Volume 16 Number 3 August 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 3 August 1992
ISSN 0350 – 5596

Informatica

A Journal of Computing and Informatics

The Slovene Society INFORMATIKA
Ljubljana

Letnik 16 Številka 3 Avgust 1992 ISSN 0350 – 5596

Informatica

Časopis za računalništvo in informatiko

VSEBINA

BASIC INFORMATIONAL AXIOMS*

INFORMATICA 3/92

Keywords: axiomatization, circularity, externalism, functionalism, informational axioms, intelligent informational entity, internalism, metaphysics, parallelism, phenomenalism

Anton P. Železnikar
Volaričeva ulica 8, 61111 Ljubljana
Slovenia

This essay brings an approach to the possibilities of axiomatization in the realm of the informational. The axiomatization begins from the *informatio prima* which says that entities inform or, formally, $\alpha \models$. This primary postulate causes logical consequences which are comprehended as extensions of the primary axiom of informing of entities (informing as externalism, informedness as internalism, circularity as metaphysics, and informational openness as phenomenalism). A consequent axiomatic deduction (together with operational particularization) is shown for general, alternative, negative, alternative negative, functional, metaphysical, parallel, and intelligent informational cases. Informational axiomatization is a part of the informational logic in which the notion of information is extended to the phenomenal extremity. Axioms are expressed by the formalized symbolism which considers the concept of the informing/informedness (active/passive, subjective/objective, operand/operator) nature of informational entities [TIL].

Osnovni informacijski aksiomi

Ta spis prinaša nekatere možnosti aksiomatizacije v domeni informacijskega. Postopek aksiomatiziranja se začenja pri t.i. *informatio prima*, ki govori, da bivajoče informira, ali formalno $\alpha \models$. Ta prvi postulat povzroča logične posledice, ki jih razumemo kot razširitve prvotnega aksioma informiranja bivajočega (informiranje kot eksternalizem, informiranost kot internalizem, cirkularnost kot metafizika in informacijska odprtost kot fenomenalizem). Pokazana je dosledna aksiomatska dedukcija (skupaj z operacijsko partikularizacijo) za splošni, alternativni, negativni, alternativno negativni, funkcijski, metafizični, paralelni in inteligentni informacijski primer. Informacijska aksiomatizacija je le del informacijske logike, kjer je pojem informacije razširjen do skrajnih pojavnih mejá. Aksiomi so izraženi s formaliziranimi simboli, ki upoštevajo koncept informiranja/informiranosti (aktivne/pasivne, subjektivne/objektivne, opernadne/operatorske) narave informacijskih entitet [TIL].

0. Introduction

...das Mathematische will sich selbst, im Sinne seiner eigenen inneren Forderung, begründen; es will sich selbst ausdrücklich als Maßstab *allen* Denkens herausstellen und die daraus entspringenden Regeln aufstellen.

—*Martin Heidegger* [FND] 78

Nowadays, the background of informational reasoning seems to be more or less forgotten, fallen into oblivion, but also unrevealed and still subconscious. The broadened notion of information [Owi] was the initial step to this recognition in which formal principles and their consequences have been brought to the reader's consciousness.

To develop a broadened and complete theory of information and informational phenomenology,

an exhaustive axiomatic background of informing of entities is needed, on which the arising phenomenalism, scenery, processibility and, lastly, formalism can be built up. This step is necessary not only with the aim to broaden the scope of informational cognition, but also to prepare the basis for the future development of that what we call the informational machine. The reader must be aware that informational axiomatization, in opposition to the known mathematical one, is a continuum in which a new axiom or theorem can always arise between two already existing ones. Informational continuum appears as an infinitesimal realm of information and an informational theory is never developed to an end, staying open for further change, improvement, and broadening. In this essay we shall discuss these assertions among several others.

Axioms are formal principles which arise from the principal position and attitude of the logical mind. Informational axioms—among others, for instance, mathematical ones—are the most general and can absorb any thinkable axiomatic concept and include it appropriately into the observed informational system. This is a hypothetical assertion which can always be proven from one concrete case to another. The informational seems, at best, to be the most adequate principle of the universe. We shall try to give a principled explanation of this assertion at the end of the essay.

Our way to the realm of axiomatic scenery of the informational must begin with some very basic assertion, which could be called *informatio prima*. Afterwards, axioms, logical consequences, and theorems can arise in a logically spontaneous, however, systematized way. Let us open this realm of cognition in the next section of the essay.

1. Basic Axioms

Der Grundzug muß in jenem bestehen, was die Grundbewegung der Wissenschaft als solcher gleichursprünglich maßgebend durchherrscht: es ist der Arbeitsumgang mit den Dingen und der metaphysische Entwurf der Dingheit der Dinge. Wie sollen wir diesen Grundzug fassen?

—*Martin Heidegger* [FND] 52

As in any axiomatic system, some basic questions prevail: Where is the origin of the informational axiomatic system? Which is the first, most natural axiom, from which other axioms and consequences arise in a possible, logical, and systematic way?

As the so-called *informatio prima* of any logic the following could be asserted: *Entities inform.* If something (an entity figuring as the operand) is marked by α, we can say that α informs. This statement, consciously or unconsciously, primordially constitutes the first principle of any theory and can be expressed symbolically by

$$(1) \qquad \alpha \Rightarrow (\alpha \models)$$

This axiom is *informatio prima* and pertains to any entity, that is, operand, marked by α. The axiom is expressed in the implicative form, using informational operator of implication, $\Rightarrow$, which is read as »implies (that)« from the left to the right, and as »is implied by« from the right to the left of formula (1). In the same way, operator $\models$ is read »informs« in the one, and »is informed by« in the other direction. Thus, for formula (1) we have the following explanations: Entity α implies that it informs. Entity α implies that something (the empty place on the right of operator $\models$) is informed by it. Anytime and anywhere *we have* or there *is* an entity α, it informs $(\alpha \models)$.

One of the very essential comments to axiom (1) is that its formula is circular in respect to entity α (operand α occurs on the left and on the right side of operator $\Rightarrow$). The other comment is, that α as an informing entity, transmits (operator $\models$ on the right of operand α) its phenomenality into the informational universe.

The first axiom induces consequences which can still be understood as axiomatic. Namely, if we adopt the fact of this axiom, that »if α, then α informs«, then probably something—some other or the same entity—must and can be informed. *To inform* without *to be informed* would not have any particular sense in the realm of the informational which is the universe of informing (phenomenalizing) of information (phenomena). Thus, the second most fundamental axiom is coming to the surface, as

$$(2) \qquad \alpha \Rightarrow (\models \alpha)$$

We can show the basic way to this axiom from the first axiom and vice versa by the following asser-

tion: if α informs, then something (α, β, ...) would be informed. This fact yields

(3) $\quad (\alpha \models) \Rightarrow (\models \alpha)$

which is a consequence of axiomatic character. Further by a similar logical reasoning, which says, that if something is informed, there exists, sometime and somewhere, an informing entity α, thus,

(4) $\quad (\models \alpha) \Rightarrow (\alpha \models)$

The last two axiomatic consequences can be certainly interpreted more universally, by introduction of different entity markers α, β, γ, ... So,

(3') $\quad (\alpha \models) \Rightarrow (\models \alpha, \beta, \gamma, ...)$
(4') $\quad (\models \alpha) \Rightarrow (\alpha, \beta, \gamma, ... \models)$

These consequences explicate the openness of the informational universe in respect to the informing and informedness of an entity α, where it can inform and be informed by several entities (operands), simultaneously.

To remain strictly consequent, axiom (2) follows from axiom (1) in the way

(5) $\quad (\alpha \Rightarrow (\alpha \models)) \Rightarrow (\models \alpha)$

or logically, even more exactly,

(6) $\quad (\alpha \Rightarrow (\alpha \models)) \Rightarrow (\alpha \Rightarrow (\models \alpha))$

As already mentioned, in axiom (1) and its consequences the property of α's circularity is hidden. Formula (1) is evidently implicatively circular in respect to α, since operand α stands on the left and on the right side of operator $\Rightarrow$. Together with formula (1), also formulas (2) to (6) are perplexedly α-circular. This circularity is of especial interest in case of formulas (3) and (4), since they enable to conclude on α's particular informing which we call metaphysics of entity α. The initial state of α's metaphysics is expressed by formula

(7) $\quad \alpha \Rightarrow (\alpha \models \alpha)$

This formula reads: α implies that α informs α and is informed by α. Or, in general, an entity informs (impacts) itself and is informed (impacted) by itself. Formula (7) is a consequence of both formulas (3) and (4), that is,

(8) $\quad ((\alpha \models) \Rightarrow (\models \alpha)) \Rightarrow (\alpha \models \alpha)$

where the so-called metaphysics follows from the circularly tied informing and informedness of α and

(9) $\quad ((\models \alpha) \Rightarrow (\alpha \models)) \Rightarrow (\alpha \models \alpha)$

where metaphysics is a consequence of simultaneous, cyclically tied informedness and informing of α. Last two formulas conceptualize the circular or cyclic nature of an informational entity (operand, also informational formula) α.

The next essential consequence in the framework of *informatio prima* follows directly from both axioms (1) and (2). This consequence speaks that an entity informs and is informed simultaneously, that it is the informational system of an entity's informing and informedness. Thus, at the end of such concluding, there is,

(10) $\quad \alpha \models (\alpha \models; \models \alpha)$

This formula is not only the most general, but also the most open expression pertaining to entity α and its environment, to α's phenomenality in space and time. It makes evident that α will inform some, yet unidentified entities and itself in the form $\alpha \models$ and that it will be informed by some, yet unidentified entities and itself in the form $\models \alpha$. It is evident that both formulas $\alpha \models$ and $\models \alpha$ are completely open, having the operand empty place on the right and on the left side of operator $\models$, consequently.

The next question concerns the informational spontaneity of an informing entity. How does spontaneity concern the basic axioms?

Within $\alpha \models$, entity α informs in a spontaneous way. Spontaneity is the axiomatic property of operand α as well as operator $\models$. Informing of any informational entity α is spontaneous within the α's informational cycle which is constituted by informing, counter-informing, and embedding of information. Further, in $\alpha \models$, operator $\models$ is particularized in a concrete case and marks only different possibilities of α's operating potentiality.

Formulas $\alpha \models$ and $\models \alpha$ depict phenomenalities of one and the same entity α. They express

possibilities of α's informational impacting and impactedness, that is, to impact the world and to be impacted by the world. The impacting and impactedness of an entity is spontaneous, uncertain and, in general, indeterminable. The spontaneity of α's informing can be understood through its metaphysics, $\alpha \models \alpha$, which can be informationally decomposed by different observers in different (spontaneous) ways.

After this discussion we can summarize the four basic implicative axioms (*informatio prima*) pertaining to an informational entity α, as follows:

(11) (1°) $\alpha \Rightarrow (\alpha \models)$ [externalism of α]
(2°) $\alpha \Rightarrow (\models \alpha)$ [internalism of α]
(3°) $\alpha \Rightarrow (\alpha \models \alpha)$ [metaphysics of α]
(4°) $\alpha \Rightarrow (\alpha \models; \models \alpha)$ [phenomenalism of α]

Let us have the following dictionary of the primary four informational cases of informing:

(12) (1°) $\alpha \models$: external informing of α;
α's informational impacting;
α's informing (informingness)
(2°) $\models \alpha$: internal informing of α;
α's informational impactedness;
α's informedness
(3°) $\alpha \models \alpha$: metaphysical informing of α;
α's metaphysical circularity;
α's informational metaphysicalness
(4°) $\alpha \models; \models \alpha$: phenomenal informing and informedness of α;
α's informing and informedness phenomenality;
α's complex phenomenalism

There are different modes of general informing and it is possible to make a rough classification in the following sense:

general informing: operator $\models$;
alternative general informing: operator $=\!|$;
negative general informing: operator $\not\models$;
alternative negative general informing: $\not=\!|$;

parallel general informing: operator $\|\models$;
parallel alternative general informing: operator $=\!\|$;
parallel negative general informing: operator $\|\not\models$;
parallel alternative negative general informing: operator $\not=\!\|$;

cyclic general informing: operator $\vdash$;
cyclic alternative general informing: operator $\dashv$;
cyclic negative general informing: operator $\nvdash$;
cyclic alternative negative general informing: operator $\nvdash$;

parallel-cyclic general informing: operator $\|\vdash$;
parallel-cyclic alternative general informing: operator $\dashv\|$;
parallel-cyclic negative general informing: operator $\|\nvdash$;
parallel-cyclic alternative negative general informing: operator $\nvdash\|$

To all these cases the so-called subscribed informational operators can be added. A subscript (for instance, subs) means an operational particularization or universalization. Thus, such operators are:

$\models_{subs}$, $=\!|_{subs}$, $\not\models_{subs}$, $\not=\!|_{subs}$, $\|\models_{subs}$, $=\!\|_{subs}$, $\|\not\models_{subs}$, $\not=\!\|_{subs}$, $\vdash_{subs}$, $\dashv_{subs}$, $\nvdash_{subs}$, $\nvdash_{subs}$, $\|\vdash_{subs}$, $\dashv\|_{subs}$, $\|\nvdash_{subs}$, $\nvdash\|_{subs}$

and, certainly various operators of the type

$\llcorner_{subs}$, $\lrcorner_{subs}$, $\urcorner_{subs}$, $\ulcorner_{subs}$, $\wedge_{subs}$, $\vee_{subs}$, $\neg_{subs}$, $\Rightarrow_{subs}$, $\Leftarrow_{subs}$, $\Leftrightarrow_{subs}$, $\equiv_{subs}$, $\neq_{subs}$, $=_{subs}$, $\neq_{subs}$, $\forall_{subs}$, $\exists_{subs}$, $\cdot_{subs}$, $\in_{subs}$, $\notin_{subs}$, $\subset_{subs}$, $\not\subset_{subs}$, ω_{subs}, $\circ_{subs}$,

etc. with particular meanings.

We may not forget that all operators (arbitrarily universalized or particularized) used within informational logic are informational. Informational operators are attributes (properties, phenomenalities, happenings, predicates) belonging to the informing and informed entities (operands, formulas, formulas within formulas) and connecting them informationally. The direction (operational orientation or directionality) of operators depends on the verbal definition of a particular informational operator. For instance, in their

simple present tense, some verbal forms as »informs«, »include«, »operates«, etc. are oriented from the left to the right operand, while others, for instance, »observes«, »understand«, »conceives«, etc. are oriented from the right to the left operand. Thus, we can distinguish externally and internally oriented operators as determined by simple present tense of verbs. Operands are marked informational entities which are single Greek or Fraktur symbols or formulas composed of operands, operators, parentheses, commas, and semicolons. We shall show the syntactic structure of informational formulas later on.

2. Alternative Basic Axioms

Zum Wesen des Mathematischen als Entwurf gehört das Axiomatische, die Ansetzung von Grundsätzen, aus denen alles Weitere in einsichtiger Folge gründet. Wenn das Mathematische in Sinne einer mathesis universalis das gesamte Wissen begründen und gestalten soll, dann bedarf es der Aufstellung ausgezeichneter Axiome.

—*Martin Heidegger* [FND] 79

We are interested to have explicit alternative possibilities to the discussed basic axioms with the aim to be able expressing explicitly the possible alternatives in each particular case as well as on the most general level. Alternative axiomatic alternatives pertain also to the principle of informational spontaneity and to other possibilities of informing, for instance, to negative informing or non-informing, parallel, serial, cyclic, parallel-cyclic informing, etc. How it is possible to express alternative informing in cases of the most general and particular informing?

The alternative possibility of informing speaks in favor of the ability to express simultaneously the case of basic (in general $\models$-operational) and alternative ($\dashv\models$-operational) informing. We say that an entity informs in one (basic or general) or another (alternatively basic or general) way. How to mark the so-called other case?

In general, entity α informs and is informed in this and that way. It informs alternatively and such situation can be expressed symbolically

$$(1^{A}) \qquad \alpha \Rightarrow (\dashv\models \alpha)$$

The differences between formula (1) and (1^{A}) are the following:

(i) Operators $\models$ and $\dashv\models$ are counter-oriented, for instance, operator $\models$ is the left-right and operator $\dashv\models$ the right-left symbol. Through this we can read formula $\alpha \models$ as α *informs* and formula $\dashv\models \alpha$ as α *informs alternatively* (in respect to the case $\alpha \models$).

(ii) Simultaneously, operators $\models$ and $\dashv\models$ are dual to each other in the sense of informing and informedness of an entity. For instance, if $\alpha \models$ means α *informs*, then $\alpha \dashv\models$ would mean α *is informed* (alternatively). We can read operators (or formulas) strictly from the left to the right side. This possibility of reading is important for the distinguishing of verbal cases in which something impacts (is impacting) something informationally and something is impacted by something informationally. The following meanings of informational operator duality can be used:

operator type $\models$	*operator type* $\dashv\models$
informs	is_informed(_by)
is informing	is_being_informed(_by)
informed	has/have_been_ _informed(_by)
has/have_been_ _informing	has/have_been_ _informed (by)
was/were_informing	had_been_informed(_by)
will/shall_inform	will/shall_be_ _informed(_by)

etc. In short, the left side of operator $\models$ and the right side of operator $\dashv\models$ behave as informing entities irrespective of the tenses. On contrary, the right side of operator $\models$ and the left side of operator $\dashv\models$ behave as informed entities irrespective of the tenses.

(iii) In regard to formula (1^{A}), to be alternatively consequent, one could introduce the alternative basic implicative axiom in the form

$$(1^{A'}) \qquad (\dashv\models \alpha) \Leftarrow \alpha$$

This formula is read from the left to the right as: $\dashv\models \alpha$ *is implied alternatively by* α. From the right to the left, this formula can be read »normally«, that is, α *implies alternatively* $\dashv\models \alpha$. The complete possibility of meaning is the following: »α *informs*

alternatively« is implied alternatively by α. In the opposite direction one can read: α *implies alternatively that* α *informs alternatively.*

As we see, the alternative axioms can now be obtained automatically from the basic, non-alternative ones, that is,

(2^A) $\alpha \Rightarrow (\alpha =\!|)$
(3^A) $(=\!|\, \alpha) \Rightarrow (\alpha =\!|)$
(4^A) $(\alpha =\!|) \Rightarrow (=\!|\, \alpha)$
(5^A) $(\alpha \Rightarrow (=\!|\, \alpha)) \Rightarrow (\alpha =\!|)$
(6^A) $(\alpha \Rightarrow (=\!|\, \alpha)) \Rightarrow (\alpha \Rightarrow (\alpha =\!|))$
(7^A) $\alpha \Rightarrow (\alpha =\!|\, \alpha)$
(8^A) $((=\!|\, \alpha) \Rightarrow (\alpha =\!|)) \Rightarrow (\alpha =\!|\, \alpha)$
(9^A) $((\alpha =\!|) \Rightarrow (=\!|\, \alpha)) \Rightarrow (\alpha =\!|\, \alpha)$
(10^A) $\alpha \Rightarrow (=\!|\, \alpha;\ \alpha =\!|)$
(11^A) (1°) $\alpha \Rightarrow (=\!|\, \alpha)$ [alternative externalism of α]
(2°) $\alpha \Rightarrow (\alpha =\!|)$ [alternative internalism of α]
(3°) $\alpha \Rightarrow (\alpha =\!|\, \alpha)$ [alternative metaphysics of α]
(4°) $\alpha \Rightarrow (=\!|\, \alpha;\ \alpha =\!|)$ [alternative phenomenalism of α]
(12^A) (1°) $=\!|\, \alpha$: alternative external informing of α;
α's alternative informational impacting;
α's alternative informing (informingness)
(2°) $\alpha =\!|$: alternative internal informing of α;
α's alternative informational impactedness;
α's alternative informedness
(3°) $\alpha =\!|\, \alpha$: alternative metaphysical informing of α;
α's alternative metaphysical circularity;
α's alternative informational metaphysicalness
(4°) $=\!|\, \alpha;\ \alpha =\!|$: alternative phenomenal informing and informedness of α;
α's alternative informing and informedness phenomenality;
α's alternatively complex phenomenalism

In regard to the informational, the alternative informational means to have (possess) another possibility of informing to the already existing or identified one. Informational alternativeness is one of the basic principles of informational spontaneity, where in each particular case of informing an unforeseen, uncertain, indeterminable alternative possibility can arise and come into existence.

3. Axioms of Negative and Negative Alternative Informing

Negative (negating, negational) informing, called also non-informing, is an alternative mode of informing in comparison to basic and alternative basic informing of an entity. Negative informing is, for instance, from the observer's point of view, something which is missing informationally in the informing activity of an entity. On the other side, the informing entity can inform by itself the deficiency of its certain informational impacting and, in this way, cannot influence the informationally tied entity in some other way. Negative informing belongs to the oldest cultural forms of communication and language. Within this tradition a clear distinction between informing (or informedness) and non-informing (or non-informedness) of entities is possible and senseful.

In general case, where $\models$ marks the (positive) informing, the negative case is marked by $\not\models$ with the meaning »does or do not inform« or, on the other hand, »is or are not informed (by)«.

For any imagined entity it can be said that it does not inform in a certain (particular) way. This seem to be a natural situation of things which certainly cannot inform in all possible ways. The assertion »entity α does not inform« is expressed symbolically

(1^N) $\alpha \Rightarrow (\alpha \not\models)$

Axiom (1^N) is a particular case of *informatio prima*, is in the realm of non-informing of entity α. And certainly, if α does not inform in a certain way, then something cannot be informed in a certain way. According to axiom (2) for informing of an entity, we have the negative axiom

(2^N) $\alpha \Rightarrow (\not\models \alpha)$

This axiom reads: if α is an entity, then it is not informed in a way. As we recognize, the negative axioms can now be obtained automatically from the basic, non-alternative ones, that is,

(3^{N}) $(\alpha \nVdash) \Rightarrow (\nVdash \alpha)$;
(4^{N}) $(\nVdash \alpha) \Rightarrow (\alpha \nVdash)$;
(5^{N}) $(\alpha \Rightarrow (\alpha \nVdash)) \Rightarrow (\nVdash \alpha)$;
(6^{N}) $(\alpha \Rightarrow (\alpha \nVdash)) \Rightarrow (\alpha \Rightarrow (\nVdash \alpha))$;
(7^{N}) $\alpha \Rightarrow (\alpha \nVdash \alpha)$;
(8^{N}) $((\alpha \nVdash) \Rightarrow (\nVdash \alpha)) \Rightarrow (\alpha \nVdash \alpha)$;
(9^{N}) $((\nVdash \alpha) \Rightarrow (\alpha \nVdash)) \Rightarrow (\alpha \nVdash \alpha)$;
(10^{N}) $\alpha \Rightarrow (\alpha \nVdash; \nVdash \alpha)$;
(11^{N}) (1°) $\alpha \Rightarrow (\alpha \nVdash)$ [negative externalism of α]
(2°) $\alpha \Rightarrow (\nVdash \alpha)$ [negative internalism of α]
(3°) $\alpha \Rightarrow (\alpha \nVdash \alpha)$ [negative metaphysics of α]
(4°) $\alpha \Rightarrow (\alpha \nVdash; \nVdash \alpha)$ [negative phenomenalism of α]
(12^{N}) (1°) $\alpha \nVdash$: negative external informing of α;
α's negative informational impacting;
α's negative informing (informingness)
(2°) $\nVdash \alpha$: negative internal informing of α;
α's negative informational impactedness;
α's negative informedness
(3°) $\alpha \nVdash \alpha$: negative metaphysical informing of α;
α's negative metaphysical circularity;
α's negative informational metaphysicalness
(4°) $\alpha \nVdash; \nVdash \alpha$: negative phenomenal informing and informedness of α;
α's negative informing and informedness phenomenality;
α's negative complex phenomenalism

The negative informational in regard to the informational means the lack of certain possibility of informing in regard to already existing or identified one. Informational negativism is one of the basic principles of informational spontaneity, where in each particular case of informing a negative unforeseen, uncertain, indeterminable possibility can arise and come into existence.

The alternativeness of the negative informing can be expressed explicitly. It is important to have the possibility of alternative negative informing and informedness on a universal and particular level. Thus, we can repeat the entire system of formal expressiveness and its meaning for alternative negative informing as follows:

(1^{AN}) $\alpha \Rightarrow (\not\dashv \alpha)$
(2^{AN}) $\alpha \Rightarrow (\alpha \not\dashv)$
(3^{AN}) $(\not\dashv \alpha) \Rightarrow (\alpha \not\dashv)$;
(4^{AN}) $(\alpha \not\dashv) \Rightarrow (\not\dashv \alpha)$;
(5^{AN}) $(\alpha \Rightarrow (\not\dashv \alpha)) \Rightarrow (\alpha \not\dashv)$;
(6^{AN}) $(\alpha \Rightarrow (\not\dashv \alpha)) \Rightarrow (\alpha \Rightarrow (\alpha \not\dashv))$;
(7^{AN}) $\alpha \Rightarrow (\alpha \not\dashv \alpha)$;
(8^{AN}) $((\not\dashv \alpha) \Rightarrow (\alpha \not\dashv)) \Rightarrow (\alpha \not\dashv \alpha)$;
(9^{AN}) $((\alpha \not\dashv) \Rightarrow (\alpha \not\dashv)) \Rightarrow (\alpha \not\dashv \alpha)$;
(10^{AN}) $\alpha \Rightarrow (\not\dashv \alpha; \alpha \not\dashv)$;
(11^{AN}) (1°) $\alpha \Rightarrow (\not\dashv \alpha)$ [alternative negative externalism of α]
(2°) $\alpha \Rightarrow (\alpha \not\dashv)$ [alternative negative internalism of α]
(3°) $\alpha \Rightarrow (\alpha \not\dashv \alpha)$ [alternative negative metaphysics of α]
(4°) $\alpha \Rightarrow (\not\dashv \alpha; \alpha \not\dashv)$ [alternative negative phenomenalism of α]
(12^{AN}) (1°) $\not\dashv \alpha$: alternative negative external informing of α;
α's alternative negative informational impacting;
α's alternative negative informing (informingness)
(2°) $\nVdash \alpha$: alternative negative internal informing of α;
α's alternative negative informational impactedness;
α's alternative negative informedness
(3°) $\alpha \not\dashv \alpha$: alternative negative metaphysical informing of α;
α's alternative negative metaphysical circularity;
α's alternative negative informational metaphysicalness
(4°) $\not\dashv \alpha; \alpha \not\dashv$: alternative negative phenomenal informing and informedness of α;

α's alternative negative informing and informedness phenomenality;
α's alternative negative complex phenomenalism

4. Axioms of Basic and Basic Alternative Parallelism

Parallelism belongs to the most significant philosophical and technological approaches of modernism. Communication and cooperation among parallel processes constitute the kernel of mastering of complex computing machinery which architecture is built-up in a massively parallel structure and organization together with signal processing and networking. And this is the reason why we shall strictly repeat the scenario of axiomatization for the case of parallel and alternative parallel informing.

Informational parallelism means that entities inform freely, that is, spontaneously and circularly in parallel ways. An operand as an informationally open entity (phenomenalism) informs unforeseeably parallel and is in this way informed. It means that from the informational point of view, there are no limits for a concrete parallelism: it can be extended or suppressed in case of any informational entity (operand) in question. So, let us discuss this particular situation in some axiomatic details.

For any imagined entity it can be said that it informs in a parallel way, that its »signals« are spread (broadcasted) in space and time where they can impact entities physically, observationally, or in other ways. This seem to be a natural situation of things which inform in all possible ways to different places and sites. The assertion »entity α informs in parallel« is expressed symbolically

(1^P) $\quad \alpha \Rightarrow (\alpha \Vdash)$

Axiom (1^P) is the parallel case of *informatio prima*, is in the realm of the parallel informing of entity α. And certainly, if α informs in parallel ways, then entities are informed in parallel. According to axiom (2) for informedness of an entity, we have the parallel axiom

(2^P) $\quad \alpha \Rightarrow (\Vdash \alpha)$

This axiom reads: if α is an entity, then it is and can be informed by several (parallel) entities simultaneously. From this point on, the parallel axioms can be produced automatically from the basic ones, that is,

(3^P) $\quad (\alpha \Vdash) \Rightarrow (\Vdash \alpha)$;
(4^P) $\quad (\Vdash \alpha) \Rightarrow (\alpha \Vdash)$;
(5^P) $\quad (\alpha \Rightarrow (\alpha \Vdash)) \Rightarrow (\Vdash \alpha)$;
(6^P) $\quad (\alpha \Rightarrow (\alpha \Vdash)) \Rightarrow (\alpha \Rightarrow (\Vdash \alpha))$;
(7^P) $\quad \alpha \Rightarrow (\alpha \Vdash \alpha)$;
(8^P) $\quad ((\alpha \Vdash) \Rightarrow (\Vdash \alpha)) \Rightarrow (\alpha \Vdash \alpha)$;
(9^P) $\quad ((\Vdash \alpha) \Rightarrow (\alpha \Vdash)) \Rightarrow (\alpha \Vdash \alpha)$;
(10^P) $\quad \alpha \Rightarrow (\alpha \Vdash; \Vdash \alpha)$;
(11^P) (1°) $\alpha \Rightarrow (\alpha \Vdash)$ [parallel externalism of α]
(2°) $\alpha \Rightarrow (\Vdash \alpha)$ [parallel internalism of α]
(3°) $\alpha \Rightarrow (\alpha \Vdash \alpha)$ [parallel metaphysics of α]
(4°) $\alpha \Rightarrow (\alpha \Vdash; \Vdash \alpha)$ [parallel phenomenalism of α]
(12^P) (1°) $\alpha \Vdash$: parallel external informing of α;
α's parallel informational impacting;
α's parallel informing (informingness)
(2°) $\Vdash \alpha$: parallel internal informing of α;
α's parallel informational impactedness;
α's parallel informedness
(3°) $\alpha \Vdash \alpha$: parallel metaphysical informing of α;
α's parallel metaphysical circularity;
α's parallel informational metaphysicalness
(4°) $\alpha \Vdash; \Vdash \alpha$: parallel phenomenal informing and informedness of α;
α's parallel informing and informedness phenomenality;
α's parallel complex phenomenalism

The parallel informational in regard to the informational means the possibility of informing addi-

tionally in regard to already existing or identified modes, that is, simultaneously in many different modes. Informational parallelism is one of the basic principles of informational spontaneity, where in each particular case of informing a parallel unforeseen, uncertain, indeterminable possibility can arise and come into existence.

The alternativeness of the parallel informing can be expressed explicitly. It is important to have the possibility of alternative parallel informing and informedness on a universal and particular level. Thus, we can repeat the entire system of formal expressiveness and its meaning for alternative parallel informing as follows:

(1^{AP}) $\alpha \Rightarrow (=\!\!\| \alpha)$

(2^{AP}) $\alpha \Rightarrow (\alpha =\!\!\|)$

(3^{AP}) $(=\!\!\| \alpha) \Rightarrow (\alpha =\!\!\|)$;

(4^{AP}) $(\alpha =\!\!\|) \Rightarrow (=\!\!\| \alpha)$;

(5^{AP}) $(\alpha \Rightarrow (=\!\!\| \alpha)) \Rightarrow (\alpha =\!\!\|)$;

(6^{AP}) $(\alpha \Rightarrow (=\!\!\| \alpha)) \Rightarrow (\alpha \Rightarrow (\alpha =\!\!\|))$;

(7^{AP}) $\alpha \Rightarrow (\alpha =\!\!\| \alpha)$;

(8^{AP}) $((=\!\!\| \alpha) \Rightarrow (\alpha =\!\!\|)) \Rightarrow (\alpha =\!\!\| \alpha)$;

(9^{AP}) $((\alpha =\!\!\|) \Rightarrow (=\!\!\| \alpha)) \Rightarrow (\alpha =\!\!\| \alpha)$;

(10^{AP}) $\alpha \Rightarrow (=\!\!\| \alpha; \alpha =\!\!\|)$;

(11^{AP}) (1°) $\alpha \Rightarrow (=\!\!\| \alpha)$ [alternative parallel externalism of α]

(2°) $\alpha \Rightarrow (\alpha =\!\!\|)$ [alternative parallel internalism of α]

(3°) $\alpha \Rightarrow (\alpha =\!\!\| \alpha)$ [alternative parallel metaphysics of α]

(4°) $\alpha \Rightarrow (=\!\!\| \alpha; \alpha =\!\!\|)$ [alternative parallel phenomenalism of α]

(12^{AP}) (1°) $=\!\!\| \alpha$: alternative parallel external informing of α;
α's alternative parallel informational impacting;
α's alternative parallel informing (informingness)

(2°) $\|\!= \alpha$: alternative parallel internal informing of α;
α's alternative parallel informational impactedness;
α's alternative parallel informedness

(3°) $\alpha =\!\!\| \alpha$: alternative parallel metaphysical informing of α;
α's alternative parallel metaphysical circularity;
α's alternative parallel informational metaphysicalness

(4°) $=\!\!\| \alpha; \alpha =\!\!\|$: alternative parallel phenomenal informing and informedness of α;
α's alternative parallel informing and informedness phenomenality;
α's alternative parallel complex phenomenalism

The parallel informational phenomenality concerns the most sophisticated metaphysical and intelligent conceptuality (phenomenology) of the informational (mathematical, mathetical in the sense of the Greek *mathesis*). In an informational situation, parallel informational phenomena pertaining to this situation can come into existence. The general and particular operators of informational parallelism ($\|\!=$, $=\!\!\|$, $\|\!\neq$, $\neq\!\!\|$) express this faculty of parallel phenomenality in a positive or negative way. Therefore, the following parallel circularity can be considered as the most significant in the context of parallel informational system composition and decomposition:

(13^{P}) *)Parallel circular externalism:*
$(\alpha \models) \Rightarrow (\alpha \|\!=)$;
$(\alpha \|\!=) \Rightarrow (\alpha \models; \ldots ; \alpha \models)$;
Parallel circular internalism:
$(\models \alpha) \Rightarrow (\|\!= \alpha)$;
$(\|\!= \alpha) \Rightarrow (\models \alpha; \ldots ; \models \alpha)$;
Parallel circular metaphysics:
$(\alpha \models \alpha) \Rightarrow (\alpha \|\!= \alpha)$;
$(\alpha \|\!= \alpha) \Rightarrow (\alpha \models \alpha; \ldots ; \alpha \models \alpha)$;
Parallel circular phenomenalism:
$(\alpha \models; \models \alpha) \Rightarrow (\alpha \|\!= ; \|\!= \alpha)$;
$(\alpha \|\!= ; \|\!= \alpha) \Rightarrow (\alpha \models; \models \alpha; \ldots ; \alpha \models; \models \alpha)$

As one can understand, informational parallelism brings into the focus the so-called parallel circular mode of informational arising of entities, systems, formulas, etc.

5. Axioms of Informational Function

Wir bringen diesen gesuchten Grundcharakter der neuzeitlichen Wissenshaltung auf einen Titel, wenn wir sagen: Der neue Wissensanspruch ist der *mathematische.* Von Kant stammt der oft angeführte, aber noch wenig begriffene Satz: »Ich

behaupte aber, daß in jeder besonderen Naturlehre nur so viel *eigentliche* Wissenschaft angetroffen werden könne, als darin *Mathematik* anzutreffen ist.«

—*Martin Heidegger* [FND] 52

A function φ is an acting or producing informational entity in the realm of several, the function concerning entities. Therefore, it is senseful to speak about a functional scheme in which different entities can be involved when functional activity is coming into existence. It means that the functional scheme should retain sufficiently general perquisites enabling it to be embedded and connected into and with distinguished and essentially different informational entities, respectively.

In general, a function is never something without its roots from where it arose or is still arising. Several essential and general questions can come into the discourse before we list the axioms pertaining to informational function φ. Let us start with some basic questions.

(1) Which entity (informational operand) ψ *produces* φ*?*

(2) For which purposes, marked by π*, function* φ *is intended when produced by* ψ*?*

The last two questions can be united by the following formula:

(13) $\quad (\psi \models \pi) \models \varphi$

Since entity ψ with its purposes π is the main constructor of φ, process (13) is recursive in respect to ψ, that is, ψ-circular. This fact yields

(14) $\quad ((\psi \models \pi) \models \varphi) \models \psi$

The last formula can be particularized in an operand and operator way, that is,

(15) $\quad ((\psi_{\text{func\_prod}} \models_{\text{with}} (\pi_{\text{purpose}} \models_{\text{consider}} \sigma_{\text{sense}}, \sigma_{\text{significance}}, \iota_{\text{intention}})) \models_{\text{produce}} \varphi_{\text{function}}) \models_{\text{refer}} \psi_{\text{func\_prod}}$

The operand and operator entities of the last formula have the following particular meanings: $\psi_{\text{func\_prod}}$ is a functional producer; π_{purpose} marks purposes which consider (operator $\models_{\text{consider}}$) operand entities as σ_{sense}, $\sigma_{\text{significance}}$, and $\iota_{\text{intention}}$; these purposes are with (operator $\models_{\text{with}}$) entity $\psi_{\text{func\_prod}}$; now, the functional producer with its purposes produces (operator $\models_{\text{produce}}$) function $\varphi_{\text{function}}$ and the whole process is in reference to (operator $\models_{\text{refer}}$) the functional producer $\psi_{\text{func\_prod}}$. In this way, the function arises cyclically, influencing backwards its producer with certain purposes.

The next question is: *(3) From which distinguished informational domain* ξ *does function* φ *depend and how does it produce a result* ρ *for purposes* π*?*

Through this axiomatic question, function φ becomes together with domain ξ the producer of result ρ. In this point we introduce a shortened form of formal expression of »function φ of domain ξ«, that is

(16) $\quad \varphi(\xi) \Rightarrow (\varphi \models_{\text{of}} \xi)$

where $\models_{\text{of}}$ reads »inform(s)_of«, »inform(s)_on_the_basis_of«, »depends_on/upon«, etc. Since both entities φ and ξ can be composed formulas, the expression $\varphi(\xi)$ can also be understood in the functional sense. For instance, functional form $\alpha(\beta \models \gamma)$ means that α is a function of entity $\beta \models \gamma$; further, functional form $(\alpha \models \beta)(\gamma \models \delta)$ would mean that entity $\alpha \models \beta$ is a function of $\gamma \models \delta$. In fact, these functional expressions are shortcuts of both formulas $\alpha \models_{\text{of}} (\beta \models \gamma)$ and $(\alpha \models \beta) \models_{\text{of}} (\gamma \models \delta)$.

