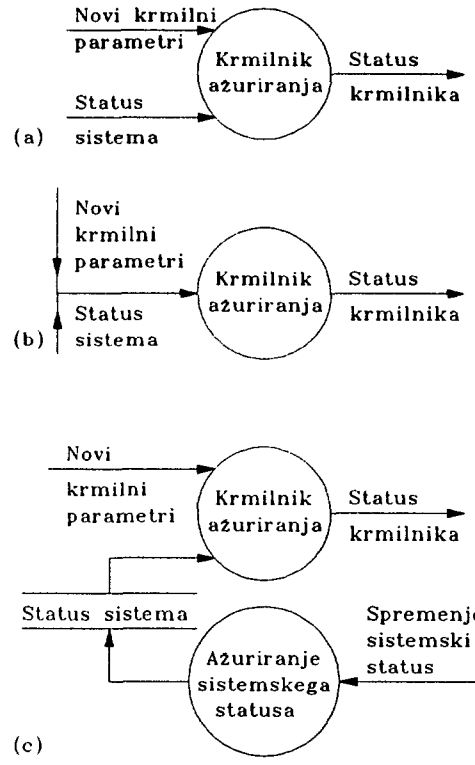
RT sistemi ne vsebujejo samo časovno diskretnih podatkov. Logični in krmilni sistemi imajo časovno zvezne podatke, kot tudi dogodkovne podatke (logične signale). Pri časovno zveznem podatku smatramo, da je podatek določen na neprekinjenem časovnem spektru (Pomni, da se pri načrtovanju podatek celo med vzorčenjem obravnava kot zvezni podatek). Ker obravnavamo različne tipe podatkov in dogodkov, mora biti notacija podatkovnega pretoka ustrezno razširjena.

9.1. Podatkovna transformacija

Primere časovno kontinuiranih transformacij kaže slika 21. Dvojna puščica na linijah pretoka podatkov označuje neprekinjeno naravo podatkov.



Slika 21. Časovno zvezni podatkovni pretok.



Slika 22. Sinhroni podatkovni pretoki:
a) dvoumni diagram
b) spojeni tokovi (merged flows)
c) uporaba podatkovnega pomnilnika

Pri diagramu časovno diskretne transformacije (slika 22) je možna dvoumnost, saj lahko diagram interpretiramo na dva različna načina: Po prvem načinu se transformacija izvrši tedaj, ko se pojavita oba vhoda hkrati; po drugem načinu pa je transformacija možna tudi v primeru, da se vhoda ne pojavita istočasno. Vhod, ki se pojavi prvi, se shrani. Ko se pojavi še drugi in sta tako prisotna oba hkrati, je izpolnjen pogoj, da se transformacija izvrši. Da ne more priti do opisane dvoumnosti, je bil vpeljan koncept sinhrone transformacije.

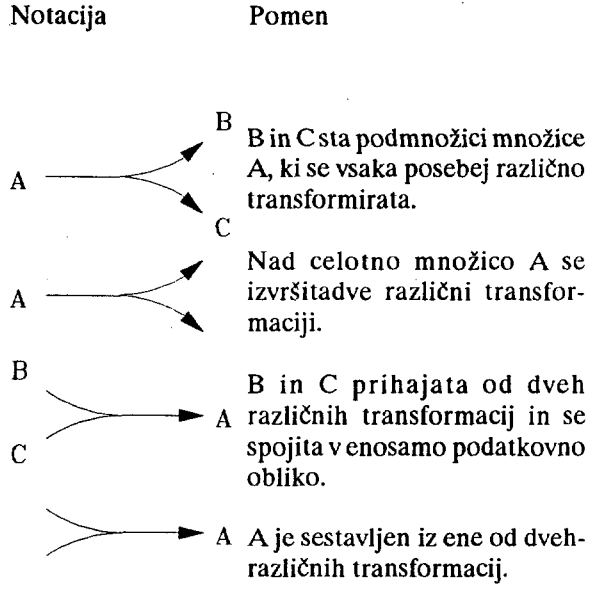
Pri sinhroni transformaciji moramo upoštevati naslednje dogovore:

- Možen je samo en diskreten vhod (vhodi za shranjevanje podat kov in vzbujanaja /glej spodaj/ niso upoštevani)
- Diskretni vhod je lahko sestavljen. Da se transformacija izvrši, morajo biti prisotni vsi elementi.
- Če obstaja več diskretnih izhodov, se morajo le-ti medsebojno izključevati.

Če upoštevamo gornja pravila, lahko transformacijo na sliki 22a predstavimo kot transformacijo na sliki 22b s sestavljenim vhodom ali kot transfor-

macijo na sliki 22c s podatkovnim pomnilnikom, ki pomni status sistema.

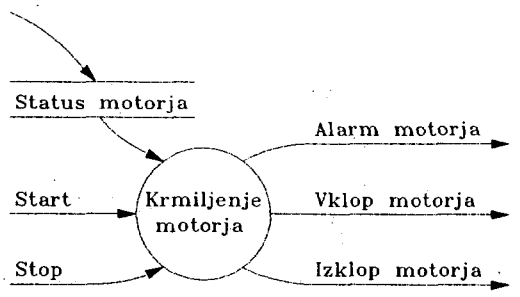
Podatkovni pomnilnik, ki je na sliki označen z dvema paralelnima črtama, kaže, da je podatek med dvema časovno diskretnima podatkoma zadržan. Pomen ostalih notacij t.im. spojenih podatkovnih pretokov je dan na sliki 23.



Slika 23. Notacije podatkovnih pretokov.

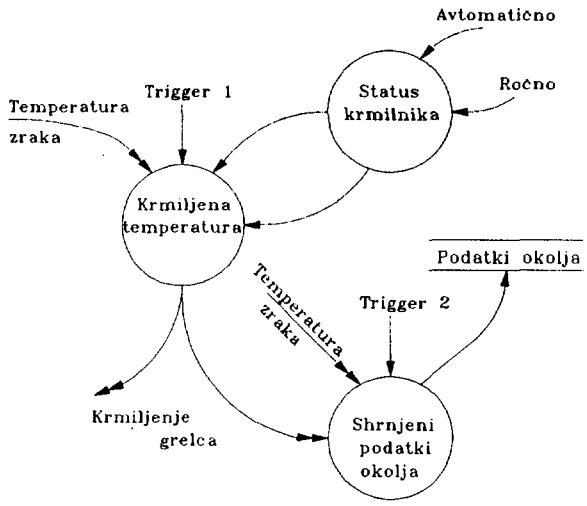
9.2. Krmilne transformacije in pozivi

Notacija krmilnih transformacij je pokazana na primeru na sliki 24. Dogodkovni pretoki in transformacije so označeni s prekinjenimi črtami. Vsaka sprememba krmiljenja je vezana na dogodek. Podatki o dogodkih okolja (ki v našem primeru predstavljajojo status motorja) se hranijo v posebnem pomnilniku.



Slika24. Krmilni podatkovni pretoki.

Koncept pozivov omogoča razmejitev podatkovnih in dogodkovnih pretokov. Ločimo tri pozive: ENABLE, DISABLE in TRIGGER. Njihova uporaba je pokazana na sliki 25. Pozivi se ne obravnavajo kot vhodi v pretvorbo (transformacijo), ampak preprosto kot stikala, ki so uporabljena zato, da se transformacija vklopi ali izklopi. Poziv TRIGGER požene transformacijo in se lahko uporablja za transformacijo časovno zveznih podatkov v časovno diskretne podatke.



Slika 25. Podatkovni pretok, ki kaže uporabo pozivov.