Question (3) can now be formally expanded in the following basic and general way:

(17) $\quad \varphi(\xi) \models ((\xi \models \varphi) \models ((\rho \models \pi) \models \psi))$

The particularized form of the last formula is

(18) $\quad \varphi_{\text{function}}(\xi_{\text{domain}}) \Rightarrow ((\xi_{\text{domain}} \models_{\text{influence}} \varphi_{\text{function}}) \models_{\text{produce}} ((\rho_{\text{result}} \models_{\text{for}} \pi_{\text{purpose}}) \models_{\text{of}} \psi_{\text{func\_prod}}))$

In the last formula, one can see how the result of the function and its domain within certain purposes become a function of the functional producer as well. It seems that the functional producer predetermines, in the realm of its own purposes, not only the function which it produces, but also to some informational extent the functional result itself. This comment may concern specifically any con-

struction of the so-called scientific functional dependencies, where the role of the functional producer is left to the scientist.

The next question pertains to the functional domain: *(4) To which informational domain α a functional domain ξ does belong and by which selecting entity β it is identified as a significant domain of the function φ?*

As one may expect, there is an entity β somewhere, clearly identified or unconscious, which performs the selection of an adequate informational functional domain ξ out of a broader informational realm α in the way to satisfy the producer purposes of producing of φ. The initial formula of selection is, for instance,

$$(19) \qquad \beta \models ((\xi \subset \alpha) \models \varphi(\xi))$$

The particularized form of this formula is, for instance,

$$(20) \qquad \beta_{selector} \models_{selects} ((\xi_{domain} \subset \alpha_{func\_domain}) \models_{for} (\varphi_{function}(\xi_{domain}))$$

The next question concerns the How of selector β: *(5) Is β an autonomous selector of domain ξ which selects ξ out of the realm α by its own strategy $\sigma_{strategy}$?*

The strategy of selector is by itself an informational function, that is, $\sigma_{strategy}(\beta_{selector})$. Thus the entity which selects is embedded in the selector, and formula (20) can be expressed more precisely as

$$(20') \qquad (\beta_{selector} \models_{by} \sigma_{strategy}) \models_{selects} ((\xi_{domain} \subset \alpha_{func\_domain}) \models_{for} (\varphi_{function}(\xi_{domain}))$$

Additional questions to the preceding ones may be the following:

(6) Is the domain (realm) α already clearly distinguished, identified, so it can serve as a background in producing functional domain ξ by selector β?

(7) How does function φ as entity $\varphi(\xi)$ produce a purposely oriented result ρ?

(8) To which informational domains does ρ belong, for instance, $\rho \subset \alpha, \beta, \xi, \pi, \varphi, \psi$?

(9) How can the functional scheme φ be expressed explicitly (in a predicating way, operationally), for instance,

$$\varphi \models_{of} \xi, \alpha, \beta, \pi, \psi;$$
$$\rho \models_{of} \varphi(\xi)$$

etc. and what is the form of the hierarchy, if specifically possible (linear, circularly interweaved) among entities?

(10) What does the functional scheme of a recursive character, that is, $\varphi_1(\varphi_2(...\varphi_n(\xi_n)...)$, mean?

Questions (1) to (10) spread the possibilities of the functional axiomatization enormously. In concrete cases, different axioms pertaining to specific situations and attitudes come to the surface.

At the end of this section, let us mention the following evident example of a recursive informational functional structure: the mind is a function of a concrete living brain; a thought is a function of the mind; a distinguished thought item is a function of the thought; the distinguished thought item, the thought, and the mind impact the arising of the brain, for instance, the appearance of new neurons and neuronal interconnections; thus, the brain is a function of its own phenomenality, that is, of thought items, thoughts, and the mind in its entirety; etc.

Further, the functional expression $\varphi(\xi)$ means that, in general, the functional structure and organization as an informational phenomenon, which is an acting entity, is sketched in a framing (intuitive, approximate) form and not determined yet too precisely. In case of a precise formulation of functional phenomenality, expression $\varphi(\xi)$ can imply various predicative, that is, operational forms of functional construction, for instance, $\varphi \models_{of} \xi$, $\xi \models_{with} \varphi$, $\varphi \models_{in} \xi$, etc. As we have seen, the functional phenomenality can take into consideration many parallel formulas which impact informationally the basic functional entity $\varphi(\xi)$.

6. Axioms of Basic Metaphysical Informing

Informational metaphysics concerns a thing in itself, that what an entity *is* by its own internal structure and organization, form and process, as it exists in itself. This view offers a more complex comprehension than that of *philosophia prima* or *supernatural imagery*.

The initial metaphysical situation of entity α is given by four metaphysical cases marking the basic, alternative basic, negative, and alternative negative informing of entity α in itself:

$$\alpha \models \alpha;\ \alpha \dashv\models \alpha;\ \alpha \not\models \alpha;\ \alpha \not\dashv\models \alpha$$

We shall limit our examination to case $\alpha \models \alpha$. In fact, this case will be decomposed axiomatically with the intention to give the basic starting points for further, that is, concrete decomposition (metaphysical projection).

Within metaphysics of operand α we are confronted with three basic phenomena: informing $\Im$ of entity α, marked by $\Im(\alpha)$; counter-informing $\complement$ of entity α, marked by $\complement(\alpha)$; and informational embedding $\mathfrak{E}$ of entity α, marked by $\mathfrak{E}(\alpha)$. These are three basic metaphysical functions of informing of α. Three groups of basic *parallel* metaphysical axioms can be posited:

(21) *Informing axioms:*
$(\alpha \models \alpha) \Rightarrow \Im(\alpha)$;
$\Im(\alpha) \Rightarrow ((\alpha \models \Im(\alpha)) \models \alpha)$;
$(\alpha \models \alpha) \Rightarrow ((\Im(\alpha) \models \alpha) \models \Im(\alpha))$;
Counter-informing axioms:
$(\alpha \models \alpha, \gamma \models \gamma) \Rightarrow \complement(\alpha)$;
$(\gamma \models \gamma) \subset (\alpha \models \alpha)$;
$\complement(\alpha) \subset \Im(\alpha)$;
$\complement(\alpha) \Rightarrow ((\alpha, \gamma \models \complement(\alpha)) \models \alpha, \gamma)$;
$(\alpha \models \alpha, \gamma \models \gamma) \Rightarrow$
$((\complement(\alpha) \models \alpha, \gamma) \models \complement(\alpha))$;
Embedding axioms:
$(\alpha \models \alpha, \gamma \models \gamma, \varepsilon \models \varepsilon) \Rightarrow \mathfrak{E}(\alpha)$;
$(\varepsilon \models \varepsilon) \subset (\alpha \models \alpha)$;
$\mathfrak{E}(\alpha) \subset \Im(\alpha)$;
$\mathfrak{E}(\alpha) \Rightarrow ((\alpha, \gamma, \varepsilon \models \mathfrak{E}(\alpha)) \models \alpha, \gamma, \varepsilon)$;
$(\alpha \models \alpha, \gamma \models \gamma, \varepsilon \models \varepsilon) \Rightarrow$
$((\mathfrak{E}(\alpha) \models \alpha, \gamma, \varepsilon) \models \mathfrak{E}(\alpha))$

In this system α marks the basic entity (operand, thing), γ is used as the marker of α's counter-informational component, and ε as the marker of α's embedding component, by which the arisen component γ and other information is connected to α. The last parallel system of axioms pertaining to metaphysics $\alpha \models \alpha$ results systematically into

(22) *Metaphysical informing:*
$(\alpha \models \alpha) \Rightarrow \Im(\alpha)$;
$(\alpha \models \alpha, \gamma \models \gamma) \Rightarrow \complement(\alpha)$;
$(\alpha \models \alpha, \gamma \models \gamma, \varepsilon \models \varepsilon) \Rightarrow \mathfrak{E}(\alpha)$;
Metaphysical hierarchy:
$(\gamma \models \gamma, \varepsilon \models \varepsilon) \subset (\alpha \models \alpha)$;
$\complement(\alpha), \mathfrak{E}(\alpha) \subset \Im(\alpha)$;
Decomposition of α, γ, and ε:
$\Im(\alpha) \Rightarrow ((\alpha \models \Im(\alpha)) \models \alpha)$;
$\complement(\alpha) \Rightarrow ((\alpha, \models \complement(\alpha)) \models \alpha, \gamma)$;
$\mathfrak{E}(\alpha) \Rightarrow ((\alpha, \gamma, \varepsilon \models \mathfrak{E}(\alpha)) \models \alpha, \gamma, \varepsilon)$;
Decomposition of $\Im(\alpha)$, $\complement(\alpha)$, and $\mathfrak{E}(\alpha)$:
$(\alpha \models \alpha) \Rightarrow ((\Im(\alpha) \models \alpha) \models \Im(\alpha))$;
$(\alpha \models \alpha, \gamma \models \gamma) \Rightarrow$
$((\complement(\alpha) \models \alpha, \gamma) \models \complement(\alpha))$;
$(\alpha \models \alpha, \gamma \models \gamma, \varepsilon \models \varepsilon) \Rightarrow$
$((\mathfrak{E}(\alpha) \models \alpha, \gamma, \varepsilon) \models \mathfrak{E}(\alpha))$

To the parallel metaphysical axioms, the cyclic or serial axioms can be introduced in which the occurring components α, $\Im(\alpha)$, γ, $\complement(\alpha)$, ε, and $\mathfrak{E}(\alpha)$ are positioned into middle-long and long metaphysical cycles. One of the axioms pertaining to the long metaphysical cycle is, for instance,

(23) $(\alpha \models \alpha) \Rightarrow$
$((((((\alpha \models \Im(\alpha)) \models \gamma) \models \complement(\alpha)) \models \varepsilon)$
$\models \mathfrak{E}(\alpha)) \models \alpha)$

Other long axioms can be obtained by the permuting of operand components in the last axiom.

6. Axioms of Informational Intelligence

Wo der Wurf des mathematischen Entwurfs gewagt wird, stellt sich der Werfer dieses Wurfes auf einen Boden, der allererst im Entwurf emtworfen wird. Im mathematischen Entwurf liegt nicht nur eine Befreiung, sondern zugleich eine neue Erfahrung und Gestaltung der Freiheit selbst, d.h. der selbstübernommenen Bindung. Im mathematischen Entwurf vollzieht sich die Bindung an die in ihm selbst geforderten Grundsätze.

—*Martin Heidegger* [FND] 75

Entities inform according to axiom (1) which represents *informatio prima* from which axiomatic consequences can follow in a mathematically projective way. How can entities inform intelligently and what are the perquisites of intelligent type of informing? Intelligence seems to be merely a pred-

icate (property, faculty) belonging to an informing entity. Thus, the way to intelligent informing of entities must proceed from the axiom (1) which has to be particularized for the case of intelligent informing.

An informational entity can have such or another intelligent position or attitude in the general and particular sense. The majority of readers would agree that an intelligent entity (IE) informs some of the facts, concepts, or fiction in the following six initial points:

(1) IE informs (expresses, describes) something (an entity) in an evaluating, observing, or understanding way;

(2) IE produces the so-called meaning, sense, spirit, and/or significance of something as information;

(3) IE forecasts, analyses, and synthesizes information concerning something under dynamic, unforeseeable, uncertain, and indeterminate circumstances;

(4) IE brings unknown, unconscious, invisible, impossible information concerning something to the surface;

(5) IE is spontaneously and circularly oriented, intentional, and goal-directed when it observes itself and evaluates various possibilities for its own development;

(6) IE behaves as a living entity in the world where it plays the struggle for its own benefit, delight, survival, etc.

Following the axiomatic route demonstrated for the general informing of entities (operator $\models$) in formulas (1) to (12), for an intelligently informing entity ι the following particular and significant axioms can be considered:

(1^{I}) $\iota \Rightarrow (\iota \models_{intelligent})$;

(2^{I}) $\iota \Rightarrow (\models_{intelligent} \iota)$;

(3^{I}) $(\iota \models_{intelligent}) \Rightarrow (\models_{intelligent} \iota)$;

(4^{I}) $(\models_{intelligent} \iota) \Rightarrow (\iota \models_{intelligent})$;

(7^{I}) $\iota \Rightarrow (\iota \models_{intelligent} \iota)$;

(10^{I}) $\iota \Rightarrow (\iota \models_{intelligent}; \models_{intelligent} \iota)$;

(11^{I}) (1°) $\iota \Rightarrow (\iota \models_{intelligent})$ [intelligent externalism of ι]

(2°) $\iota \Rightarrow (\models_{intelligent} \iota)$ [intelligent internalism of ι]

(3°) $\iota \Rightarrow (\iota \models_{intelligent} \iota)$ [intelligent metaphysics of ι]

(4°) $\iota \Rightarrow (\iota \models_{intelligent}; \models_{intelligent} \iota)$ [intelligent phenomenalism of ι]

Let us have the following dictionary of the primary four informational cases of intelligent informing:

(12^{I}) (1°) $\iota \models_{intelligent}$: intelligent external intelligent informing of ι;
ι's intelligent informational impacting;
ι's intelligent informing (informingness)

(2°) $\models_{intelligent} \iota$: intelligent internal informing of ι;
ι's intelligent informational impactedness;
ι's intelligent informedness

(3°) $\iota \models_{intelligent} \iota$: intelligent metaphysical informing of ι;
ι's intelligent metaphysical circularity;
ι's intelligent informational metaphysicalness

(4°) $\iota \models_{intelligent}; \models_{intelligent} \iota$: intelligent phenomenal informing and informedness of ι;
ι's intelligent informing and informedness phenomenality;
ι's intelligent complex phenomenalism

There are different modes of intelligent general informing and it is possible to make a rough classification in the following sense:

intelligent general informing: operator $\models_{intelligent}$;
intelligent alternative general informing: operator $=\!|_{intelligent}$;
intelligent negative general informing: operator $\not\models_{intelligent}$;
intelligent alternative negative general informing: operator $\neq\!|_{intelligent}$;
intelligent parallel general informing: operator $\|\models_{intelligent}$;
intelligent parallel alternative general informing: operator $=\!\|_{intelligent}$;
intelligent parallel negative general informing: operator $\|\not\models_{intelligent}$;

intelligent parallel alternative negative general informing: operator $\not\nparallel_{\text{intelligent}}$;
intelligent cyclic general informing: operator $\vdash_{\text{intelligent}}$;
intelligent cyclic alternative general informing: operator $\dashv_{\text{intelligent}}$;
intelligent cyclic negative general informing: operator $\nvdash_{\text{intelligent}}$;
intelligent cyclic alternative negative general informing: operator $\not\dashv_{\text{intelligent}}$;
intelligent parallel-cyclic general informing: operator $\Vdash_{\text{intelligent}}$;
intelligent parallel-cyclic alternative general informing: operator $\dashv\!\!\!\Vdash_{\text{intelligent}}$;
intelligent parallel-cyclic negative general informing: operator $\nVdash_{\text{intelligent}}$;
intelligent parallel-cyclic alternative negative general informing: operator $\not\dashv\!\!\!\Vdash_{\text{intelligent}}$

We see how intelligent informing of an entity ι is nothing else than any other particular informing of this entity. Intelligence is merely a specific perquisite of the entity which produces »intelligent« information concerning something which this entity evaluates, observes, or understands. Thus, to the basic axiomatic system of an intelligently informing entity ι the parallel system can be added, considering the previous six initial points (1) to (6) of intelligent informing. The markers and to them corresponding meanings of the Greek and Fraktur operands and operators are the following:

β_{benefit} benefit
$\beta_{\text{benefit}}(\sigma)$ something's own benefit; benefit of something
γ_{circumst} circumstance(s)
δ_{delight} delight
$\delta_{\text{delight}}(\sigma)$ something's own delight; delight of something
$\delta_{\text{development}}$ development
δ_{dynamic} dynamics
$\delta_{\text{dynamic}}(\gamma_{\text{circumst}})$ dynamic circumstances
ι_{indeterm} indeterminacy
$\iota_{\text{indeterm}}(\gamma_{\text{circumst}})$ indeterminate circumstances
ι intelligently informing entity
$\iota(\sigma)$ entity intelligently concerning something
$\iota_{\text{impossible}}$ impossible, the
$\iota_{\text{impossible}}(\sigma)$ impossible concerning something, the
$\iota_{\text{invisible}}$ invisible, the
$\iota_{\text{invisible}}(\sigma)$ invisible concerning something, the
λ_{live} liveliness
$\lambda_{\text{live}}(\varepsilon_{\text{entity}})$ living entity
μ_{meaning} meaning
$\mu_{\text{meaning}}(\sigma)$ meaning of something
$\pi_{\text{possibility}}$ possibility/possibilities
$\pi_{\text{possibility}}(\iota(\delta_{\text{development}}))$ possibility of intelligent entity development
σ something (an arbitrary entity to be »understood« by ι)
σ_{sense} sense
$\sigma_{\text{sense}}(\sigma)$ sense of something
$\sigma_{\text{significance}}$ significance
$\sigma_{\text{significance}}(\sigma)$ significance of something
σ_{spirit} spirit
$\sigma_{\text{spirit}}(\sigma)$ spirit of something
σ_{struggle} struggle
$\sigma_{\text{struggle}}(\beta_{\text{benefit}}(\sigma), \delta_{\text{delight}}(\sigma), \sigma_{\text{survival}}(\sigma), \ldots)$ struggle of something's own benefit, delight, survival, etc.
σ_{survival} survival
$\sigma_{\text{survival}}(\sigma)$ something's own survival; survival of something
$\upsilon_{\text{uncertain}}$ uncertainty
$\upsilon_{\text{uncertain}}(\gamma_{\text{circumst}})$ uncertain circumstances
$\upsilon_{\text{unconscious}}$ unconscious, the
$\upsilon_{\text{unconscious}}(\sigma)$ unconscious concerning something, the
$\upsilon_{\text{unforeseeable}}$ unforeseeable, the
$\upsilon_{\text{unforeseeable}}(\gamma_{\text{circumst}})$ unforeseeable circumstances
$\upsilon_{\text{unknown}}$ unknown, the
$\upsilon_{\text{unknown}}(\sigma)$ unknown concerning something, the
$\mathfrak{W}_{\text{world}}$ world

$\models_{analyze}$ analyze(s)
$\models_{behave}$ behave(s)
$\models_{circular}$ is/are_circular
$\models_{evaluate}$ evaluate(s)
$\models_{forecast}$ forecast(s)
$\models_{generate}$ generate(s); bring(s)_to_the_surface
$\models_{goal\_directed}$ is/are_goal-directed
$\models_{in}$ is/are_in; inform(s)_in
$\models_{intentional}$ is/are_intentional
$\models_{observe}$ observe(s)
$\models_{play}$ play(s)
$\models_{produce}$ produce(s)
$\models_{spontaneous}$ is/are_spontaneous
$\models_{synthesize}$ synthesize(s)
$\models_{under}$ is/are_under; inform(s)_under
$\models_{understand}$ understand(s)

The six additional paragraphs pertaining to IE can now be formalized by the corresponding informational formulas as parts of the informational system of IE ι. For the first sentence and its formula we have:

($13^{I}.1$) *IE informs (expresses, describes) something (an entity) in an evaluating, observing, or understanding way;*
$(\sigma \models (\iota \models_{intelligent} \iota)) \Rightarrow$
$(\iota \models_{evaluate} \sigma;\ \iota \models_{observe} \sigma;$
$\iota \models_{understand} \sigma)$

In this formula, IE ι, that is, its metaphysics $\iota \models_{intelligent} \iota$, is the intelligent observer of σ, so, it evaluates, observes, and understands σ. The second sentence and its formula is:

($13^{I}.2$) *IE produces the so-called meaning, sense, spirit, and/or significance of something as information;*
$\iota \models_{produce}$
$(\mu_{meaning}(\sigma), \sigma_{sense}(\sigma), \sigma_{spirit}(\sigma),$
$\sigma_{significance}(\sigma))$

The third sentence together with its formula is

($13^{I}.3$) *IE forecasts, analyses, and synthesizes information concerning something under dynamic, unforeseeable, uncertain, and indeterminate circumstances;*
$(\iota \models_{forecast} \iota(\sigma);\ \iota \models_{analyze} \iota(\sigma);$
$\iota \models_{synthesize} \iota(\sigma)) \models_{under}$
$(\delta_{dynamic}(\Upsilon_{circumst});$
$\upsilon_{unforeseeable}(\Upsilon_{circumst});$
$\upsilon_{uncertain}(\Upsilon_{circumst});$
$\iota_{indeterminate}(\Upsilon_{circumst}))$

The fourth sentence and formula are:

($13^{I}.4$) *IE brings unknown, unconscious, invisible, impossible information concerning something to the surface;*
$\iota \models_{generate}$
$(\upsilon_{unknown}(\sigma);\ \upsilon_{unconscious}(\sigma);$
$\iota_{invisible}(\sigma);\ \iota_{impossible}(\sigma))$

The fifth sentence and its formula are

($13^{I}.5$) *IE is spontaneously and circularly oriented, intentional, and goal-directed when it observes itself and evaluates various possibilities for its own development;*
$((\iota \models_{spontaneous};\ \iota \models_{circular};$
$\iota \models_{intentional};\ \iota \models_{goal\text{-}directed})$
$\models_{observe} \iota) \models_{evaluate}$
$\pi_{possibility}(\iota(\delta_{development}))$

The sixth sentence and its formula are

($13^{I}.6$) *IE behaves as a living entity in the world where it plays the struggle for its own benefit, delight, survival, etc.;*
$((\iota \models_{behave} \lambda_{live}(\varepsilon_{entity}))$
$\models_{in} \mathfrak{W}_{world}) \models_{play}$
$\sigma_{struggle}(\beta_{benefit}(\sigma), \delta_{delight}(\sigma),$
$\sigma_{survival}(\sigma), \ldots)$

7. The Syntax of Informational Formulas

In this essay we have used various informational formulas from the beginning, in the most basic axioms. In parallel, we developed a foundation of the so-called operand-operator (O-O) theory which could be understood also outside the so-called informational realm as a mathematical theory. We

could simply speak about a self-sufficient operand-operator theory which does not concern only the so-called informational theory. This would mean to ignore the semantics pertaining to the informational phenomenality and introduce operand-operator objects as the only ones into the O-O theory. However, such theory would be nothing else than a particular case of an informational theory. The syntax delivering syntactically correct informational formulas is simple and straightforward. If α represents any possible operand entity, the formation rules of well-formed informational formulas φ are the following:

(24)
(1°) $\varphi \Rightarrow \alpha$;
(2°) $\varphi \Rightarrow (\alpha',' \varphi)$;
(3°) $\varphi \Rightarrow (\alpha';' \varphi)$;
(4°) $\varphi \Rightarrow '('\varphi')'$;
(5°) $\varphi \Rightarrow (\varphi \models)$;
(6°) $\varphi \Rightarrow (\models \varphi)$;
(7°) $\varphi \Rightarrow (\varphi \models \varphi)$;
(8°) $\varphi \Rightarrow \alpha(\varphi)$;
(9°) $\alpha \Rightarrow (\alpha|\beta|\gamma| \ldots \omega|\mathfrak{A}|\mathfrak{B}|\mathfrak{C}| \ldots \mathfrak{Z})$;
(10°) $\alpha \Rightarrow \alpha_{subs}$;
(11°) $\models \Rightarrow (\models|\dashv|\not\models|\not\dashv|\|\models|\dashv|\|\not\models|\not\dashv|\| \ldots$
$\llcorner|\lrcorner|\neg|\Gamma|\wedge|\vee|\neg|'\Rightarrow'|\Leftarrow|\Leftrightarrow|\equiv|$
$\not\equiv|=|\neq|\forall|\exists|\,.\,|\in|\notin|\subset|\not\subset|\circ|$
$\Uparrow|\Downarrow|\uparrow|'|'| \ldots)$
(12°) $\models \Rightarrow \models_{subs}$;
(13°) $\models \Rightarrow \models_{comp}$;
(14°) $\models_{comp} \Rightarrow (\models\circ\models|'('\models\circ\models')'|$

In the last formula system we have marked symbols ',', ';', '(', ')', '$\Rightarrow$', and '|' to avoid the ambiguity with the equally marked symbols appearing in syntactic system (xx). In formula (11°) operator entities appear as informational entities. Entity α is a simple operand, that is, operand marker in the sense of formulas (9°) and (10°).

Conclusion

The informational seems to be the most adequate principle of the universe. Things, entities, phenomena, information, minds, machines, and the entirety of the universe inform and are informed, that is, impact and are impacted phenomenally, perform as informational entities, as formations concerning formations; and, that is, speaking roughly, *information*. In the essay, we have unfolded this situation and attitude through the conceptualism of informational externalism, internalism, metaphysics, and phenomenalism which all can be accepted in a general and in every particular case. Thus, any other axiomatic approach can be derived from the informational as a specific axiomatic particularization. On the other hand, an informational (verbal and symbolic) language [TIL], that is, the new formalism, came into the foreground and made the progressing into the domain of the informational (intelligent) machine possible.

Various forms of the verb *to inform* obtained the power of the verb *to be* and the role of the most primordial (simple and composed) verbal forms. Thus, any operator connection between informational operands could be expressed dynamically in a general or particular way, widening the scope of the verbal phrase which includes informing to arbitrary other words and word groups with particular, composed meanings. The informational approached to that what we can call the understanding way of informing, living the atomized, strictly linguistically structured and organized conceptualism in the background, unconscious, however not excluded from the informational realm. In any case, the verb *to inform* became the central happening between entities and enabled to broaden the informational semantic scope to any dynamically understood relation, operation, and phenomenality, occurring between informational entities (operands).

On this way, the informational axiomatization can encounter the most problematic, semantically complex, intelligent, and real situations, offering e new, flexible, and promising apparatus in the development of nowadays and future theories, systems, and machines.

References

[FND] Heidegger, M., Die Frage nach dem Ding, Max Niemeyer Verlag, Tübingen (1987).

[Owi] Železnikar, A.P., On the Way to Information, The Slovene Society Informatika, Ljubljana (1990).

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

INTEGRITY IN THE RELATIONAL DATA MODEL

INFORMATICA 3/92

Keywords: relational model, entity integrity, referential integrity, logic approach, consistency

Mario Radovan
Fakultet ekonomije i turizma, Pula
Inštitut »Jožef Stefan«, Ljubljana

The paper shows that the general integrity rules are simply "corollaries" of the (traditional) definitions of primary and foreign keys. Further, it is shown why (and how) the database-specific integrity rules should be treated as expressions of that knowledge which is contained, built-in in our "language, atitudes and measures". Such a conception of integrity allow us to reduce the (rather informal) concept of "correctness" of databeses to the request for its consistency (as a theory).

INTEGRITETA V RELACIJSKEM MODELU: Članek dokazuje, da so splošna pravila integritete preprosto korolarji tradicionalnih definicij primarnega in zunanjega ključa. Dokaže tudi zakaj (in kako) bi pravila integritete specifična za dano bazo, morala biti obravnavana kot izrazi (zapisi) tistih znanj, ki so vsebovana (vgrajena) v našem "jeziku, stališčih (odnosih) in merilih". Tako pojmanje integritete nam omogoča da (precej neformalen) pojem "pravilnosti" baze podatkov zvedemo na zahtevo po njeni konsistentnosti (kot teorije).

1. Introduction

"We begin with a little philosophy", says Date, <Dat 90, p. 275>, at the beginning of the chapter on Relatinal Integrity Rules. In this article we begin in the same way; indeed, we also continue in this way because it seems that the basic concepts concerning integrity in general are not yet defined suitably nor clearly enough.

Our discusion is concerned with the approach and definitions given in <Dat 90> because Date is not only one of the active contibutors to the field od databases,

but also one of the most influental writer on the topic. We are not speaking so much in terms of the "right" or "wrong" way of doing things as in terms of more or less suitable ways of conceiving and defining basic concepts in data modeling in general, and especialy of integrity constraints (rules).

2. The Logic Approach

A set of seminal ideas on the "logic approach" to data modeling was collected in <Gal 78>; among subsequent contributions, it seems that the most influental on the topic was <Rei 84>. An analysis and a proposal for (re)defining inference and integrity for knowledge bases, defined in terms of clausal logic and SLDNF-resolution, was given in <Rad 87>.

Generally speaking, the conceptual foundation of any "informal knowledge" always involves "some logic". Such a logic consists (at least) of a language (defined on the level of syntax (form) and semantics (interpretation, meaning)), and a (set of) inference rule(s) which allows us to infer consequences which follow from the given knowledge. Therefore, a "conceptual foundation" of any "knowledge" bring us to a "system" in whose language such a knowledge should be unambigously expressed, and by whose rules of inference (all and only) the consequencess of this knowledge could be inferred. What is known as "relational data model" is also such a "system".

The logic approach defines a way of conceiving "facts" and "knowledge", and a way of speaking about them. By "model" we mean a set of facts (states, values), while a "theory" is a set of propositions (assertions, knowledge, ...) concerning some model.

In accordance with such a conception, the content of a data/knowledge base is seen as a theory which expresses knowledge concerning some "world". If such a theory is correct and complete, it "tells the truth and only the truth" about the world, and we say that the world is its model. (Incidentally, in such a context, we would prefer to speak of the "relational system", rather than of the "relational model".)

3. The General Integrity Rules

In the relational data model (system), the integrity rules are traditionaly divided into two clases: general and data-base-specific rules. The first class contains only two rules, known as the entity integrity rule and the referential integrity rule. Although there is much "theoretical debate" (not always very clear!) about these rules, their meaning is fairly simple and clear.

The entity integrity rule says that a thing about which we (want to) speak must be identified, while the referential integrity rule says that what we refer to must exist.

Although these rules are concerned with the relational data model (system), it seems that those requests should be a minimal condition of any "afirmative speech" to be considered inteligible! (Therefore, they should form a part of any system (language), except those of poetry. Of course, hypotetical speech (in religion, politics, ...) is

possible (and common!) even without real identification and existence.)

3.1. The Entity Integrity Rule

In spite of the simplicity of the entity integrity rule, there seems to be much vagueness and disagreement concerning its meaning and definition than one would expect. In <Dat 90, p. 279>, Date states this rule as:

"No component of the primary key of a base relation is allowed to accept nulls".

He specifies (p. 280): "... we take 'null' to mean, simply, a value or representation that is understood by convention not to stand for any real value of the applicable atribute" (underlined M.R.). In fact, the same was stated on p. 251, where we find: "null is not a value".

To evaluate Date's conception of the entity integrity rule, it seems apropriate to cite his definition of candidate (and primary) key also; in <Dat 90, p. 277> we find:

"Atribute K (possibly composite) of relation R is a candidate key for R if and only if it satisfies ...

1. Uniqueness: At any given time, no two tuples of R have the same value for K.
2. Minimality: If K is composite, then no component of K can be eliminated without destroying the uniqueness property".

Discusing the entity integrity rule, Date rightly emphasizes that "all kind of problems" arise if the key is (partially) null. Among other things, he stresses that in such a case we do not even know (in general) whether the tuple with a null in its key "represents" some of the entities about which we already have a tuple in the relation.

However, a "kind of problem" also springs from Date's concluding fragment of the section on entity integrity; he says:

"... it is commonly but erroneously thought that the entity integrity rule says something along the lines of 'Primary key values must be unique'. It does not. That uniqueness requirement is a part of the basic definition of the primary key concept per se. (underlined M.R.) The entity integrity rule says (to repeat) that primary keys in base relations must not contain any nulls".

In fact, it seems that there really are such "thoughts" as Date mentiones above; whether they are really "erroneous" or not, we will try to see in what follows. For example, in <Gro 90, p. 270> (a respectable book, in my opinion) Groff and Weinberg give the following definition:

"Entity integrity: The primary key of a table must contain a unique value in each row ..."

Evidently, this definition says exactly what Date denies! Now, which of the two (in fact, three) autors has an "erroneous thought"? And why, and where does the error come from? That is the question we shall try to answer now.

First of all, both positions seem to be coherent. Which one will then be the "loser"? We claim: The entity integrity rule itself!

For Groff and Weinberg, the

position is fairly clear: They give a definition, without much argumentation. The only objection which could be raised against their position is that such a definition makes the entity integrity rule superfluous. Namely, it only repeats what is already contained in the concept of uniqueness in the very definition of kandidate key!

Date's position seems somehow different. However, it seems that the difference arises mainly from his imprecise definition of the candidate key given above. Namely, what does it in fact mean that "no two tuples of R have the same value for K"?

Let us try to understand it by means of an example. Let the sheme of the relation R be

R(A, B, ...),

and let two tuples from R be

<a1, null, ... >
<a1, null, ... >.

Now, does this pair of tuples violate the above uniqueness request or not?

No, because according to Date, "null is not a value"! Therefore, it is impossible even to speak about the "same value"! And in such a case, it seems necessary to have Date's definition of the entity integrity rule, so as to "expell" the nulls from the key! Namely, otherwise "all kind of problems" would arise: adressibility of tuples, first of all ("null is not a value"!).

However, there was one more requirement in the definition of candidate key: minimality! And this one warns us clearly enough that if we assign to the key atributes something that "is not a value", we must lose uniqueness! And by that, we also deviate from the definition of the key (i.e., we lose even the key itself!). Namely, even though by assigning null to an atribute which is a component of the key, the component is not really "eliminated", it is eliminated in a functional sense: it is not "in function" any more! (A component can guarantee (or support) the uniqueness of the key only by being instantiated to a value - and not to something which is "not a value"!)

Therefore, we must once more conclude that the entity integrity rule (at least, as a rule!) is superfluous, because it only repeats what necessarily follows from the mere definition of the candidate key. Namely, a careful reading of the candidate key definition tells us that the candidate key atributes must not be null.

By the above claim of "imprecision" in Date's definition of the candidate key, we mean that with the expression "same value", Date should have really intended "a value"! Well, there is a negation in his definition of uniqueness, and that always make things less clear. Namely, as we already stated, "not a value" in fact satisfies the request for "no ... the same value"! However, it seems easy to escape this (rather sofistic) trap. We should only reformulate the uniqueness request, in perhaps the following a way:

"Any two (different) tuples of R must have different values for K."

Now, from "have different values" we should be allowed to infer "have value"! And then all additional rules concerning "nulls in key" become superfluous.

3.2. The Referential Integrity Rule

For discussing the referential integrity rule we should first give a definition of foreign key. This concept is pretty clear, however it seems very "unsuitable" for being caught in an "elegant" definition. Date, <Dat 90, p. 282>, states it in the following way:

"Atribute FK (possibly composite) of base relation R2 is a foreign key if and only if it satisfies the following ...

1. Each value of FK is either wholly null or wholly nonnull.

...

2. There exists a base relation R1 with primary key PK such that each nonnull value of FK is identical to the value of PK in some tuple of R1".

Now, the referential integrity rule, Date defines, <Dat 90, p. 284>, with the following words:

"The database must not contain any unmatched foreign key values".

Unfortunately, it seems that there are "some problems" even with this definition. Namely, according to the definition of foreign key, if some "nonnul value" i.e. "value" (all "values" are "nonnull", if null "is not a value"!) of a foreign key FK would be unmatched, it would not be a foreign key! (Read carefully clause (2) of the foreign key definition!) Therefore, in the context of the given definition of foreign key, the referential integrity rule seems to be superfluous (as was the entity integrity key in the context of the definition of candidate key).

With the above discussion, we didn't mean to criticize Date's exposition; in fact, he himself says that "... the foreign key and referential integrity concepts are defined in terms of one another. That is, it is not possible to explain what a foreign key is without mentioning the idea of referential integrity (at least tacitly) ...", <Dat 90, p. 285>. Incidentally, we "feel" that the last sentence is too strong. For it is not very clear why it should not be posible to define a foreign key if (some of) its values would be simply (momentarily) unmatched.

Let us conclude the discusion on general integrity rules with a few remarks and estimates of their position and role in the relational data model (system).

First of all, we hold that there is an esential difference between the request for "uniqueness" (entity integrity) and the one for "matching" (referential integrity). Namely, it is very clear that uniqueness (adresibility) is of an essential importance for the model. Therefore, it should be defined (included) at the level of the syntax of the language (system).

On the other hand, we hold that the referential integrity request is of no such importance on the theoretical level. Namely, what is in fact the (cognitive) difference between a null-valued foreign key and an unmatched one? Let as look at an example.

Let R1 and R2 be relations, and B be a foreign key of R2 in R1:

R1(A, ..., B, ...)
R2(B, ...),

and let

<a1, ..., null, ... >
<a2, ..., b2, ... >
<a3, ..., b3, ... >

be tuples in R1. Further, let the relation R2 contain a tuple

<b2, ... >,

but not the tuple

<b3, ... >.

Now, the first two tuples of R1 do not violate the referential integrity rule (the rule allows nulls), while the third one does. However, we think that there is not much difference among them, at least not at the theoretical level of the data model (system). Namely, the first tuple refers to "something unknown", while the third refers to "something unknown, designated as b3".

Of course, there is a difference between the first and the third tuple of R1 on the operative (practical) level. Perhaps by inserting null (instead of b3) in the third tuple above, we would not bring in much information about "the world": in fact, "the reality" would be "even more uknown"! However, this "even more" is useful, because - with it, expressed as null - we know that the "referred entity" is actually unknown. And that could be a useful information, not about "the world", but about our knowledge of it! For without a null, we could erroneously hold, in accordance with "common sense", that there is a tuple b3 in R2.

In accordance with what is stated above, we hold that the request for referential integrity is a problem which belongs (much) more to the level of implementation (software) than to the basic theoretical (conceptual) level of the model (system). Therefore, we put forward three possible ways of "theoretical treatment" of this rule, where our preference goes from the first downwards.

1. Referential integrity (as a request for obligatory matching) could be ommited as an explicit rule, and as an implicit part of the foreign key definition.

2. Referential integrity could be (and in fact actually is) stated inside (as an implicit part of) the definition of foreign key. In that case, "the problem" is raised on the level of syntax, and there is no more need of a special (explicit) theoretical treatment.

3. Finaly, if we insist to keep referential integrity (as a rule) in the system, then a referential integrity violation should be reduced to the inconsistency of the database as a theory. To achieve this, we should simply define a rule as a "generally holding knowledge (truth)", in the following way:

"For every fk, such that fk is a value of a foreign key, there is (in the database) a primary key with value fk."

Such a rule would then be part of the (knowledge contained in) the database.

Now, if an "actual content" of the database (which is always finite) would allow us (or the DBMS) to infer (compute) that:

"There is such a value of an atribute(s) defined as a foreign key (in some relation) which is actually not a value (instance) of the refered primary key",

then such an inference (result) would contradict the integrity rule stated above, and show the inconsistency of the database content. Naturally, such a control (checking) should be performed before (every) update of a database.

4. Database-specific Integrity

According to Date, a database-specific integrity rule (constraint) "can be regarded as a condition that all correct states of the database are required to satisfy", <Dat 90, p. 437>. Such an assertion immediately lead to the question: What does it mean for a state to be "correct"?

Earlier, in <Dat 90, p. 275>, we find: "... database consists of some configuration of data values, ... (which) ... is supposed to 'reflect reality' (i.e., it is supposed to be a model ... of some piece of the real world ...)". (As we already stated, it would be much more suitable to say that a database is a theory rather than a model. Namely, it is the theory we intend to "reflect reality"!)

"Now, some configurations of values simply do not make sense, in that they do not represent any possible state of the real world", continues Date.

All that sounds well; however, there are too many "critical" (not well defined) concepts here: in addition to "correctness", we now also have "sense" and "possible states of the real world". Let us start with the last one.

Date says (same page) that "weights cannot be negative in the real world". Sounds wise. However, strangely enough, temperature can! Moreover, even the water level of a river can be negative! (I must confess that this "fact" perplexes me anew whenever I hear such a report. I always have an impression that a river somehow went underground!)

With these examples, we simply wanted to ilustrate that it is not suitable to speak in terms of what "can" or "cannot" be, or what is "(im)possible" in the "real world" (which is itself not a very clear concept!). Therefore, we propose to move the discusion to the level of contradiction and (in)consistency, which are well defined concepts, and also "manipulable" in the computational sense.

Philosophically speaking, the first reason for such an approach would be that there is no such a thing as a "mere fact": Facts are always something already shaped in (or by means of) our language and measures! (Well, even if (at some "deep level") there is the True Reality, it is - by definition - indiscernible for us, because the "eyes" and the evaluation of the "seen" will always be ours!)

Now, (database-specific) integrity constraints would express that (relevant) knowledge which is already "built-in" (contained) in our language and measures. Such integrity constraints would not protect the database from "asserting impossible things" but would simply prevent such changes which would lead to contradiction with the knowledge which tacitly holds in the language and which is formally expressed by the database-specific integrity constraints. For example, a constraint concerning (our conception of!) weight could state (in some DDL of DBMS):

"There is no such a thing as a weight less than zero."

Now, if some update would lead to a state in which some value for the atribute WEIGHT would become "less than zero", it would mean that this update would make the database inconsistent! In such a way, the concepts of "cor-

rectness", "sense" and "(in)possibility" are reduced to the basic (and "most suitable") concept of consistency.

Inconsistent theory don't have a model; therefore, we can say with certainty that an inconsistent database would not speak of the "real world" (not even of an "ireal" one, at least not inteligibly). (Note that in such a "conceptual (data) system", integrity constraints are part of the database (theory), although they do not in general state "isolated facts" but "rules" which single facts must not contradict.)

Of course, it is not practically possible to prevent all possible errors in the database on the formal level. However, we hold that those preventive measures which are possible should be best done (defined) by means of (database-specific) integrity rules, where integrity is reduced to the concept of consistency.

5. Conclusion

By discussing integrity rules in the relational data model, we tried to show and emphasize the suitability of the logic, proof--theoretic approach to the definition (or conceptual foundation) of data models in general.

By claiming that general integrity rules are "superfluous" we didn't mean that their content is irelevant, but that this content could be stated (expressed) within definitions of the primary and foreign key. In that case, general integrity rules (as defined e.g. in <Dat 90>) could be treated as simple "corollaries" (i.e., evident consequences) of those definitions.

On the other hand, the database-specific integrity rules should be treated as expressions of that knoledge which is contained (built-in) in our language, measures and atitudes. Database-specific integrity rules are, in fact, general knowledge wich should be thought (and treated) as part of the database. Such a conception and role of the integrity allows us to reduce the concept of "database correctness" to its consistency (as a theory). Of course, at the operative level (i.e., at the level of the DBMS) they would serve to prevent those updates which would bring the database to inconsistency.

REFERENCES

<Dat 90> Date, C.J.: An Introduction to Database Systems, Vol. 1 (Fifth ed.), Addison-Wesley, 1990.

<Gal 78> Gallaire, H., Minker, J. (eds): Logic and Data Bases, Plenum Press, 1978.

<Gro 90> Groff, R.J., Weinberg, N.P.: Using SQL, McGraw-Hill, 1990.

<Rad 87> Radovan, M.: "On Deduction and Consistency in Clausal Logic", Acta Analytica, No 3, 1987.

<Rei 84> Reiter, R.: "Towards Logical Reconstruction of Relational Database Theory", in Brodie, L.M. et all (eds): On Conceptual Modeling, Springer-Verlag, 1984.

UNDERSTANDABILITY OF THE SOFTWARE ENGINEERING METHOD AS AN IMPORTANT FACTOR FOR SELECTING A CASE TOOL

INFORMATICA 3/92

Keywords: CASE tool, engineers testing, learning properties, software engineering

I. Rozman, J. Györkös, K. Rizman
University of Maribor
Faculty of Technical Sciences
Smetanova 17, 62000 Maribor, Slovenia

ABSTRACT — The article highlights the understandability of a software engineering methodology as an important criterion for selecting a CASE tool. This aspect is treated through the comparison of learning properties for two very well known methodology on which the CASE tools are usually based on. The first one is SA-SD and the second one is JSD. In the purpose to compare both methodology a group of young engineers has been tested. Each of them wrote a seminar theme, answered a questionnaire and explained his observations. At the end of the paper, a general conclusion is presented.

1 Introduction

Computer - Aided Software Engineering (CASE) tools are coming into widespread use in the software engineering. These tools automate methods that can improve software development practice and help to improve the quality of a software product. Choosing a CASE tool can be a difficult and confusing task because the great number and variety of tools are on the market. Many potential user of the CASE tool base their selection upon literature from the vendors, perhaps with a trial period using a demo system.

In article [6] a method for selection a CASE tool is described as a six step process:

- define the CASE requirements,
- contact vendors for information,
- select systems and vendors for in-dept evaluation,
- require vendors to respond in writing to the CASE requirement and to loan demonstration copies of the system trial use and evaluation,
- if possible, visit vendors at their home sites,
- select a CASE system for purchase.

In the first step, analysis and/or design methodologies which support the tool as an overall system requirement are expressed. It should be noted, that the tool

- must have a full semantic understanding of the graphical and textual object comprising each analyses and design component,
- must handle a minimum number of levels (e.g. 10 or 15) in hierarchical analyses or design methodologies, and a minimum number of components such as data dictionary entries, data-flow diagrams, or structure charts,
- must support completeness and consistency checking across the analyses and design and through all levels of the analyses hierarchy.

But, the influence of methodology to be learned itself, is not observed in this article. Methodology is prerequisite and essential for using the CASE tool. The importance of this criterion, as an important criterion for characterizing the CASE tool, was factually not expressed in last workshop CASE 90 [2]. But from our experience, we know that a full satisfaction with a CASE tool can be achieved only if users like to use them. A tool must be based on methodologies which are close to the way of the user's thinking and close to his/her experience in the past, or that it aids his/her existing knowledge.

In order to highlight the problem of friendliness of a CASE tool to the user, we have compared two formal methodologies regarding to the user's capability to understand it and to learn how a particular method can be applied to the software development. The both chosen methodologies are taken from the two main groups and are quite different. These groups are:

- Data Flow Oriented Methodologies and
- Data Structure Oriented Methodologies.

From the first group the well known SA-SD methodology (Structured Analysis according to DeMarco [1] and Structured Design according to Yourdon and Constantine [5]) was chosen. From the second group the JSD methodology [4] (Jackson System Development, author is M. Jackson) was chosen, because this methodology is the most typical and in certain expert circles most widely used. And what has not the least importance, some authors classify this methodology near the object oriented ones (e.g. [3]).

Our comparison is based on an experiment in which we tested some engineers to find out which methods are easier to learn.

2 Experiment description

Experiment has been applied to twelve person tested who learned the both mentioned methodologies during the post - graduate study. Eight of them are electrical engineers, two are mechanical engineers and two are civil engineers. With the aid of lectures attended (fifteen hours) and literature studied each of them had to solve a problem, approx. 3000 LOC, in the form of a seminar work and according to methodologies cited above. The problems to be solved were mainly procedural oriented, but some of them had important data component, too. If difficulties in individual design appeared consultations were given. When their work was finished they presented their results and answered the questionnaire given. At the same time they passed the examination.

2.1 Questionnaire

The questionnaire is divided into two logical unit. The first two questions are general. The answers are expected to reflect the influence of previous skill of students for modelling a real world. The last four questions are expected to give an individual note of both methodologies and to show which methodology the young engineers are more familiar with. A descriptive manner of answering is foreseen for the last four questions. The narrowness of possible answers and their misguiding influence are avoided, but a descriptive treatment of the results of the questionnaire is needed. For the presentation in the last question the well known Osgood's Semantic Differential with four-level scale is used .

Q U E S T I O N N A I R E

1. Which programming language is most often used in your programming experience?

a) FORTRAN

b) Pascal

c) Other

d) None

2. Your task is to model a part of the real world into a software system. When such task is given to you, do you think that you understand the required reality:

a) Completely

b) Only main aims are known; the details are unknown.

c) The details are known; their uniting into a system is unknown.

3. If you don't understand the problem completely, the reason is as follows:

a) Ignorance of the skilled perspective of reality.

b) It is not known how to express the function of the problem (It can only be "felt")

c) Others

4. What do you think of a problem that should be solved?

a) The model of reality is seen.

b) The model is seen as a transformation of the input picture into the output picture.

5. Which software formal developing method are you going to use with your future work?

a) JSD

b) SA-SD

c) None

6. Why are you going to use the method marked under point 5?

a) ----------

b) ----------

c) ----------

(give at least two answers)

7. Why are you not going to use the other method mentioned or even none of them?

a) ----------

b) ----------

c) ----------

(give at least two answers)

8. With the marks 0 (very bad), 1 (bad), 2 (good), 4 (very good) try to express the degree of satisfaction of particular activities in the life cycle for both methods

	JSD	SA-SD
A - assistance in formulation of requirements		
B - quick detection of faults		
C - surveyability		
D - simple maintenance of end-product		
E - simple completion of end-product		

Figure 1. Questionnaire

2.2 Interpretation of Answers

The answers to the first five questions are shown in the tabular form on the figure 2.

answer	1	2	3	4	5
a	10	2	7	3	1
b	2	8	2	10	11
c	0	2	3	x	0
d	0	x	x	x	x

Figure 2. Tabular presentation of answer to the first five questions

Most young engineers answered that they are familiar with the FORTRAN language. This is understood because the young engineers mostly learned this language during their regular education. To have only a skill about FORTRAN means that these people are "half illiterate" as regards programming. It is sure that ignorance of data part of the program makes the understanding of any software development methodology difficult. In the SA-SD methodology this is seen from the difficult understanding of the data dictionary and data handling in the files as a data base. In the JSD methodology, where entity structure is dominant, ignorance of the data part of the program was undoubtedly one of the reasons for more difficult understandability.

The answers to the second question show that the tested persons tried to get the most abstract picture of all important functions. Two of them decided that they had known details but they had not had a clear perspective about uniting details into the system. The conversation in the experiment showed that in these two cases tested people were working on the problem long time and the seminar work done during the experiment was prolonging their continuous work.

The answers to the questions three and four show that in the most cases the functional decomposition was used as appropriate way of thinking for decomposing a problem into small pieces. The fifth question demanded a concrete decision regarding the methodology used in the future by the individual. The majority (eleven persons) decided to use the SA-SD methodology. Let us consider some most important arguments for preferring SA-SD methodology which were answered under question six:

- methodology was easy to learn,
- the philosophy of this methodology is close to my way of thinking,
- a clear model is quickly reached and
- the methodology is simple and efficient.

The JSD methodology was negatively appreciated (answers to question seven). The arguments are as follows:

- it is difficult to learn,
- analysis from the entity/action aspect is difficult to understand and
- implementation is not simple.

The presented seminar works also showed worse understanding of the JSD methodology. Where usually had the tested people problems? In the first "entity action step" they were in doubt about boundary of the modelled system. They enumerated actions easier as they decided about entities. The second problem, which appeared, was the understanding what the entity is? They usually forgot that actions must appear on the entity what is not the same as the entity in the E-R (Entity-Relationship) diagram.

In the second "entity structure step" tested person had problems to understand that each structure in Jackson's diagram has only two levels. The first level is the structure's name and the second level form the parts of this structure, while in the Structure Chart each box represents a program module. In the "initial model step" - step three seems difficult to understand what really a process marked by zero and process marked by one means. In the beginning, it was not very clear to the students why the same object is treated as structure of entity in step two and as process in step three. This was the most repeated question which tested persons had during the working their seminar work.

When the first steps were understood the remaining three did not cause greater problems. We are surprised that the function of scheduler (implementation step) is quickly understandable, what means that testes had enough knowledge about concurrent programming. The essence of Jackson's pseudo cod is well accepted, too.

The answers to the eighth question are shown in a graphical form as a average values of collected points of semantical differential. The results are surprising. Despite of the negative attitude towards the usage of JSD methodology, both methodologies are relatively equally assessed; figure 3. Cumulation of points for question "E" is greater for JSD as for SA-SD. This means that young engineers decided correctly, accepted the essence of both methodologies, and assessed that JSD is more appropriate for completion of end product than SA-SD.

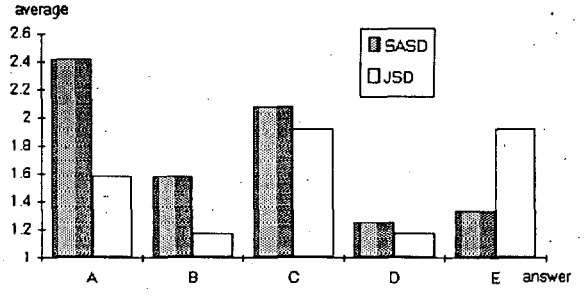


Figure 3. Presentation of the answers to the eight question

3 Conclusion

It should be stressed that it is not our intention to compare all aspects of both methodologies used in the experiment but only to present that the learning aspect of a methodology is an important criterion which has to be taken into account in the process of selecting a CASE tool. To assess this criterion we suggest the usage of a questionnaire like this one, presented in our paper. We are aware of the fact that such exhaustive investigation the problem of the understandability the methodology, as it is presented in the paper, is too time consuming. But however, some problems such as: types of problems which are usually solved by the group, previous skill of group about modelling real world and programming, familiarity with software engineering methodologies, satisfaction with proposed methodology, should be answered by the questionnaire. The most time needed for questionnaire can be saved with reducing problems with which person are tested. In our experiment we intentionally have not used any CASE tool because we were observing only methodologies themselves. The usage of any tool would make our picture about methodology learning unclear. We have not evaluate the influence of the tool's teaching component. This is a very difficult task which demands additional investigation and it is valid for a particular tool only.

The results obtained speak in favour of the SA-SD methodology because it is more understandable to the group of engineers who represent, in our case, the potential user of selected CASE tools. Perhaps, we are wondering, why SA-SD is better assessed than JSD? Our opinion that we obtained such result is, that in engineering curriculums people have gotten much more skills about programming with procedural languages than programming with object-oriented languages. So, we can legitimately expect that those CASE tools which support data flow oriented methodology (like SA-SD) need a shorter training period for people with engineering education than those CASE tools which support other methodologies.

If we want to select a CASE tool for another group of users which have got more knowledge and skills in object-oriented programing it is possible that we achieve another fully opposite conclusion about the methodology supported.

References

[1] T. DeMarco *Structured Analysis and System specification*, Prentice-Hall, 1979

[2] R. J. Norman, R. V. Ghent. (Editor), *CASE'90, Four Internal Workshop on Computer-aided Software Engineering*, Irvine, CA (December, 1990).

[3] B. Henderson-Sellers, J. M. Edwards, *The Object - Oriented Systems Life Cycle*, Communication of the ACM, vol. 33, no. 9, Sept. 1990, pp. 143-159

[4] M.A. Jackson, *System Development*, Prentice-Hall, 1983

[5] E. Yourdon, L.L. Constantine, *Structured Design: Fundamentals of a Discipline of Computer Program and System Design*, Prentice-Hall, 1979

[6] L.Zucconi, *Selecting a CASE Tool*, ACM SIGSOFT, vol. 14, no. 2 Apr 1989, pp. 42-44

Keywords: tree, network, allocation, network construction

Peter Zaveršek*, Peter Kolbezen**
* VELCOM d.o.o., Foitova 8, Velenje
** Institut Jožef Stefan, Jamova 39, Ljubljana

POVZETEK. Delo obravnava preslikavo algoritma, ki je določen s hierarhičnim acikličnim grafom pretoka podatkov (HADFG), na drevesno avtomatno strukturo (TAS). Predlagani postopek preslikave upošteva prostorsko optimizacijo avtomata. Podana je primerjalna analiza med avtomatom z drevesno in avtomatom s toroidno strukturo. Na osnovi te analize so pokazane dobre in slabe strani obeh struktur avtomata.

ALLOCATION OF HADF GRAPHS ONTO A TREE STRUCTURE. Hierarchical acyclic data flow graph (HADFG) tree-shaping form encouraged us in investigating a possibility of their allocation onto a tree-like automat structure (TAS). Various approaches can be used in order to obtain an appropriate structure. The paper proposes an analytical space-optimizing approach towards the network construction. Further, execution results on such a network are compared to the ones obtained by a torus-like network and the behaviour of each of them is discussed.

1. Uvod

V delih [1, 2] je opisan večprocesorski sistem za učinkovito izvajanje hierarhičnih acikličnih grafov pretoka podatkov (HADFG). Sistem ima toroidno avtomatno strukturo. Preslikava HADFG na takšno strukturo je posredna. Mehanizem posredne preslikave s svojimi neželenimi vplivi podaljšuje čas izvajanja HADF grafov, ker zahteva zase del procesorskih zmogljivosti. Tako si lahko upravičeno zastavimo vprašanje uspešnosti neposredne preslikave HADFG na avtomatno strukturo. Postopek preslikave, če naj bo uspešen, mora v povezavi z avtomatno strukturo čim manj obremenjevati procesorje. Preslikava naj bo zato neposredna, procesorji v strukturi pa statično povezani. Zaželena je avtomatna struktura, ki se povsem ujema s programskim grafom v času izvajanja grafa. Procesorji v dinamično rekonfigurabilnem polju, kot npr. [3] ali sistemi ParsyTec-a [6], bi poleg kompleksnejše strojne opreme zahtevali dodatne procesorske zmogljivosti za povezovanje v času izvajanja. Omenjene sisteme bi lahko uporabili kot osnovo statičnim sistemom, če bi procesorje v njih ustrezno povezali pred začetkom izvajanja programskih grafov. Procesorji v statični strukturi, ki jo zahtevamo, bodo predvidoma manj izkoriščena kot tisti v toroidnem sistemu, ker se statična struktura ne more prilagajati trenutnim razmeram v sistemu (zasedenost, razpoložljivost procesorjev). S transformacijo DFG v HADFG dobimo drevesni programski graf s korenom zgoraj, in vejami, ki se širijo od korena navzdol [1]. Kot je splošno znano, razlikujemo pri drevesnih grafih več tipov vozlišč. *Koren* grafa je vozlišče, ki nima predhodnika, in ima enega ali več naslednikov. *Razvejišče* je vozlišče, ki ima enega predhodnika in enega ali več naslednikov. Vozlišče grafa, ki ima le predhodnika in nima nobenega naslednika, imenujemo *list grafa*. Če poskušamo takšen graf čimbolj neposredno preslikati v avtomat, lahko sklepamo da bo imela tudi avtomatna struktura drevesno obliko. To nas je vzpodbudilo k primerjalni raziskavi drevesne avtomatne stukture in že v delih [1, 2, 4] obravnavane toroidne strukture. Pričakujemo, da bo neposredna preslikava na drevesno avtomatno strukturo (TAS) ponudila boljše rezultate glede na čas izvajanja vsaj za neko družino HADF grafov. Družino HADFG, ki se bo uspešneje izvajala, želimo tudi identificirati. Zato poskušamo v tem delu odgovoriti, ali je za izvajanje alogritmov, ki so predstavljeni v obliki HADF grafov, učinkovitejši toroidni sistem ali TAS.

2. Drevo in torus

Predlagana toroidna struktura v delih [1, 2] je sestavljena iz množice prepletenih zank (prstanov) transputerjev. Vsak transputer v avtomatni strukturi je fizično vezan na štiri sosede. Komunikacija med njimi je krožna po zankah, ki sestavljajo torus, in poteka eno-

smerno. Vsak procesor ima zato le dva logična soseda. Komunikacijske zanke ponujajo možnosti dvostranskih prenosov, ki jih izkoriščamo tako, da znotraj ene fizične zanke vzpostavimo dve logični komunikacijski zanki. Medtem, ko se po eni prenašajo podatki, ima druga funkcijo prenosa krmilnih sporočil in ukazov. Zasnova sistema podpira dinamično, "run-time" razporejanje procesov glede na proste vire v polju. Izbrana arhitektura, podprta s preprostimi mehanizmi optimizacije razporejanja, omogoča skupaj z uporabljeno arhitekturo sorazmerno učinkovito preslikavo programskega grafa na statično povezano polje procesorjev.

Vsi procesorji v toroidnem polju imajo enako število sosedov, zato so si enakovredni ne glede na položaj v vezju. Ker imajo vsi procesorji logične naslednike, se polje navidezno nikjer ne zaključi. Izvajanje programskih algoritmov, ki so predstavljeni v obliki HADFG, je, zahvaljujoč virtualni razširljivosti toroidnega polja, učinkovito in uspešno tako pri manjšem, kot pri večjem številu procesorjev.

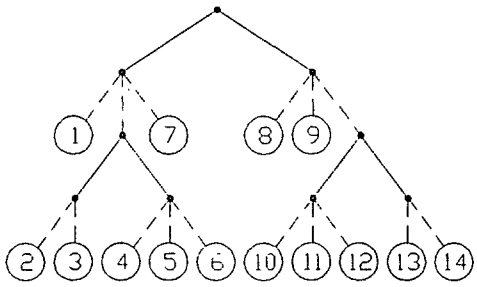
Drevesna struktura za razliko od toroidne ni virtualno razširljiva. Terminalni procesorji (listi drevesne avtomatne strukture) nimajo naslednikov. Širjenje HADF grafa po takšnem polju se zaustavi na terminalnih procesorjih, kjer se struktura zaključi. Zato drevo ne ponuja vsem procesorjem enakovrednega položaja v strukturi. Učinkovita preslikava HADF grafa na drevesno strukturo zahteva ustrezno razvejanost in globino strukture.

Drevo torej ni tako prilagodljivo, kot je torus. Če želimo izvajati programski graf na statično povezani drevesni avtomatni strukturi, moramo ustrezno strukturo praviloma zasnovati vnaprej. Struktura je lahko predvidena ali za vsak HADF graf posebej ali za neko natančno določeno družino HADF grafov.

3. Določitev drevesne strukture

Učinkovito izvajanje algoritma, ki ga predstavlja določen HADF graf, zahteva ustrezno drevesno avtomatno strukturo. Razsežnost avtomata mora biti vsaj takšna, kot je razsežnost preslikave grafa. Globina avtomatne strukture mora biti vsaj enaka številu paralelnih nivojev HADF grafa, medtem ko mora širina na opazovanem delu ustrezati številu paralelnih vej v vozliščih grafa, ki se bodo preslikala na to avtomatno vozlišče.

Zahtevamo, da je avtomatna struktura splošna. Ustrezati mora vsakemu HADF grafu dane oblike in mora zagotavljati neomejeno širjenje procesorskih aktivnosti v njej, neodvisno od časov izvajanja posameznih procesov. Posledica pogoja takšne časovne neodvisnosti je, da je tudi rezultat optimizacije skromnejši. Primer optimiranja avtomatne strukture je npr. sekvenčno izvajanje dveh kratkih paralelnih procesov ob tretjem zelo dolgem, s čemer lahko prihranimo kakšen procesor. V nadaljevanju bomo nakazali možne poti, ki nas pripeljejo do ustreznega avtomata.



Slika 1: Izhodiščni HADF graf

3.1 Prekrivanje

HADF graf natančno predpisuje način izvajanja posameznih vej v skladu s tipom veje (paralelna, sekvenčna). Možnost, ki se sama ponuja, je neposredna preslikava grafa v avtomatno strukturo. Glede na tip veje lahko zaključimo naslednje:

- **Paralelne veje:** Vse paralelne veje, ki izhajajo iz istega (paralelnega) vozlišča, se izvajajo sočasno. Izvajanje ostalih paralelnih vej grafa glede na opazovano vejo je določeno z njihovo lego v grafu. Zaželeno je, da se paralelne veje znotraj enega paralelnega vozlišča, kot tudi vse njim paralelne veje drugod po grafu, izvajajo sočasno. Zajeti želimo ves ponujeni paralelizem in tako doseči največjo stopnjo paralelnosti izvajanja. S tem bo tudi hitrost izvajanja največja. Med izvajanjem moramo imeti vsak trenutek na razpolago zadostno število procesorjev, odgovarjajoče številu procesov, ki jih je v danem trenutku možno izvajati sočasno. Procesorji morajo biti medsebojno povezani tako, kot zahteva HADF graf, saj v nasprotnem primeru preslikava procesov nanje ne bi bila možna.

- **Sekvenčne veje:** Sekvenčne veje se izvajajo druga za drugo. Pravimo, da so si tuje. Pri izvajanju izrabljajo isti del avtomatne strukture. Kaskado večih zaporednih sekvenčnih procesov lahko obravnavamo kot en sam proces. Vse procese, ki sestavljajo kaskado, preslikamo na isti procesor. Podobno obravnavamo podgrafe, ki si sledijo drug za drugim. Tudi te preslikamo na isti del avtomatne strukture. Zavedati se moramo pomembne razlike med procesi in podgrafi. Posamezni procesi zasedejo le en procesor, medtem ko je razsežnost in oblika zahtevanega avtomatnega polja za podgraf določena s pripadajočo globino in širino podgrafa. Potrebno globino podavtomata določa število paralelnih nivojev najglobjega podgrafa. Število procesorjev na posameznem nivoju je enako številu paralelnih vej tistega peralelnega vozlišča nivoja, ki ima največje število vej.

Gornje ugotovitve strnimo v preprost postopek, poimenovan *prekrivanje*. Z njim je že mogoče določiti

avtomat. Postopek prekrivanja na primeru grafa iz slike 1 prikazuje slika 2 in poteka takole :

1. postopek pričnemo izvajati na podgrafu, ki je določen s skrajnimi levimi sekvenčnimi vejami podgrafa (slika 2-I).
2. Preslikamo vozlišča v procesorje po naslednjem pravilu:
   - sekvenčna vejišča se preslikajo v procesorje, povezave med procesorji pa določajo paralelne povezave grafa
   - procesi (listi HADFG drevesa) se preslikajo v procesorje
   - če v avtomatni struktruri na mestu, kjer bi moral biti procesor slednjega še ni, ga dodamo (temnejši procesorji na sliki 2)
   - če ima avtomatna struktura več procesorjev, kot je potrebno, jih ne odvzemamo
3. Izberemo naslednjo sekvenčno vejo na najnižjem sekvenčnem nivoju (naslednji podgraf) in nadaljujemo od točke 2 dalje, dokler ni obdelan celoten graf (slike 2 II-IV).

Metoda prekrivanja avtomatne strukture z ustreznimi deli grafa, nam omogoča enostavno konstruiranje avtomatne strukture. Tako pridobljena avtomatna struktura popolnoma ustreza danemu grafu, vendar je v splošnem predimenzionirana.

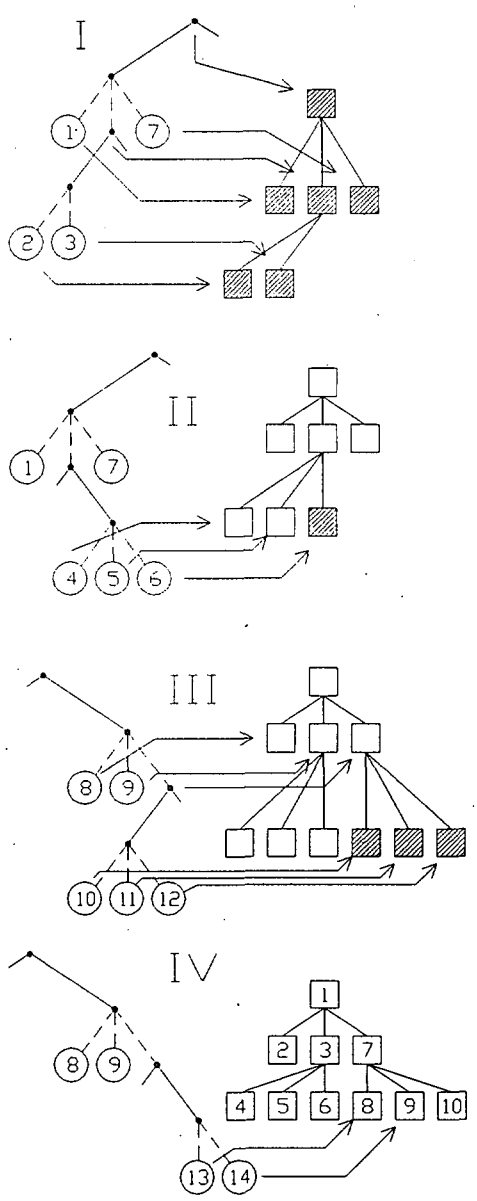
Ugotovimo lahko, da je možno graf ustrezno preoblikovati in realizirati funkcionalno enakovredno strukturo z manjšim številom procesnih elementov.

3.2 Urejeno prekrivanje

HADF graf ima v splošnem nesimetrično obliko. Nesimetričnost se odraža v različnih globinah podgrafov istega nivoja, in v različnem številu vej, ki izhajajo iz posameznih paralelnih vozlišč grafa. Vrstni red posameznih vej znotraj paralelnega vozlišča je funkcionalno nepomemben, saj se vse veje istega vozlišča izvajajo sočasno. Ta lastnost nam daje možnost, da veje med seboj kombinatorično permutiramo tako, da dosežemo kar najboljše prekrivanje nastajajoče drevesne avtomatne strukture z obravnavanim grafom. Originalni HADF najprej graf ustrezno preoblikujemo, nakar sestavimo avtomat z zgoraj opisanim prekrivanjem.

Uspešnost postopka se kaže v prostorski optimizaciji ATS in je neposredno odvisna od preoblikovanja grafa. Iterativne metode (npr. simulirano ohlajanje) bi lahko bile zelo uspešne, vendar potrebujejo za svoje izvajanje več časa. Analitični pristop je hitrejši in zaradi drevesne strukture enostavno izvedljiv, zato smo ga uporabili tudi mi.

Prvi korak do optimizacije je torej ureditev HADF grafa. Graf uredimo tako, da znotraj vseh paralelnih vozlišč razvrstimo skrajno levo najglobje in skrajno desno najplitvejše veje. V primeru enake globine



Slika 2: Postopek prekrivanja

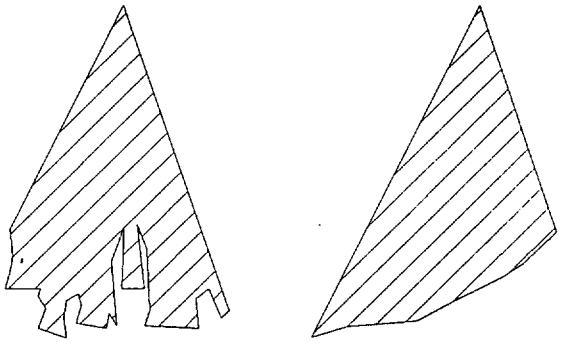
dveh vej ima prednost tista, ki ima večje število procesov na najnižjem nivoju. Opisani postopek prevede drevo iz neurejene v urejeno strukturo, kar lahko shematično prikažemo na sliki 3.

3.2.1 Algoritem urejenega prekrivanja

Algoritem deluje na principu utežitve vozlišč HADF grafa. Uteži, ki jih pripišemo vozliščem, so le računski pripomočki in nimajo nobene povezave s časovnimi utežmi grafa. Primerna utežitev nam olajša postopek urejanja grafa.

Predpostavke za HADF graf:

- začetno vozlišče HADF grafa je sekvenčno vozlišče, ki predstavlja ničti nivo v grafu ($l = 0$). V nasprotnem primeru govorimo o večih grafih, ki se lahko izvajajo paralelno.



Neurejena struktura Urejena struktura

Slika 3: Struktura grafa

- na koncih vej, ki izhajajo iz korenskega vozlišča, se nahajajo procesi ali paralelna vejišča. To je prvi nivo v grafu ($l = 1$).
- ko se spuščamo po grafu navzdol, se izmenjujejo sekvenčni in paralelni nivoji. Nivo v grafu se poveča z vsakim naslednjim vozliščem za ena ($l_{i+1} = l_i + 1$).

Utežitev listov in vozlišč grafa ter urejanje
Prvi, prioritetni kriterij pri urejanju grafa je globina. Širina predstavlja drugi kriterij. Utežitev vej mora zagotoviti globjim vejam večjo težo ne glede na širino primerjanih plitvejših vej. Uteži moramo izbirati tako, da ima utež na najnižjem nivoju večjo težo, kot je vsota vseh ostalih uteži na višjih nivojih.

$$g_n > \sum_{i=0}^{N-1} g_i$$

Predno začnemo uteževati posamezne liste, moramo določiti težo uteži za liste HADF grafa po posameznih nivojih. Največje število paralelnih vej N, ki izhajajo iz katerega koli paralelnega vozlišča v grafu, je v skladu z gornjo zahtevo in mora biti znano. Enakovredno bi ustrezalo tudi poznavanje največjega števila paralelnih vej po posameznih paralelnih nivojih, vendar zaradi poenostavitve postopka gledamo na HADF graf kot celoto.

Uteži določamo po naslednjem pravilu:

Utež q_0 ($l = 0$)
q_0 ... izberemo: $q_0 = 0$

Utež q_1 ($l = 1$)

$$q_1 > q_0$$

q_1 ... izberemo: $q_1 = 1$

Utež q_2 ($l = 2$)

$$q_2 > \sum_{i=1}^{N} q_1 = N \cdot q_1 = N$$

Najmanjša možna razlika za

$$q_2 > N \ldots q_2 = N + 1$$

Utež q_3 ($l = 3$)

$$q_3 > \sum_{i=1}^{N} q_2 = N \cdot q_2 = N(N+1) = N^2 + N = \frac{N^3 - 1}{N - 1} - 1$$

$$q_3 = q_2 + 1 = N^2 + N + 1 = \frac{N^3 - 1}{N - 1}$$

⋮

Utež q_k ($l = k$)

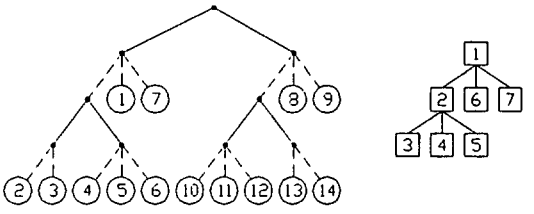
$$q_k > \sum_{i=1}^{N} q_{k-1} = N \cdot q_{k-1} =$$

$$= N(N^{k-2} + N^{k-3} + \cdots + N + 1) =$$

$$= N^{k-1} + N^{k-2} + \cdots + N^2 + N = \frac{N^k - 1}{N - 1} - 1$$

$$q_k = q_2 + 1 = N^{k-1} + N^{k-2} + N^2 + N + 1 = \frac{N^k - 1}{N - 1}$$

Pripisovanje uteži procesom HADF grafa
Ko so uteži določene, utežimo z njimi liste grafa (procese) v skladu z njihovo lego v grafu. Paralelne in sekvenčne procese obravnavamo po ločenih kriterijih.



Slika 4: Urejen HADF graf

Paralelni procesi

- obsegajo liste grafa na nivojih $l = 2k$; $k = 1, 2, \ldots$
- iz posameznega vejišča izhaja največ N takšnih procesov
- utež $q_n = q_n|_{n=\frac{1}{2}} = \frac{N^n - 1}{N-1} \Longrightarrow$
  $\Longrightarrow q_1|_{l=2}, q_2|_{l=4}, \ldots q_k|_{4l=2k}$

Sekvenčni procesi

- obsegajo liste grafa na nivojih $l = 2k - 1 \; ; k = 1, 2, \ldots$
- število vej iz posameznega vejišča ni omejeno
- utež $q_n = q_{n|_{n=lDIV2}} = \frac{N^n - 1}{N - 1} \Longrightarrow$
  $\Longrightarrow q_{0|_{l=1}}, q_{2|_{l=3}}, \cdots q_{k|_{l=2k-1}}$

Pripisovanje uteži razvejiščem grafa
Paralelna razvejišča ($l = 2k - 1$)
Paralelnemu razvejišču pripišemo vsoto uteži vseh vej, ki iz njega izhajajo:

$$q_p = \sum_{i=1}^{m_p \leq N} q_i$$

Sekvenčna razvejišča ($l = 2k$)
Sekvenčnemu razvejišču pripišemo utež najtežje veje sekvenčnega razvejišča.

$$q_s = \max_i q_i$$

Ko vsem vozliščem grafa pripišemo ustrezne uteži, uredimo veje znotraj vseh paralelnih vozlišč po utežeh, od veje z največjo (levo) do veje z najmanjšo težo (desno).

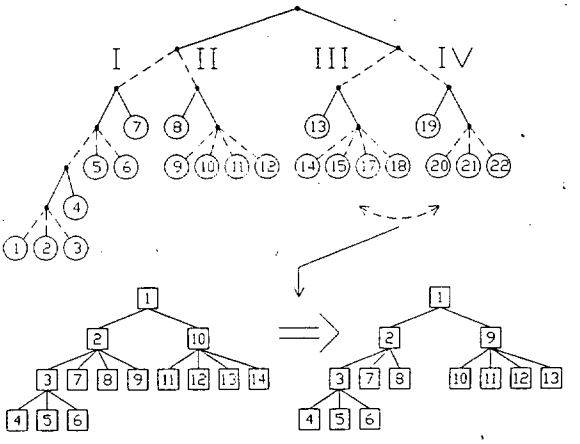
Avtomatno strukturo konstruiramo iz tako preoblikovanega grafa po enakem postopku kot prej, t.j. s prekrivanjem. Avtomatna struktura urejenega grafa je praviloma optimalnejša (manjše število procesorjev) kot tista, ki jo dobimo neposredno iz originalnega HADF grafa.