9.3. Pregled metod

Sistem podpira omejeno grafično notacijo za predstavitev splošne strukture RT sistema. Pravila, povzeta v spodnji tabeli 2, omogočajo preprosto preverjanje povezav vmesnikov med različnimi

Transformacija	Vhodi	Izhodi
Podatki	Podatkovni pretok	Podatkovni pretok
	Pozivi	Dogodkovni pretok
Krmiljenje	Dogodkovni pretok	Dogodkovni pretok
	Poziv	Poziv

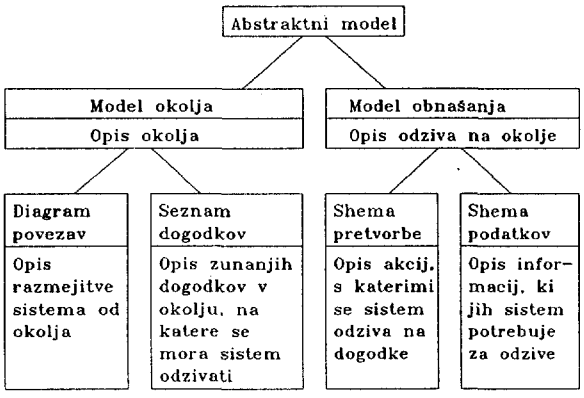
Tabela 1. Pregled vhodov in izhodov transformacij.

transformacijami. Ker so podatkovni pretoki jasni, je pri odločanju o delitvi sistema na module preprosta tudi uporaba načrtovalne heuristike za minimizacijo podatkovnih pretokov med posamez-

nimi deli sistema. Pri tem se moramo zavedati, da notacija upošteva spojene podatkovne pretoke in zato lahko enojna linija predstavlja kompleksno podatkovno strukturo in kompleksni vmesnik.

9.4. Zgradba modela

Opisano formalno metodo sta izdelala Ward in Mellor /WaM85/. Metoda sloni na dveh modelih: abstraktnem in izvedbenem. Modela se gradita sočasno, ker lahko visoko-nivojske izvedbene nadrobnosti vplivajo na nižje nivoje abstraktnega modela. Posamezne faze gradnje abstraktnega modela so razvidne iz slike 26.



Slika 26. Abstraktno modeliranje po Ward/Mellor-ju.

Na prvi stopnji se sistem določa glede na povezave z okoljem. Pri ten se uporablja diagram povezav (povezovalna shema) in seznam dogodkov.

Na drugi stopnji modeliranja se razvija vedenjski model (model obnašanja). Ta model kaže, kako se sistem odziva na aktivnosti v okolju. Razviti ga je mogoče na dva načina: po shemi transformacije ali po shemi podatkov.

Vsaka transformacija je oštevilčena. Z razčlenitvijo posameznih transformacij se lahko število množic diagramov povečuje, dokler operacije znotraj vsake transformacije niso več deljive. Pretvorbena specifikacija pripada vsaki transformaciji diagrama.

Pri načrtovanju opravila (kot enojnega opravila) lahko uporabimo običajne podatkovno pretočne metode. Alternativno metodo pa predstavlja uporaba nekega ustreznega višjega programskega jezika, n.pr. Module 2 /Bud85/.

10. ZAKLJUČEK

Obravnavali smo tri pristope k načrtovanju programske opreme RT sistemov, ki so zasnovani na enojnem programu, na priviligeranih programih ali na izvajanju več opravil hkrati. Zadnja metoda je najbolj splošna. Pred ostalima dvema ima prednost, ker uporablja abstraktne pojme (nabiralnike, kanale, signale in aktivnosti v MASCOT notaciji; ali podatkovne in dogodkovne pretoke s podatki in krmilnimi transformacijami v podatkovno pretočni notaciji) in omogoča načrtovalcu sistema, da se osredutoči na uporabniške vidike problema. Na kasnejši stopnji načrtovanja, ko je ugotovljena istovetnost različnih aktivnosti, obstaja možnost, da se načrtovalec opre na eno od ostalih dveh metod, če katera od njih omogoča boljše rešitve.

Metoda se v prvih korakih načrtovanja programske opreme ogradi od nadrobnih rešitev materialne opreme. Notranja struktura aktivnosti se lahko načrta tako, kot da so sekvenčno programirane. Dejanske izvedbene podrobnosti nabiralnikov, kanalov, signalov, aktivnosti podatkovnih in krmilnih transformacij postanejo ločen problem. Aktivnosti (ali transformacije) se lahko načrtujejo kot sekvenčni programi z uporabo standardnih tehnik /Pres82, Coo90/.

V novejšem času prevzemajo vse večjo vlogo CASE orodja. Z njimi lahko sistematično oblikujemo programsko in materialno opremo skozi celotni razvojni cikel. CASE orodja, ki so danes v uporabi, se najpogosteje opirajo na Ward/Mellor in Hatley/Pirbhai razširitvijo Yourdonove metodologije /HaP87, You88/. Primerjavo obeh metod najdemo v delu /KoK92/. Hatley/Pirbhai metodologija je predvsem zmogljivejča na nivoju specifikacij sistema in kot taka dobro dopolnjuje Ward/Mellor metodologijo. Skupaj omogočata učinkovito strukturno analizo opreme namenskih RT sistemov, ki smo jo v pričujočem delu poskušali pokazati na opisanih formalnih pristopih načrtovanja.

11. LITERATURA

/All81/ S.T.Allworth, Introduction to Rial-time Software Design. Macmillan, 1981.

/BenL84/ Bennett and D.A.Linkens (eds), Real-Time Computer Control. Peter Peregrinus, Stevenage, 1984.

/Bud85/ D.Budgen, Combining MASCOT with Modula-2 to aid the engineering of real-time systems. Software: Practice and Experience, 15(8), pp.767-93.

/Coo86/ J.E.Cooling, Real-Time Interfacing. Van Nostrand Reinhold (UK), ISBN 0-442-31755-7, 1986.

/Coo90/ J.E.Cooling, Software Design for Real-Time Systems. Chapman and Hall, ISBN 0-412-341-808, 1990.

/Gom84/ H.Gomma, A software design method for real-time systems. Communications ACM, 27(9), pp.938-49,1984.

/Hat84/ D.J.Hatley, A structured analysis method for large real- time systems. The Heap, DESIG Structured Lnguages, 7(2), pp.21- 38, 1984.

/Ise84/ R.Iserman, Process fault detection based on modelling and estimation methods-a survey, Automatica 20(4), pp.387-404.

/JaS75/ K.Jackson and H.R.Simpson, MASCOT-A modular approach to software construction, operation and test. IRE Tech.Note, No.778.

/KoK92/ B.Koroušić and P.Kolbezen, Razširitvi Yourdonove SADT metodologije za razvoj namenskih sistemov, Zbornik ERK 92, Portorož, B: 27-30, 1992.

/May78/ G.J.Myers, Composte Structured Design. Van Nostrand Reinhold, 1978.

/Par72/ D.L.Parnas, On the criteria to be used in decomposing systems into modules. Communications ACM 15(12), pp.1053-66, 1972.

/Pres82/ R.S.Pressman, Software Enginnering: a Practitioner's Approach. McGraqw-Hill, 1982.

/StMC74/ W.P.Stewens, G.I.Myers and L.L. Constantine, Structured design. IBM Systems J.13,p.15, 1974

/Sim84/ H.R.Simpson, MASCOT 3. IEE Colloquium, Paper No.3, 1984.

/WaM86/ P.T.Ward and S.J.Mellor, Structured Development for Real- Time Systems. Englewood Cliffs, NJ:Yourdo Press, ISBN: 0-13- 854787-4, 1985.

/You88/ E.Yourdon, Modern Structured Analysis. A Prentice-Hall Campany, 1988.

/You79/ E.N.Yourdon, Classics in Software Engineering. Yourdon Press, 1979.

/Zav84b/ P.Zave, An overview of the PAISLey project-1984. Tech. Rep., AT and T Bell Laboratories, 1984.

COMPUTERS IN MATHEMATICS IN SLOVENIA

Nika Gams
Moste High School, Zaloška 49, Ljubljana, Slovenia
AND
Matjaž Gams
Institute Jožef Stefan, Jamova 39, Ljubljana, and
Faculty of pedagogy, Kardeljeva pl. 16, Ljubljana, Slovenia
E-mail: matjaz.gams@ijs.si

Keywords: computers, education, mathematics

Edited by: Rudolf Murn
Received: November 1992 **Revised:** November 1992 **Accepted:** November 1992

An introduction to computers in Slovenian schools is first presented through a short history, and comparisons with other countries. An overview of the ten best schools in the world is presented in order to analyse successful models, especially mathematical courses in the Netherlands, German high schools, teacher training in Germany and Japan's scientific programmes. Then, the basics of introducing computers in educational courses are proposed. Finally, two programs for symbolic computing, Derive and Mathematica, are briefly presented.