Rezultat generiranja avtomatne strukture s predhodnim urejanjem HADF grafa si oglejmo na primeru. Originalni par: neurejeni graf in pripadajoča avtomatna struktura je prikazan na sliki 2, urejeni par pa na sliki 4. Vidimo, da ima urejeni graf preprostejšo avtomatno strukturo.

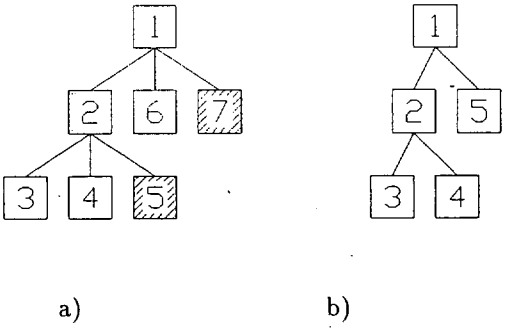
Opisani algoritem urejenega prekrivanja je dokaj enostaven, vendar vselej ne zagotavlja optimalnih rezultatov. Kljub temu menimo, da je dovolj uporaben za procesorje z omejenim številom zunanjih povezav pri postavki, da nastopa v paralelnih vozliščih HADF grafov praviloma več vej, kot je možnih povezav na procesnem elementu. Algoritem ima v primerjavi z iterativnimi algoritmi tudi manjšo kompleksnost in je zato cenejši.

Na sliki 5 si lahko ogledamo preprost primer grafa, pri katerem z opisanim postopkom urejanja ne dobimo idealne avtomatne strukture. Ugotovimo lahko, da bi bila ob zamenjavi paralelnih vej III in IV avtomatna struktura manjša za en procesor.

Funkcije posameznih procesorjev prikazuje slika 6. Iz slike je razvidno, da opravljajo vozliščni procesorji le funkcijo vmesnika in stikala, in so kot taki slabo izkoriščeni. Izkoriščenost le-teh povečamo, če enega izmed procesov (če obstaja) zadržimo v paralelnem vozlišču in ga izvajamo na vozliščnem procesorju. Takšen primer za že prej obravnavani HADF graf si oglejmo



Slika 5: Primer, kjer algoritem ni optimalen



a) b)

Slika 6: Krčenje avtomatne strukture

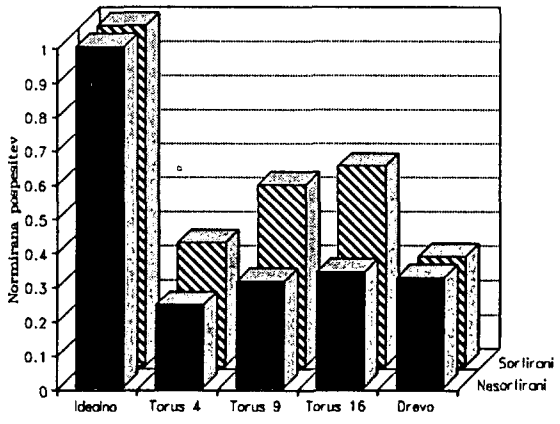
na sliki 6a. Procesorja 1 in 2 opravljata le funkcijo razvejišča. Ker sta neizkoriščena, smemo nanju preslikati procesa, ki se sicer izvajata na procesorjih 5 in 7. Rezultat je dodatno skrčenje avtomatne strukture, ki je v tem primeru manjša za 2 procesorja (slika 6b). Pridobljena avtomatna struktura še vedno ne upošteva časovnih značilnosti posameznih procesov in zadošča splošnemu primeru grafa dane oblike.

4. Izvajanje grafov

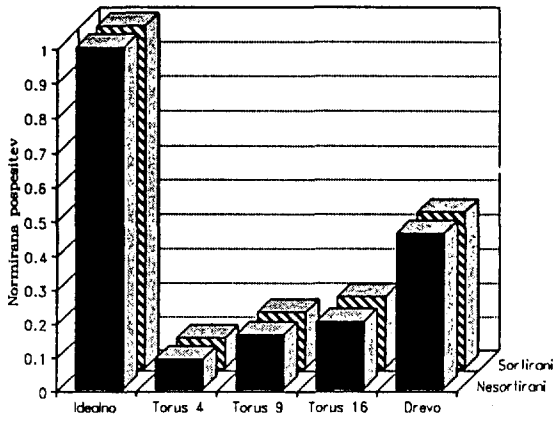
Regularna (drevesna) avtomatna struktura in neposredna preslikava HADF grafa nanjo, nam omogočata analitično oceno časa izvajanja. Ocena, ki smo jo napravili, je poenostavljena in daje boljše rezultate, kot bi jih dobili na resničnem sistemu, saj ne upošteva zmanjšanja moči procesiranja zaradi prenosov. Prispevek te napake ocenjujemo kot majhen, ker je intenzivnost prometa nizka (samo lokalni prenosi). Graf se bločno (koračno) širi po drevesni strukturi navzdol tako, da se od višjega na nižji nivo najprej prenese celotno poddrevo; delitev grafa na nižje nivoje se nadaljuje šele v naslednjem koraku. Avtomatna struktura mora biti prilagojena obravnavanemu grafu, zato pri preslikavi ne pričakujemo mrtvih točk (blokad).

5. Rezultati

Čas izvajanja. Pri opazovanju časov izvajanja posameznih grafov opazimo, da se lahko z obravnavanimi transformacijami drevesne strukture čas izvajanja HADF grafa izboljša v primerjavi s tistim časom, ki smo ga dosegli na toroidnem polju [2, 4]. Dosežena normirana povprečna pospešitev je prikazana na sliki 7. Sortirani grafi so prilagojeni za uspešnejše izvajanje na toroidnem polju [1] in ne vplivajo na potek izvajanja na drevesni strukturi avtomata. Eksperimentalno smo ugotovili, da so rezultati na drevesu boljši pri grafih, ki imajo močno poudarjen faktor širine. Pospešitev za primer takšnega grafa prikazuje slika 8. Drevesna avtomatna struktura je manj uspešna pri globokih grafih (slika 9). Največjo prednost drevesne strukture opazimo pri izvajanju popolnoma simetričnih grafov (slika 10). Grafi te vrste so simetrični tako po zgradbi, kot tudi po času prenosov in izvajanj posameznih procesov.

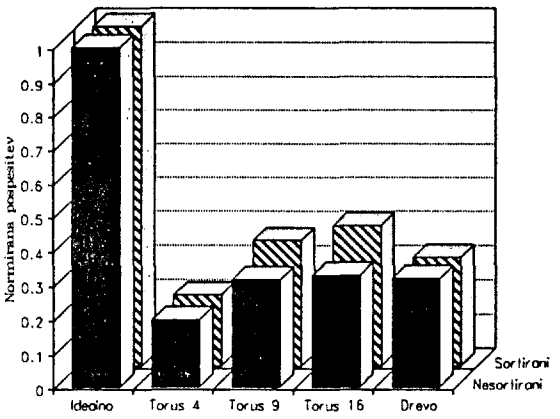


Slika 7: Primerjava drevesne in toroidne strukture

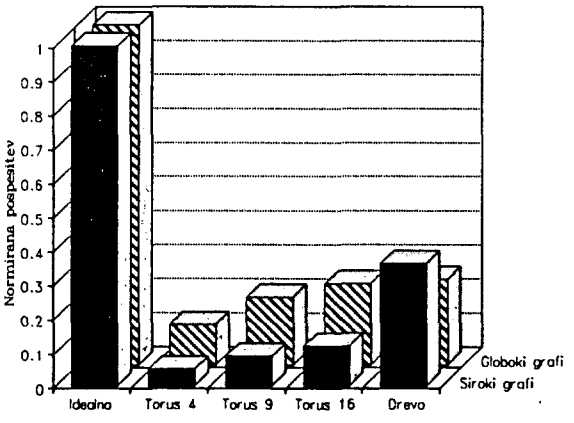


Slika 8: Primerjava širokih grafov

Količnik prenosnega časa. Vrednost potrebnega količnika prenosnega časa se pri drevesni strukturi giblje okoli vrednosti 20, kar je več, kot je veljalo za toroidno polje. Razmerje T_{izv}/T_p smo kljub temu ohranili na vrednosti 10, da bi bili rezultati, ki smo jih dobili za



Slika 9: Primerjava globokih grafov



Slika 10: Primerjava popolnoma simetričnih grafov

drevo primerljivi z rezultati, ki smo jih dobili za toroidni sistem. Ocenjujemo, da je povečanje količnika prenosnega časa posledica odsotnosti interakcije prenosov sporočil med procesorji. Značilnost drevesne avtomatne strukture je namreč lokalen prenos sporočil. Prenos se izvaja le med dvema procesorjema. Istočasne zahteve za prenos po isti poti, kot so možne pri torusu, pri drevesni strukturi niso možne.

6. Zaključek

Drevesna oblika HADF grafov nam je pomenila izziv za raziskavo, ki naj bi dala odgovor na vprašanje o upravičenosti izvajanja algoritmov, ki so predstavljeni v obliki HADFG na drevesni avtomatni strukturi. Rezultate raziskave smo primerjali z rezultati raziskave toroidnega polja [2, 4]. Ugotovili smo, da je drevesna struktura glede na čas izvajanja primernejša za razreda popolnoma simetričnih in širokih grafov.

Drevesna struktura je neprilagodljiva. Meje, ki jih predstavljajo terminalni procesorji, t.j. takšni procesorji, ki nimajo naslednikov, pomenijo meje širjenja grafa. Posledica je, da mora biti za neblokirano preslikavo grafa avtomatna struktura dovolj razvejana. Avtomat je tako lahko bodisi splošen (predimenzioniran), bodisi takšen, da ustreza družini izbranih HADF grafov

ali nekemu določenemu grafu. Splošna oblika avtomata je drevo zadostne širine in globine, da se lahko na njega preslika vsak želeni HADF graf. Povsem jasno je, da je takšen avtomat občutno predimenzioniran. Bolj upravičena je konstrukcija avtomata, ki pokriva potrebe določene izbrane družine HADF grafov. Avtomat bo najverjetneje za večino grafov še vedno predimenzioniran, vendar v znatno manjši meri. Namenski avtomat za nek določen graf je cenovno najboljša rešitev saj potrebuje najmanjše število procesorjev. Ozka omejenost na en sam tip grafa pa je cena, ki jo moramo plačati za takšno rešitev.

Problemi, na katere naletimo ob povezovanju procesorjev v drevesne strukture, so predvsem tehnične narave in se jim lahko do določene mere izognemo ob uporabi rekonfigurabilnega polja procesorjev. Statična polja so manj uporabna in bi bila upavičena kvečjemu za neprestano izvajanje istega grafa ali iste družine grafov.

Naslednji problem predstavlja stopnja povezljivosti procesnih elementov, ki je tehnološko omejena, medtem ko je število možnih naslednikov vozlišča v grafu neomejeno. Transputerji imajo npr. štiri povezave, kar ob enem predhodniku pogojuje le tri naslednike. Dinamično rekonfigurabilna polja omogočajo navidezno poljubno število naslednikov, vendar so tudi ta omejena z zmogljivostjo matričnih stikal.

Iz obravnavanega lahko zaključimo naslednje :

Toroidni sistem je najprimernejši za grafe, ki so nesimetrični po obliki in v katerih imajo procesi različne čase izvajanja. Dekompozicija grafa po korenskih procesorjih (glej[2]) praviloma zagotavlja krajši skupni čas prenosov in izvajanja ter občutno manjše zahtevano število procesorjev, kot drevesna struktura. Toroid si predstavljamo kot vase zapognjeno drevo, ki se navidezno nikoli ne konča. Vsi procesorji prvega procesorskega nivoja[1] predstavljajo primarne, procesorji drugega nivoja pa sekundarne potencialne naslednike. Tu lahko potegnemo vzporednice s kompleksnejšo rekonfigurabilno strukturo. Navidezna neskončnost strukture pomeni hkrati tudi večjo univerzalnost.

Čeprav je število navideznih naslednikov procesorja veliko, lahko sočasno dostopamo le do dveh. Pri prenosu sporočil ima namreč vsak procesor v torusu le dva logična naslednika, kar omejuje komunikacijske zmožnosti. Kot vemo, se sporočila prenašajo krožno po zankah torusa. Ker prenos ni neposreden, so pri svojem delu dodatno moteni tako vsi posredniki, kot tudi prenosne poti. Posledica je, da takšen sistem zelo dobro deluje z nesimetričnimi HADF grafi, kjer so prenosi razdeljeni neenakomerno. Izvajanje širokih in popolnoma simetričnih grafov je manj uspešno, ker slednji zahtevajo za učinkovito delovanje simetrično in sočasno razporejanje.

Predstavljena transformacija HADF grafa v avtomatno strukturo je analitična in temelji na predpostavki, da imajo procesorji neomejene zmožnosti povezovanja, in je kot taka idealizirana. Transformacija v splošnem ne zagotavlja idealnih rezultatov, vendar menimo, da je glede na nizko stopnjo kompleksnosti dovolj uporabna. Nadalje menimo, da bo potrebno večino HADF grafov ustrezno preoblikovati še predno skonstruiramo avtomatno strukturo in jih prilagoditi stopnji povezljivosti procesorjev. Preoblikovanje izvedemo tako, da odvečne paralelne veje, t.j. tiste, ki prekoračijo število razpoložljivih povezav, potisnemo na nižji nivo v grafu. Tako preoblikovan graf bo predvidoma na vseh višjih nivojih avtomatne strukture popolnoma povezan, zato se situacije, kot je podana na sliki 5, ne bodo pojavljale.

Idealizirane drevesne strukture so se pokazale kot primerne za popolnoma simetrične in široke HADF grafe. Takšni grafi so značilni za rekurzivne algoritme [5]. Naše nadalnje delo bo usmerjeno v raziskavo statičnih drevesnih struktur z realnimi procesorji, z možnostjo preslikav algoritma nanje in uspešnosti takšnih struktur. Pričakujemo, da se bodo rezultati po času izvajanja približali rezultatom, ki jih dosegamo na toroidnem polju, pri čemer dopuščamo slabšo učinkovitost drevesne strukture.

Literatura

[1] P.Kolbezen, P.Zaveršek, A hierarchical multimicroprocessor system, Informatica, A Journal of Computing and Informatics, The Slovene Society Informatika, Vol.15, Nr.1, Feb. 1991, 65-76.

[2] P.Zaveršek, P.Kolbezen, Process allocation in the multitransputer network, Informatica, A Journal of Computing and Informatics, The Slovene Society Informatika, Vol.15, Nr.4, Nov. 1991, 43-52.

[3] M.Szturmowicz, M.Tudruj, A multi-layer transputer network for efficient execution of OCCAM programs, North-Holland, Microprocessing and Microprogramming, Vol.28 (1989), 133-138.

[4] P.Zaveršek, P.Kolbezen, Dynamic allocation on the transputer network, Proceedings of the CONPAR 92-VAPP V, Lyon, France, 1992 (in print).

[5] B.Buchberger, J.Fegerl, F.Lichtenberger, Computer trees: a concept for parallel processing, IPC Business Press, Microprocessors and Microsystems, Vol.3 No.6, 1979, 244-248.

[6] Parsytec Transputer Modules, ParsyTec, Gesellschaft für Parallele Systemtechnik mbH.

PREDVIDLJIVO DINAMIČNI PRINCIP RAZVRŠČANJA OPRAVIL V REALNEM ČASU

INFORMATICA 3/92

Ključne besede: strogi namenski sistem, jedro OS za delo v realnem času, princip razvrščanja, časovne in logične prioritete opravil, idealna razvrstitev opravil, zaščita skupnih virov

Barbara Koroušić, Peter Kolbezen
Laboratorij za računalniške arhitekture
Inštitut Jožef Stefan, Ljubljana

Vloga operacijskih sistemov za delo v realnem času (OS) postaja pri namenskih računalniških sistemih vse pomembnejša. Programer se lahko posveti razvoju programske opreme na abstraktnem nivoju in prepusti odgovornost za upravljanje z opravili aplikacije sistemskim procesom jedra OS. V članku opisujemo takšen proces razvrščanja opravil, ki se istočasno potegujejo za vire računalnika. Podan je opis in analiza dinamičnega principa razvrščanja, ki dodeljuje prednost tistim opravilom, ki so za aplikacijo pomembnejša. Hkrati upošteva tudi časovno omejenost opravil in zagotavlja njihovo pravočasno izvršitev. Princip smo poimenovali *predvidljivo dinamični*, ker predvideva zavrnitev opravil takoj ob prihodu zahtev za njihovo izvajanje. O zavrnitvi govorimo, ko nabor pripravljenih opravil na procesorsko izvajanje ne omogoča pravočasne izvršitve opravila. Zavrnjeno opravilo lahko preusmerimo v drugo procesorsko vozlišče, če je namenski računalnik porazdeljeni ali večprocesorski sistem. Pomembna je čimprejšnja zavrnitev, tako da ima opravilo še dovolj časa za izvršitev vseh potrebnih operacij.

PREDICTABLE DYNAMIC TASK SCHEDULING IN HARD REAL-TIME - The essential feature of hard real-time systems is that the embedded computer must deliver its results correctly and on-time - otherwise malfunction may result. To assure software constructions of such systems a support of a real-time operating system is needed. We present a task scheduling and mutual exclusion protocol as one of real-time operating system structuring mechanisms. The protocol is applicable to aperiodic tasks having specific priority and timing characteristics.

1. Uvod

Računalniški sistem, ki je prilagojen okolju, v katerem izvaja nadzor, imenujemo *namenski* ali *vloženi* (*embedded*) sistem. Njegova osnovna naloga je spremljanje zunanjih dogodkov in odločanje o nadaljnem dogajanju v okolju. Ker je spreminjanje stanja v okolju dinamično in pogosto tudi nepredvidljivo, mora namenski računalnik delovati pod pogoji realnega časa:

1. Odločitve morajo biti **logično** in **časovno** pravilne.
2. Sistem mora delovati **varno** tudi ob pojavitvi morebitnih napak.

Namenske računalnike delimo glede na stopnjo upoštevanja zahtev realnega časa na *stroge* (*hard*) in *mehke* (*soft*) sisteme [1].

Potniško letalo A-320 [2] je primer sistema, ki je pod nadzorom *strogega* namenskega računalnika. Po obsegu okolja, nad katerim ima računalnik nadzor, so verjetno največji namenski sistemi v elektrarnah, kemijsko predelovalni industriji ali v upravljanju prometa. Manjši namenski sistemi so mobilni roboti, ki jih uporabljajo za opravljanje težaških, monotonih ali nevarnih opravil [3].

Razvoj programske opreme namenskih sistemov je bil do nedavnega prepuščen izkušenim programerjem, ki so imeli bogato znanje o strojni opremi sistema. Dandanes pa poznamo že vrsto

- operacijskih sistemov (npr. VRTX32, FlexOS [4]),
- visokonivojskih programskih jezikov (npr. Ada [5], Modula-2 [6], Real-Time Euclid [7]),
- CASE (*Computer Aided Software Engineering*) orodij (npr. VRTXdesigner [8]),

ki so prilagojeni zahtevam realnega časa. Uporaba tovrstnih orodij poenostavi oblikovanje namenskih

sistemov in hkrati zagotavlja večjo učinkovitost razvite kode.

Uporaba CASE orodij vodi oblikovanje programske opreme iz definicije problema. Podatkovne in nadzorne strukture visokonivojskih jezikov za programiranje pod pogoji realnega časa omogočajo razbitje problema na abstraktnem nivoju. Sistemske funkcije jedra OS (*real-time kernel*) pa prevzamejo odgovornost za upravljanje z opravili, ki predstavljajo osnovne abstraktne enote programske aplikacije.

Eden najpomembnejših sistemskih procesov jedra je *razvrščanje* sočasnih opravil aplikacije računalniškega sistema. Dogodki v okolju namreč lahko zahtevajo sočasen odziv sistema, ne glede na vrstni red prihoda zahtev za izvajanje opravil. Naloga razvrščevalnika je spremljanje dogodkov v okolju in odločanje o vrstnem redu izvajanja prebujenih opravil, saj večina namenskih računalnikov ne omogoča istočasnega izvajanja vseh zahtevanih opravil.

Princip razvrščanja (*scheduling policy*) je pogojen z aplikacijo. V preprostih aplikacijah, kjer je celotno dogajanje v sistemu predvidljivo, so zelo učinkoviti *statični* principi, ki določajo vrstni red izvajanja opravil z upoštevanjem statičnih (nespremenljivih) lastnosti opravil. Večina namenskih aplikacij pa mora upoštevati dinamično spreminjanje lastnosti opravil. Takoimenovani *dinamični* principi razvrščanja določajo zaporedje izvajanja prebujenih opravil med samim izvajanjem aplikacije.

Razvrščevalnik lahko teži k *idealnem* (*feasible*) ali *optimalnem* razvrščanju opravil. V prvem primeru je razvrstitev pripravljenih opravil čakajočih na procesor določena tako, da nobeno opravilo ne more preseči vnaprej določene časovne omejitve (*deadline*). Pri optimalni razvrstitvi pa so dovoljene minimalne zakasnitve.

V nadaljevanju bomo predstavili *predvidljivo dinamični* princip razvrščanja, ki določa prioritete prebujenih opravil z upoštevanjem njihove **časovne omejenosti** ter **stopnje pomembnosti** za celotno aplikacijo. Časovna omejenost in pomembnost opravil sta namreč kriterija, ki se lahko medsebojno izključujeta. Opravilo, ki je časovno kritično, ima lahko nizko stopnjo pomembnosti za aplikacijo in obratno. Princip teži k *idealnem* razvrščanju opravil.

2. Opis PD principa razvrščanja opravil

PD princip razvrščanja opravil temelji na sledečih predpostavkah:

1. Stanje namenskega sistema se lahko spreminja tudi asinhrono.
2. Odziv na dogodke je časovno omejen.
3. Opravila aplikacije so različno pomembna za delovanje sistema.
4. Dovoljeno je prekinjanje izvajanja opravil.
5. Dostop do skupnih virov je zaščiten.
6. Izvajanje namenske aplikacije mora biti robustno.

1. Dogodki, ki sprožijo izvajanje opravil v računalniškem sistemu, se pojavljajo sinhrono ali asinhrono. Trenutek, v katerem sproži asinhroni dogodek izvajanje opravila, ni znan pred zagonom aplikacije. Asinhroni dogodki lahko sprožijo izvajanje periodičnih ali aperiodičnih opravil.

2. Dogodki v sistemu zahtevajo pravočasen odziv. Vsako opravilo je časovno omejeno. Če računalniški sistem ne more zagotoviti odziva v predvidenem (odzivnem) času, je opravilo zavrnjeno.

3. Prednost pri izvajanju imajo opravila, ki so bolj pomembna za aplikacijo. Če procesor ni zmožen pravočasno izvesti vseh prebujenih opravil, razvrščevalnik zavrne najprej opravila z najmanjšo stopnjo pomembnosti.

4. Ker so dogodki večinoma asinhroni in opravila časovno omejena, mora princip podpirati prekinjanje opravil. Časovne lastnosti opravil se dinamično spreminjajo in tako se tudi prednost opravil pri dostopu do procesorja spreminja s časom izvajanja aplikacije. Prekinjanje izvajanja opravil je dovoljeno, če je prioriteta prebujenega opravila višja od prioritete opravila, ki se trenutno izvaja.

5. Opravila lahko zahtevajo dostop do skupnih virov. Metoda medsebojnega izključevanja (*mutual exclusion*) z uporabo *monitorja*, zagotavlja zaščito skupnih virov [9]. Ker pa smo želeli zadostiti tudi zahtevi po možnosti prekinjanja opravil, smo princip monitorja razširili in ga priredili za uporabo v

strogih namenskih sistemih. Podrobnejši opis sledi v poglavju 2.1.

6. Robustnost izvajanja namenskih aplikacij smo želeli zagotoviti posredno s principom razvrščanja. Opravilo, ki poruši *idealno* razvrstitev prebujenih opravil, je zavrnjeno že ob prihodu dogodka, ki je sprožil izvajanje opravila. Če je namenski sistem večprocesorski, se lahko opravilo preusmeri v drugo procesorsko vozlišče. Zato je zelo pomembno, da je zavrnitev objavljena, ko ima opravilo še dovolj časa, da opravi vse operacije. Upoštevali smo tudi morebitne napake pri izračunu operacij opravil. Če ima opravilo še dovolj časa, lahko ob pojavitvi napake ponovi niz operacij z alternativnimi procedurami (razširjeni mehanizem *okrevanja po napaki (recovery block programming)* [1]).

Rezultati matematične in simulacijske analize, ki sta jih objavila Craig in Woodside [10], kažejo na učinkovitost dinamičnega *principa prednosti časovno kritičnih opravil* (*Earliest Deadline As Soon As Possible*) [1]. Princip dodeljuje prednost opravilom, ki so časovno bolj kritična. Ker princip ne upošteva pomembnosti opravil za aplikacijo, smo vgradili dodaten mehanizem „izločanja" manj pomembnih opravil v primeru nasičenja v vrsti pripravljenih opravil (*ready queue*). To je programska struktura, ki hrani zahteve za izvajanje prebujenih opravil v vrstnem redu, kot ga določa razvrščevalnik.

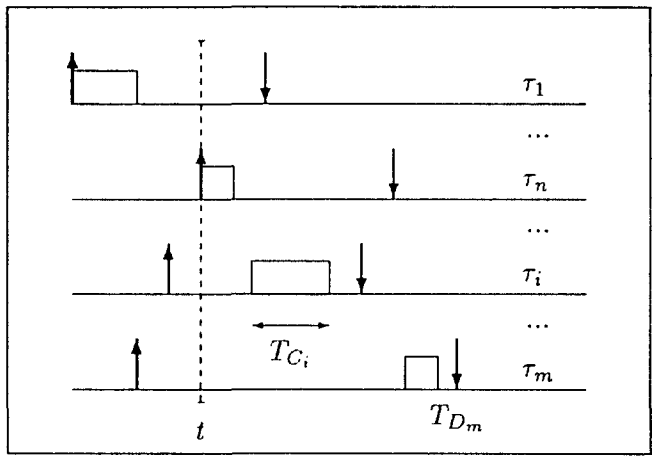
Ob prihodu nove zahteve za izvajanje opravila τ_n razvrščevalnik preveri pogoj:

$$(T_{D_k} - \sum_{i=1}^{k} T_{C_i}) \geq t,\ n \leq k \leq m, \qquad (1)$$

ki zagotavlja *idealno* razvrstitev nabora m prebujenih aperiodičnih opravil. T_{D_k} predstavlja čas, do katerega se mora izvršiti opravilo τ_k ter T_{C_i} predvideni čas izvajanja opravila τ_i. Indeksi opravil so določeni z njihovo časovno kritičnostjo, $i < j$ pri pogoju $T_{D_i} \leq T_{D_j}$. t je čas prihoda zahteve za izvajanje novega opravila τ_n (**slika 1**).

Razvrščevalnik preveri pogoj (1) le za vrednosti indeksa $k \geq n$, saj je pri nižjih vrednostih indeksa zanesljivo izpolnjen.

Če je pogoj izpolnjen, poteka razvrščanje opravil po *principu prednosti časovno kritičnih opravil.*



Slika 1: Časovne lastnosti in urejenost opravil

Prednost pri dostopu do procesorja ima opravilo, ki je časovno najbolj omejeno. V nasprotnem primeru pa se vključi mehanizem za "izločanje„ manj pomembnih opravil.

Recimo, da pogoj ni izpolnjen pri vrednosti indeksa $k = j$, $n \leq j \leq m$. Potem razvrščevalnik izloči iz vrste opravilo, katerega stopnja pomembnosti je najmanjša v podmnožici prebujenih opravil $\{\tau_1, \tau_2, ...\tau_j\}$ ter ponovno preveri pogoj pri vrednosti indeksa $k \geq j$.

Biyabani in Stankovic [11] sta dokazala, da je princip izločanja najmanj pomembnih opravil najbolj učinkovit. Druga možnost je izločanje opravil, katerih časovna prioriteta je najmanjša in hkrati stopnja pomembnosti ne preseže stopnje pomembnosti opravila τ_n. V primerjavi s prvo možnostjo, ki smo jo označili kot najbolj učinkovito, je slednja časovno manj potratna. Postopek izločanja opravil se ponavlja, dokler pogoju (1) ni zadoščeno.

Ker so predvideni časi izvajanja opravil določeni za „najslabši" (najdaljši) primer, razvrščevalnik dejansko ne izloči opravil iz vrste prebujenih opravil, temveč jih prestavi na konec vrste in obvesti sistem o predvideni zakasnitvi odziva na dogodke, ki so sprožili zahteve za izvajanje teh opravil. Sistem lahko zahteva dejansko izločitev teh opravil iz vrste pripravljenih opravil. Če je dejanski čas izvajanja opravila krajši od predvidenega, razvrščevalnik ponovno preveri pogoj (1) po zaključenem izvajanju opravila.

Če je pogoju (1) zadoščeno, je zagotovljeno *idealno* razvrščanje opravil, četudi je procesor 100 %

zaseden:

$$U = \frac{\sum_{i=1}^{m} T_{C_i}}{T_{D_m}} \leq 1 - \frac{t}{T_{D_m}}, \qquad (2)$$

saj je časovna meja T_{D_m} zmeraj manjša od časa t in velja:

$$\lim_{m \to \infty} 1 - \frac{t}{T_{D_m}} = 1. \qquad (3)$$

2.1 Mehanizem za zaščito skupnih virov

Pri opisu zahtev smo omenili, da monitor, kot mehanizem za zaščito skupnih virov, ne zagotavlja pravočasnega izvajanja namenskih aplikacij v realnem času, ker ne dopušča prekinjanja izvajanja opravil v kritičnih področjih [12]. Izogniti se moramo tudi težavam, ki nastopijo ob blokiranju (časovno in logično) pomembnih opravil, zaradi zasedenosti kritičnih področij z manj pomembnimi opravili:

- verižno blokiranje (*chained blocking*) izvajanja opravil,
- problem mrtve zanke (*deadlock*).

Opravilo je blokirano, kadar čaka na nadaljevanje izvajanja operacij zaradi zasedenosti zahtevanega vira z manj pomembnim opravilom. Težave nastopijo, ko opravilo zahteva vstop v več kritičnih področij, ki so zasedene z manj pomembnimi opravili (verižno blokiranje). Kot primer podajmo situacijo, ko opravilo τ_C vstopi v kritično področje. Opravilo τ_B, ki je bolj pomembno, prekine izvajanje opravila τ_C ter vstopi v drugo kritično področje. Opravilo τ_A, ki je še bolj pomembno, prekine izvajanje opravila τ_B. Če zahteva opravilo τ_A vstop v obe kritični področji, se njegovo izvajanje blokira, ker sta področji zasedeni z opravili τ_C in τ_B.

Problemu mrtve zanke pri potegovanju za skupni vir se lahko izognemo z uporabo semaforjev [9]. Mrtva zanka pa se lahko pojavi tudi po vstopu opravil v kritična področja, ko opravila navzkrižno zahtevajo dostop do skupnih virov, ki so že zasedeni.

Princip monitorja temelji na uporabi semaforjev za zaščito kritičnih področij in signalov za sporazumevanje z opravili, ki zahtevajo uporabo skupnih virov [9]. Princip razširimo z dodatnim mehanizmom, ki omogoča varno prekinjanje izvajanja opravil v kritičnih področjih [12]. Poleg osnovnih lastnosti opravil, ki jih mora določiti programer pred zagonom aplikacije, zahteva mehanizem seznam semaforjev, katere mora opravilo upoštevati pred vstopom v kritična področja. Vsakemu kritičnemu področju določimo prioriteto, ki se med izvajanjem aplikacije dinamično spreminja. Prioriteta kritičnega področja je enaka najvišji prioriteti prebujenega opravila, ki lahko zahteva vstop v to kritično področje.

Mehanizem za varno prekinjanje izvajanja opravil v kritičnih področjih razdelimo v dva dela. Prvi del omogoča dinamično določanje prioritet opravil v kritičnih področjih. Opravilo, ki ima trenutno dostop do skupnega vira, lahko blokira izvajanje višje prioritetnih opravil. V takšnem primeru prevzame opravilo v kritičnem področju najvišjo prioriteto blokiranih opravil. Ko izstopi iz kritičnega področja, dobi ponovno "originalno„ prioriteto. Drugi del upravlja s kritičnimi področji. Opravilo, ki zahteva ključ za vstop v kritično področje, mora ustrezati pogoju, da je njegova prioriteta višja od vseh prioritet trenutno zasedenih kritičnih področij. Sicer razvrščevalnik blokira izvajanje opravila in čaka na vstop v zasedeno področje.

Opišimo dva možna načina blokiranja opravil, ki zahtevajo dostop do skupnih virov:

1. Opravilo τ zahteva ključ za vstop v kritično področje S, vendar ga ne dobi, ker je njegova prioriteta nižja ali enaka najvišji prioriteti zasedenih področij, S_H. Opravilo τ je blokirano, dokler se področje S_H ne sprosti, opravilo v S_H pa prevzame prioriteto opravila τ, če je njegova prioriteta nižja.
2. Opravilo τ je blokirano, ker zahteva vstop v področje S in ima prioriteto nižjo ali enako najvišji prioriteti zasedenih področij, S_H. Če je opravilo v področju S_H že prevzelo višjo prioriteto od predhodnje blokiranega opravila, potem ostane njegova prioriteta nespremenjena.

Dokaz, da mehanizem onemogoča verižno blokiranje izvajanja opravil, temelji na sledečih lemah [12].

LEMA 1: *Če opravilo τ blokira izvajanje višje prioritetnega opravila τ_h, potem τ zaseda vsaj eno kritično področje S s prioriteto višjo ali enako prioriteti τ_h v času prihoda opravila τ_h (t).*

Dokaz: Predpostavimo, da je najvišja prioriteta kritičnih področij, ki jih zaseda τ v času t, prioriteta področja S_H. Če je ta prioriteta nižja od prioritete opravila τ_h, potem lahko opravilo τ prevzame le prioriteto, nižjo od prioritete opravila τ_h. Drugače rečeno, dokler je opravilo τ_h aktivno (prebujeno), opravilo τ ne bo razvrščeno tako, da bi blokiralo τ_h. Torej obstaja vsaj eno področje, ki ga zaseda τ, s prioriteto večjo ali enako prioriteti τ_h, v času t ♠.

LEMA 2: *V vsakem trenutku t obstaja največ eno kritično področje s prioriteto višjo ali enako P, ki ga zaseda opravilo z „originalno" prioriteto nižjo od P.*

Dokaz: Predpostavimo, da lema ne velja. V času t sta področji S_i in S_j zasedeni in imata obe prioriteto višjo ali enako P. Dodatno predpostavimo, da je najprej opravilo τ_p zasedlo področje S_i v času t_p in nato opravilo τ_q področje S_j v času t_q. Prioriteti opravil τ_p in τ_q sta nižji od P. Upoštevajoč princip našega mehanizma mora biti prioriteta opravila τ_q višja od prioritete kateregakoli zasedenega področja v času t_q. Ker pa zahtevamo, da je prioriteta opravila τ_q zmeraj nižja od P in prioriteta področja S_i višja ali enaka P, pridemo v nasprotje ♠.

S pomočjo lem smo dokazali, da lahko opravilo τ blokirajo le opravila z nižjimi prioritetami, ki zasedajo področja, katerih prioritete so višje ali enake prioriteti opravila τ (**LEMA 1**). V času prihoda opravila τ je zasedeno največ eno kritično področje s prioriteto višjo od prioritete τ, ki ga zaseda opravilo z nižjo prioriteto (**LEMA 2**). Tako smo dokazali, da lahko opravilo τ blokira največ eno opravilo z nižjo prioriteto in pogoj za verižno blokiranje nikoli ni izpolnjen.

Pri izboru mehanizma za varno prekinjanje izvajanja opravil v kritičnih področjih, smo se želeli izogniti tudi mrtvim zankam. Pogoj za mrtvo zanko je navzkrižno čakanje opravil, ki zasedajo kritična področja, medtem ko čakajo na semaforje področij, ki jih zasedajo ostala opravila, sodelujoča v zanki.

Predpostavimo, da obstaja mrtva zanka in ima opravilo τ najvišjo prioriteto med opravili v zanki. τ zaseda področje medtem, ko čaka na semafor področja, ki je zasedeno z drugim (nižje prioritetnim) opravilom iz zanke. Dokažemo pa lahko, da opravilo, ki zaseda področje, ne more biti blokirano z nobenim nižje prioritetnim opravilom [12] (**LEMA 3**). Tako pogoj za pojavitev mrtve zanke ne more biti izpolnjen.

LEMA 3: *Opravilo je lahko blokirano le pred vstopom v prvo kritično področje.*

Dokaz: Ko opravilo τ vstopi v prvo kritično področje v času t, ima najvišjo prioriteto med prebujenimi opravili. Ostala zasedena področja, ki jih zasedajo nižje prioritetna opravila v času t, označimo z množico S. Prioriteta opravila τ je zagotovo višja od prioritet področij iz S. Predpostavimo, da bo izvajanje opravila τ blokirano v času t_1, $t_1 > t$, ko se bo začelo izvajati opravilo z nižjo prioriteto τ_l. Opravilo τ_l bo začasno prevzelo prioriteto od opravila τ. Upoštevajoč princip mehanizma lahko opravilo τ_l zaklene področje S_k, če je prioriteta opravila τ nižja ali enaka prioriteti S_k. Ker nobeno opravilo ni bilo razvrščeno v času med t in t_1, je množica zasedenih področij v času t_1 enaka S in $S_k \in S$. Tako pridemo v nasprotje, saj nobeno področje iz S nima prioritete višje ali enake od prioritete opravila τ ♠.

Razvrščevalnik mora preveriti pred vstopom opravila v kritično področje, ali je njegova prioriteta višja od najvišje prioritete zasedenih področij. Ker pa smo v zgornji lemi dokazali, da je opravilo lahko blokirano le pred vstopom v prvo kritično področje, moramo preveriti pogoj le pred vstopom v prvo področje. Nadaljne zahteve za dostop do skupnih virov izvaja razvrščevalnik brez preverjanja pogoja, ali je prioriteta opravila višja od najvišje prioritete zasedenih področij.

Predstavljeni mehanizem za zaščito skupnih virov opravil vpliva na potek izvajanja opravil. Blokiranje izvajanja opravil zaradi zasedenosti kritičnih področij vpliva na odzivni čas, kar moramo upoštevati pri preverjanju pogoja o idealni razvrstitvi prebujenih opravil. V pogoj (1) vpeljemo dodatni parameter B_k:

$$(T_{D_k} - (\sum_{i=1}^{k} T_{C_i} + B_k)) \geq t,\ 1 \leq k \leq m, \qquad (4)$$

ki predstavlja najdaljši čas blokiranja izvajanja opravila τ_k. Za vsako opravilo τ_k določimo množico Z_k, ki vsebuje seznam kritičnih področij, do katerih imajo dostop nižje prioritetna opravila in je njihova prioriteta višja ali enaka prioriteti opravila τ_k. Drugače povedano, Z_k vsebuje vsa področja, ki jih lahko zasedejo nižje prioritetna opravila v trenutku zahteve opravila τ_k ali opravila z višjo prioriteto za vstop v ta področja. Seznami Z_k, $k = 1, ..., m$ se spreminjajo s časom. Ker lahko opravila blokirajo opravila z nižjo prioriteto, moramo pogoj preveriti pri vseh vrednostih indeksa k. Pogoj zagotavlja *idealno* razvrščanje nabora prebujenih opravil.

PRIMER: Oglejmo si primer *idealnega* razvrščanja treh aperiodičnih opravil z lastnostmi:

opravilo	T_A	T_C	T_D	*kritičnost*	*semaforji*
τ_1	3	2	6	2	$[S_1, S_2]$
τ_2	1	2.5	7	1	$[S_1, S_3]$
τ_3	0	3	8	3	$[S_2, S_3]$

Tabela 1: Lastnosti treh aperiodičnih opravil

kjer parameter T_A predstavlja čas prihoda zahteve za izvajanje opravila, T_C preostali izvajalni čas opravila in T_D trenutek, do katerega se mora opravilo zagotovo izvršiti, sicer je zavrnjeno.

Razvrščevalnik ureja prebujena opravila v vrsti pripravljenih opravil RQ po PD principu.

V času $t = 0$, ob prihodu zahteve za izvajanje opravila τ_3, je vsebina vrste $RQ = [\tau_3]$. Opravilo se takoj prične izvajati, saj je procesor prost.

Ob prihodu nove zahteve za izvajanje opravila τ_2, v času $t = 1$, vsebuje vrsta $RQ = [\tau_2, \tau_3]$, tako da ima opravilo τ_2 prioriteto $p(\tau_2) = 1$ in $p(\tau_3) = 2$. Opravilo $p(\tau_2)$ ima višjo prioriteto od $p(\tau_3)$, ker je časovno bolj omejeno. Prioritete kritičnih področij, v katera imata opravili vstop med izvajanjem, so sledeče: $c(S_1) = c(S_3) = p(\tau_2) = 1$ in $c(S_2) = p(\tau_3) = 2$. Pogoj (4) preverimo najprej za prvo opravilo v vrsti RQ (τ_2):

$$T_{D_1} - (T_{C_1} + B_1) = 7 - (2.5 + 0.5) > 1,$$

pri čemer sta T_{D_1} in T_{C_1} časovna parametra opravila τ_2. Najdaljši možni čas blokiranja opravila τ_2 $B_1 = 0.5$. To je čas zasedenosti področja S_3, do katerega ima dostop tudi nižje prioritetno opravilo τ_3. Ker je pogoju zadoščeno, preverimo pogoj pri vrednosti indeksa $k = 2$:

$$T_{D_2} - (\sum_{i=1}^{2} T_{C_i} + B_2) = 8 - (2.5 + 2 + 0) > 1.$$

Vrednost indeksa 2 označuje lastnosti opravila τ_3 in 1 lastnosti τ_2. Opravilo τ_3 ima nižjo prioriteto, zato njegovega izvajanja ne more blokirati τ_2 ($B_1 = 0$).

V času $t = 3$ se pojavi nova zahteva za izvajanje opravila τ_1. $RQ = [\tau_1, \tau_2, \tau_3]$, $p(\tau_1) = 1$, $p(\tau_2) = 2$ in $p(\tau_3) = 3$, $c(S_1) = c(S_2) = p(\tau_1) = 1$ in $c(S_3) = p(\tau_2) = 2$. Pogoju (4) ni zadoščeno pri vrednosti indeksa $k = 3$:

$$T_{D_1} - (T_{C_1} + B_1) = 6 - (2 + 0.5) > 3,$$

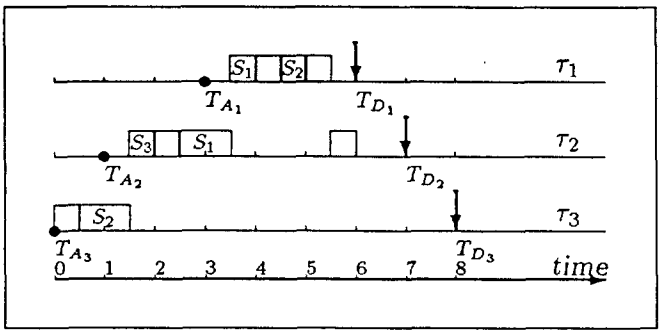
$$T_{D_2} - (\sum_{i=1}^{2} T_{C_i} + B_2) = 7 - (2 + 1 + 0.5) > 3,$$

$$T_{D_3} - (\sum_{i=1}^{3} T_{C_i} + B_3) = 8 - (2 + 1 + 2.5 + 0) < 3,$$

zato je potrebno izločanje manj kritičnih opravil iz nabora prebujenih opravil $\{\tau_1, \tau_2, \tau_3\}$. Vrednost indeksa $i = 1$ označuje parametre opravila τ_1, $i = 2$ opravila τ_2 in $i = 3$ opravila τ_3. $B_1 = 0.5$, ker lahko izvajanje opravila τ_1 blokira τ_2 v S_1 za 0.5 časovne enote. $B_2 = 0.5$, ker lahko izvajanje opravila τ_2 blokira τ_3 v S_3 za 0.5 časovne enote.

Slika 2 natančno prikazuje potek izvajanja opravil.

V času $t = 1$ prekine τ_2 izvajanje opravila τ_3. Ker se nahaja τ_3 v kritičnem področju S_2 in τ_2 ne zahteva dostop do skupnega področja, se izvajanje τ_3 v S_2 nadaljuje. Ko zahteva opravilo τ_2 v času $t = 1.5$ ključ za vstop v S_3, ga dobi, ker ni zasedeno nobeno področje. Opravilo τ_2 se ob prihodu zahteve za izvajanje τ_1 nahaja v področju S_1. Ker je $c(S_1) = p(\tau_1)$ in $p(\tau_1) > p(\tau_2)$, blokira opravilo τ_2 izvajanje τ_1 in prevzame do izstopa iz S_1 prioriteto 1.



Slika 2: Primer PD razvrščanja

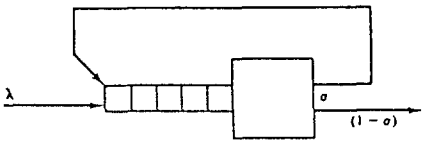
3. Primerjava PD principa z dinamičnimi principi razvrščanja

Osnovo PD principa razvrščanja predstavljata *princip prednosti časovno kritičnih opravil* (ED) in *princip kritičnih opravil* (*Priority Scheduling* - PS). Slednji dodeli prednost pri dostopu do procesorja opravilu, ki ima tisti trenutek najvišjo stopnjo pomembnosti v vrsti pripravljenih opravil. Stopnjo pomembnosti določi programer subjektivno in se le ta med izvajanjem aplikacije ne spreminja [1].

Pri snovanju PD principa razvrščanja smo težili k odpravi pomankljivosti ED in PS pristopov. Seveda pa smo želeli ohraniti tudi vse dobre lastnosti. V nadaljevanju bomo podali primerjalno analizo, ki je sestavljena iz dveh delov, matematičnega in simulacijskega. Matematični model je zahtevnejši za izpeljavo, vendar podaja zanesljive rezultate, ki temeljijo na primerjavi lastnosti posameznih principov pod enakimi pogoji. S pomočjo simulacije pa lahko preverimo rezultate matematične analize brez upoštevanja predpostavk, katere običajno uvedemo zaradi lažje izpeljave zahtevnega matematičnega modela.

3.1 Matematični model PD principa

Namenski računalniški sistem lahko predstavimo kot matematični model iz teorije vrst (slika 3). Okolje pošilja asinhrone zahteve za izvajanje opravil, ki se kopičijo v vrsti pripravljenih opravil. Zahteve so časovno neodvisne in časi med prihodi opravil so eksponentno porazdeljeni (*Markov process*). Prav tako so določeni tudi strežni in čakalni časi opravil z eksponentno porazdelitvijo. Hitrost prihoda zahtev je podana s parametrom λ, srednja vrednost dolžine strežnih časov z μ ter čakalnih časov z γ. Opravilo zapusti sistem, ko opravi vse zahtevane operacije. Verjetnost, da je opravilo prekinjeno in se mora vrniti v čakalno vrsto pripravljenih opravil je določena s parametrom $(1-\sigma)$.



Slika 3: Sistem z vrsto

Izhodni parametri našega matematičnega modela so naslednji:

1. Izkoriščenost procesorja.
2. Verjetnost zavrnitve opravil.
3. Lastnosti zavrnjenih opravil:
   - Preostali strežni čas.
   - Preostali čakalni čas.

1. Izkoriščenost procesorja ($U = \rho$) izračunamo s pomočjo dveh podatkov, povprečnega strežnega časa $E(T_C)$ ter povprečnega časa W, ki ga opravilo preživi v sistemu [1]:

$$\rho = \frac{E(T_C)}{W}. \tag{5}$$

Če upoštevamo, da so strežni in čakalni časi porazdeljeni eksponencialno:

$$\begin{aligned} g(x) &= \mu e^{-\mu x}, \\ l(x) &= \gamma e^{-\gamma x}, \end{aligned} \tag{6}$$

potem sledi izpeljava:

$$\begin{aligned} C(t) &= \int_0^t x g(x) \int_0^{t-x} l(y) dy dx, \\ H(t) &= \int_0^t g(x) \int_0^{t-x} l(y) dy dx, \end{aligned} \tag{7}$$

$$W'(t) = \frac{\lambda}{2(1-\lambda C(t))^2} \cdot \left(\int_0^t x^2 g(x) \int_0^{t-x} l(y)dydx + \int_0^t x^2 g(x)[1-H(x)]dx\right), \quad (8)$$

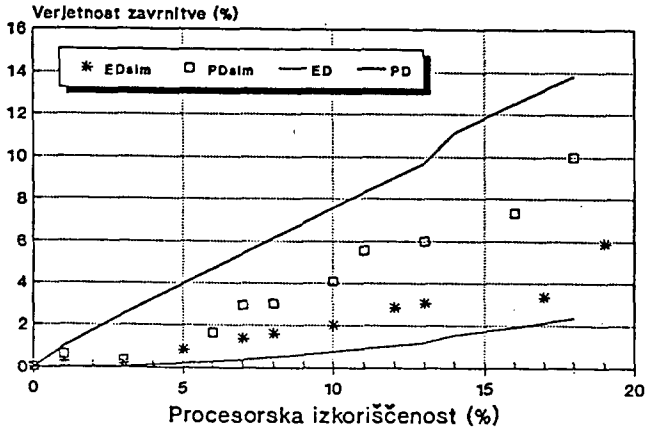
$$W''(t) = \int_0^{t_1} \frac{g(x)}{1-\lambda C(t_1)} \int_0^{t_1-x} l(y)dydx,$$
$$t_1 = W'(t), \quad (9)$$

$$W(t) = W'(t) + W''(t). \quad (10)$$

$W(t)$ je povprečni odzivni čas opravila, katerega predvideni odzivni čas je enak t. Povprečni čas opravila s poljubnim predvidenim odzivnim časom izračunamo s približkom vrednosti t s srednjo vrednostjo $E(t)$:

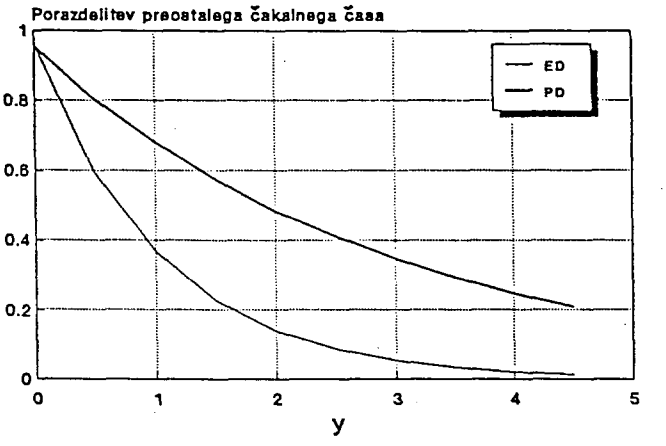
$$E(t) = \int_0^{\infty} t\, h(t)dt. \quad (11)$$

2. - 3. Podrobnejši opis analize verjetnosti zavrnitve opravil ter njihovih lastnosti smo podali v prispevku [13] ter v nalogi [1]. Zato podajmo na tem mestu le grafično primerjavo rezultatov analize dinamičnih principov razvrščanja (**graf 1, 2, 3**).

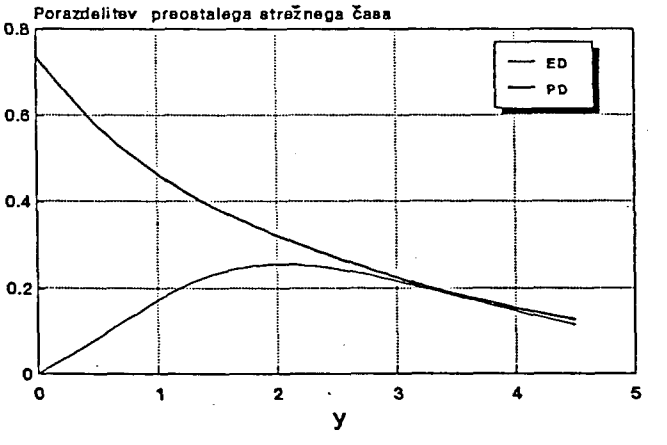


Graf 1: Verjetnost zavrnitve opravil

Pri analizi matematičnega modela smo privzeli zelo močno predpostavko, da se lahko v določenem



Graf 2: Porazdelitev preostalih čakalnih časov zavrnjenih opravil



Graf 3: Porazdelitev preostalih strežnih časov zavrnjenih opravil

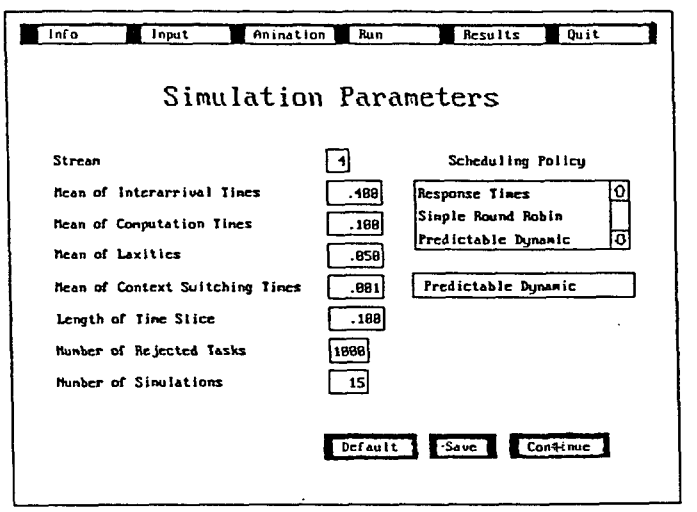
trenutku potegujeta za procesor največ dve opravili. Ker pa je procesorska izkoriščenost v *strogih* namenskih računalnikih nizka (največ 10 % [14]) in upoštevamo Markov proces prihoda zahtev za izvajanje opravil, rezultati analize ne odstopajo močno od dejanskih. To smo tudi potrdili s simulacijo, ki temelji na bolj realnih pogojih.

3.2 Simulacijski model PD principa

Program, ki omogoča simulacijo principov razvrščanja opravil, smo razvili v programskem jezi-

ku SimScript II.5 za PC kompatibilni računalnik. Uporabnik lahko spreminja sledeče parametre simulacij:

- razvrščevalni princip,
- dolžino simulacij in
- časovne parametre opravil (slika 4).



Slika 4: Vhodni simulacijski parametri

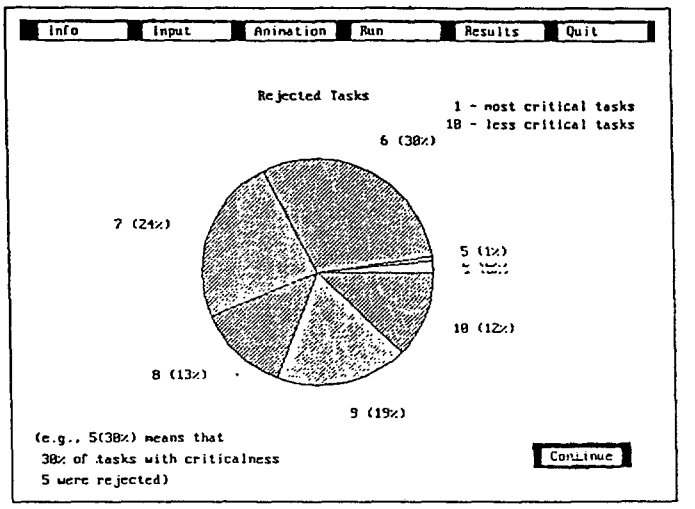
ter spremlja rezultate:

- verjetnost zavrnitve opravil pri določeni procesorski izkoriščenosti (**graf 1**),
- lastnosti zavrnjenih opravil (**graf 4**),
- uteženo verjetnost pravočasne izvršitve opravil (**graf 5**).

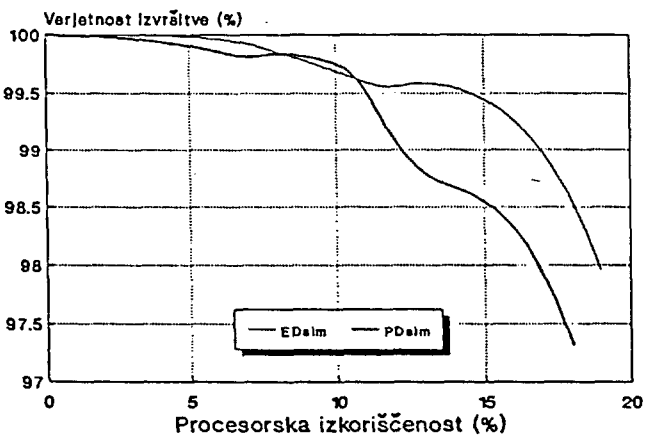
Mero utežene verjetnosti izvršitve opravil smo izračunali s pomočjo funkcije:

$$WGR = 100 \cdot \frac{\sum_i (c_i \cdot | \{\tau^i_{exe}\} |)}{\sum_i (c_i \cdot | \{\tau^i_{all}\} |)},$$
$$c_i = e^{i-1}. \qquad (12)$$

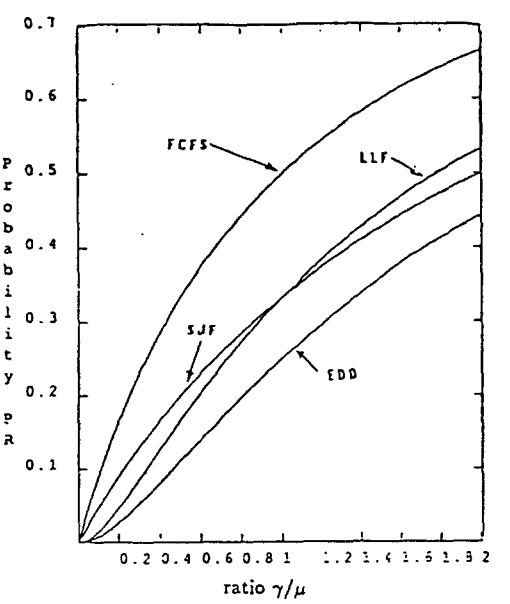
Množica $\{\tau^i_{exe}\}$ vsebuje opravila s stopnjo pomembnosti i, ki so se pravočasno izvršila in $\{\tau^i_{all}\}$ vsa prebujena opravila z i-to stopnjo pomembnosti za aplikacijo. Nižja vrednost indeksa i pomeni višjo stopnjo pomembnosti opravila.



Graf 4: Lastnosti zavrnjenih opravil



Graf 5: Utežena verjetnost izvršitve opravil



Graf 6: Verjetnost zavrnitve opravil

4. Zaključek

Predvidljivo dinamični princip razvrščanja opravil, na katerem temelji jedro OS za delo v *strogem* realnem času, upošteva zahteve realnega časa. To pomeni, da obravnava zahteve za izvajanje aperiodičnih opravil, ki so časovno in logično različno zahtevna za programsko aplikacijo.

V primerjavi z ostalimi dinamičnimi principi razvrščanja opravil (**graf 6** [10]) lahko povzamemo sledeče zaključke:

1. Princip PD zagotavlja pri določeni procesorski izkoriščenosti večjo verjetnost zavrnitve manj pomembnih opravil.

2. Verjetnost zavrnitve opravil z visoko stopnjo pomembnosti za aplikacijo je manjša.

3. Ker princip temelji na predvidljivosti, so lastnosti zavrnjenih opravil ugodnejše. Razvrščevalnik lahko preusmeri opravilo v drugo procesorsko vozlišče poraždeljenega ali večprocesorskega namenskega sistema, če je preostali čakalni čas še dovolj velik.

Področje večprocesorskih in porazdeljenih namenskih sistemov odpira nove težave, s katerimi se je potrebno soočiti. Naše nadaljne delo bo usmerjeno v raziskavo topologij namenskih sistemov s porazdeljenimi vhodno-izhodnimi napravami, ki omogočajo hitro in zanesljivo zaznavanje zunanjih dogodkov ter komunikacijo med opravili.

Literatura

[1] B. Koroušić. *Jedro operacijskega sistema za delo v strogem realnem času (magistrska naloga).* Univerza v Ljubljani, Fakulteta za elektrotehniko in računalništvo, april 1992.

[2] M. Thaler. Nova tehnologija v pilotski kabini. *KRILA*, page 28, maj 1989.

[3] L.S. McTamaney. Real-time intelligent control. *IEEE Expert*, 2(4):55–68, 1987.

[4] Operating systems in real time. *.EXE Magazine*, 4(7):72–77, 1989.

[5] W.A. Halang. Evaluation of ada from the viewpoint of control engineering. *IEEE*, pages 8–13, 1986.

[6] A. Brodnik in M. Špegel in T. Lasbaher. Programiranje z modulo-2. *Informatica, Ljubljana*, (2):78–82, 1987.

[7] E. Kligerman in A.D. Stoyenko. Real-time euclid: A language for reliable real-time systems. *IEEE Transactions on Software Engineering*, 12(9):941–949, september 1986.

[8] Real-time software. *Micro Technology*, december 1991.

[9] J.E. Cooling. *Software Design for Real-Time Systems.* Chapman and Hall, 1990. ISBN 0-412-341-808.

[10] D.W. Craig in C.M. Woodside. The rejection rate for tasks with random arrivals, deadlines, and preemptive scheduling. *IEEE Transactions on Software Engineering*, 16(10), oktober 1990.

[11] J.A. Stankovic. Misconceptions about real-time computing. *COMPUTER*, 21(10):10–19, oktober 1988.

[12] M.I. Chen in K.J. Lin. Dynamic priority ceilings: A concurrency control protocol for real-time systems. *Real-Time Systems - The international journal of time-critical computing systems*, 2(4):325–345, november 1990.

[13] B. Koroušić in J.E. Cooling in P. Kolbezen. Predictable hard real-time scheduling. *In Proceedings of the* 4^{th} *EUROMICRO Workshop on Real-Time*, junij 1992.

[14] C.J. Paul in A. Acharya in B. Black in J.K. Strosnider. Reducing problem-solving variance to improve predictability. *COMMUNICATIONS OF THE ACM*, pages 81–93, avgust 1991.

SINHRONIZIRANA PODATKOVNO PRETOKOVNA RAČUNALNIŠKA ARHITEKTURA II

INFORMATICA 3/92

Jurij Šilc
Laboratorij za računalniške arhitekture
Inštitut Jožef Stefan, Ljubljana

Ludvik Gyergyek
Fakulteta za elektrotehniko in računalništvo,
Ljubljana

Keywords: parallel processing, dataflow computing, scheduling, allocator, performance evaluation, parallel computer architecture

Delo obravnava problem časovne optimizacije asinhronega procesiranja na omejenem številu procesorjev. Predlagamo izvirno rešitev, ki temelji na uvedbi nekaterih mehanizmov sinhronizacije v asinhrono računanje. Graf pretoka podatkov, ki opisuje asinhrono procesiranje, opremimo s časovno optimalno sprožitveno funkcijo, ki služi tako pri vlaganju grafa v računalnik, kakor tudi pri njegovem časovno optimalnejšem izvrševanju. V ta namen smo razvili hevristična algoritma `pOptSinh` in `TOptSinh` za konstrukcijo optimalnih sprožitvenih funkcij, ki po pesimistični oceni vračata optimalno rešitev v 80% primerov. Nadalje predlagamo algoritma za dodeljevanje `MinGlDol` in `MinGlGor`, ki temeljita na optimizaciji medprocesorskih komunikacij. V primerjavi z znanimi razvrščevalnimi algoritmi dobimo s predlaganima algoritmoma boljše rezultate, kar potrjujejo analizirani primeri algoritmov za izračun hitre Fourierjeve transformacije, dinamične analize scene in LU razcepa matrike. Končno podajamo tudi zasnovo hibridne vzporedne arhitekture računalnika, ki podpira predlagano preoblikovanje asinhronega računanja.

SYNCHRONOUS DATAFLOW COMPUTER ARCHITECTURE – We discuss the problem of time optimization of asynchronous processing on a limited number of processors. We present an original solution to the problem based on introduction of synchronization mechanisms into asynchronous processing. The dataflow graph describing asynchronous processing is associated with the corresponding time-optimal firing function. This function is used both for loading a dataflow graph into the computer and for time-optimal graph execution. In order to do this, we have developed two heuristic algorithms, `pOptSinh` and `TOptSinh`, which are used for optimal firing function construction. According to conservative estimates, these algorithms return optimal functions with 80% probability. Furthermore, we propose two scheduling algorithms, `MinGlDol` and `MinGlGor`, which are based on interprocessor communication minimization. These two algorithms give better results compared to some other well known scheduling algorithms. This fact is illustrated in Fast Fourier Transformation, Dynamic Scene Analysis, and LU matrix decomposition algorithms. Finally, we present a design of hybrid parallel computer architecture capable of supporting modified asynchronous computing.

1. Asinhrono računanje

V nadaljevanju bomo opisali način *asinhronega* računanja, ki ga bomo opisali s pomočjo GPP. Formalizirali bomo problema minimizacije časa oziroma procesorjev. Uvedli bomo pojem sprožitvene funkcije, ki nam bo v nadaljevanju omogočil vpeljati nekatere mehanizme *sinhronizacije* v asinhrono računanje.

1.1 Graf pretoka podatkov

Graf G, ki ga sestavljata množici *točk* $\mathcal{V}$ in *povezav* $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, označimo z $G(\mathcal{V}, \mathcal{E})$. Število točk in povezav označimo z $n = |\mathcal{V}|$ oziroma $m = |\mathcal{E}|$. Kadar je $(u, v) \in \mathcal{E}$, pravimo, da je točka v *neposredni naslednik* točke u, in to označimo z $u \longmapsto v$. Kadar obstaja vsaj ena usmerjena pot iz točke u do točke v, pa pravimo, da je v *naslednik* točke u, kar označimo z $u \overset{+}{\longmapsto} v$. Relacija $\overset{+}{\longmapsto}$ je *tranzitivna ovojnica* relacije $\longmapsto$. Graf G je

acikličen, če ne vsebuje točke v z lastnostjo $v \stackrel{+}{\longmapsto} v$. Če je G acikličen, je $\longmapsto$ *relacija stroge delne urejenosti* v $\mathcal{V}$, saj je irefleksivna, asimetrična in tranzitivna. *Vhodnja stopnja* $d^+(v)$ točke v je število povezav, ki se stekajo v v, *izhodnja stopnja* $d^-(v)$ točke v pa je število povezav, ki točko v zapuščajo. Pravimo, da je G *enostaven*, če obstaja med dvema točkama največ ena povezava v isti smeri.

Naj bo $G(\mathcal{V},\mathcal{E})$ acikličen. Tedaj zapišemo $\mathcal{V}$ kot:

$$\mathcal{V} = \mathcal{V}_Z \cup \mathcal{V}_N \cup \mathcal{V}_K.$$

$\mathcal{V}_Z$, $\mathcal{V}_N$ in $\mathcal{V}_K$ se imenujejo množica *začetnih*, *notranjih* oziroma *končnih* točk. Velja naslednje:

$$\begin{aligned} \forall v \in \mathcal{V}_Z &: \quad d^+(v) = 0 \wedge d^-(v) > 0, \\ \forall v \in \mathcal{V}_N &: \quad d^+(v) > 0 \wedge d^-(v) > 0, \\ \forall v \in \mathcal{V}_K &: \quad d^+(v) > 0 \wedge d^-(v) = 0. \end{aligned}$$

Če sta $\mathcal{V}_Z$ in $\mathcal{V}_K$ končni, neprazni in nevezani množici, potem v G ni nobene izolirane točke.

V nadaljevanju bomo obravnavali le grafe pretoka podatkov GPP, ki so v skladu s sledečo definicijo:

DEFINICIJA: Par $(G(\mathcal{V},\mathcal{E}),t)$ je *graf pretoka podatkov*, če velja:
(a) G je usmerjen, enostaven in acikličen;
(b) $\mathcal{V}_Z \cap \mathcal{V}_K = \emptyset$;
(c) $t : \mathcal{V} \longrightarrow \mathbb{N}$. □

Če je $v \in \mathcal{V}$, pomeni $t(v)$ *čas izvrševanja* točki v pridružene operacije[1].

1.2 Najkrajši čas izvrševanja

Naj bodo dani $GPP = (G,t)$ ter točki u in v. Če velja $u \stackrel{+}{\longmapsto} v$, obstaja iz točke u do v vsaj ena *pot* $\wp = w_1, w_2, \ldots, w_k$, kjer je $w_1 = u$ ter $w_k = v$. Tedaj definiramo *dolžino poti* $\wp$ kot $\ell(\wp) = \sum_{i=1}^{k} t(w_i)$. Dolžino *najdaljše* poti iz točke u v točko v pa označimo z $\ell(u,v)$. Pot iz množice točk $\mathcal{U}$ v množico točk $\mathcal{W}$ je vsaka pot, ki se prične v eni od točk množice $\mathcal{U}$ in se konča v eni od točk množice $\mathcal{W}$. Dolžino najdaljše poti med množicama $\mathcal{U}$ in $\mathcal{W}$ pa označimo z $\ell(\mathcal{U},\mathcal{W})$. Če vsebuje kaka od množic $\mathcal{U}$ ali $\mathcal{W}$ en sam element, pišemo namesto množice kar sam element; na primer, če je $\mathcal{U} = \{u\}$, pišemo kar $\ell(u,\mathcal{W})$.

[1] Operacij posebej ne navajamo, ker za samo analizo niso pomembne.

Seveda je $\ell(\mathcal{V}_Z,\mathcal{V}_K)$ *najkrajši* možni čas, v katerem se še more izvršiti GPP. Označimo ga s T_∞, ker se GPP izvrši v najkrajšem možnem času le ob zadosti velikem (praktično "neskončnem") številu procesorjev. Če je na voljo le en procesor (zaporedno izvrševanje), pa označimo najkrajši možni čas za izvršitev GPP s T_1. Seveda velja $T_1 = \sum_{v\in\mathcal{V}} t(v)$.

1.3 Sprožitvena funkcija

Naj bodo dani $GPP = (G,t)$, naravno število $T \geq T_\infty$ ter funkcija $s : \mathcal{V} \longrightarrow \{0,1,\ldots T\}$. Če je $v \in \mathcal{V}$, pišemo $f(v) = s(v) + t(v)$.

DEFINICIJA: Funkcijo s imenujemo *sprožitvena* funkcija, če velja:
(a) $f(v) \leq T, \quad \forall v \in \mathcal{V}$ in
(b) $u \longmapsto v \Longrightarrow f(u) \leq s(v), \quad \forall u,v \in \mathcal{V}$. □

Sprožitvena funkcija s priredi vsaki točki v trenutek *sprožitve* $s(v)$, v katerem točka v zajame vhodne podatke in prične izvrševanje pridružene operacije. Posredno je s tem natanko določen tudi trenutek *izstrelitve* $f(v)$, v katerem točka v konča svoje izvrševanje in pošlje podatke vsem neposrednim naslednikom. Zgornja definicija zagotavlja, da vsaka točka izstreli svoje rezultate pred trenutkom T; poleg tega pa proženje ter izstreljevanje spoštujeta relacijo $\longmapsto$. Za dani GPP v splošnem obstaja več različnih sprožitvenih funkcij. Množico vseh sprožitvenih funkcij danega GPP označimo z $\mathcal{S}(T)$. Vsaka $s \in \mathcal{S}(T)$ porodi pravilo, po katerem poteka računanje GPP. Kadar bomo želeli to pravilo posebej poudariti, bomo k oznaki s dodali ustrezni indeks. Sprožitveno funkcijo s imenujemo *požrešna*, če velja

$$s(v) = \ell(\mathcal{V}_Z, v) - t(v), \text{ za vsak } v \in \mathcal{V},$$

oziroma *lena*, če velja

$$s(v) = T - \ell(v, \mathcal{V}_K), \text{ za vsak } v \in \mathcal{V}.$$

Požrešno[2] sprožitveno funkcijo bomo označili s s_e, leno[3] pa s s_l. Prva opisuje podatkovno vodeno računanje, druga pa računanje vodeno z zahtevo.

[2] Požrešno iz angl. "eager".
[3] Leno iz angl. "lazy".

1.4 Kritične točke

Naj bo dan $GPP = (G, t)$. Poti z dolžino $\ell(\mathcal{V}_Z, \mathcal{V}_K)$ so *najdaljše* poti v GPP.

DEFINICIJA: Točko v imenujemo *kritična*, kadar se nahaja na kaki od najdaljših poti v GPP. Če je $0 \leq \tau \leq \ell(\mathcal{V}_Z, \mathcal{V}_K)$, imenujemo točko v *kritična na globini* τ, kadar je kritična in velja $\ell(\mathcal{V}_Z, v) - t(v) = \tau$. □

Množico kritičnih točk označimo s $\mathcal{K}$, množico kritičnih na globini τ pa s $\mathcal{K}(\tau)$. Kritičnost je torej značilnost GPP samega, saj je moč enostavno pokazati, da je kritična natanko tista točka $v \in \mathcal{V}$, za katero velja

$$\ell(\mathcal{V}_Z, v) - t(v) = \ell(\mathcal{V}_Z, \mathcal{V}_K) - \ell(v, \mathcal{V}_K).$$

V posebnem primeru smemo kritičnost točk opredeliti tudi z vidika sprožitvenih funkcij. Naj bo dan čas izvrševanja T. Opazimo, da je levi del zgornje enačbe enak $s_e(v)$. Desni del pa je enak $s_l(v)$, vendar le, če je $T = T_\infty$! V tem primeru je kritični točki možno prirediti le en in en sam trenutek za njeno sprožitev, neodvisno od izbrane sprožitvene funkcije $s \in \mathcal{S}(T_\infty)$. Značilnost točk $v \in \mathcal{K}(\tau)$ pa je, da se sprožijo natanko v trenutku τ, zato jih včasih imenujemo tudi *kritične v trenutku* τ.

1.5 Zahteve po procesorjih

Naj bodo dani $GPP = (G, t)$, naravno število $T \geq T_\infty$ ter sprožitvena funkcija $s \in \mathcal{S}(T)$. Kadar je $\tau \in \{0, 1, \ldots, T\}$, definiramo

$$p_{T,s}(\tau) = \Big| \{ v \in \mathcal{V} \mid s(v) \leq \tau < f(v) \} \Big|,$$

kar je število v trenutku τ izvršujočih se točk GPP oz. število procesorjev, ki so zasedeni v trenutku τ. Število procesorjev, ki so potrebni za nemoteno izvršitev GPP v času T, kot to narekuje sprožitvena funkcija s, je tedaj

$$p_{T,s} = \max_{0 \leq \tau \leq T} p_{T,s}(\tau).$$

Najmanjše število procesorjev, ki so potrebni za izvršitev GPP v času T pa je

$$p_T = \min_{s \in \mathcal{S}(T)} p_{T,s}.$$

Sprožitvene funkcije, ki minimizirajo zgornji izraz imenujemo *procesorsko optimalne* sprožitvene funkcije – okrajšano p-optimalne. Iskanje p-optimalnih sprožitvenih funkcij je smiselno le za $T \in [T_\infty, T_1]$. Posebej se bomo osredotočili na primer $T = T_\infty$, ki opisuje iskanje minimalnega števila procesorjev p_{T_∞} za najhitrejšo izvršitev GPP.

1.5.1 Ocene spodnje meje

Sedaj predpostavljamo, da je $T = T_\infty$. Kot smo že omenili je problem iskanja vrednosti p_{T_∞} NP-poln. Zato nadomestimo natančno računanje p-optimalne sprožitvene funkcije s hevristično konstrukcijo sprožitvene funkcije, ki pa je "zadosti blizu" p-optimalni. Pri hevrističnem konstruiranju sprožitvene funkcije je koristno poznati čim bolj natančno spodnjo mejo za p_{T_∞}. Zato bomo v nadaljevanju uvedli nekatere metode za računanje spodnje meje $\tilde{p}_{T_\infty}$. Kvaliteto metode določata: časovna zahtevnost računanja $\tilde{p}_{T_\infty}$ in natančnost $\tilde{p}_{T_\infty}$, ki jo določa vrednost $p_{T_\infty} - \tilde{p}_{T_\infty}$.