1 Introduction

Twenty years ago in Slovenia, high school students as well as faculty students slowly began using calculators while the majority of teaching and calculating was still performed by logarithmic tables. With the coming boom in calculators, teachers warned that such tools could effectively decrease understanding of mathematics. Similar questions were raised a long time ago. For example, here are some remarks from the Teacher's Conference (Soloway 1991): "Students today can't prepare to calculate their problems. They depend upon their ...s which are much more expensive. What will they do when the ... is dropped and it breaks?" Before replacing dots with 'calculator' let us remind you that it was the year 1703 and the dots should actually be replaced by 'slots'. And in 1815, similar remarks emerged (Soloway 1991): "Students today depend upon paper too much. They don't know how to write on a slate without getting chalk dust all over themselves. They can't clean a slate properly. What will they do when they run out of paper?".

It would be too naive to say that these hesitations were caused by prejudices from uninnovative teaching staff. Furthermore, Slovenia should very carefully choose the appropriate model since mistakes in a country with just two universities might get very expensive. Therefore, we shall try to analyse existing and forthcoming changes regarding the use of computers in mathematics, and education in general.

2 Comparisons With Other Countries

Relatively small and underdeveloped countries like Slovenia often compare to and copy from major powers like the USA and Germany or from their developed neighbours (Italy and Austria in our case). On the other hand, comparisons with less developed countries could be quite informative and helpful to curb unrealistic attempts. Let us compare, for example, Romania. For most of the last 20 years, Elena Ceausescu personally ran the Romanian science and technology sector (Goodman 1991). Drawbacks in a highly centralised, censored and insufficient scientific and educational structure are well known. Yet, computing was favoured under the Ceaucescus and the Romanian company ICE is still building IBM/PC-like microcomputers and duplicates of Digital's VAX 11/730. At the time of the first democratic election in 1991 (about the same time as in Slovenia), all of their roughly 40 county-sized electoral districts were connected to the national centre in Bucharest via telephone lines and a simple star-like dial-in topology. In comparison, results from Slovenian electoral districts were transported by cars on floppy disks. The resulting astonishing delay of a couple of days was predicted in advance since the inefficient and complicated counting algorithm needed a second pass. Furthermore, one of the two main Slovenian computer companies, the one which was producing Digital-like computers went bankrupt a couple of years ago. Similar comparisons with Hungary or Bulgaria show that many neighbour-

ing countries possess quite developed educational systems and computer facilities.

On the other hand, there are great comparative advantages in Slovenia. For example, there are 700,000 registered vehicles per 2,000,000 inhabitants. In the informatics and computer sector, there are 15,000 people which represents nearly 1% of the whole population. Computer equipment as well as their introduction into mathematical courses is quite comparable to those of the developed neighbouring countries like Italy or Austria. Indeed, the number of personal computers in private homes can be estimated to be about the same level as in the most developed countries, while the number of programs int the average computer is quite probably even higher. This situation is a result of several factors, among them:

- computer is one of the main status or prestige symbols in Slovenia
- technological improvement at the lower level (Sinclair, Commodore, IBM PC) of the computer spectrum enabled relatively cheap and yet powerful computers (Gams, Hribovšek 1992)
- several hardware and software products were obtained without customs costs.

The two main disadvantages of introducing computers in Slovenian educational programmes are:

- troubled economy, and
- insufficient number of staff experienced in dealing with computers, resulting in poor maintenance.

However, even relatively cheap stand-alone IBM PC 386 supports the most powerful system for symbolic computing, named Mathematica. Even the old IBM PC 286 with 360K floppy disks is powerful enough to support Derive, another program for symbolic computing.

3 The World's Ten Best Schools

An overview with this title was presented in Newsweek, December 2, 1991. It is very consistent with what can be obtained from other comparative studies. The subtitle is: "And What Can We Learn From Them", suggesting that a good education no longer stops at borders. In the presented study it is claimed, that while many nations worry that their schools have stumbled, there are many models to follow around the globe. The report openly admits that USA high schools are not at the level of the world's best high schools while their postgraduate and Ph.D. study is one of the best in the world.

In another study in Scientific American, October 1992, it is claimed that the performance of US students in math and science is abysmal. For example, the Carnegie Commission's report observed that 47% of the nation's 17-year-olds cannot converge nine parts out of 100 to a percentage. Bill G. Aldridge, executive director of the National Science Teachers Association, complained that when teaching density, one third of middle-school teachers could not use the gradations on a ruler to measure fractions of an inch. It is claimed, that their error-filled textbooks are often designed with one eye more to the religious sensitivities of adoption committees in key markets than to accuracy. Furthermore, local authorities who provide most of the funds, often follow their interest and directions while national standards in the USA are just being debated.

It is well known that one of the most important single factors is the esteem of educators. For example, in the USA people with faculty education earn roughly 1/3 more than people who have completed high school. A similar comparison holds for university teachers versus teachers in high or elementary schools. Being educated in developed European countries is generally more important than in America. In Japan and even among the "boat people" in the USA, students and their parents attach greater importance to academic achievements and more closely cooperate than do their average counterparts in the USA. In this superpower, the proportion of students answering "No" to the question "Do you like science" in recent years increased from 20 to 35%. In addition, Japan and Germany spent more on teacher salaries than the United States. The United States tends to spend more money on buildings and administration. Germany and Japan both spend about 50% less per student than the United States does, yet rank higher. In both countries, the schooling period is longer than in the USA, compared from 240 days in Japan and 210 in Germany to 180 days a year in America. Similarly, the average schooling period per worker is shorter in the USA compared to Japan or Germany.

In comparison, Slovenia has around half the percentage of graduates in the whole population while the number of high school students is comparable to developed countries. Despite constant complaints by a very strong union, teacher's salaries are not too low compared to average incomes in industry. There are small differences in incomes between elementary and high schools. However, teacher's jobs are not very respected especially at elementary and high schools where their work is highly regulated. At the moment, faculties in Slovenia have academic freedom even to the level of anarchy since new regulations have not yet been accepted. It is the professors that in most of the schools retain relatively high status of universities and also high schools in Slovenia. Namely, in most of the

comparisons Slovenian schools rate somewhere in the middle.

The 10 Best Schools report proposes the best ten pockets of educational excellence in the world: reading in New Zealand, early childhood in Italy, math in the Netherlands, science in Japan, languages in the Netherlands, art in the United States, high school in Germany, teacher training in Germany, graduate school in the United States, and adult education in Sweden.

We shall analyse in detail mathematical courses in Netherlands, high school and teacher training in Germany, and scientific education in Japan.

It is not that Dutch children rank the best in the world. Several other nations like Japan do as well. However, children in Netherlands do like math, and do score high or even highest on standardised tests, and it is interesting to represent their (European) model. First, 90% of Dutch high school students take advanced math courses. As if the old story repeats again and again - to be successful you simply have to pay sufficient attention to it, be it time or money or both. Interactions with teachers and the use of calculators are frequent, but they are frequent in several other countries as well. It is the revolutionary new system for math instruction called Realistic Mathematics that draws attention. Developed over the past 15 years by researchers at Utrecht University's Freudenthal Institute, Realistic Math starts with the premise that children have a natural way of doing arithmetic in real-life tasks. They begin by using sticks and dots, and later they learn at their own pace reinventing mathematical concepts for themselves. Instead of memorising and applying strict formulas they tend to visualise a problem and estimate the answer rather than using strict hierarchy which is utilised in other countries. Each subject is introduced or at least supported by a real-life problem. Instead of strict division into arithmetics, algebra and geometry, Realistic Math integrates all of these skills at every level.

All high school students complete Realistic Math. At the level of high school and college, future technological graduates switch to more structured and abstract math more similar to common approaches in other countries. However, abstract mathematics is introduced only to students who will quite probably use it in their work. The Freudenthal researchers argue that abstract mathematics does not serve any useful purpose in later life of most of students. Interesting to notice, the number of female students in mathematics has not increased. It goes along with the latest studies of the differences between brains of the two sexes. (It is not clear why similar percentages hold for minority groups in the USA, especially for black people and with the exception of Asians.)

Analysing education in Japan one has to observe extreme differences compared to the United States. Japan is a highly homogeneous country with very high standards we might call ethical. For example, cheating is something very rare compared to the USA and even more to Slovenia which is - unfortunately - not paying sufficient attention to regulate such very common transgressions (smoking and lately drugs would be another example). In Japan, it is just not honourable to do so while students in Slovenia often give admiration to their colleagues for successful cheating at exams. Therefore, it is up to the teachers to control and prevent such cases. Although they are rather successful at that task, and the situation is generally improving with separation from the Balkan philosophy, the remaining point of difference to overcome is the basic relation to and treatment of it.

Japanese (and Korean) students rank highest or close to the highest in the majority of tests from science to mathematics. Around 25% of Japanese elementary-school time goes to science and math and they produce more engineers than lawyers. One should ask oneself, how come that most of the Nobel Prizes go to researchers in the USA? The answer seems to be at hand. In 1968 they introduced the first and in 1977 the second reform which moved the curriculum away from pure science and toward applied. In recent years, Japan has been performing the third revolution with the intention to make learning more interactive and innovative. Students typically discover the laws of nature through live experiments and not by reading a textbook. This covers Maxwell's laws of electromagnetism, fixing the toaster, creating a solar car or genetics. An interesting remark - teaching in schools in Japan is far less automated than believed in the rest of the developed world. Students squirm, punch, and do a lot of other things children generally do. Yet, disciplinary actions are rarely used. Pupils listen to the teachers.

The high school system in Germany (and Switzerland) is considered the best in the world. As in Japan, it is oriented towards practical skills and integration with basic fundamental principles. Students are trained to work, not simply to receive diplomas. One third attends Gymnasiums (Grammar Schools, i.e. college-preparatory schools) and the rest goes to vocational and technical schools that feature job training and academics. The parting comes after 10th grade. Compared to Slovenia, the basic approach seems similar, although results indicate that our high schools are not as successful. For one thing, German business feels it is its responsibility to cooperate and share responsibility to train the next generation. High attention is being paid to develop the future labor pool. The schools are also quite elitist. Only about 12% of Gymnasium students come from working families, roughly similar to Slovenia. Summarily, it is higher attention, better salaries, discipline, and better system, better cooperation with industry and more

real-life examples that contribute to the difference. For example, 100% of teachers in Germany have double academic majors, they are better trained and advised during first teaching exercises, and not unimportant, their salary is sufficient to support a middle-class family comfortably.

In comparison, it is claimed that money in the USA would be best spent on reducing class sizes and increasing teacher's pay. A recent study in Texas by a public policy scholar Ronald F. Ferguson statistically shows important correlation between higher salaries of teachers and higher scores of their students.

4 Computers In Schools

Computers are the most successful tool in teaching. They are useful in teaching computer science, mathematics, geography, pedagogy etc. In fact, there is rarely any teaching discipline which has not been already affected by computers.

After initial complaints that computer hackers and game-playing children are becoming socially alienated, it has become accepted that it is not the case. Today, it is known that children do learn through playing computer games at different levels (Soloway 1991). Children not only learn how to use keyboards and train their reflexes, they also quickly progress their mental skills at all levels. Games like Tetris were studied in detail at the level of activities in human brains. Furthermore, children are not becoming alienated. They converse about problem situations with their colleagues, teachers and parents. They read and exchange magazines about games, and intensely practise various techniques, including fine learning activities like reading, critiquing, talking and sharing, and trying out ideas. In Slovenia, Dr. Vid Pečjak was one of the first to point out that positive effects strongly outweigh negative ones.

If playing games is so attractive and children learn so much, why don't we perform learning in that way? Clearly, motivation and interest are one of the major factors in the successfulness of learning. In fact, several of the successful teaching programmes in the world resemble the game with interesting cases and active cooperation of students. The opportunities at hand increase with constant progress in computer technology. For example, multimedia is becoming reality (Soloway 1992), and even in Slovenia there are programs which enable students to make their own cartoons or videos on IBM PCs. Teachers can prepare lessons as videos, therefore avoiding several problems like temporary headaches or bad moods.

How come that in Slovenia teachers mainly use chalk and blackboard or slide projectors at most? Why don't they have a videoprojector or a PC with attached projector? First, there is not enough financial support. While videos are progressing in America and teachers there are making around $30,000 a year, even they usually don't have a phone on their desk (Soloway 1992). In that light, talking about BBSs and networks sounds a little unusual. Second, technological solutions are prone to errors. Machines break, the glass soon becomes scratched or dirty beyond recognition, small pins on the cables and connectors are bent out of shape. Indeed, even good alternatives like a large screen video monitor, faster and with better colors than projectors, are sensitive machines that easily get out of order.

Soloway (Soloway 1992) proposes that computers in schools meet the following basic needs:

- There should be enough readily available computers for
  - routine tasks on computers: wordprocessing, spreadsheeting, drawing, calculating, and telecommunications.
  - supporting didactic instruction. In every class, there should be a computer with a suitable projection mechanism, TV and overhead projector.
- Each course and each student population has its own special needs. Most of them can be fulfilled with general computers and special additional equipment such as scanners, TVs or music instruments.
- The computers should be capable of growing as the students, teachers and administrators mature together with new technologies. Computers last for 10 years and buying new ones because of inexperience means throwing money through the windows.

Anything has its positive and negative sides. Among negative sides of using computers, it is mainly health that matters. Rather surprisingly, since there is no obvious hazard like when driving cars or inhaling particles and gases when smoking. However, there is sitting, typing and watching a screen from a close distance. It all results in a physical stress accumulated over months or years of practice, or simply as a lack of normal physical activities which are essential in the growth of a normal human. Indeed, it is somehow sad to see each generation having more and more troubles with their backbones and eyes. Filters on screens and exercises at least once per hour do help, however, additional physical exercise is more than needed.

5 Programs For Mathematics

Around 20 years ago programs like Macsyma were regarded as artificial intelligence systems solving complex symbolic tasks in geometry and integration. They

were using heuristic methods. From then on, new systems were emerging on the market, each better and better. Now they are based on a special yet exact branch of mathematics.

Probably the best mathematical system available on the market and running on IBM PC (386 or more) is Mathematica (Wolfram 1991). It also runs on Windows, Macintosh, CONVEX, DEC VAX, HP 9000, IBM RISC System/6000, NeXT, Silicon Graphics, Sun workstations, SPARC, just to mention some of them. Mathematica was first released in 1988 and soon received several awards as one of the best software products. The company Wolfram Research, Inc., has now approximately 100 employees. Mathematica is a general software system for mathematical and other applications. It is currently used by around 100,000 researchers, engineers, and analysts, as well as students from high school to graduate school. The applications of Mathematica span all areas of science, technology and business where quantitative methods are used. The manual has around 1000 pages. It includes a full range of interactive numerical, graphical, and symbolic computation capabilities, all linked to the powerful built-in Mathematica language including sound generation and flexible external program communication. Mathematica has been published in dozens of books, and it has its own quarterly journal.

No doubt that Mathematica is the world leader in scientific, engineering and faculty surroundings. But for high schools and nontechnical faculties Derive (Soft Warehouse 1991) might be even more interesting. First, there is a great difference in size. Mathematica takes at least 8M while Derive is available on one single-sided single-density diskette with 360K. The manual for Derive is only one quarter of the size of the manual of Mathematica, indicating that Derive is smaller and more comprehensible. These are the reasons why Mathematica is being presented in postgraduate studies in Slovenia and Derive is the favourite for high schools. In recent years, Derive has been systematically introduced into mathematical courses in Slovenia through the Ministry of Education, and mathematical and computer communities in Slovenia. For example, special courses were given to high school teachers, and a Slovenian shorter version of the user manual was published (Lokar 1991). Derive offers many possibilities to solve different problems. For example, it takes around half a minute to input

$$\int_0^r \int_0^{\sqrt{r^2-x^2}} x\,y\,dy\,dx$$

and Derive computes it correctly to be $r^4/8$ in a fraction of a second. As for another example, it takes around 10 seconds to compute 1000! on all 2567 digits on IBM PC 386. Last but not least, one can plot functions on the screen as 2- or 3-dimensional figures.