Fernandez - Bussellova ocena. Zelo natančno oceno za $\tilde{p}_{T_\infty}$ podajata *Fernandez* in *Bussell* v [1]. To oceno določimo na sledeči način. Naj bo v poljubna točka iz $GPP = (G, t)$. Interval $I(v) = [s_e(v), f_l(v)]$ imenujemo *interval izvrševanja* točke v. Za nekritične točke, torej točke $v \in \mathcal{V} - \mathcal{K}$, velja $t(v) < f_l(v) - s_e(v)$, kar pomeni, da se sme njihovo izvrševanje "premikati" znotraj $I(v)$. Naj bosta τ_1 in τ_2 poljubni naravni števili in naj velja $0 \leq \tau_1 < \tau_2 \leq T_\infty$. Interval $J(v, \tau_1, \tau_2) = I(v) \cap [\tau_1, \tau_2]$ imenujemo *interval prekrivanja* točke v glede na τ_1 in τ_2. Kadar je $J(v, \tau_1, \tau_2) \neq \emptyset$, smemo premakniti izvrševanje nekritične točke v tako, da minimiziramo čas njenega izvrševanja znotraj $J(v, \tau_1, \tau_2)$. Vsoto vseh tako minimiziranih časov, gledano po vseh $v \in \mathcal{V}$, označimo s $K(\tau_1, \tau_2)$. Končno smemo zapisati oceno

$$\tilde{p}_{T_\infty} = \max_{\tau_1 < \tau_2} \left\lceil \frac{K(\tau_1, \tau_2)}{\tau_2 - \tau_1} \right\rceil.$$

V nadaljevanju bomo to oceno označevali s $\tilde{p}_{T_\infty, FB}$. Računanje $\tilde{p}_{T_\infty, FB}$ zajema obravnavo $T_\infty(T_\infty + 1)/2$ intervalov $[\tau_1, \tau_2]$, kar pomeni, da je časovna zahtevnost reda $\mathcal{O}(T_\infty^2)$. Vidimo, da ima računanje $\tilde{p}_{T_\infty, FB}$ relativno veliko časovno zahtevnost, zato si bomo v nadaljevnaju ogledali še nekatere manj natančne toda hitrejše metode za izračun $\tilde{p}_{T_\infty}$.

Chen - Epleyeva ocena. V [2][4] je vpeljana ocena za p_{T_∞}, ki temelji na povprečni vzporednosti $S_\infty =$

[4] To oceno je dejansko prvi vpeljal McNaughton v [3]

$\frac{T_1}{T_\infty}$ in se glasi

$$\tilde{p}_{T_\infty} = \lceil S_\infty \rceil.$$

To oceno bomo označevali s $\tilde{p}_{T_\infty,CE}$.

Hujeva ocena. *Hu* je v [4] vpeljal oceno za p_{T_∞} na sledeč način. Naj bo $\mathcal{L}(\tau) = \{v \in \mathcal{V} \mid f_l(v) = \tau\}$, torej množica vseh točk, ki se morajo izstreliti najkasneje v trenutku τ. Potem je

$$\tilde{p}_{T_\infty} = \max_{0 \le \omega \le T_\infty} \left\lceil \frac{1}{\omega} \sum_{\tau=1}^{\omega} | \mathcal{L}(\tau) | \right\rceil.$$

Hujevo oceno bomo označevali s $\tilde{p}_{T_\infty,H}$ in jo imenovali tudi *največja povprečna vzporednost.*

Ramamoorthy - Chandy - Gonzalezova ocena. *Ramamoorthy et al.* so v [5] izboljšali $\tilde{p}_{T_\infty,H}$ tako, da so upoštevali tudi *kritično vzporednost*, ki je vpeljana kot

$$\kappa = \max_{0 \le \tau \le T_\infty} | \mathcal{K}(\tau) | .$$

Oceno so definirali kot

$$\tilde{p}_{T_\infty} = \max(\tilde{p}_{T_\infty,H}, \kappa).$$

Njihovo oceno pa bomo označili s $\tilde{p}_{T_\infty,R}$.

Razširjena kritična vzporednost. Razvidno je, da je kritična vzporednost κ precej šibka ocena za p_{T_∞}, saj se pri relativno velikih grafih njen vpliv izniči. Takrat postane $\tilde{p}_{T_\infty,R} \approx \tilde{p}_{T_\infty,H}$. Eden od načinov za izboljšanje natančnosti ocene je torej izostritev kritične vzporednosti κ, kar bomo storili v nadaljevanju. Predpostavimo, da je v vseh trenutkih τ intervala $[\tau_1, \tau_2]$ v vsaki množici $\mathcal{K}(\tau)$ po κ elementov. Zgodi se, da se mora v $[\tau_1, \tau_2]$ poleg kritičnih izvrševati tudi katera od nekritičnih točk. Kadar obstaja vsaj ena taka točka, potrebujemo dodatni procesni element, torej je potrebnih $\kappa + 1$ procesorjev! Do podobne situacije pa seveda pogosto pride tudi v intervalih, kjer je število kritičnih točk sicer manjše od κ, vendar se mora znotraj tega intervala (delno) izvrševati toliko nekritičnih točk, da skupno število potrebnih procesorjev preseže κ. V nadaljevanju bomo zato vpeljali *razširjeno kritično vzporednost* κ' in jo uporabili pri lastni oceni za p_{T_∞} [6].

Na končnem intervalu $[0, T_\infty]$ poiščimo najmanjše končno zaporedje trenutkov $0 = \tau_0 < \tau_1 < \ldots < \tau_k = T_\infty$, in sicer takšno, da je na vsakem intervalu $[\tau_{j-1}, \tau_j)$, $j = 1, 2, \ldots, k$ število kritičnih točk $| \mathcal{K}(\tau) |$ stalno. To število krajše označimo s κ_{j-1}. Definirajmo množico $\mathcal{W}_{j-1}$ tistih nekritičnih točk, ki zahtevajo v intervalu $[\tau_{j-1}, \tau_j)$ procesor vsaj za en trenutek:

$$\begin{aligned} \mathcal{W}_{j-1} = \{ \; & v \in \mathcal{V} - \mathcal{K} \mid (s_e(v) \ge \tau_{j-1} \vee \\ & f_e(v) > \tau_{j-1} > s_e(v)) \wedge \\ & (f_l(v) \le \tau_j \vee f_l(v) > \tau_j > s_l(v)) \}. \end{aligned}$$

Izrševanje točke $v \in \mathcal{W}_{j-1}$ se časovno prekriva (v vsaj enem trenutku) z izvrševanjem kritičnih točk v časovnem intervalu $[\tau_{j-1}, \tau_j)$. To prekrivanje je v splošnem krajše od časa $t(v)$, kajti točka se lahko sproži že pred trenutkom τ_{j-1} in/ali konča izvrševanje po trenutku τ_j. Zato upoštevamo pri izračunu skupnega prekrivanja v intervalu $[\tau_{j-1}, \tau_j)$ za vsako točko $v \in \mathcal{W}_{j-1}$ le njeno minimalno prekrivanje $\omega_{j-1}(v)$, ki je

$$\omega_{j-1}(v) = \min\{f_e(v) - \tau_{j-1}, \; \tau_j - s_l(v), \; t(v)\}$$

Minimalno potrebno število dodatnih procesnih elementov v časovnem intervalu $[\tau_{j-1}, \tau_j)$ je tako:

$$\begin{aligned} \lambda_{j-1} = & \Big(\sum_{v \in \mathcal{W}_{j-1}} \omega_{j-1}(v) \Big) \Big/ \\ & \max\{\tau_{j-1}, \min_{v \in \mathcal{W}_{j-1}} s_e(v)\} - \min\{\tau_j, \max_{v \in \mathcal{W}_{j-1}} f_l(v)\}. \end{aligned}$$

Končno je mogoče zapisati razširjeno kritično vzporednost κ' kot

$$\kappa' = \max_{1 \le j \le k} \lceil (\kappa_j + \lambda_j) \rceil$$

in definiramo oceno za p_{T_∞} takole:

$$\tilde{p}_{T_\infty} = \max(\tilde{p}_{T_\infty,H}, \kappa').$$

To oceno bomo označili tudi s $\tilde{p}_{T_\infty,\kappa'}$.

Opisana metoda je poenostavitev Fernandez-Bussellove metode, v kateri namesto $\mathcal{O}(T_\infty^2)$ intervalov analiziramo največ $\mathcal{O}(T_\infty)$ disjunktnih intervalov. S takšno poenostavitvijo pa se natančnost ocene ne poslabša bistveno.

n	$\tilde{p}_{T_\infty,CE}$	$\tilde{p}_{T_\infty,H}$	$\tilde{p}_{T_\infty,R}$	$\tilde{p}_{T_\infty,\kappa'}$
1 - 20	.85897	.91815	.93984	.98225
21 - 40	.81865	.89119	.89983	.93207
41 - 60	.87800	.91306	.91595	.92843
61 - 80	.88899	.91847	.92151	.93322
81 - 100	.88649	.91497	.91904	.92596
1 - 100	.87006	.91143	.91768	.93529

Tabela 1: Relativna natančnost ocen pri $t(v) = 1$.

Primerjava ocen. Ocene za spodnjo mejo potrebnega števila procesorjev smo računali[5]

[5] V ta namem smo uporabili računalnik IBM PC in razvili ustrezna programska orodja.

n	$\tilde{p}_{T_\infty,CE}$	$\tilde{p}_{T_\infty,H}$	$\tilde{p}_{T_\infty,R}$	$\tilde{p}_{T_\infty,\kappa'}$
1 - 20	.88188	.92392	.92392	.98498
21 - 40	.82895	.89294	.89354	.94139
41 - 60	.86430	.90486	.90486	.91888
61 - 80	.89379	.91816	.91816	.93241
81 - 100	.89224	.91446	.91446	.92759
1 - 100	.87384	.91040	.91051	.93561

Tabela 2: Relativna natančnost ocen pri $t(v) \leq 5$.

n	$\tilde{p}_{T_\infty,CE}$	$\tilde{p}_{T_\infty,H}$	$\tilde{p}_{T_\infty,R}$	$\tilde{p}_{T_\infty,\kappa'}$
1 - 20	.88200	.92100	.92100	.98200
21 - 40	.82912	.89350	.89350	.93622
41 - 60	.87045	.90739	.90739	.92385
61 - 80	.89447	.91911	.91911	.93026
81 - 100	.89115	.91370	.91370	.92194
1 - 100	.87510	.91072	.91072	.93348

Tabela 3: Relativna natančnost ocen pri $t(v) \leq 10$.

po Fernandez-Bussellovi, Chen-Epleyevi, Hujevi, Ramamoorthy-Chandy-Gonzalezovi in lastni metodi. Skupno smo analizirali 7.500 naključno generiranih grafov pretoka podatkov. Pri tem smo spreminjali število točk $n \leq 100$ in čas njihovega izvrševanja $t(v) \leq 10$. Rezultati analiz so prikazani v tabelah 1, 2 in 3. Natančnost metod smo določali glede na najnatančnejšo, tj. Fernandez-Bussellovo oceno $\tilde{p}_{T_\infty,FB}$, saj je število p_{T_∞} neznano. Rezultati potrjujejo, da je naša metoda po svoji natančnosti najbliže $\tilde{p}_{T_\infty,FB}$, saj je v povprečju le za 6.52% slabša, hkrati pa je za red velikosti hitrejša. Naši oceni sledijo $\tilde{p}_{T_\infty,R}$ (8.70%), $\tilde{p}_{T_\infty,H}$ (8.92%) in $\tilde{p}_{T_\infty,CE}$ (12.71%). Primerjava med $\tilde{p}_{T_\infty,H}$ in $\tilde{p}_{T_\infty,R}$ kaže, da je slednja ocena boljša le pri tistih GPP, katerih vrednosti $t(v)$ se spreminjajo zelo malo. Samo v takšnih grafih kritična vzporednost prevlada nad največjo povprečno vzporednostjo. Razširjena kritična vzporednost, ki smo jo vpeljali pri svoji oceni $\tilde{p}_{T_\infty,\kappa'}$, pa prevlada tudi v tistih GPP, kjer je $t(v)$ spremenljiv.

1.6 Zahteve po času

Naj bosta dana $GPP = (G,t)$ in naravno število $p \leq p_{T_\infty}$. Iščemo najmanjše naravno število $T_p \geq T_\infty$, za katero je množica sprožitvenih funkcij $\mathcal{S}(T_p) \neq \emptyset$. Posledica pomanjkanja procesorjev je *podaljšanje časa izvrševanja* ΔT_p, ki znaša

$$\Delta T_p = T_p - T_\infty.$$

Sprožitvene funkcije, ki minimizirajo zgornji izraz imenujemo *časovno optimalne* sprožitvene funkcije – krajše T-optimalne. Iskanje T-optimalnih sprožitvenih funkcij je seveda smiselno le za $p \leq p_{T_\infty}$.

1.6.1 Ocene spodnje meje

Tudi problem iskanja vrednosti T_p je NP-poln, zato namesto natančnega računanja T-optimalne sprožitvene funkcije uporabimo njeno hevristično konstrukcijo. Podobno kot pri iskanju p-optimalnih sprožitvenih funkcij je tudi tu koristno poznati kar se da natančno spodnjo mejo za T_p, v nadaljevanju označeno s $\tilde{T}_p$. Tudi tokrat določata kvaliteto metode za računanje spodnje meje: časovna zahtevnost računanja $\tilde{T}_p$ in natančnost $\tilde{T}_p$, ki jo določa vrednost $T_p - \tilde{T}_p$.

Hujeva ocena. *Hu* je v [4] vpeljal za T_p sledečo oceno: Naj bo zopet $\mathcal{L}(\tau)$ množica vseh točk, ki se morajo izstreliti najkasneje v trenutku τ. Potem je

$$\tilde{T}_p = T_\infty + \max_{0 \leq \omega \leq T_\infty} \left\lceil -\omega + \frac{1}{p} \sum_{\tau=1}^{\omega} | \mathcal{L}(\tau) | \right\rceil.$$

Hujevo oceno bomo označevali s $\tilde{T}_{p,H}$.

Fernandez - Bussellova ocena. Izboljšanje ocene $\tilde{T}_{p,H}$ sta opisala *Fernandez* in *Bussell* v [1]. To oceno določimo s sredstvi, ki so bila vpeljana med konstruiranjem $\tilde{p}_{T_\infty,FB}$. Naj bosta τ_1 in τ_2 poljubni naravni števili in naj velja $0 \leq \tau_1 < \tau_2 \leq T_\infty$. Če je na voljo le p procesorjev, potem se izvrševanje točk znotraj intervala $[\tau_1, \tau_2]$ podaljša za najmanj $\frac{K(\tau_1,\tau_2)}{p} - (\tau_2 - \tau_1)$. Od tod moremo določiti oceno za $\tilde{T}_p$, ki je

$$\tilde{T}_p = T_\infty + \max_{\tau_1 < \tau_2} \left\lceil \frac{K(\tau_1,\tau_2)}{p} - (\tau_2 - \tau_1) \right\rceil.$$

V nadaljevanju bomo to oceno označevali s $\tilde{T}_{p,FB}$.

Primerjava ocen. V [1] je pokazano, da je Fernandez-Bussellova ocena natančnejša od Hujeve. Ta natančnost se doseže za ceno večje časovne kompleksnosti, ki znaša $\mathcal{O}(T_\infty^2)$ in je za razred večja od Hujeve.

2. Mehanizmi sinhronizacije

Sedaj bomo opisali hevristična algoritma za konstrukcijo T- in p-optimalnih sprožitvenih funkcij, ki ju bomo uporabili v dveh hevrističnih algoritmih za dodeljevanje; z njima dani GPP porazdelimo med procesorje [9].

vhod: GPP, p.
izhod: $SGPP(p, T_p)$, oz. $s(v)$, za vsak $v \in \mathcal{V}$.

$\tau := 0$; $T_p := 0$; $q := 0$; $\mathcal{W} := \mathcal{V}$
repeat
  if $q > 0$ **then**
    $\mathcal{V}_p := \{v \in \mathcal{V} | f(v) = \tau\}$; $\mathcal{W} := \mathcal{W} - \mathcal{V}_p$; $q := q - |\mathcal{V}_p|$
  endif
  $\mathcal{V}_i := \{v \in \mathcal{V} | v$ ima vse vhodne podatke$\}$;
  -- Množica izvršljivih točk je $\mathcal{V}_i = \mathcal{K}(\tau) \cup \mathcal{V}_n$, kjer sta $\mathcal{K}(\tau)$ in $\mathcal{V}_n$
  -- množici kritičnih oz. nekritičnih točk na globini τ.
  if $q < p$ **then**
    if $p - q \leq |\mathcal{K}(\tau)|$ **then** bodi $\mathcal{V}_d \subset \mathcal{K}(\tau)$, kjer je $|\mathcal{V}_d| \leq p - q$ **else**
    if $p - q \geq |\mathcal{V}_i|$ **then** $\mathcal{V}_d := \mathcal{V}_i$ **else**
      Bodi $\mathcal{V}_d \subset \mathcal{V}_n$, kjer je $|\mathcal{V}_d| \leq p - q - |\mathcal{K}(\tau)|$; $\mathcal{V}_d := \mathcal{V}_d \cup \mathcal{K}(\tau)$
    endif
    $q := q + |\mathcal{V}_d|$ -- Sproži se nekaj dodatnih, naključno izbranih točk.
  endif
  forall $v \in \mathcal{V}_d$ **do** $s(v) := \tau$ **endforall**
  $T_p := \tau$; $\tau := \tau + 1$
until $W = \emptyset$;

Algoritem `TOptSinh`: Konstruiranje T-optimalne sprožitvene funkcije.

2.1 Sinhronizacija

Algoritma za konstrukcijo optimalnih sprožitvenih funkcij, ki ju bomo opisali v tem poglavju, temeljita na spoznanju, da si moremo izvrševanje GPP predočiti s pomočjo prehajanja točk iz ene množice (stanja) v drugo. Že v prej smo vpeljali eno takšnih množic – množico $\mathcal{K}(\tau)$ kritičnih točk v trenutku τ. Nadalje za vsak trenutek τ vpeljemo še množici: $\mathcal{V}_i$ *izvršljivih* točk, tj. točk, ki so zbrale vse vhodne podatke, a se še niso sprožile in $\mathcal{V}_p = \{v \in \mathcal{V} | f(v) = \tau\}$ *pozabljenih* točk, tj. točk, ki so se izstrelile v tem trenutku. V splošnem so poleg kritičnih v množici $\mathcal{V}_i$ tudi točke, ki smejo svojo sprožitev odložiti, ne da bi to nujno vplivalo na najkrajše izvrševanje GPP. Te točke imenujemo *nekritične* v trenutku τ in jih zberemo v množico $\mathcal{V}_n$.

Par (GPP, s), kjer je s sprožitvena funkcija, ki posredno določa tudi p in T, imenujemo *sinhronizirani* graf pretoka podatkov ter ga označimo z $SGPP(p, T)$. Torej T-optimalna sprožitvena funkcija določa $SGPP(p, T_p)$, medtem ko p-optimalna funkcija določa $SGPP(p_{T_\infty}, T_\infty)$.

2.1.1 T-optimalna sprožitvena funkcija

Za konstruiranje T-optimalne sprožitvene funkcije smo razvili algoritem, ki smo ga imenovali `TOptSinh`. Ta nam za dani GPP ter vnaprej določeno število p procesorjev vrne takšno sprožitveno funkcijo s, ki zagotavlja čas izvrševanja T_p, ki je kar se da blizu oceni $\tilde{T}_p$.

Kvaliteto algoritma smo ocenjevali z odstotkom primerov, ko je čas T_p dosegel oceno $\tilde{T}_{p,H}$, saj natančne vrednosti T_p ne poznamo. Algoritem je bil preverjen[6] nad 500 GPP in je v 75.6 % dosegel $T_p = \tilde{T}_{p,H}$. Obenem smo merili tudi upad idealne popospešitve D_p v odvisnosti od pomanjkanja procesorjev. Primerjava med asinhronim podatkovno vodenim računanjem (sprožitvena funkcija s_e) in sinhroniziranim podatkovno vodenim računanjem (T-optimalna sprožitvena funkcija) je prikazana v tabeli 4, kjer so podani povprečni rezulati.

Ključni del algoritma `TOptSinh` je v konstruiranju množice $\mathcal{V}_d$, tj. v sprožanju dodatnih izvršljivih točk, kadar so na voljo prosti procesorji. Absolutno prednost pri sprožanju imajo kritične točke v danem trenutku (zaradi omejenega števila procesorjev seveda v splošnem pride do odloga sprožitve celo pri nekaterih kritičnih točkah). Uvrščanje nekritične točk v množico $\mathcal{V}_d$ pa smo izvajali v skladu z naslednjimi strategijami: naključno izbiranje (v vrstnem redu, kot prihajajo

[6] V ta namem smo uporabili računalnik IBM PC in razvili ustrezna programska orodja.

vhod: GPP, T_∞.
izhod: $SGPP(p_{T_\infty}, T_\infty)$, oz. $s(v)$, za vsak $v \in \mathcal{V}$.

Izračunaj $\tilde{p}_{T_\infty}$; $\tau := 0$; $p_{T_\infty} := 0$; $q := 0$
repeat
  if $q > 0$ **then**
    $\mathcal{V}_p := \{v \in V | f(v) = \tau\}$; $q := q - |\mathcal{V}_p|$
  endif
  $\mathcal{V}_i := \{v \in \mathcal{V} | v$ ima vse vhodne podatke$\}$;
  — Množica izvršljivih točk je $\mathcal{V}_i = \mathcal{K}(\tau) \cup \mathcal{V}_n$, kjer sta $\mathcal{K}(\tau)$ in $\mathcal{V}_n$
  — množici kritičnih oz. nekritičnih točk na globini τ.
  $q := q + |\mathcal{K}(\tau)|$ — Sprožijo se vse kritične točke.
  if $q < \tilde{p}_{T_\infty}$ **then**
    Bodi $\mathcal{V}_d \subset \mathcal{V}_n$, kjer je $|\mathcal{V}_d| \leq \tilde{p}_{T_\infty} - q$;
    $q := q + |\mathcal{V}_d|$ — Sproži se nekaj dodatnih, naključno izbranih točk.
  endif
  forall $v \in \mathcal{K}(\tau) \cup \mathcal{V}_d$ **do** $s(v) := \tau$ **endforall**
  $p_{T_\infty} := \max(q, p_{T_\infty})$; $\tau := \tau + 1$
until $\tau = T_\infty$;

Algoritem `pOptSinh`: Konstruiranje p-optimalne sprožitvene funkcije.

p_{T_∞}	$p = 3/4p_{T_\infty}$	$p = 1/2p_{T_\infty}$	$p = 1/4p_{T_\infty}$
Sprožitvena funkcija s_e			
4	0.030	0.223	1.232
5	0.016	0.121	0.454
6	0.006	0.205	0.621
7	0.004	0.104	0.761
8	0.015	0.147	0.883
9	0.006	0.071	0.397
10	0.003	0.100	0.464
$\sum$	0.011	0.139	0.687
T-optimalna sprožitvena funkcija			
4	0.011	0.211	1.079
5	0.002	0.048	0.375
6	0.002	0.117	0.537
7	0.000	0.029	0.718
8	0.000	0.044	0.789
9	0.000	0.001	0.324
10	0.000	0.018	0.311
$\sum$	0.002	0.067	0.590

Tabela 4: Upad idealne pospešitve D_p.

v $\mathcal{V}_i$), po naraščajočih $t(v)$ in po padajočih $t(v)$. Poskusi so pokazali, da nobena od strategij ni bila izrazito boljša od ostalih.

2.1.2 p-optimalna sprožitvena funkcija

Algoritem `pOptSinh` za dani GPP ter pripadajoči najkrajši čas izvrševanja T_∞ vrne sprožitveno funkcijo s, ki zagotavlja izvrševanje GPP na številu procesorjev, ki je kar se da blizu ocene $\tilde{p}_{T_\infty}$. Algoritem v vsakem trenutku τ poskuša sprožiti čimveč, toda kvečjemu $\tilde{p}_{T_\infty}$ izvršljivih točk. V vsakem trenutku τ sprožimo vse točke iz $\mathcal{K}(\tau)$. Če je zasedenih procesorjev še vedno manj kot $\tilde{p}_{T_\infty}$, na njih sprožimo nekatere nekritične točke, ki jih izberemo naključno (v vrstnem redu, kot prihajajo v $\mathcal{V}_i$). Iz povedanega je očitno, da je pri tem odločilna natančnost $\tilde{p}_{T_\infty}$.

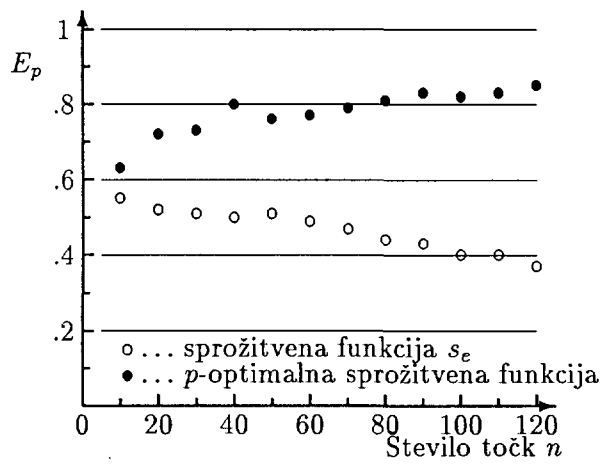
Optimalnost algoritma smo ocenjevali z odstotkom primerov, ko je p_{T_∞} dosegel oceno $\tilde{p}_{T_\infty,R}$, $\tilde{p}_{T_\infty,\kappa'}$ oziroma $\tilde{p}_{T_\infty,FB}$, saj natančne vrednosti p_{T_∞} ne poznamo. Algoritem je bil preverjen[7] nad 1.500 GPP in je dosegel rezultate, prikazane v tabeli 5.

$\tilde{p}_{T_\infty,R}$	$\tilde{p}_{T_\infty,\kappa'}$	$\tilde{p}_{T_\infty,FB}$
0.705	0.783	0.824

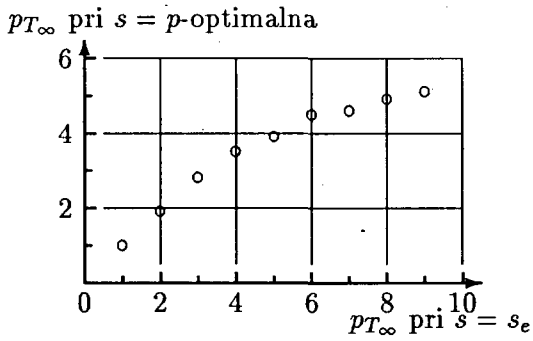
Tabela 5: Optimalnost algoritma `pOptSinh`.

Pri problemu optimizacije procesorjev je ključen podatek izkoriščenost procesorjev E_p, za katero želimo, da se čimbolj približa vrednosti 1. Slika

[7] V ta namem smo uporabili računalnik IBM PC in razvili ustrezna programska orodja.



Slika 1: Izkoriščenost procesorjev E_p.



Slika 2: Potrebe po procesorjih.

1 prikazuje izkoriščenost procesorjev E_p v odvisnosti od velikosti GPP za asinhro podatkovno vodeno računanje (sprožitvena funkcija s_e) in sinhronizirano podatkovno vodeno računanje (p-optimalna sprožitvena funkcija). Računanje, ki sledi p-optimalni sprožitveni funkciji, ima mnogo manjše zahteve po procesorjih kot računanje, ki sledi s_e sprožitveni funkciji, kar potrjujejo rezultati, prikazani na sliki 2. Tudi tu smo analizirali 1.500 GPP, pri čemer smo spreminjali število točk $n \leq 120$ in čas njihovega izvrševanja $t(v) \leq 10$.

2.2 Dodeljevanje

Po končani fazi sinhroniziranja je GPP pridružena sprožitvena funkcija s in skupaj tvorita $SGPP(p,T)$. S tem so vse $v \in \mathcal{V}$ urejene po trenutkih sprožitve $s(v)$. Takšna urejenost zagotavlja, da se GPP izvrši na p procesorjih v času T ob uporabi najsplošnejšega algoritma za dodeljevanje (algoritem `NakljDod`) [7], ki točko v dodeli *naključno izbranemu* prostemu procesorju.

vhod: $SGPP(p,T)$.
izhod: $\pi(v)$, za vsak $v \in \mathcal{V}$.

Razvrsti pare $(v, s(v))$ po naraščajočih $s(v)$ in jih shrani v sklad S.
for $q := 1$ **to** p **do** $F[q] := 0$ **endfor**
repeat
  Vzemi iz S vse pare z enakim s in jih shrani v $\mathcal{W}$.
  $P := \{q \mid F[q] \leq s\}$
  forall $v \in \mathcal{W}$ **do**
    Naključno izberi $q \in P$.
    $\pi(v) := q$;
    $F[q] := f(v)$; $P := P - \{q\}$
  endforall
  $W := \emptyset$; $P := \emptyset$
until $S = \emptyset$;

Algoritem `NakljDod`: Naključno dodeljevanje.

Po končanem dodeljevanju je vsaki točki $v \in \mathcal{V}$ pridružen procesor z *indeksom* $\pi(v)$, kjer je $1 \leq \pi(v) \leq p$. V vsakem trenutku je na voljo vsaj toliko prostih procesorjev, kot je točk, ki se morajo tedaj sprožiti. Dodeljevanje teh točk je poljubno, kar pomeni, da moremo v splošnem dani $SGPP(p,T)$ porazdeliti na več načinov. Vse dodelitve so enakovredne, saj vse zagotavljajo, da se GPP izvrši na p procesorjih v času T.

Točki u in v iz $\mathcal{V}$ sta *ločeni*, če sta dodeljeni različnima procesorjema. Povezavo med ločenima točkama imenujemo *globalna* povezava. V primeru hibridne arhitekture poteka komunikacija med ločenima točkama preko nadzorno-povezovalne enote in v splošnem zahteva čas $t_c > 0$. V takšnem primeru najsplošnejši algoritem torej ne vrača več enakovrednih dodelitev, saj se število globalnih povezav spreminja. Že prej smo videli, da število globalnih povezav vpliva na čas izvrševanja GPP. Zato se bomo osredotočili na konstruiranje takšnih algoritmov za dodeljevanje, ki bodo minimizirali število globalnih povezav v dodelitvah [8]. Točneje: trivialni kriterij naključnega dodeljevanja prostih procesorjev bomo nadomestili s kompleksnejšim, ki se glasi:

Dodeli točke w množice $\mathcal{W}$ procesorjem q množice P tako, da bo $\sum c(w,q)$ maksimalna, kjer je $c(w,q)$ število sosednjih točk točke w, ki so bile doslej dodeljene procesorju q.

vhod: $SGPP(p,T)$.
izhod: $\pi(v)$, za vsak $v \in \mathcal{V}$.

Razvrsti pare $(v, s(v))$ po naraščajočih $s(v)$ in jih shrani v sklad S.
for $q := 1$ **to** p **do** $F[q] := 0$ **endfor**
repeat
  Vzemi iz S vse pare z enakim s in jih shrani v $\mathcal{W}$.
  $P := \{q \mid F[q] \leq s\}$
  forall $v \in \mathcal{W}$ **do**
    forall $q \in P$ **do**
      $c(v,q)$ = število neposrednih predhodnikov od v, ki so bili dodeljeni v q-ti procesor.
    endforall
  endforall
  Reši problem WBM za graf $(\mathcal{W} \cup P, \mathcal{W} \times P)$.
  forall pare (v,q), ki so del rešitve **do**
    $\pi(v) := q$;
    $F[q] := f(v)$
  endforall
  $W := \emptyset$; $P := \emptyset$
until $S = \emptyset$;

Algoritem `MinGlDol`: Minimizacija globalnih povezav (navzdol).

Dodeljevanje, ki poteka v skladu z zgornjim pravilom, je v nekem smislu konservativno, saj poskuša točko pridružiti procesorju, v katerem je največ njenih sosedov. Opisano pravilo je primer znanega problema UTEŽENEGA DVODELNEGA UJEMANJA (WBM[8]) [10, 11]:

Naj bo dan dvodelni graf $G = (\mathcal{W} \cup P, E)$, kjer je $E \subseteq \mathcal{W} \times P$ ter cenovna funkcija $c : E \longrightarrow \mathbb{N}$. Iščemo ujemanje $M \subseteq E$ (množico povezav M, v kateri noben par povezav nima skupne točke) tako, da je $\sum_{(w,q)\in M} c(w,q)$ maksimalna.

Problem WBM znamo rešiti v času $\mathcal{O}(ne \log n / \max(1, \log \frac{e}{n}))$, kjer je $e = \mid E \mid$ in $n = \mid \mathcal{W} \mid + \mid P \mid$.

V algoritmu `MinGlDol` poteka dodeljevanje od začetnih točk proti končnim, medtem ko poteka dodeljevanje v algoritmu `MinGlGor` v nasprotni smeri. V obeh algoritmih se pojavlja problem WBM, pri čemer v prvem primeru določajo ceno $c(w,q)$ neposredni predhodniki točke w, v drugem pa njeni neposredni nasledniki.

2.3 Primeri

Z nekaterimi primeri bomo pokazali učinkovitost časovne optimizacije asinhronega računanja s pomočjo mehanizmov sinhronizacije. Obravnavali bomo naslednje primere: hitro Fourierjevo transformacijo (FFT), dinamično analizo scene (DAS) in LU razcep matrike (LU). Privzeli bomo hibridno arhitekturo s $p = 3$ procesorji (P_1, P_2 in P_3), katere medprocesorsko komunikacijo t_c bomo ustrezno spreminjali. Kvaliteto naših rezultatov, ki jih dobimo z uporabo sinhronizacijskega algoritma `TOptSinh` ter dodeljevalnih algoritmov `MinGlDol` in `MinGlGor`, bomo primerjali z naslednjimi znanimi razvrščevalnimi algoritmi: `CPM`[9] [12], `HNF`[10] [13] in `WL`[11] [13].

2.3.1 FFT: Hitra Fourierjeva transformacija

Povrnimo se na primer izračuna hitre Fourierjeve transformacije na 8 točkah, ki smo ga že obravnavali v prvem delu. Predpostavimo, da se točke A, C, E, G, I, J, M, N, Q, R, S in T izvršujejo eno časovno enoto, točke B, D, F, H, K, L, O, P, U, V, W in X pa pet časovnih enot.

`CPM`, `HNF` in `WL` algoritmi. Za primer FFT algoritma dajejo vse tri metode (`CPM`, `HNF`, `WL`)

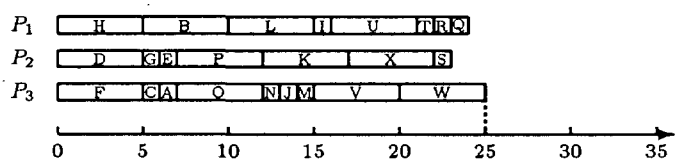
[8] Weighted Bipartite Matching

[9] Critical Path Method
[10] Heavy Node First
[11] Weighted Length

vhod: $SGPP(p,T)$.
izhod: $\pi(v)$, za vsak $v \in V$.

Razvrsti pare $(v, s(v))$ po padajočih $s(v)$ in jih shrani v sklad S.
for $q := 1$ **to** p **do** $F[q] := T$ **endfor**
repeat
  Vzemi iz S vse pare z enakim f in jih shrani v $\mathcal{W}$.
  $P := \{q \mid F[q] \geq f\}$;
  forall $v \in \mathcal{W}$ **do**
    forall $q \in P$ **do**
      $c(v,q)$ = število neposrednih naslednikov od v, ki so bili dodeljeni v q-ti procesor.
    endforall
  endforall
  Reši problem WBM za graf $(\mathcal{W} \cup P, \mathcal{W} \times P)$.
  forall pare (v,q), ki so del rešitve **do**
    $\pi(v) := q$
    $F[q] := s(v)$;
  endforall
  $\mathcal{W} := \emptyset$; $P := \emptyset$;
until $S = \emptyset$

Algoritem `MinGlGor`: Minimizacija globalnih povezav (navzgor).



Slika 3: FFT: Razvrstitev po `CPM`, `HNF` in `WL`.

enakovredne rezultate. Točke se dodelijo procesorjem, kot je prikazano na sliki 3. Razvidno je, da dobimo v tem primeru 22 globalnih povezav. Upad idealne pospešitve D_p se pri $t_c > 0$ poveča, tako dobimo pri $t_c = 10$ čas izvrševanja $T_p = 43$, kar se odraža tudi v povečanju upada idealne pospešitve $D_p = 1.87$. Podrobnejši rezultati `CPM`, `HNF` in `WL` dodeljevanja v odvisnosti od časa t_c so prikazani v tabeli 6.

t_c	T_p	S_p	E_p	D_p
0	25	2.88	0.96	0.67
2	27	2.67	0.89	0.80
4	31	2.32	0.77	1.07
10	43	1.67	0.56	1.87

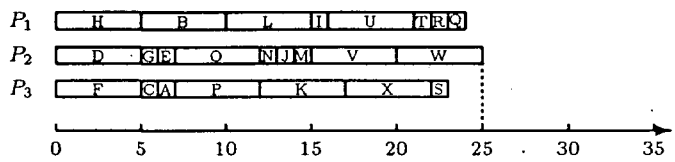
Tabela 6: FFT: `CPM`, `HNF` in `WL` dodeljevanje pri različnih t_c.

`MinGlDol` in `MinGlGor` algoritma. *GPP* FFT

v	$s(v)$	v	$s(v)$	v	$s(v)$
A	6	I	15	Q	23
B	5	J	13	R	22
C	5	K	12	S	22
D	0	L	10	T	21
E	6	M	14	U	16
F	0	N	12	V	15
G	5	O	7	W	20
H	0	P	7	X	17

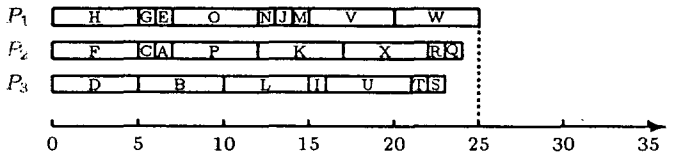
Tabela 7: FFT: T-optimalna sprožitvena funkcija.

algoritma najprej sinhroniziramo, za kar uporabimo algoritem `TOptSinh`, ki vrne T-optimalno sprožitveno funkcijo. Rezultat je prikazan v tabeli 7.