6 Summary

Lessons to be learned are clear enough: give teachers good incomes and thorough education, make a good and stimulative educational system with strong stimulations for successful teaching and learning, quickly eliminate all disciplinary problems by a strict and consistent approach thus avoiding repression and confrontation, mix theory with many real-life examples throughout learning, use as many learning tools as possible, and combine the learning programmes with existing national industrial and infrastructure activities.

On the other hand, it is important to proportionally weigh practical exercises and theory. For example, there are courses in Slovenia which are nearly out of touch with reality while others hardly have any theory in them.

Computers are far the most useful educational tool. Although Slovenia is rather successful in introducing computers in schools, much more can be done with relatively small expenses. Hopefully, some of it will happen since Slovenia is on its way to restructure its schooling system once more, and this time at least basic directions are well known. Systematic introduction of computers into all levels of computing is one of those proper actions to continue with.

Acknowledgement: We would like to thank Aram Karalić for helpful remarks.

References

[1] Soloway E. (1991): *How The Nintendo Generation Learns*, Communications of the ACM, Vol. 34, No. 9.

[2] Goodman S.E. (1991): *Computing and the Resuscitation of Romania*, Communications of the ACM, Vol. 34, No. 9.

[3] Gams, M., Hribovšek B. (1992): *Trends in Computer Progress*, Informatica, Vol. 16, No. 1.

[4] Soloway E. (1992): *Buying Computers for Your School: A Guide for the Perplexed*, Communications of the ACM, Vol. 35, No. 7.

[5] Wolfram S. (1991): *Mathematica*, User Manual, Second Edition, Addison-Wesley.

[6] Soft Warehouse (1991): *Derive*, User Manual, Version 2, Soft Warehouse.

[7] Lokar M. (1991): *Derive, program for symbolic computing*, Raček (in Slovenian).

real-life examples that contribute to the difference. For example, 100% of teachers in Germany have double academic majors, they are better trained and advised during first teaching exercises, and not unimportant, their salary is sufficient to support a middle-class family comfortably.

In comparison, it is claimed that money in the USA would be best spent on reducing class sizes and increasing teacher's pay. A recent study in Texas by a public policy scholar Ronald F. Ferguson statistically shows important correlation between higher salaries of teachers and higher scores of their students.

4 Computers In Schools

Computers are the most successful tool in teaching. They are useful in teaching computer science, mathematics, geography, pedagogy etc. In fact, there is rarely any teaching discipline which has not been already affected by computers.

After initial complaints that computer hackers and game-playing children are becoming socially alienated, it has become accepted that it is not the case. Today, it is known that children do learn through playing computer games at different levels (Soloway 1991). Children not only learn how to use keyboards and train their reflexes, they also quickly progress their mental skills at all levels. Games like Tetris were studied in detail at the level of activities in human brains. Furthermore, children are not becoming alienated. They converse about problem situations with their colleagues, teachers and parents. They read and exchange magazines about games, and intensely practise various techniques, including fine learning activities like reading, critiquing, talking and sharing, and trying out ideas. In Slovenia, Dr. Vid Pečjak was one of the first to point out that positive effects strongly outweigh negative ones.

If playing games is so attractive and children learn so much, why don't we perform learning in that way? Clearly, motivation and interest are one of the major factors in the successfulness of learning. In fact, several of the successful teaching programmes in the world resemble the game with interesting cases and active cooperation of students. The opportunities at hand increase with constant progress in computer technology. For example, multimedia is becoming reality (Soloway 1992), and even in Slovenia there are programs which enable students to make their own cartoons or videos on IBM PCs. Teachers can prepare lessons as videos, therefore avoiding several problems like temporary headaches or bad moods.

How come that in Slovenia teachers mainly use chalk and blackboard or slide projectors at most? Why don't they have a videoprojector or a PC with attached projector? First, there is not enough financial support. While videos are progressing in America and teachers there are making around $30,000 a year, even they usually don't have a phone on their desk (Soloway 1992). In that light, talking about BBSs and networks sounds a little unusual. Second, technological solutions are prone to errors. Machines break, the glass soon becomes scratched or dirty beyond recognition, small pins on the cables and connectors are bent out of shape. Indeed, even good alternatives like a large screen video monitor, faster and with better colors than projectors, are sensitive machines that easily get out of order.

Soloway (Soloway 1992) proposes that computers in schools meet the following basic needs:

- There should be enough readily available computers for
  - routine tasks on computers: wordprocessing, spreadsheeting, drawing, calculating, and telecommunications.
  - supporting didactic instruction. In every class, there should be a computer with a suitable projection mechanism, TV and overhead projector.
- Each course and each student population has its own special needs. Most of them can be fulfilled with general computers and special additional equipment such as scanners, TVs or music instruments.
- The computers should be capable of growing as the students, teachers and administrators mature together with new technologies. Computers last for 10 years and buying new ones because of inexperience means throwing money through the windows.

Anything has its positive and negative sides. Among negative sides of using computers, it is mainly health that matters. Rather surprisingly, since there is no obvious hazard like when driving cars or inhaling particles and gases when smoking. However, there is sitting, typing and watching a screen from a close distance. It all results in a physical stress accumulated over months or years of practice, or simply as a lack of normal physical activities which are essential in the growth of a normal human. Indeed, it is somehow sad to see each generation having more and more troubles with their backbones and eyes. Filters on screens and exercises at least once per hour do help, however, additional physical exercise is more than needed.

5 Programs For Mathematics

Around 20 years ago programs like Macsyma were regarded as artificial intelligence systems solving complex symbolic tasks in geometry and integration. They

were using heuristic methods. From then on, new systems were emerging on the market, each better and better. Now they are based on a special yet exact branch of mathematics.

Probably the best mathematical system available on the market and running on IBM PC (386 or more) is Mathematica (Wolfram 1991). It also runs on Windows, Macintosh, CONVEX, DEC VAX, HP 9000, IBM RISC System/6000, NeXT, Silicon Graphics, Sun workstations, SPARC, just to mention some of them. Mathematica was first released in 1988 and soon received several awards as one of the best software products. The company Wolfram Research, Inc., has now approximately 100 employees. Mathematica is a general software system for mathematical and other applications. It is currently used by around 100,000 researchers, engineers, and analysts, as well as students from high school to graduate school. The applications of Mathematica span all areas of science, technology and business where quantitative methods are used. The manual has around 1000 pages. It includes a full range of interactive numerical, graphical, and symbolic computation capabilities, all linked to the powerful built-in Mathematica language including sound generation and flexible external program communication. Mathematica has been published in dozens of books, and it has its own quarterly journal.

No doubt that Mathematica is the world leader in scientific, engineering and faculty surroundings. But for high schools and nontechnical faculties Derive (Soft Warehouse 1991) might be even more interesting. First, there is a great difference in size. Mathematica takes at least 8M while Derive is available on one single-sided single-density diskette with 360K. The manual for Derive is only one quarter of the size of the manual of Mathematica, indicating that Derive is smaller and more comprehensible. These are the reasons why Mathematica is being presented in postgraduate studies in Slovenia and Derive is the favourite for high schools. In recent years, Derive has been systematically introduced into mathematical courses in Slovenia through the Ministry of Education, and mathematical and computer communities in Slovenia. For example, special courses were given to high school teachers, and a Slovenian shorter version of the user manual was published (Lokar 1991). Derive offers many possibilities to solve different problems. For example, it takes around half a minute to input

$$\int_0^r \int_0^{\sqrt{r^2-x^2}} x\,y\,dy\,dx$$

and Derive computes it correctly to be $r^4/8$ in a fraction of a second. As for another example, it takes around 10 seconds to compute 1000! on all 2567 digits on IBM PC 386. Last but not least, one can plot functions on the screen as 2- or 3-dimensional figures.

6 Summary

Lessons to be learned are clear enough: give teachers good incomes and thorough education, make a good and stimulative educational system with strong stimulations for successful teaching and learning, quickly eliminate all disciplinary problems by a strict and consistent approach thus avoiding repression and confrontation, mix theory with many real-life examples throughout learning, use as many learning tools as possible, and combine the learning programmes with existing national industrial and infrastructure activities.

On the other hand, it is important to proportionally weigh practical exercises and theory. For example, there are courses in Slovenia which are nearly out of touch with reality while others hardly have any theory in them.

Computers are far the most useful educational tool. Although Slovenia is rather successful in introducing computers in schools, much more can be done with relatively small expenses. Hopefully, some of it will happen since Slovenia is on its way to restructure its schooling system once more, and this time at least basic directions are well known. Systematic introduction of computers into all levels of computing is one of those proper actions to continue with.

Acknowledgement: We would like to thank Aram Karalić for helpful remarks.

References

[1] Soloway E. (1991): *How The Nintendo Generation Learns*, Communications of the ACM, Vol. 34, No. 9.

[2] Goodman S.E. (1991): *Computing and the Resuscitation of Romania*, Communications of the ACM, Vol. 34, No. 9.

[3] Gams, M., Hribovšek B. (1992): *Trends in Computer Progress*, Informatica, Vol. 16, No. 1.

[4] Soloway E. (1992): *Buying Computers for Your School: A Guide for the Perplexed*, Communications of the ACM, Vol. 35, No. 7.

[5] Wolfram S. (1991): *Mathematica*, User Manual, Second Edition, Addison-Wesley.

[6] Soft Warehouse (1991): *Derive*, User Manual, Version 2, Soft Warehouse.

[7] Lokar M. (1991): *Derive, program for symbolic computing*, Raček (in Slovenian).

PROLOGUE TO THE SOUL OF A NEW SCIENCE

(A Review of a Book)

"All noble things are as difficult as they are rare"
Spinoza

There are books to which we return to, like the invisible places in our imagination, when doubts overshadow the reasons of functionality and the benefits of pragmatism. These books, as they should do, raise more questions then give answers. The issues they address are complex and difficult ,and possibly avoid any rigorous definitions. And yet, there is no single scientific endeavor that has not reached to these issues belonging to the very roots, reexamining the foundations and the existence of the entities that constitute the discipline. These are the moments when philosophical arguments become inevitable, an intervals of time in which the necessity of why supplants the contingency of how.

One of those books has been written by Professor Zeleznikar under the title "On the Way to Information". The readers, the ones that will persist through the book,will possibly be few. It requires a stamina and involvement, it demands a new way of thinking about things related to the science of informatics. A redefinition of the principles and ideas present for a long time in the realm of informatics is a premise for understanding the text not a consequence of it. The set of paradigms postulated will eventually become basis for the "structure of the informational revolution".

The definitions of mankind range from man as a rational animal, man as a social animal, to man as the tool-making animal. The last one, a homo faber concept of human existence, while interacting with others, is the predominant one and closely related to the perception of human nature. It has been conceived in the myth of Prometheus who as the bearer of the fire is considered to be the father of our tool-oriented civilization. The other aspect of this myth, to be the symbol of foreseeing consequences or homo sapiens concept has been much to often neglected.

The advent of computer, like no other tool in the history of man, has confirmed the tool-making orientation. Moreover, it appears that for the first time it has been made possible to have an artefact involved in performing activities usually termed mental. Thus the computer, as a transducer of information, either by itself or as part of another man-made devices was promoted into indispensable technological instrument of our civilization.

It is the nature of that civilization and its phenomena, from language to sociology, from biology to artificial intelligence, Zeleznikar laboriously dissects in the attempt to identify one common denominator: information. So, the insight into the quintessence of information reveals a

rudimentary and primitive notion, yet so complex that almost any imaginable system can be explicated and represent in terms of information. Anything can be entailed by information and in the same time everything can be reduced to it. Systems based and expressed through the information have minimal ontology postulated on a single entity known as information. Both, the criteria of ontological commitment and the standards of ontological admissibility, formulated by Quine, are genuinely met.

A particular focus is placed on the differences and similarities relating to the informational phenomena concerning humans and machines. By precluding man from pure communication, Zeleznikar rightfully acknowledges the plethora of notions (termed informational additions) which need explication in order to produce a devices such as the "information machine". But then by equating intelligence and information, the information machine becomes an intelligent machine. Implementing, at least some of its characteristics, will require a new philosophy, theory and technology of information. Moreover, it will insists on a massive parallel architecture, a field where the emphasis on the technical matter has left out the issue of meaning.

Similar argument is easily extended in the critique of standard artificial intelligence. No one can deny the progress in various areas of problem solving (both on an algorithmic and heuristic level) introduced by artificial intelligence research communities. Most of it has been a result of an unselfish amount of effort built into the final products as opposed to the more mundane and routine work put into ordinary software programming. The accent on the rationalist tradition should not be easily dismissed. It is another form of the tool making paradigm, and what Professor Dujmovic, in his dignified and warm review of the book, naturally named "the engineering approach". This kind of course is dictated by the opportunism of everyday life which is a series of problems best solved by the constructive properties of engineering. But, as stated earlier, sciences mature and become a valid philosophical discourse when they start asking why, instead of how. Many disciplines, touched and profoundly influenced by the information revolution have abandoned the reflexive, philosophical direction.

Formal systems insist on completeness or consequential closure. By doing so, they lose the opportunity of becoming ampliative. There is no change in the informational level in these systems. Actually, a closed system comes with a new syntax while maintaining the old semantics. The author claims that from a viewpoint of autopoesis there are not closed systems, nor if there are will be of any interest. The autopoetic characteristic of the information, i.e. its ability to self produce is the one distinguishing it from any other entity and is the necessary condition to equate notion of information with the concept of existence or being. The manifestation of being is creativity, a highly individual and antisocial act. And this idea leads to the God as information.

There is no substitute for a philosophical argument. As the author points out, the systematic research of information philosophy has been incidental and precluded from the integrative approach. The totality of the issues traversed makes this book on information an informational avalanche. The number of the concepts and notions related to information which are novel and original is overwhelming. On numerous occasions the reader is left to walk through a linguistic nightmare. The definitions, by the character of information, are recursive providing an additional difficulty to the process of understanding. Yet, the horizons opened and visions implied are worthwhile the exceptional effort on the part of the reader.

Finally, it might indeed be considered peculiar that all of the reviews referring to "On the Way to Information" included some personal note about the author. Knowing the personality of Zeleznikar it is clearly to be expected. He was my professor at the University of Ljubljana, and he still is. In the years bygone, he has become a fragment of the fabric one cherishes in life. And as Robert Frost ones wrote "two roads diverged in a wood, and I took the one less traveled by, and that has made all the difference". The difference for Dr. Zeleznikar and for all of us who have the privilege of knowing him.

The articles recently published confirm the author's continuing dedication to work on the formalization of the ideas presented in this text. A work that will be a prelude to the mind of new science. In the years to come, "the mind and the soul" will signfy a contribution to the philosophy and the ideology of homo informaticus.

Skopje, Doc.Dr.Oliver B. Popov
June 1992 University of Skopje

EDITORIAL BOARD, BOARD OF REFEREES, PUBLISHING COUNCIL

Informatica is a journal primarily covering the European computer science and informatics community; scientific and educational as well as technical, commercial and industrial. Its basic aim is to enhance communications between different European structures on the basis of equal rights and international refereeing. It publishes scientific papers accepted by at least two referees outside the author's country. In addition, it contains information about conferences, opinions, critical examinations of existing publications and news. Finally, major practical achievements and innovations in the computer and information industry are presented through commercial publications as well as through independent evaluations.