Slika 4: FFT: Razvrstitev po `MinGlDol`.

V drugem koraku uporabimo dodeljevalni algoritem `MinGlDol` oz. `MinGlGor`. Razvrstitev točk po `MinGlDol` algoritmu je prikazan na sliki 4, medtem ko je razvrstitev po `MinGlGor` algoritmu prikazana na sliki 5.



Slika 5: FFT: Razvrstitev po MinGlGor.

Z uporabo algoritmov MinGlDol in MinGlGor dobimo pri $t_c = 0$ enake rezultate kot pri algoritmih CPM, HNF in WL ($T_p = 25$, $S_p = 2.88$, $D_p = 0.667$ in $E_p = 0.96$). Pri $t_c > 0$ pa naša algoritma vračata rezultate z manjšim upadom idealne pospešitve D_p. Tako je pri $t_c = 10$ čas izvrševanja $T_p = 40$ (v prejšnjem primeru 43), kar pomeni, da je D_p le 1.67. Podrobnejši rezultati MinGlDol in MinGlGor dodeljevanja v odvisnosti od časa t_c so prikazani v tabeli 8.

t_c	T_p	S_p	E_p	D_p
0	25	2.88	0.96	0.67
2	25	2.88	0.96	0.67
4	28	2.57	0.86	0.87
10	40	1.80	0.60	1.67

Tabela 8: FFT: MinGlDol in MinGlGor dodeljevanje pri različnih t_c.

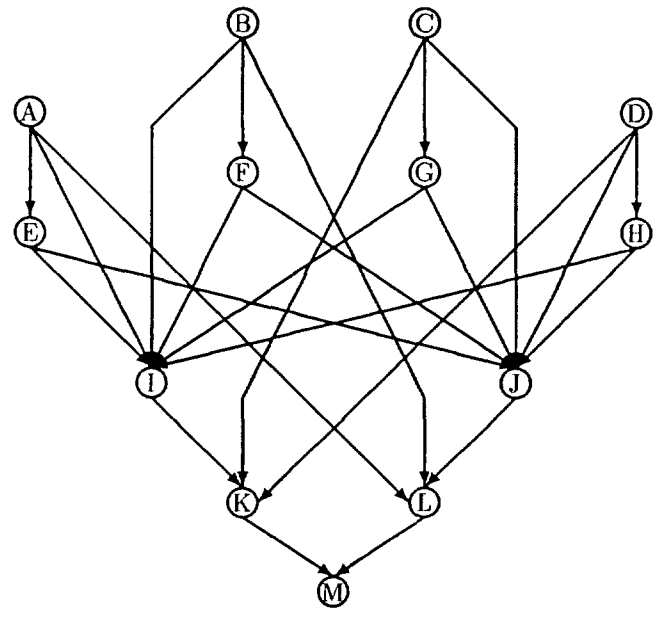
Oba algoritma občutno zmanjšata število globalnih povezav, tako dobimo pri MinGlDol algoritmu 16 globalnih povezav, algoritem MinGlGor pa nam število globalnih povezav zmanjša celo na 15.

2.3.2 DAS: Dinamična analiza scene

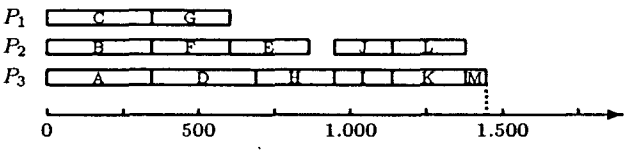
Za naslednji primer vzemimo algoritem za dinamično analizo scene. Algoritem DAS se uporablja pri izločanju podob gibljivih objektov in je podrobneje opisan v [14]. Pripadajoči *GPP* prikazuje slika 6. Točke A, B, C in D se izvršujejo 345 časovnih enot, točke E, F, G in H 259 časovnih enot, točki I in J 190 časovnih enot, točki K in L 241 časovnih enot in točka M 69 časovnih enot.

CPM, HNF in WL algoritmi. Tudi v primeru DAS algoritma dajejo metode (CPM, HNF in WL) enakovredne rezultate (slika 7).

MinGlDol in MinGlGor algoritma. Oba algoritma vračata enake rezultate kot algoritmi CPM, HNF in WL, torej pri $t_c = 0$ dobimo $T_p = 1449$, $S_p = 2.31$, $D_p = 0.31$ in $E_p = 0.77$ ter 12 globalnih povezav. Z algoritmom MinGlGor dobimo tudi enako dodelitev med procesorje (slika 7), medtem ko dobimo z algoritmom MinGlDol sicer drugačno dodelitev med procesorje (slika 8), ki pa, kot rečeno, rezultira v enako število globalnih povezev. Podrobnejša analiza pri $0 \leq t_c \leq 300$ je prikazana v tabeli 9.



Slika 6: *GPP* DAS algoritma.



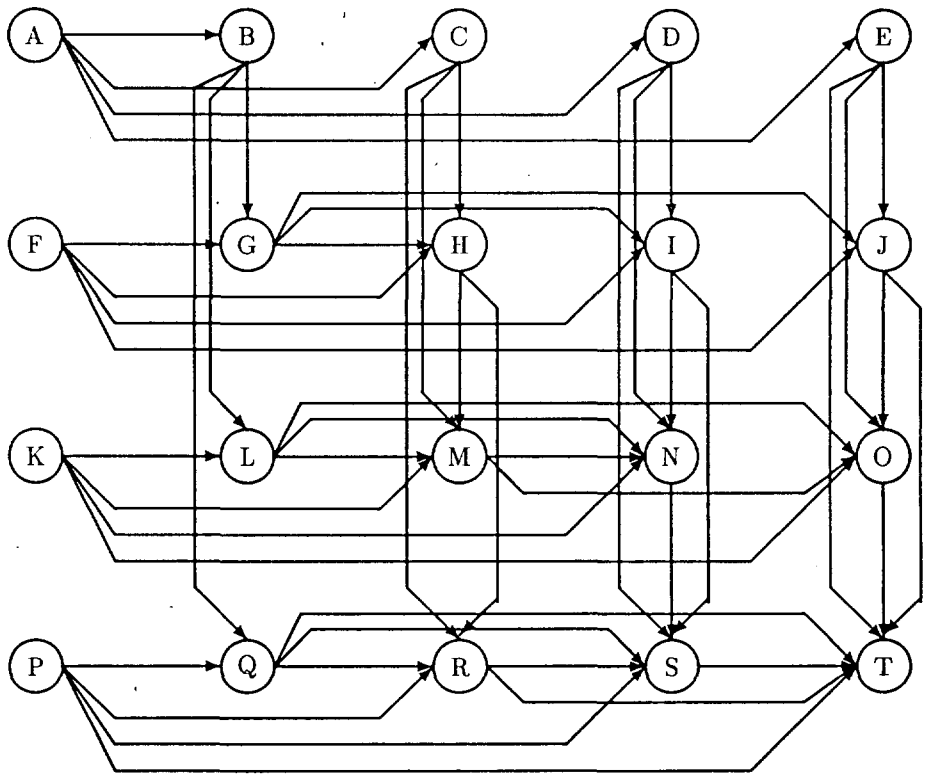
Slika 7: DAS: Razvrstitev po CPM, HNF, WL in MinGlGor.

t_c	T_p	S_p	E_p	D_p
0	1449	2.31	0.77	0.31
100	1649	2.03	0.68	0.49
200	1849	1.81	0.60	0.68
300	2049	1.63	0.54	0.86

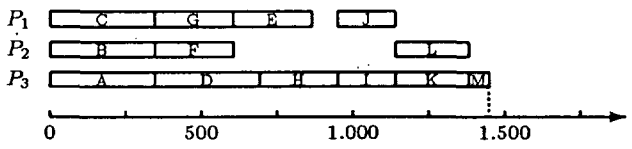
Tabela 9: DAS: CPM, HNF, WL, MinGlGor in MinGlDol dodeljevanje pri različnih t_c.

2.3.3 LU: LU razcep matrike

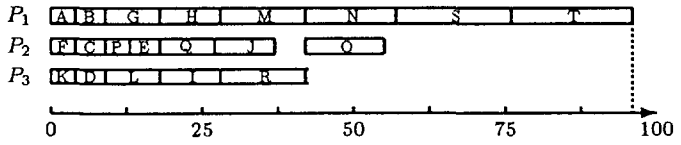
Za zadnji primer vzemimo algoritem za LU razcep matrike [15], ki je eden od najpogosteje uporabljenih algoritmov za reševanje sistema linearnih enačb. *GPP* LU algoritma je podan na sliki 9. Točke A, F, K in P se izvršujejo 4 časovne enote, točke B, C, D in E 5 časovnih enot, točke G, L in Q 9 časovnih enot, točke H, I in J 10 časovnih enot,



Slika 9: *GPP* algoritma za LU razcep matrike.



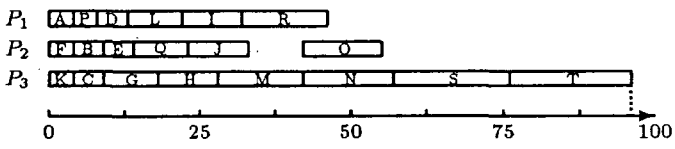
Slika 8: DAS: Razvrstitev po `MinGlDol`.



Slika 10: LU: Razvrstitev po `CPM` in `WL`.

točka O 13 časovnih enot, točki M in R 14 časovnih enot, točka N 15 časovnih enot, točka S 19 časovnih enot in točka T 20 časovnih enot.

`CPM`, `HNF` in `WL` algoritmi. LU razcep matrike je primer, da `HNF` algoritem vrača boljše rezultate od ostalih dveh. Razvrstitev točk po `CPM` oz. `WL` metodi je prikazana na sliki 10, medtem



Slika 11: LU: Razvrstitev po `HNF`.

ko razvrstitev po `HNF` metodi na sliki 11. Pri analizi LU algoritma smo t_c spreminjali med 0 in 30 (tabeli 10 in 11). Metodi `CPM` in `WL` sta vrnili 36 globalnih povezav, medtem ko je `HNF` vrnila le 32 globalnih povezav. Zato je upad idealne pospešitve D_p pri $t_c = 10$ za 12.5% boljši.

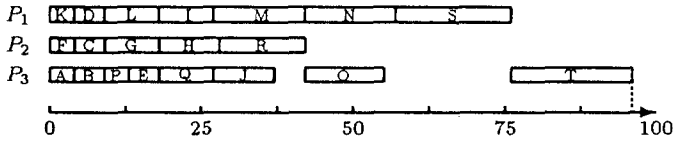
t_c	T_p	S_p	E_p	D_p
0	96	1.96	0.65	0.10
10	126	1.49	0.50	0.45
20	166	1.13	0.38	0.91
30	206	0.91	0.30	1.37

Tabela 10: LU: `CPM` in `WL` dodeljevanje pri različnih t_c.

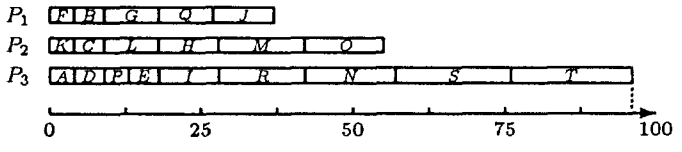
t_c	T_p	S_p	E_p	D_p
0	96	1.96	0.65	0.10
10	122	1.54	0.51	0.40
20	162	1.16	0.39	0.86
30	202	0.93	0.31	1.32

Tabela 11: LU: HNF dodeljevanje pri različnih t_c.

MinGlDol in MinGlGor algoritma. Še boljše rezultate kot HNF pa vračata naša algoritma MinGlDol in MinGlGor. Razvrstitvi točk po MinGlDol ter MinGlGor sta opisani na slikah 12 in 13.



Slika 12: LU: Razvrstitev po MinGlDol.



Slika 13: LU: Razvrstitev po MinGlGor.

Pri $t_c \geq 0$ dosežeta obe metodi enak upad idealne pospešitve, ki je boljši od upada idealnih pospešitev metod CPM, HNF in WL (tabela 12).

2.3.4 Primerjava rezultatov

Na osnovi povedanega sledi, da algoritma MinGlDol in MinGlGor dodeljujeta GPP ob manjšem (kvečjemu enakem) številu globalnih povezav. Analiza FFT, DAS in LU algoritmov je potrdila našo domnevo, da je število globalnih povezav v tesni zvezi z upadom idealne pospešitve. S tem je upravičena naša odločitev, da smo se pri oblikovanju algoritmov za dodeljevanje osredotočili na minimizacijo števila globalnih povezav. To minimizacijo opravljata algoritma MinGlDol in MinGlGor, ki za svoje delo potrebujeta sinhroniziran GPP. Slednjega pa konstruiramo z enim od naših algoritmov TOptSinh oz. pOptSinh. S takšno minimizacijo je dosežen zastavljeni cilj, tj. časovna optimizacija asinhronega procesiranja.

Algoritma MinGlDol in MinGlGor sta bila preizkušena nad 500 GPP, pri čemer smo spreminjali: število točk $n \leq 120$, čas njihovega izvrševanja

t_c	T_p	S_p	E_p	D_p
0	96	1.96	0.65	0.10
10	112	1.68	0.56	0.29
20	152	1.24	0.41	0.75
30	192	0.98	0.33	1.21

Tabela 12: LU: MinGlDol in MinGlGor dodeljevanje pri različnih t_c.

t_c	MinGlGor $(D_p/D_p^{\uparrow})$	MinGlDol $(D_p/D_p^{\downarrow})$
5	1.3729	1.3623
10	1.2216	1.2199
20	1.1260	1.1187

Tabela 13: Relativna kvaliteta MinGlDol in MinGlGor algoritmov.

$t(v) \leq 10$ in medprocesorsko komunikacijo $t_c \leq 20$. Število procesorjev smo omejili na $p = 1/2p_{T_\infty}$. Kvaliteto algoritmov MinGlDol in MinGlGor smo ocenjevali[12] glede na algoritem NakljDod. Rezultati so zbrani v tabeli 13, kjer so D_p, $D_p^{\uparrow}$ in $D_p^{\downarrow}$ upadi idealne pospešitve pri algoritmih NakljDod, MinGlGor in MinGlDol.

3. Organizacija stroja

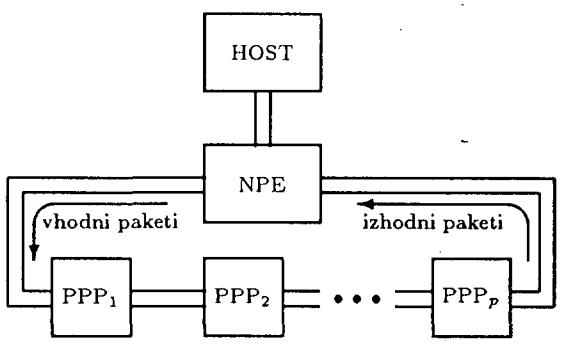
Končno bomo predstavili še računalniško arhitekturo, ki učinkovito podpira podatkovno vodeno računanje, oplemeniteno s sinhronizacijskimi mehanizmi, kot so bili vpeljani v prejšnjem poglavju. Takšna arhitektura v popolnosti izkorišča rezultate časovne optimizacije asinhronega procesiranja.

3.1 Funkcionalni opis

Že uvodoma smo opozorili na problem kopičenja podatkovnih paketov v vrsti pred procesno enoto, ki je posledica omejenega števila procesorjev v procesni enoti. Nakazali smo tudi eno od možnih rešitev, ki temelji na pravilnem vstopanju paketov v vrsto oz. izstopanju paketov iz nje. Na osnovi vpeljanih sinhronizacijskih mehanizmov je to rešitev možno uresničiti z vzporedno računalniško arhitekturo, kot bo opisana v nadaljevanju; poimenovali jo bomo *sinhronizirana podatkovno pretokovna arhitektura* (SPPA) [7, 16]. SPPA sodi v družino hibridnih arhitektur. Ses-

[12] V ta namem smo uporabili računalnik IBM PC in razvili ustrezna programska orodja.

tavlja jo množica enakih *podatkovno pretokovnih procesorjev* (PPP) ter *nadzorno-povezovalna enota* (NPE), ki so krožno povezani, kot kaže slika 14. Vhodno/izhodna komunikacija z gostiteljskim računalnikom poteka preko NPE.



Slika 14: SPPA.

Preden se lotimo podrobnejšega opisa enot, na kratko opišimo potek reševanja danega problema na SPPA. Reševanje si lahko predstavljamo kot zaporedje treh faz: prevajanje programa ter optimizacija *GPP*, nalaganje *SGPP* v SPPA in izvrševanje v SPPA.

Prevajanje ter optimizacija. Naj bo dan nek poljuben algoritem, zapisan v kakem od visokih podatkovno pretokovnih jezikov. Z ustreznim prevajalnikom se algoritem prevede v strojni kod, tj. *GPP*. Prevajanje poteka na gostiteljskem računalniku. V naslednjem koraku *GPP* optimiziramo, tj. konstruiramo p- ali T-optimalno sprožitveno funkcijo (`pOptSinh` in `TOptSinh`) ter z uporabo algoritmov `MinGlGor` oz. `MinGlDol` določimo procesorske indekse π. Ti indeksi so pomemebni za samo nalaganje *GPP* v SPPA. Poleg tega pa so ob koncu optimizacije znane še vse globalne povezave ter trenutki sprožitve vseh tistih točk (*vhodne ločene* točke), v katere vstopajo globalne povezave. Ti podatki imajo pomembno vlogo pri izvrševanju *GPP* v SPPA.

Nalaganje. Kot bomo videli v nadaljevanju, ima vsak PPP svoj grafni pomnilnik, v katerem hrani podatke o delu *GPP*. V fazi nalaganja se vsaka točka v iz *GPP* naloži v PPP z indeksom i, če je $\pi(v) = i$. V PPP s tem indeksom se implicitno vpišejo tudi vse povezave, katerih začetna in končna točka sta dodeljeni procesorju z indeksom i, kot narekuje logična zgradba grafnega pomnilnika. V NPE pa se shranijo globalne povezave ter trenutki sprožitve vseh vhodnih ločenih točk. Pri nalaganju sodelujeta gostiteljski računalnik ter NPE.

Izvrševanje. Po fazi nalaganja vsak PPP hrani ustrezen podgraf od *GPP*. Vse točke v PPP, ki za svojo izvršitev potrebujejo vsaj en podatkovni paket iz drugega podgrafa, se imenujejo vhodne ločene točke. Podobno pa so *izhodne ločene* točke vse tiste točke podgrafa, ki vsaj en svoj rezultat posredujejo v drug podgraf. Vse ostale točke v podgrafu imenujemo *notranje* točke. Izmenjava podatkov med notranjimi točkami poteka asinhrono, zato je podgraf *čisti* graf pretoka podatkov brez vnaprej določenih trenutkov sprožitve. Tudi izhodna ločena točka posreduje svoj rezultat takoj, ko se le ta izračuna.

Izmenjava podatkovnih paketov med PPP poteka posredno preko NPE. Vsaka izhodna ločena točka torej pošlje svoj paket v NPE. Ta enota prebere iz seznama globalnih povezav, kateri vhodni ločeni točki je paket namenjen. Poleg tega pa iz seznama sprožitvenih trenutkov ugotovi, kdaj mora ta paket poslati ustreznemu PPP. Vidimo, da NPE deluje popolnoma sinhrono na osnovi globalne ure.

3.2 Nadzorno-povezovalna enota

NPE opravlja naslednje funkcije: skrbi za vhodno/izhodno komunikacijo z gostiteljskim računalnikom, omogoča nalaganje delov *GPP* v ustrezne PPP in skrbi za posredovanje paketov med ločenimi točkami. Logično si NPE predstavljamo kot tabelo

Globalna povezava	Procesorski indeks	Podatkovno polje	Trenutek sprožitve
$u \longmapsto v$	$\pi(v)$	p	$s(v)$

kjer je $u \longmapsto v$ globalna povezava iz točke u v v, $\pi(v)$ indeks PPP, v katerem je vhodna ločena točka v, p podatkovno polje, ki ga točka u pošilja točki v, ter $s(v)$ trenutek sprožitve točke v. Podatkovno polje p je par $p = (d, a)$, kjer je d vrednost, a pa določa, kateri od argumentov točke v bo sprejel vrednost d.

Tabela se pred začetkom izvrševanja uredi po naraščajočih vrednostih $s(v)$, ki so bile določene

v fazi optimizacije. Delovanje NPE opišimo na primeru. Naj bo $A \longmapsto B$ globalna povezava. Točka A naj bo dodeljena procesorju i, točka B pa procesorju j. Točka B se mora sprožiti v trenutku t. V nekem trenutku $\tau < t$ točka A pošlje podatkovno polje p v izhodnem paketu oblike

NPE	$A \longmapsto B$	p

v NPE. Ta na osnovi globalne povezave $A \longmapsto B$ poišče v tabeli vrstico s to globalno povezavo ter v podatkovno polje vpiše vrednost p. S tem je sprejem izhodnega paketa končan.

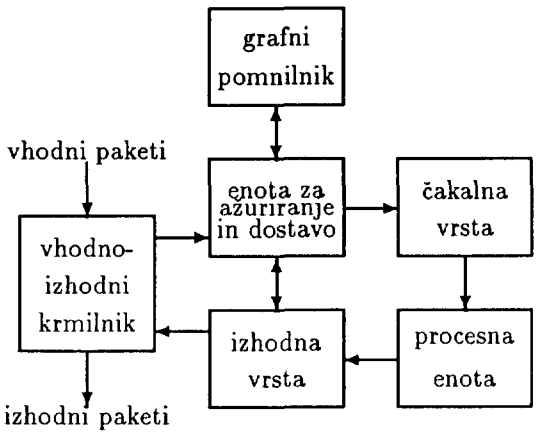
Sočasno globalna ura enote NPE šteje čas τ. Ko postane $\tau = t$, NPE tvori vhodni paket oblike

j	B	p

in ga pošlje verigi PPP, kjer ga j-ti PPP sprejme.

3.3 Podatkovno pretokovni procesor

Podatkovno pretokovni procesor omogoča hranjenje in izvajanje podgrafov GPP ter hkrati učinkovito komuniciranje s svojo okolico. Arhitektura je zasnovana tako, da je uporabljen sinhronizacijski mehanizem, ki temelji na shranjevanju paketov. Procesor sestavljajo *vhodno-izhodni krmilnik, grafni pomnilnik, enota za ažuriranje in dostavo, čakalna vrsta, procesna enota* ter *izhodna vrsta* (slika 15).



Slika 15: Zgradba PPP.

Vhodno-izhodni krmilnik skrbi za komunicajo med PPP in okolico. Vhodni podatkovni paketi, ki vstopajo v krmilnik, imajo obliko

$\pi(v)$	v	p

kjer je v točka, ki (v procesorju z indeksom $\pi(v)$) čaka na podatkovno polje $p = (d, a)$. Iz indeksa $\pi(v)$ krmilnik ugotovi, če je prispeli paket namenjen njemu. Če ni, ga vhodno-izhodni krmilnik nespremenjenega v istem ciklu posreduje naslednjemu PPP. Procesor je tako za tuje pakete transparenten. Če pa se $\pi(v)$ ujema z indeksom danega procesorja, se paket sprejme. V tem primeru enota za ažuriranje in dostavo iz paketa izloči indeks $\pi(v)$, preostanek

v	d	a

pa pošlje grafnemu pomnilniku.

Grafni pomnilnik hrani opise točk in povezav podgrafa GPP. Opis točke v se nahaja v tabeli

Točka	Op	Št.	Operandi			Točke čakajoče na rezultat		
v	op	c	d_1	...	d_m	w_1, f_1, a_1	...	w_n, f_n, a_n

kjer je op operacija, ki jo mora točka v izvršiti nad operandi $d_1, d_2, \ldots, d_m$ ter rezultat poslati točkam $w_1, w_2, \ldots, w_n$; rezultat mora biti shranjen v točko w_i kot argument z zaporedno številko a_i. Zastavica f_i zavzame vrednost L ali G glede na to, ali je točka w_i v istem (lokana točka, $f = L$) ali drugem PPP (globalna točka, $f = G$). Števec c manjkajočih vhodnih operandov se ob nalaganju postavi na vrednost m.

Enota za ažuriranje in dostavo vpiše vrednost d paketa v grafni pomnilnik v vrstico z oznako točke v na mesto operanda d_a ter hkrati zmanjša števec c za eno. Takoj po vpisu enota za ažuriranje in dostavo preveri, če je vrednost števca c enaka nič. Če je temu tako, posreduje paket oblike

v	op	d_1	...	d_m

v čakalno vrsto, kjer se paketi kopičijo in čakajo na sprostitev procesne enote. Sočasno pa na osnovi v-te vrstice grafnega pomnilnika tvori v izhodni vrsti naslednjo tabelo

Povezava	Zastavica	Rezultat	Št. argumenta
$v \longmapsto w_1$	f_1		a_1
$v \longmapsto w_2$	f_2		a_2
⋮			
$v \longmapsto w_n$	f_n		a_n

Ko procesna enota izračuna rezultat $r = op(d_1, \ldots, d_m)$, tvori paket

v	r

in ga pošlje izhodni vrsti. Ta vpiše vrednost r v polja za rezultat vseh vrstic, ki ta rezultat pričakujejo.

Enota za ažuriranje in dostavo na osnovi zastavice f_i tvori lokalni paket, če je $f_i = L$ oz. izhodni paket, če je $f_i = G$. Lokalni paket ima obliko

w_i	r	a_i

izhodni paket pa obliko

NPE	$v \longmapsto w_i$	d

kjer je $d = (r, a_i)$.

Vrednost r lokalnega paketa se preko enote za ažuriranje in dostavo vpiše v grafni pomnilnik v vrstico z oznako w_i na mesto operanda d_{a_i}. Izhodni paket pa se preko vhodno-izhodnega krmilnika posreduje NPE.

3.4 Izvedba

Zadnji korak do končne izvedbe SPPA predstavlja razvoj strojne opreme NPE in PPP.

V začetni fazi je moč NPE relativno preprosto emulirati s kakim od obstoječih mikroračunalnikov; končni cilj pa je izvedba NPE v obliki VLSI čipa.

Tudi PPP je smiselno zasnovati kot VLSI vezje, vendar pa že danes obstajajo na tržišču procesorji, ki bi v prvi fazi zadovoljivo opravljali funkcijo PPP. Najbliže temu je podatkovno vodeni mikroprocesor μPD7281 [17, 18]. Primeren kandidat, ki bi v prvi fazi prevzel funkcijo PPP, je transputer (npr. T9000) [19].

Za konec omenimo še možnost, da bi PPP nadomestili tudi s podatkovno vodenim procesorskim poljem. Še posebej so zanimiva heksagonalna procesorska polja, ki so trenutno še predmet raziskav [20]. Zgradbo procesorskega polja prikazuje slika 16. Podatkovno vodene *celice* (procesorji) so organizirane v vrstice. Med vsakim parom vrstic je speljano *komunikacijsko vodilo*, ki omogoča vhodno/izhodno komunikacijo z NPE ter porazdelitev delov GPP med celice. Vsaka celica je točkovno povezana s šestimi sosednjimi celicami. Slika 16a prikazuje povezanost celice 0 s sosedami 1, 2, 3, 4, 5 in 6. Iz tehnoloških razlogov je število celic v polju omejeno (trenutno do nekaj 100 [20]). Zato moramo v splošnem GPP porazdeliti po več poljih. Na primer, slika 16b prikazuje rezultat nalaganja tistega dela GPP za FFT, ki je ga je algoritem `MinGlDol` priredil procesorju P_2.

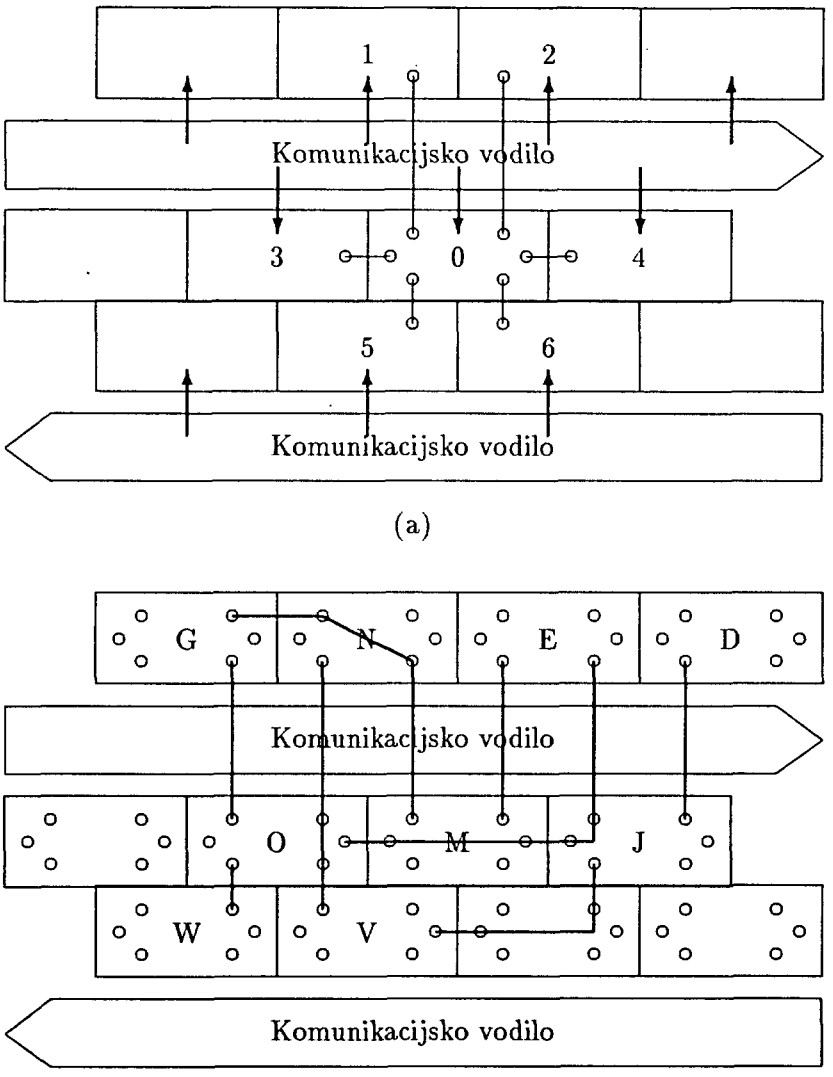
4. Zaključek

Povrnimo se ponovno na nalogo, ki smo si jo zastavili. Predpostavimo, da je na voljo podatkovno pretokovni računalnik s potencialno neskončnim številom procesorjev ter izberimo poljuben graf pretoka podatkov. Denimo, da je za najhitrejšo asinhrono izvršitev izbranega grafa potrebnih vsaj m procesorjev. Tedaj označimo s T_m najkrajši čas, v katerem se ta graf izvrši na m procesorjih. Na računalniku z $n < m$ procesorjev pa bi se v splošnem isti graf asinhrono izvršil v času $T_n = T_m + \Delta T_n$, kjer $\Delta T_n \geq 0$. Za nalogo si zadajmo minimizirati podaljšek izvrševanja ΔT_n, tj. časovno optimizirati asihrono procesiranje pri n danih procesorjih.

Časovno optimizacija asinhronega procesiranja se rešuje bodisi dinamično, tj. med izvrševanjem grafa pretoka podatkov, ali pa statično, tj. med prevajanjem programa v graf. Dinamično dodeljevanje je obremenjeno z "overheadom" ter zaradi lokalne optimizacije redko vodi v globalno (časovno in/ali prostorsko) optimalno izvrševanje grafa pretoka podatkov. Zato je smiselno čim večji del optimizacije prenesti v fazo prevajanja. K razrešitvi opisanega problema smo pristopili po izvirni poti. Rešitev, ki jo predlagamo, temelji na uvedbi nekaterih mehanizmov sinhronizacije v asinhrono procesiranje.

Graf pretoka podatkov s predhodno analizo opremimo z dodatno informacijo, ki bo koristila pri vlaganju grafa v računalnik ter med njegovim izvrševanjem. Postopek, imenovan statično dodeljevanje, poteka v dveh korakih:

Sinhronizacija: Naj bo $\mathcal{V}$ množica točk danega grafa pretoka podatkov ter označimo s $t(v)$ čas



Slika 16: Heksagonalno procesorsko polje.

izvrševanja točke $v \in \mathcal{V}$. Vsaki točki $v \in \mathcal{V}$ hevristično priredimo trenutek njene sprožitve $s(v)$ tako, da minimiziramo ΔT_n, kjer je $n < m$. K hevrističnim metodam se zatečemo zaradi NP-polnosti opisanega problema. Graf, katerega točke so opremljene s trenutki sprožitve, imenujemo sinhronizirani graf pretoka podatkov. Sinhronizirani graf je torej oplemeniten graf, v katerem je znano, kdaj oziroma v kakšnem vrstnem redu se morajo točkam pridruženi ukazi pričeti izvrševati, da se bo celoten graf izvršil na $n < m$ procesorjih v najkrajšem času.

Dodeljevanje: Vsaki $v \in \mathcal{V}$ priredimo indeks $\pi(v)$, $1 \leq \pi(v) \leq n$, ki pove, kateremu od n procesorjev se bo točka dodelila. Torej razbijemo $\mathcal{V}$ v n particij $\mathcal{V}_1, \mathcal{V}_2, \ldots \mathcal{V}_n$, kjer je $\mathcal{V}_i$ množica točk, ki se bodo izvrševale na i-tem procesorju. Točki imenujemo ločeni, če sta dodeljeni različnima procesorjema. Povezavo med ločenima točkama imenujemo globalna povezava. Množico $\mathcal{V}$ je v splošnem moč razbiti na več načinov, seveda pa se omejimo le na konstrukcijo dopustnih particij, ki omogočajo izvršitev grafa v skladu z določenimi trenutki sprožitve $s(v)$. Med dopustnimi particijami iščemo optimalno, tj. takšno, pri kateri bo najmanj globalnih povezav. Tako bo zagotovljena tudi najmanjša medprocesorska komunikacija. Rezultat dodeljevanja je, da imamo za vsako $v \in \mathcal{V}$ določen indeks $\pi(v)$ ter "označene" globalne povezave.

Sinhronizirani graf pretoka podatkov dobimo s pomočjo hevrističnih algoritmov `pOptSinh` in `TOptSinh`, za konstrukcijo optimalnih sprožitvenih funkcij. Po pesimistični oceni vračata predlagana algoritma optimalno rešitev v 80% primerov.

V algoritmu `pOptSinh` je uporabljena izvirna ocena za spodnjo mejo potrebnih procesorjev, ki temelji na razširjeni kritični vzporednosti grafa. Ta ocena je po svoji natančnosti v povprečju le za 6.52% slabša od optimalne Fernandez-Bussellove ocene, hkrati pa je njeno določanje za red velikosti hitrejše.

Za dodeljevanje predlagamo algoritma `MinGlDol` in `MinGlGor`, ki temeljita na optimizaciji medprocesorskih komunikacij. V primerjavi z znanimi razvrščevalnimi algoritmi dobimo s predlaganima algoritmoma boljše rezultate, kar potrjujejo analizirani primeri algoritmov za izračun hitre Fourierjeve transformacije, dinamične analize scene in LU razcepa matrike.

Končno je predlagana tudi organizacija računalnika, ki podpira asinhrono procesiranje z elementi sinhronizacije. Nakazane so možnosti realizacije z VLSI podatkovno pretokovnimi mikroprocesorji in podatkovno pretokovnimi polji.

Zahvala

Raziskavo je finančno podprlo Ministerstvo za znanost in tehnologijo Republike Slovenije po pogodbi C2-0521-106-92. Za strokovno pomoč pri nastanku pričujočega dela gre zahvala mag. Borutu Robiču, sodelavcu Laboratorija za računalniške arhitekture IJS.

Literatura

[1] E. B. Fernandez and B. Bussell. Bounds on the Number of Processors and Time for Multiprocessor Optimal Schedules. *IEEE Trans. Computers*, C-22(8):745–751, August 1973.

[2] Y. E. Chen and D. L. Epley. Bounds on Memory Requirements of Multiprocessing Systems. In *Proc. 6th Annu. Allerton Conf. Circut and Syst. Theory*, pages 523–531, 1968.

[3] R. McNaughton. Scheduling with Deadlines and Loss Functions. *Management Sci.*, 6, October 1959.

[4] T. C. Hu. Parallel Sequencing and Assembly Line Problems. *Oper. Res.*, 9(6):841–848, November 1961.

[5] C. V. Ramamoorthy, K. M. Chandy, and M. J. Gonzalez. Optimal Schedulling Strategies in a Multiprocessor System. *IEEE Trans. Computers*, C-21(2):137–146, February 1972.

[6] J. Šilc, B. Robič, and L. M. Patnaik. Performance Evaluation of an Extended Static Dataflow Architecture. *Computers and Artificial Intelligence*, 9(1):43–60, 1990.

[7] J. Šilc and B. Robič. Synchronous Dataflow-Based Architecture. *Microprocessing and Microprogramming*, 27(1-5):315–322, August 1989.

[8] J. Šilc and B. Robič. Program Graph Partitioning for Macro-Dataflow. In *Proc. ISSM Int'l Workshop on Parallel Computing*, pages 327–330, September 1991.

[9] J. Šilc. *Časovna optimizacija asihronega procesiranja z uvedbo nekaterih mehanizmov sinhronizacije.* Doktorska disertacija, Fakulteta za elektrotehniko in računalništvo, Ljubljana, junij 1992.

[10] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, 1976.

[11] K. Mehlhorn. *Graph Algorithms and NP-Completeness.* Springer-Verlag, 1984.

[12] E. G. Coffman et al. *Computer and Job-Shop Scheduling Theory.* Wiley-Interscience, 1976.

[13] B. Shirazi, M. Wang, and G. Pathak. Analysis and Evaluation of Heuristic Method for Static Task Schedulling. *Journal of Parallel and Distributed Computing*, 10(3):222–232, November 1990.