Editing and refereeing are distributed. Each editor can conduct the refereeing process by appointing two new referees or referees from the Board of Referees or Editorial Board. Referees should not be from the author's country. If new referees are appointed, their names will appear in the Refereeing Board. Each paper bears the name of the editor who appointed the referees. Each editor can propose new members for the Editorial Board or Board of Referees. Editors and referees inactive for a longer period can be automatically replaced. Changes in the Editorial Board and Board of Referees are confirmed by the Executive Editors.

The coordination necessary is made through the Executive Editors who examine the reviews, sort the accepted articles and maintain appropriate international distribution. The Executive Board is appointed by the Society Informatika. Informatica is partially supported by the Slovenian Ministry of Science and Technology.

Each author is guaranteed to receive the reviews of his article. When accepted, publication in Informatica is guaranteed in less than one year after the Executive Editors receive the corrected version of the article.

INFORMATION FOR CONTRIBUTORS

1 Submissions and Refereeing

Please submit three copies of the manuscript with good copies of the figures and photographs to one of the editors from the Editorial Board or to the Contact Person. At least two referees outside the author's country will examine it and they are invited to write as many remarks as possible directly on the manuscript, from typing errors to global philosophical disagreements. The chosen editor will send the author copies with remarks, and if accepted also to the Contact Person. The Executive Board will inform the author that the paper is accepted, meaning that it will be published in less than one year after receiving original figures on separate sheets and the text on an IBM PC DOS floppy disk or through e-mail – both in ASCII and the Informatica LaTeX format. Style and examples of papers can be obtained through e-mail from the Contact Person.

2 News, letters, opinions

Opinions, news, calls for conferences, calls for papers, etc. should be sent directly to the Contact Person.

```
\documentstyle[twoside,informat]{article}

\begin{document}

  \title{title of the article}
   \author{First Author \\ Address \\ E-mail \\
      AND \\
      Second Author \\ Address \\ E-mail}
  \keywords{article keywords}
  \edited{editor in charge}
  \received{date}
  \revised{date}
  \accepted{date}
  \abstract{abstract -- around 200 words}
  \maketitle

  \section{Introduction}
    Text of Introduction.

  \section{Subject}
    Text of the Subject.

  \begin{figure}
    An example of a figure over 1 column.
    \caption{Caption of the figure 1}
  \end{figure*}

  \begin{figure*}
    An example of a figure over 2 columns.
    \caption{Caption of the figure 2}
  \end{figure*}

  \begin{thebibliography}{99}
    \bibitem{} First Author: {\sl Title},
      Magazine, Vol. 1, No. 1.
  \end{thebibliography}

\end{document}
```

```
\def\editionHead{
  Informatica Vol. 16, No. 4, November 1992}
\immediate\write16{'Informatica'}
\newif\iftitle \titlefalse
\hoffset=11mm \voffset=8mm
\oddsidemargin=-11.4mm \evensidemargin=-11.4mm
\topmargin=-33mm
\headheight=17mm \headsep=10mm
\footheight=8.4mm \footskip=52.5mm
\textheight=240mm \textwidth=170mm
\columnsep=5mm \columnseprule=0pt
\twocolumn \sloppy \flushbottom
\parindent 1em
\leftmargini 2em \leftmargin\leftmargini
\leftmarginv .5em \leftmarginvi .5em
\def\labelitemi{\bf --} \def\labelitemii{--}
\def\labelitemiii{--}
\setcounter{secnumdepth}{3}
\def\maketitle{\twocolumn[%
  \vbox{\hsize=\textwidth\Large\bf\raggedright
  \@title}\vss\bigskip\bigskip \vbox{
  \hsize=\textwidth \@author}\bigskip\smallskip
  \vbox{\hsize=\textwidth {\bf Keywords:}
    \@keywords}
  \bigskip \hbox{{\bf Edited by:} \@edited}
  \smallskip
  \hbox{{\bf Received:}
    \hbox to 10em{ \@received\hss}
    {\bf Revised:}\hbox to 10em{ \@revised\hss}
    {\bf Accepted:}\hbox to 10em{ \@accepted\hss}}
  \bigskip \vbox{\hsize=\textwidth
    \parskip=\baselineskip \leftskip=3em
    \rightskip=3em \sl \@abstract}
  \bigskip\bigskip]\titletrue}
\def\maketitleauthor{\twocolumn[%
  \vbox{\hsize=\textwidth \Large\bf\raggedright
  \@title}\vss \bigskip\bigskip
  \vbox{\hsize=\textwidth \@author}
  \bigskip\bigskip] \gdef\@title{}\titletrue}
\def\makeonlytitle{\twocolumn[%
  \vbox{\hsize=\textwidth \Large\bf\raggedright
  \@title}\vss\bigskip\bigskip]
  \gdef\@title{}\titletrue}
\def\@title{} \def\@author{}
\def\@keywords{} \def\@edited{} \def\@abstract{}
\def\@received{} \def\@revised{} \def\@accepted{}
\def\keywords#1{\gdef\@keywords{#1}}
\def\edited#1{\gdef\@edited{#1}}
\def\received#1{\gdef\@received{#1}}
\def\revised#1{\gdef\@revised{#1}}
\def\accepted#1{\gdef\@accepted{#1}}
\long\def\abstract#1{\gdef\@abstract{#1}}
\def\section{\@startsection {section}
  {1}{\z@}{-3.5ex plus -1ex minus -.2ex}
  {2.3ex plus .2ex}{\Large\bf\raggedright}}
\def\subsection{\@startsection{subsection}
  {2}{\z@}{-3.25ex plus -1ex minus -.2ex}
  {1.5ex plus .2ex}{\large\bf\raggedright}}
\def\subsubsection{\@startsection{subsubsection}
  {3}{\z@}{-3.25ex plus -1ex minus -.2ex}
  {1.5ex plus .2ex}{\normalsize\bf\raggedright}}
\def\@evenhead{\hbox to 3em{\bf\thepage\hss}
  {\small\editionHead\hfil \iftitle\else
  \@title \fi} \global\titlefalse}
\def\@oddhead{{\small\iftitle\else \@title \fi
  \hfil\editionHead}\hbox to 3em{\hss\bf\thepage}
  \global\titlefalse}
\def\@evenfoot{\hfil} \def\@oddfoot{\hfil}
\endinput
```

Avtorsko stvarno kazalo časopisa Informatica, letnik 16 (1992)

Authors Subject Index of the Journal Informatica, Volume16 (1992)

Članki — Articles

Šilc, J., Sinhronizirana podatkovno pretokovna računalniška arhitektura, Informatica **16** (1992), št. 2, 59-72.

Šilc, J. in L. Gyergyek, Sinhronizirana podatkovno pretokovna računalniška arhitektura, Informatica **16** (1992), št. 3, 46-63.

Varga, M., Implementacija TEX-a na računalu Atari-ST, Informatica **16** (1992), No. 4, 48-52.

Zaveršek, P. in P. Kolbezen, O preslikavi HADFG na TAS, Informatica **16** (1992), št. 3, 29-35.

Železnikar, A.P., An Informational Approach of Being-there as Understanding I, Informatica **16** (1992), No. 1, 9-26.

Železnikar, A.P., An Informational Approach of Being-there as Understanding II, Informatica **16** (1992), No. 2, 29-58.

Železnikar, A.P., Basic Informational Axioms, Informatica **16** (1992), No. 3, 1-16.

Železnikar, A.P., An Informational Approach of Being-there as Understanding III, Informatica **16** (1992), No. 3, 64-75.

Železnikar, A.P., Into a New Perspective, Informatica **16** (1992), No. 4, 1.

Železnikar, A.P., An Introduction to Informational Machine, Informatica **16** (1992), No. 4, 8-29.

Novice in zanimivosti — News

Gams, M., Pojasnilo k članku »Trends in Computer Progress«, Informatica **16** (1992), št. 2, 76.

Lavrač, Nada, Computing NoE: Network of Excellence in Computational Logic, Informatica **16** (1992), No. 3, 78-80.

Železnikar, A.P., Strokovna srečanja, Informatica **16** (1992), št. 1, 85-86.

Železnikar, A.P., Fotonika namesto elektronike, Informatica **16** (1992), št. 2, 73-76.

Železnikar, A.P., Sistemske znanosti in kibernetika, Informatica **16** (1992), št. 3, 76.

Železnikar, A.P., Spreminjanje paradigem v programirni tehniki, Informatica **16** (1992), št. 3, 76-77.

Železnikar, A.P., Strokovno povezovanje Slovenije na področju informatike, Informatica **16** (1992), št. 3, 77.

Železnikar, A.P., Nekateri zanimivi regionalni časopisi, Informatica **16** (1992), št. 3, 77.

Informatica in 1993 (Editing and Instructions), Informatica **16** (1992), No. 4, 94-95.

Knjižne recenzije — Book Reviews

Indihar, S., »Ivan Meško: Optimizacija poslovanja s programi na disketi«, Informatica **16** (1992), št. 1, 83.

Popov, O.B., Prologue to the Soul of a New Science, Informatica **16** (1992), No. 4, 91-93.

Železnikar, A.P., »Hubert L. Dreyfus: Being-in-the-World«, Informatica **16** (1992), št. 1, 84-85.

Železnikar, A.P, »R.A. de Beaugrande in W.U. Dressler: Uvod v besediloslovje«, Informatica **16** (1992), št. 2, 77-78.

Strokovna društva — Professional Societies

Ustanovitev Društva za umetno inteligenco, Informatica **16** (1992), št. 2, 76-77.

Statut Slovenskega društva Informatika, Informatica **16** (1992), št. 3, 81-86.

VSE ZA UNIX ZA VSE

UNIX je eden najstarejših operacijskih sistemov (rojen 1969 kot projekt v Bell Lab) in danes brez dvoma najbolj razširjen večuporabniški operacijski sistem. Prvih 10 let se je "izpopolnjeval" po številnih univerzah, zadnjih 10 let pa je postal eden od uporabniških standardov odprtih računalniških arhitektur in ima največji trend rasti v prodaji v poslovnem in akademskem okolju. Vsi večji proizvajalci računalniške opreme (ki so do nedavnega imeli le lastne operacijske sistem) so bili prisiljeni sprejeti miselnost odprtih arhitektur in portirati UNIX na svoje sisteme (IBM - AIX, DEC - ULTIX, HP - HP-UX, BULL - DPX).

SCO UNIX

SCO UNIX je prvi med UNIX-i saj ima prodanih več kot 5 milijonov licenc na PC-jih s procesorji 386 in 486 ali multiprocesorji. Omogoča polno 32 bitno večuporabniško (licenca je za neomejeno število uporabnikov) in večopravilno računalniško okolje, ki je cenovno in performančno prepričljivo najugodnejša rešitev za večuporabniške sisteme ali za grafične postaje. Minimalna konfiguracija zahteva: PC 386SX, 4 Mb RAM in 40 Mb trdi disk.

Vsklajen je z vsemi potrebnimi standardi kot so ANSI, IEEE POSIX in X/Open. Omogoča visoko zaščito dostopa uporabnikov in ne pozna virusov.

Open Desktop je celotni grafični operacijski sistem z vgrajenimi X Window grafičnim vmesnikom, mrežo (TCP /IP in NFS), relacijsko podatkovno bazo in DOS-UNIX integracijo.

SCO MPX je nadgradnja SCO UNIX operacijskega sistema ali Open Desktop za multiprocesor z do 16 tesno povezanimi procesorji.

KOMUNIKACIJA

SCO TCP/IP je programska oprema za komunikacijo po Ethernet mreži. Omogoča povezavo številnih heterogenih sistemov, ki lahko komunicirajo preko TCP/IP (IBM mainframe, VAX, DOS, UNIX, Novell, ...). Omogoča prijavo na oddaljen sistem (remote-login), prenos datotek, oddaljeno tiskanje (remote-printing), prenos elektronske pošte (mail-transfer), medmrežno usmerjanje (internetwork-routing) in drugo. V kombinaciji z Xsight, to je SCO X Window vmesnikom, lahko dosegamo grafične mrežne aplikacije. TCP/IP omogočą tudi komunikacijo preko standardnih serijskih vrat in preko modema.

SCO NFS (Network File System) je nadgradnja za TCP /IP. SCO NFS omogoča strukturo posluževalec / strežnik. Uporabnik lahko transparentno dosega datoteke in aplikacije na oddaljenem sistemu. Vsebuje RPC (Remote Procedure Call) in XDR (eXternal Data Representation) knjižnici za razvoj distribuiranih aplikacij. Proces na enem računalniku lahko pokliče proces na drugem sistemu preko mreže.

UNIX-DOS integracija

UNIX DOS integracija je možna na več nivojih. DOS uporabnik se lahko prijavi na UNIX po serijski liniji preko terminalskega emulatorja. DOS aplikacije kot so programi v dBase, Clipper, Lotus, WordStar in druge lahko izvajamo pod UNIX-om preko treminalskega emulatorja SCO VP/ix. S paketom XVision lahkjo DOS uporabnik, ki ima Microsoft Windows, spremeni svoj PC v X terminal na UNIX-u. Uporabnik lahko v različnih oknih hkrati na zaslonu prikazuje UNIX in DOS aplikacije in izvaja cut&paste med X in DOS aplikacijami.

INFORMIX BAZA

Informix je s 500.000 licencami prva po številu prodanih licenc med bazami na UNIX-u. To je relacijska podatkovna baza, ki ima vso podporo: od ANSI standardnega SQL-a preko 4GL do CASE orodij in objektno orientirane tehnologije. Omogoča hranjenje BLOB-ov (Binary Logical OBject), kjer shranimo skanirane slike ali zvočni zapis enako preprosto kot navadni zapis fiksne dolžine v bazi. Nekaj zmogljivostnih podatkov pove marsikaj: posamezno polje znakovnega tipa ima lahko 32 kbytov, posamezno polje tipa text ali byte ima lahko 2 Gbyta, hkrati lahko odpre 32.000 datotek ali tabel, maksimalno število uporabnikov na en računalnik je 255.000 (omejitve so v hardveru), posamezna datoteka ali tabela ima lahko do 96 indeksov itd. Minimalne zahteve za Informix so 1Mb RAM in 5 Mb prostora na disku. Informix šteje med najbolj racionalne, zanesljive in hitre podatkovne baze te vrste na UNIX-u.

Informix uporablja pri delu podatkovni stroj (Database Engine). Preprostejši Informix-SE (Standard Engine) je namenjen manjšim in srednjim aplikacijam, Informix-OLTP (On-Line-Transation-Processing) pa je visokozmogljiv podatkovni stroj z direktnim dostopom do diska, avtomatskim zrcaljenjem, podporo za delo z multiprocesorjem in sprotnim monitoringom performans celotnega sistema.

Večje aplikacije lahko realiziramo tako, da na močan strežnik dodamo novi podatkovni stroj, ali pa uporabimo Informix mrežne produkte. Informix-Net omogoča realizacijo modela odjemalec/strežnik. Podatkovni stroj skrbi za delo z bazo na strežniku. Infromix orodja kot so SQL, 4GL, CASE, ESQL za C, FORTRAN, COBOL in druga pa se izvajajo na odjemalcu.

Informix-Star je namenjen za aplikacije z distribuirano bazo. Tu lahko delamo poizvedbo po več tabelah, ki so v različnih računalnikih. Informix-Star najde optimalni način iskanja elementov, ki ustrezajo kriterijem izbora in zagotavlja najhitrejši odgovor.

NAKUP in IZOBRAŽEVANJE

PAREX d.o.o, Inštitut za računalniški inženiring in svetovanje nudi akademskim ustanovam 30% popust za SCO UNIX Operating System in možnost obročnega plačila. Pri šolanju imajo naročniki časopisa Informatica 20% popust.

Saša Prešern

Informatica

Informatica *is published four times a year in Winter, Spring, Summer and Autumn by the Slovene Society Informatika, Vožarski pot 12, 61000 Ljubljana, Slovenia.*

Volume 16 Number 4 December 1992 ISSN 0350 – 5596

Informatica

A Journal of Computing and Informatics

CONTENTS