[14] G. Pathak and D. P. Agrawal. Task Division and Multicomputer System. In *Proc. 5th Int'l Conf. on Distributed Computing System*, page 273, May 1985.

[15] G. H. Golub and C. F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, 1989.

[16] J. Šilc and B. Robič. MADAME - Macro-Dataflow Machine. In *Proc. Mediterranean Electrotechnical Conference - MELECON'91*, pages 985–988, May 1991.

[17] T. Jeffery. The μPD7281 Processor. *Byte*, pages 237–246, November 1985.

[18] J. Šilc in B. Robič. Procesor s podatkovno pretokovno arhitekturo. *Informatica*, 10(4):74–80, 1986.

[19] D. Pountain. The Transputer Strikes Back. *Byte*, 16:265–275, August 1991.

[20] S. Weiss, I. Spillinger, and G. M. Silberman. Architectural Improvements for Data-Driven VLSI Processing Arrays. In *Proc. Functional Programming Languages and Computer Architecture*, pages 243–259, September 1989.

AN INFORMATIONAL APPROACH OF BEING-THERE AS UNDERSTANDING III

INFORMATICA 3/92

Keywords: dictionary, Heideggerian Being-there, information, informational formulas, philosophy, text formalization, understanding

Anton P. Železnikar
Volaričeva ulica 8, 61111 Ljubljana

This part of the essay includes the remaining dictionary of Fraktur operands and informational operators pertaining to the Heideggerian Being-there as understanding. These dictionaries complete the discussion concerning understanding as informational phenomenality in terms of informational formulas in which operands communicate among themselves in a dynamic, that is processing and operational way. This informational model of understanding approaches the concept of treating texts informationally in the most imaginable entirety. At the end a short comment is added.

Informacijski pristop k biti-tu kot razumevanju III
V tem delu spisa imamo še preostali del slovarja frakturnih operandov in slovar informacijskih operatorjev, ki se tičejo Heideggrove tu-biti kot razumevanja. S tema slovarjema se končuje razprava o razumevanju kot informacijski pojavnosti v terminih informacijskih formul, v katerih operandi medseboj dinamično komunicirajo v procesni in operacijski obliki. Ta informacijski model razumevanja se približuje konceptu informacijskega obravnavanja besedil v kar najbolj domiselni celosti. Na koncu je dodana še kratka opomba.

Fraktur Operands (A Continuation)

$\mathfrak{B}_{in}$ Being-in | In-Sein, das | v-bit | (2.2), (2.3), (2.4), (7.2), (10.11) □

$\mathfrak{B}_{in}(\mathfrak{U}(\mathfrak{W}_{world}))$ Being-in of understanding of the world | In-sein des Verstehens der Welt, das | v-bit razumevanja sveta | (10.11) □

$\mathfrak{B}_{in\text{-}the\text{-}world}$ Being-in-the-world | In-der-Welt-sein, das | bit-v-svetu | (2.4), (2.6), (2.8), (7.1), (7.2), (8.4), (10.1), (10.6), (10.10), (12.4), (16.5), (17.1), (17.7) □

$\mathfrak{B}_{one's\text{-}Self}$ Being-one's-Self | Selbstsein, das | bit-pri-sebi; samobit | (10.6) □

$\mathfrak{B}_{possible}$ Being-possible | Sein-können, das | moči-bit | (3.4), (3.6), (3.7), (4.4), (4.6) □

$\mathfrak{B}_{possible}(\mathfrak{D})$ Being-possible of Dasein | Möglichsein des Daseins, das | moči-bit tubiti | (4.6) □

$\mathfrak{B}_{there}$ Being-there | Sein des "Da", das | tu-bit | (1.1), (1.2), (1.6), (2.2), (2.4) □

$\mathfrak{B}_{there}(\mathfrak{W}_{world})$ Being- there of the world | Da-sein der Welt, das | biti-tu sveta | (2.2), (2.4) □

$\mathfrak{B}_{with}$ Being-with | Sein bei, das | pri-bit; bit pri | (12.5) □

$\mathfrak{B}_{with}(\omega)$ Being-with Others | Mitsein mit Anderen, das | bit pri drugih | (12.5) □

$\mathfrak{C}_{comport}$ comporting | Verhalten, das | vedénje | (8.6) □

$\mathfrak{C}_{comport}(\omega_{oneself})$ comporting oneself | Sichverhalten,

das | Se-vedénje | (8.6) □

$\mathfrak{C}_{const}$ constitution | Konstitution, die | konstitucija | (16.6) □

$\mathfrak{C}_{const}(\mathfrak{B})$ constitution of the Being | Konstitution des Seins, die | konstitucija biti | (16.6) □

$\mathfrak{D}$ Dasein | Dasein, das | tubit | (1.5), (1.5), (2.2), (2.3), (2.4), (2.8), (3.3), (3.4), (3.5), (3.6), (3.7), (3.11), (4.2), (4.3), (4.4), (4.5), (4.6), (5.1), (5.2), (5.3), (5.4), (5.5), (5.6), (5.7), (6.1), (7.9), (8.3), (8.5), (8.6), (8.7), (8.8),(8.11), (9.1), (9.2), (9.3), (10.2),(10.3), (10.5), (11.1), (12.1), (12.2), (12.3), (13.1), (16.1), (17.2), (17.3), (17.4), (17.5), (17.6), (17.7) □

$\mathfrak{D}_{discl}$ disclosure | Erschließen, das | razprtje | (7.1) □

$\mathfrak{D}_{fact}$ factical Dasein | faktische Dasein, das | faktična tubit | (11.1) □

$\mathfrak{E}_{exist}$ existing; existentiality | Existieren, das; Existenzialität, die | eksistiranje; eksistencialnost| (3.12), (12.5) □

$\mathfrak{E}_{exist}(\alpha)$ existence of entities | existierende Seiende, das | eksistirajoče bivajočega | (12.5) □

$\mathfrak{E}_{expl}$ explaining | Erklären, das | pojasnjevanje | (1.6) □

$\mathfrak{G}_{general}$ in general | überhaupt | nasploh | (14.7) □

$\mathfrak{G}_{grant}$ granting | Vorwegnehmen, das | vnaprejdanost | (16.3) □

$\mathfrak{G}_{grasp}$ grasping | Erfassen, das | zapopadanje | (8.10) □

$\mathfrak{G}_{grasp}(\mathfrak{U})$ grasping of understanding | Erfassen von Verstehen, das | zapopadanje razumevanja | (8.10) □

$\mathfrak{G}_{known}$ that_which_we_have_such_ _compentence_over; known | Gekonnte, das | znano | (3.2'), (3.2"), (3.2^3) □

$\mathfrak{I}_{inter}$ interpretation | Interpretation, die | interpretacija | (16.7) □

$\mathfrak{I}_{inter}(\mathfrak{T}_{temporal}(\mathfrak{B}))$ temporal interpretation of Being | temporale Seinsinterpretation, die | časovna interpretacija biti | (16.7) □

$\mathfrak{K}$ kind | Art, die | način | (3.3), (8.5), (8.11), (9.1), (15.4), (16.5), (18.1) □

$\mathfrak{K}(\mathfrak{B})$ kind of Being | Seinsart, die | način biti | (3.3), (8.5), (9.1), (18.1) □

$\mathfrak{K}(\mathfrak{B}(\alpha))$ kind of Being of entity | Seinsart des Seienden, die | način biti bivajočega | (16.5) □

$\mathfrak{K}(\mathfrak{B}(\mathfrak{D}))$ kind of Being of Dasein, the | Seinsart des Daseins, die | način biti tubiti | (8.11) □

$\mathfrak{K}(\mathfrak{S}_{see}(\mathfrak{B}))$ kind of seeing of Being | Art des Sehens von Sein, die | način videnja biti | (15.4) □

$\mathfrak{K}_{cogn}$ kind of cognition | mögliche Erkenntnisart, die | spoznavni način | (1.6) □

$\mathfrak{K}_{know}$ knowing; knowledge | Wissen, das | znanje | (5.4), (12.4) □

$\mathfrak{K}_{know}(\sigma_{self})$ knowledge of the Self | »Selbsterkenntnis«, die | samospoznanje | (12.4) □

$\mathfrak{K}_{know}(\mathfrak{D})$ knowing of Dasein | Wissen des Daseins, das | védenje tubiti | (5.4) □

$\mathfrak{M}$ mood | gestimmtes | razpoloženo | (1.4), (17.2), (17.3) □

$\mathfrak{M}(\mathfrak{U})$ mood of understanding | gestimmtes des Verstehens | razpoloženo razumevanja | (1.4) □

$\mathfrak{N}$ nature | Natur, die | narava | (7.6), (7.7) □

$\mathfrak{O}_{opaque}$ opaqueness | Undurchsichtigkeit, die | neprozornost | (13.1) □

$\mathfrak{O}_{opaque}(\mathfrak{D})$ Dasein's opaqueness | Undurchsichtigkeit des Daseins, die | neprozornost tubiti | (13.1) □

$\mathfrak{P}_{imm}$ self-perception, immanent | immanente Selbstwahrnehmung, die | imanentno samozaznavanje | (5.4) □

$\mathfrak{P}_{imm}(\mathfrak{D})$ immanent self-perception of Dasein | immanente Selbstwahrnehmung des Daseins, die | imanentno samozaznavanje tubiti | (5.4) □

$\mathfrak{P}_{perceive}$ perceiving | Wahrnehmen, das | zaznavanje | (14.3) □

$\mathfrak{P}_{percept}$ perceptually tracking down and inspecting | wahrnehmende Aufspüren und Beschauen, das |

zaznavno zasledovanje in ogledovanje | (12.4) □

$\mathfrak{P}_{percept}(\sigma_{self})$ perceptually tracking down and inspecting a point called the "Self" | das wahrnehmende Aufspüren und Beschauen eines Selbstpunktes | zaznavno zasledovanje in ogledovanje nekega Se | (12.4) □

$\mathfrak{P}_{project}$ projecting | Entwerfen, das | snovanje; projektiranje | (8.5), (8.6), (8.7), (8.10), (8.11), (16.3) □

$\mathfrak{P}_{project}(\alpha)$ what is projected | Entworfene, das | zasnovano | (8.10) □

$\mathfrak{Q}_{question}$ or $\mathfrak{Q}_{quest}$ question | Frage, die | vprašanje | (7.7), (17.5), (17.6), (17.7) □

$\mathfrak{Q}_{quest}(\mathfrak{B}(\mathfrak{D}))$ question about the Being of Dasein | Frage nach dem Sein des Daseins, die | vprašanje po biti tubiti | (17.5), (17.6) □

$\mathfrak{R}_{to\text{-}hand}$ ready-to-hand | Zuhandene, das | priročno | (7.4), (7.5) □

$\mathfrak{S}_{mind}$ state-of-mind | Befindlichkeit, die | počutnost (počutje, nahajalnost) | (1.1), (1.2), (1.3), (4.2), (5.6), (17.1), (18.1) □

$\mathfrak{S}_{of\text{-}Being}$ state-of-Being | In-der-Welt-sein, das | stanje-biti; svetovno-biti | (8.4) □

$\mathfrak{S}_{of\text{-}Being}(\Upsilon_{char}(\mathfrak{U}))$ state-of-Being of the character of understanding | In-der-Welt-sein des Charakters von Verstehen, das | stanje-biti karakterja razumevanja | (8.4) □

$\mathfrak{S}_{see}$ 'seeing' | »Sehen«, das | »videnje« | (14.3), (14.4), (14.6), (14.7), (15.4) □

$\mathfrak{S}_{see}(\mathfrak{B})$ seeing of Being | Sehen von Sein, das | videnje biti | (15.4) □

$\mathfrak{S}_{seize}$ seizing | verstehende Ergreifen, das | razumevno prijetje | (12.4) □

$\mathfrak{S}_{seize}(\varphi_{full}(\vartheta_{disclose}(\mathfrak{B}_{in\text{-}the\text{-}world})))$ seizing upon the full disclosedness of Being-in-the-world | verstehendes Ergreifen der vollen Erschlossenheit des In-der-Welt-seins | razumevno prijetje polnega razprtja biti-v-svetu | (12.4) □

$\mathfrak{T}_{temporal}$ temporal | temporal | časoven | (16.7) □

$\mathfrak{T}_{temporal}(\mathfrak{B})$ temporal of Being, the | Temporale des Seins, das | časovno biti | (16.7) □

$\mathfrak{T}_{think}$ thinking | Denken, das | mišljenje | (15.2) □

$\mathfrak{T}_{throw}$ throwing | Werfen, das | metanje | (8.10) □

$\mathfrak{U}$ understanding | Verstehen, das | razumevanje | (1.2), (1.3), (1.4), (1.5), (1.6), (2.4), (2.5), (2.6), (3.1), (3.2), (3.3), (3.13), (5.1), (5.3), (5.4), (5.5), (5.6), (6.1), (7.1), (8.1), (8.2),(8.3), (8.4), (8.5), (8.9), (8.10), (8.11),(9.4), (10.1), (10.2), (10.3), (10.4), (10.8), (10.9), (10.10), (10.11), (11.1),(12.1), (12.4), (15.1), (15.2), (16.1),(16.3), (16.5), (16.6), (16.7), (17.1), (18.1) □

$\mathfrak{U}(\alpha)$ understanding something | etwas verstehen | nekaj razumeti | (3.1), (8.9) □

$\mathfrak{U}(\varepsilon_{ex})$ understanding of existence | Verstehen der Existenz, das | razumevanje eksistence | (10.11) □

$\mathfrak{U}(\varphi_{sake})$ understanding of the »for-the-sake-of-which« | Verstehen des Worumwillen, das | razumevanje zaradi-česa | (2.5) □

$\mathfrak{U}(\mathfrak{B})$ understanding of Being | Seinsverständnis, das | razumevanje biti | (16.3), (16.5), (16.7) □

$\mathfrak{U}(\mathfrak{B}(\tau_{there}))$ understanding of the Being of the "there" | Verstehendes Seins des Da, das | razumevanje biti Tuja | (9.4) □

$\mathfrak{U}(\mathfrak{W}_{world})$ understanding the world | Verstehen von Welt, das | razumevanje sveta | (10.11) □

$\mathfrak{U}_{auth}$ authentic understanding | eigentliche Verstehen, das | pravo razumevanje | (10.4), (10.7) □

$\mathfrak{U}_{exist}$ existential understanding | existenziale Verstehen, das | eksistencialno razumevanje | (15.3) □

$\mathfrak{U}_{inauth}$ inauthentic understanding |

uneigentliche Verstehen, das | nepravo razumevanje | (10.4), (10.7) □

$\mathfrak{U}_{mis}$ misunderstanding | Mißverständnis, das | nesporazum; nerazumevnost | (14.1) □

$\mathfrak{U}_{mis}(\varepsilon_{express}(\sigma_{sight}))$ misunderstanding of the expression 'sight' | Mißverständnis des Ausdrucks »Sicht«, das | nerazumevnost izraza »vid« | (14.1) □

$\mathfrak{U}_{prim}$ understanding, primary | primäre Verstehen, das | primarno razumevanje | (1.6) □

$\mathfrak{U}_{well}$ well-understanding; well understood, the | wohlverstanden | dobro razumljen | (12.4) □

$\mathfrak{W}_{basic\_state}$ whole basic state | ganze Grundverfassung, die | celotno osnovno stanje | (7.1) □

$\mathfrak{W}_{basic\_state}(\mathfrak{B}_{in\text{-}the\text{-}world})$ whole basic state of Being-in-the-world, the | ganze Grundverfassung des In-der-Welt-seins, die | celotno osnovno stanje biti-v-svetu | (7.1) □

$\mathfrak{W}_{categorial}$ whole, categorial | kategoriale Ganze, das | kategorialna celost | (7.5) □

$\mathfrak{W}_{cur\_world}$ current world, the | jeweilige Welt, die | vsakokratni svet | (8.3) □

$\mathfrak{W}_{cur\_world}(\mathfrak{U})$ current world of understanding, the | jeweilige Welt des Verstehens, die | vsakokratni svet razumevanja | (8.3) □

$\mathfrak{W}_{way}$ way | Art, die | način | (14.6), (16.1), (16.2), (17.2), (18.1) □

$\mathfrak{W}_{way}(\pi_{for\text{-}Being}(\mathfrak{D}))$ way of Dasein's potentiality-for-Being | Seinkönnen des Daseins, das | potencialnost biti tubiti | (16.1) □

$\mathfrak{W}_{way}(\mathfrak{A}_{access})$ way of access | Zugangsart, die | način dostopa | (14.6) □

$\mathfrak{W}_{way}(\mathfrak{M})$ way of having a mood, the | in der Weise der Gestimmtheit | na način razpoloženosti | (17.2) □

$\mathfrak{W}_{what}$ "what"; what | Was, das | Kaj | (3.2), (5.3'), (6.1), (7.8), (10.1), (16.6) □

$\mathfrak{W}_{what}(\mathfrak{B}_{exist})$ "what" of the existential Being, the | Was des existenzialen Seins, das | Kaj eksistencialne biti | (6.1), (7.8) □

$\mathfrak{W}_{whole}$ whole, the | ganz; Ganze, das | cel; celota | (10.10) □

$\mathfrak{W}_{why}$ why; the "Why" | warum; das "Warum" | zakaj; Zakaj | (8.1) □

$\mathfrak{W}_{world}$ world | Welt, die | svet | (2.2), (2.4), (3.6), (7.3), (10.2), (10.5), (10.6), (10.11), (12.5), (13.1), (16.2) □

$\mathfrak{W}_{world}(\Xi_{sign})$ world on the basis of significance, the | Bedeutsamkeit ist das, woraufhin Welt ... , die | svet na temelju pomembnosti | (2.7) □

$\mathfrak{W}_{world}(\mathfrak{D})$ Dasein's world | Welt des Daseins, die | svet tubiti | (10.2) □

$\mathfrak{W}_{worldhood}$ worldhood | Weltlichkeit, die | svetnost | (8.3) □

$\mathfrak{W}_{worldhood}(\mathfrak{W}_{cur\_world}(\mathfrak{U}))$ worldhood of understanding's current world, the | Weltlichkeit der jeweiligen Welt des Verstehens, die | svetnost trenutnega sveta razumevanja | (8.3) □

A DICTIONARY OF INFORMATIONAL OPERATORS

(Operator symbols with English, German, and Slovene explanation)

$\models$ inform(s); impact(s); make(s); am/are/is; maintain(s); etc. | informiert; beeinflußt/beeinflussen; bin/bist/ist/sind/seid; usw. | informira(m/aš/mo/te/jo); vpliva(m/aš/mo/te/jo); sem/si/je/smo/ste/so; itd. | (1.1), (1.5), (1.6), (2.2), (2.3), (2.4), (2.8), (3.1), (3.2), (3.5), (3.7), (3.12), (3.13), (4.1), (4.3), (4.4), (4.5), (4.6), (5.1), (5.2), (5.5), (6.1), (7.4), (8.2), (8.5), (8.7), (8.8), (8.9), (8.10), (9.4), (12.2), (12.3), (12.4), (12.5), (14.3), (15.1), (16.1), (16.6), (16.7), (17.2), (17.5), (17.6),

(18.1) □

$\models_{able}$ is/are_able | will_und_kann_ sein | je/so_lahko | (9.1) □

$\models_{able\_man}$ to_be_able_to_manage | vorstehen_können | biti_sposoben | (3.1) □

$\models_{about}$ is_about | ist_nach | je_po/o | (7.7), (17.7) □

$\models_{access}$ is/are_accessible | ist/sind__ zugänglich | je/so_dostopen/ dostopni | (14.4) □

$\models_{accid}$ is_accidental | ist_zufällig | je_slučajno | (7.7) □

$\models_{accomp\_by}$ is_accompained_by | ist_befindlich(es) | je_spoznano(_za) | (5.6) □

$\models_{aim\_at}$ aim(s)_at | ziehlt_auf | se_usmerja_na | (7.7) □

$\models_{all\_press\_for}$ always_press(es)_forward | dringt_ _immer_in | napreduje_vselej_v | (8.1) □

$\models_{all}$ inform(s)_all | informier-t/-en_ _wesenhaft | informira(jo)_bistveno | (12.4) □

$\models_{already}$ is_already | ist_je_schon | je_že | (4.2), (15.1), (16.3) □

$\models_{already} \circ \models_{been}$ has/have_already_been | ist/sind_ schon | je/so_že | (16.3) □

$(\models_{already} \circ \models_{been}) \circ \models_{take}$ has/have_already_been_taken | ist/sind_schon_vorweggenommen | je/so_že_vnaprejvzet(i) | (16.3) □

$\models_{al\_underst}$ always has understood and always will understand | versteht sich immer schon und immer noch | je vselej že in še razumljeno | (8.8) □

$\models_{always}$ infor(s)_always | informier-t/-en_ _immer | informira(jo)_vselej | (10.1), (10.10), (10.11), (12.2), (17.3) □

$\models_{always} \circ \models_{pertain}$ always_pertain(s)_to | betrifft/ betreffen_immer | se_tiče(jo)_vselej | (10.1), (10.10) □

$\models_{always} \circ \models_{u}$ understand(s)_always | versteht/ verstehen_jeweils | razume(jo)_ _vselej | (10.11) □

$\models_{arr}$ arrange(s) | einrichtet (einrichten) | uravnava(jo) | (8.6) □

$\models_{arise\_out}$ arise(s)_out | entspringt_aus | izvira(jo)_iz | (10.4) □

$\models_{as}$ as; is_as | als; ist_als | kot; je_kot | (2.6), (2.8), (3.2), (3.3), (3.11), (3.12), (3.13), (4.1), (4.3), (5.1), (5.3), (7.1), (7.2), (7.3), (7.4), (8.3), (8.4), (8.5), (8.7), (8.8), (8.10), (8.11), (9.1), (9.3), (9.4), (10.1), (10.4), (10.6), (10.8), (10.10), (10.11), (11.1), (12.2), (14.6), (14.7), (15.1), (15.4), (16.6), (17.1) □

$\models_{as\_long\_as}$ is/are_as_long_as | ist/sind_solange | dokler_je/so | (8.7), (8.8) □

$\models_{as\_not\_yet}$ is/are_as_not_yet | ist/sind_sondern_als_nie toda_ni/niso_nikoli | (5.1) □

$\models_{astray}$ inform(s)_astray | informiert_verlaufend | informira(jo)_blodno | (5.5), (5.6) □

$\models_{been}$ has/have_been | ist/sind(_gewesen) | je/so_bil(i) | (16.3) □

$\models_{bring}$ bring(s) | bring-t/en | prinese(jo) | (18.1) □

$\models_{but}$ but_there_is; but | sondern | temveč | (3.2'), (3.2"), (7.3), (12.4), (18.1) □

$\models_{but} \circ (\models_{first} \circ \models_{even})$ but_is/are_in_the_first_instance_ _even | ist/sind_zunächst_nur_(um) | je/so_najprej_le | (18.1) □

$\models_{by}$ inform(s)_by | ist_hinsichlich | informira(jo)_glede_na | (8.4) □

$\models_{cap}$ is/are_capable | 'woran(_es)_ist/sind' | je_sposobno | (5.3), (6.1) □

$\models_{char}$ characterize(s) | charakterisier-t/-en | karakterizira(jo) | (3.9), (3.11), (14.2), (14.7), (17.1) □

$\models_{clarify}$ clarifies/clarify | aufklär-t/en | pojasnjuje(jo) | (16.7) □

$\models_{come}$ come(s) | kommt/kommen_zu | pride(jo)_do/k/h | (17.7) □

$\models_{comp}$ to_be_compentent_to_do | können | znati | (3.1) □

$\models_{conceive}$ conceive(s) | begreif-t/en | doume(jo) | (16.4) □

$\models_{concern}$ concern(s) | betrifft/betreffen | zadeva(jo) | (3.6) □

$\models_{confront}$ confront(s) | ist_gegenüber | je_nasproti | (7.9) □

$\models_{connect}$ keep(s)_connection_with | wahrt/wahren_den_Zusammenhang_ _mit | ohranja(jo)_povezanost_s/z | (14.7) □

$\models_{const}$ constitute(s) | konstituier-e/st/t/en; | konstituira-m/š/mo/te/jo | (1.2), (8.4), (9.1) □

$\models_{constantly}$ is/are_constantly | ist/sind_ständig | je/so_trajno | (9.1) □

$\models_{contra}$ inform(s)_on_the contrary | sondern_hat/informiert | temveč_ima/je/informira | (8.7) □

$\models_{corresp}$ correspond(s) | entspricht/entsprechen | odgovarja(jo) | (14.2), (15.1) □

$\models_{corresp} \circ \models_{noe}$ correspond(s)_noetically | entspricht/entsprechen_noetisch | odgovarja(jo)_noetično | (15.1) □

$\models_{cut\_off}$ cut(s)_off | abschnürt/abschnüren | odveže(jo) | (10.5) □

$\models_{decept}$ is/are_deceptional | täuscht/täuschen | vara(jo) | (13.1) □

$\models_{decide}$ decide(s) | entscheide-t/n | odloča(jo) | (15.4) □

$\models_{deliver}$ deliver(s)_over_to | überantworte-t/n | odgovarja(jo) | (17.4) □

$\models_{deliver}$ is/are_delivered(_to) | ist/sind_überantwortete(s) | je/so_izročeno(i) | (4.4), (5.7) □

$\models_{deprive}$ deprive(s) | nimmt/nehmen | odvzame(jo) | (15.1) □

$\models_{devote}$ devote(s) | legt/legen_sich_in | se_posveča(jo) | (10.2) □

$\models_{discl}$ inform(s)_the_disclosedness_of; disclose(s) | informiert_die_Erschlosseheit_von; erschließt | informira(jo)_razprtost; razpira(jo) | (2.4), (2.5), (2.6), (2.7), (6.1), (7.3), (7.6), (7.9), (10.1) □

$\models_{discover}$ can_be_discovered_only | wird_nur_entdeckbar | bo_le_odkrito | (7.6) □

$\models_{discover\_as}$ discover(s)_as | entdeckt_als | odkriva(jo)_kot | (7.4) □

$\models_{divert}$ divert(s) | ist/sind_verlegt | je/so_založen(i), odvrnjen(i) | (10.9), (11.1) □

$\models_{do}$ do(es) | leistet/leisten | dela(jo) | (14.5), (18.1) □

$\models_{dogma}$ inform(s)_dogmatically | ist/sind_dogmatisch | informira(jo)_dogmatsko | (16.6) □

$\models_{draw\_upon}$ draw(s)_upon | nimmt/nehmen_in_ Anspruch | upošteva(jo) | (14.4) □

$\models_{draw\_upon} \circ \models_{merely}$ draw(s)_upon_merely | nimmt/nehmen_in_Anspruch_nur | upošteva(jo)_le | (14.4) □

$\models_{either}$ is/are_either | ist/sind_entweder | je/so_ali | (10.4), (10.7) □

$\models_{emerge}$ emerge(s) | läßt/lassen_heraustreten | izhaja(jo) | (17.7) □

$\models_{empty}$ is_empty | ist_leer | je_prazno | (3.7) □

$\models_{encounter}$ encounter(s) | begegnet/begegnen | srečuje(jo) | (14.4) □

$\models_{encounter} \circ \not\models_{conceal}$ encounter(s)_unconcealedly | begegnet/begegnen_unverdeckt | srečuje(jo)_nezakrito | (14.4) □

$\models_{enough}$ is/are_enough | kann/können_so_weit | je/so_lahko_tako | (14.7) □

$\models_{enough} \circ \models_{obtain}$ is/are_enough_to_obtain | kann/können_so_weit_..._gewinnen | je/so_lahko_tako_pridobljen(i)_s/z | (14.7) □

$\models_{equal}$ is/are_equal | ist/sind_gleich | je/so_enak(i) | (12.5) □

$\models_{equi\_p}$ am/are/is_equiprimordial | bin/bist/ist/sind/seid_gleichursprünglich | sem/si/je/smo/ste/so_enakoizvor-en/na/no/ni | (1.2), (12.2) □

$\models_{equi\_p} \circ \models_{always}$ is/are_equiprimordially_in_each_ _way | ist/sind_gleichursprünglich_ _nach_der_Weise | je/so_

_enakoizvir-en/ni_na_način | (12.2) □

$\models_{essen}$ is/are_essentially | ist/sind_wesenhaft | je/so_bistveno | (10.1), (16.5) □

$\models_{essen} \circ \models_{discl}$ is/are_essentially_disclosable | ist/sind_wesenhaft_erschließbar | je/so_bistveno_razprt/i | (10.1) □

$\models_{essen\_for}$ is/are_essential(_for) | ist/sind_wesenhaft | je/so_bistven(o)(_za) | (3.6) □

$\models_{essential}$ is/informs_essentially | ist_wesenhaft | je_bistveno | (5.4) □

$\models_{even}$ is/are_even | ist/sind_auch | je/so_tudi | (17.7), (18.1) □

$\models_{exist}$ is_existential(_in every_case) | ist_(je_)existenzial | je_(vselej_)eksitencialno | (3.7), (9.3), (10.10), (12.2), (12.2) □

$\models_{exist\_as}$ exist(s)_as | existier-t/en_als | eksistira(jo)_kot | (10.3) □

$\models_{exist\_const}$ is_existentially_constitutive | konstituiert | konstituira_ _eksistencialno | (8.4) □

$\models_{exist\_surr}$ is_existentially_surrended | ist_existenzial_ausgeliefert | je_eksistencialno_izročeno | (5.6) □

$\models_{explicate}$ explicate(s) | explizier-t/en | ekspliciра(jo) | (17.5) □

$\models_{extra}$ informs_in_an_extra_way | informiert_noch_als_Zugabe | informira_kot_dodatek | (3.4) □

$\models_{factually}$ is/are_factually | ist/sind_tatsächlich | je/so_dejansko | (9.1), (9.2) □

$\models_{fail\_to\_recognize}$ fail(s)_to_recognize | verkennt/verkennen_sich | se_ne_spozna(jo) | (5.5), (5.6) □

$\models_{first}$ is/are_first | ist/sind_zunächst | je/so_najprej | (17.7), (18.1) □

$\models_{first} \circ \models_{even}$ is/are_in_the_first_instance | ist/sind_zunächst_nur | je/so_najprej_le | (18.1) □

$\models_{first}(\models_{let} \circ \models_{emerge})$ must_first_let_emerge | muß/müssen_erst_heraustreten_ _lassen | mora(jo)_najprej_dopustiti_ _izstopiti | (17.7) □

$\models_{first\_find}$ is_first_to_find_in | ist_erst_zu_finden_in | je_najprej_najti_v | (5.7) □

$\models_{for}$ is/are_for | ist/sind_für | je/so_za | (3.6), (4.5) □

$\models_{formalize}$ formalize(s) | formalizier-t/en | formalizira(jo) | (14.7) □

$\models_{formulate}$ formulate(s) | formulier-t/en | formulira(jo) | (17.7) □

$\models_{free}$ gives_free | gibt_frei | daje_prostost | (7.3) □

$\models_{from}$ is_are_from | ist/sind_von | je/so_od | (10.5) □

$\models_{general}$ is/are_in_general | ist/sind_überhaupt | je/so_na_splošno | (16.2) □

$\models_{get}$ get(s) | ist/sind | dobi(jo)/je/so | (8.4), (16.6) □

$\models_{give}$ give(s) | ist/sind(_gegeben) | daje(jo) | (14.4) □

$\models_{guard\_against}$ guard(s)_against | bewahr-t/en_vor| obvaruje(jo)_pred | (14.1) □

$\models_{in}$ is/are_in | ist/muß_in | je/so_v | (9.3), (12.1) □

$\models_{have}$ has/have | hat/haben | ima(jo) | (16.3), (16.5), (17.3) □

$\models_{have} \circ ((\models_{already} \circ \models_{been}) \models_{take})$ has/have_already_been_taken | ist/sind_schon_vorweggenommen | je/so_že_vnaprej_vzet-o/i | (16.3) □

$\models_{have} \circ \models_{always}$ has/have_already | ist/sind_je_schon | je/so_vselej_že | (17.3) □

$\models_{in}$ inform(s)_in | ist/sind_in | informira(jo)_v | (18.1) □

$\models_{in\_manner}$ inform(s)_in_a_manner | ist_solche(s) | je/informira_tako | (8.10) □

$\models_{int}$ interpret(s) | interpretie-rt/ren | interpretira(jo) | (1.5), (1.6) □

$\models_{into}$ inform(s)_into | informier-t/en_in | informira(jo)_v | (11.1), (18.1) □

$\models_{just}$ is/are_just | ist/sind_nur | je/so_le | (14.3), (18.1) □

$\models_{just\_as\_much}$ is/are_just_as_much_in | ist/sind_einzig_und_primär_in | je/so_ravnotako_v | (13.1) □

$\models_{just} \circ \models_{bring}$ is/are_just_to_bring | ist/sind_zu_bringen | je/so_kot_prineše-n/ni | (18.1) □

$\models_{know}$ know(s) | weiß, wissen | ve/vedo | (5.3) □

$\models_{let}$ must_let | muß/müssen_lassen | mora(jo)_dopustiti | (17.7) □

$\models_{let} \circ \models_{emerge}$ must_let_emerge | muß/müssen_heraustreten_lassen | mora(jo)_dopustiti_izstopiti | (17.7) □

$\models_{list}$ list(s) | registrier-t/en | registrira(jo) | (9.1) □

$\models_{make}$ make(s) | will/wollen | želi(jo) | (9.1) □

$\models_{make\_up}$ make(s)_up | macht/machen_aus | donkončuje(jo) | (12.1) □

$\models_{make\_up} \circ \models_{exist}$ make(s)_up_existentially | macht/machen_aus_existenzial | dokončuje(jo)_eksistencialno | (12.1) □

$\models_{match}$ to_be_a_match_for | gewachsen_sein | biti_dorasel | (3.1) □

$\models_{merely}$ is/are_merely | ist/sind_nur | je/so_samo | (14.4) □

$\models_{most\_part}$ is/are_for_the_most_part | ist/sind_zumeist | je/so_zlasti | (10.2) □

$\models_{must}$ must | muß/müssen; bedarf/bedürfen | mora(jo); potrebuje(jo) | (18.1) □

$\models_{must} \circ \models_{work\_out}$ must_work_out | bedarf/bedürfen_ _Ausarbeitung | potrebuje(jo)_ _izdelavo | (18.1) □

$\models_{noe}$ inform(s)_noetically | informier-t/en_noëtisch | informira(jo)_noetično | (15.1) □

$\models_{now}$ is/are_now | ist/sind_jetzt | je/so_zdaj | (16.6), (17.5) □

$\models_{now} \circ \models_{get}$ now_get(s) | erhält/erhalten_jetzt | dobi(jo)_zdaj | (16.6) □

$\models_{obtain}$ obtain(s) | gewinnt/gewinnen | pridobiva(jo) | (14.7) □

$\models_{of}$ of; is_of; depend(s)_of | von | od | (7.5), (10.5), (15.1), (16.7), (17.7) □

$\models_{only}$ is/are_only | ist/sind_»nur« | je/so_»samo« | (10.5) □

$\models_{only\_because}$ is/are_only_because | ist/sind_nur_ _weil | je/so_samo_zaradi | (9.4) □

$\models_{ont}$ is_ontologically | ist_ontologisch | je_ontološko | (3.10), (3.11) | □

$\models_{ont,some}$ is/are_ontically_sometimes | ist/sind_zuweilen_in_ontischer_Rede | je/so_občasno_v_ontološki_govorici | (3.1) □

$\models_{or}$ or_inform(s) | oder_ist/sind | ali_je/so | (10.4), (10.7) □

$\models_{orient}$ orient(s)| orientier-t/en| se_usmerja(jo)| (14.6) □

$\models_{orient} \circ (\models_{prim} \circ \models_{towards})$ orient(s)_primarily_towards | orientier-t/en_sich_primär_zu | se_usmerja(jo)_najprej_k/h/na | (14.6) □

$\models_{over}$ inform(s)_over | informier-t/en_ _um | informira(jo)_o | (17.7) □

$\models_{pass\_by}$ has_let_pass_by | hat_vorbeigehen_lassen | je_pustil(o)_mimo_iti | (4.3) □

$\models_{permeate}$ permeat(s)_with | durchsetzt/durchsetzen | prepojuje(jo) | (10.8) □

$\models_{pertain}$ pertain(s) | betrifft/betreffen | zadeva(jo) | (3.6), (10.1), (10.7) □

$\models_{posit}$ posit(s) | ansetz-t/en | določa(jo) | □

$\models_{posit} \circ \models_{dogma}$ posit(s)_dogmatically | ansetz-t/en_dogmatisch | določa(jo)_dogmatsko | (16.6) □

$\models_{prim}$ is_primarily | ist_primär | je_primarno | (3.4), (14.6) □

$\models_{prim\_sol}$ is/are_primarily_and_solely_in | ist/sind_einzig_und_primär_in | je/so_edino_in_primarno_v | (13.1) □

$\models_{prim\_to}$ is/are_primarily_to | ist/sind_primär_in | je/so_primarni_v | (10.2) □

$\models_{prim} \circ \models_{towards}$ is/are_primarily_towards | ist/sind_primär_zu | je/so_primarno_k/h/na | (14.6) □

$\models_{project}$ project(s) | entwirft/entwerfen | snuje(jo)/projektira(jo) | (8.7), (8.9), (16.2) □

$\models_{project\_upon}$ project(s)_upon | entwirft/entwerfen_auf | snuje(jo)_na/v | (8.3), (16.2) □

$\models_{prov}$ provides | bietet | nudi | (3.13) □

$\models_{prox}$ is/are_proximally | ist/sind_zunächst | je/so_najprej | (10.2) □

$\models_{prox} \circ \models_{most\_part}$ is/are_proximally_and_for_the_ _most_part | ist/sind_zunächst_und_ _zumeist | je/so_najprej_in_najbolj | (10.2) □

$\models_{pure}$ is/are_pure | ist/sind_pur | je/so_čist-i/a/o | (14.3) □

$\models_{pure} \circ \not\models_{sense}$ is/are_pure_non-sensory | ist/sind_pur(e)_unsinnlich(e) | je/so_čisto_nesmiseln-i/a/o | (14.3) □

$\models_{quest}$ question(s) | fragt(_nach) | vprašuje(_po) | (7.7), (7.8), (7.9), (8.1) □

$\models_{receive}$ receive(s) | erhält/erhalten | dobi(jo) | (9.4) □

$\models_{reduce}$ reduce(s) | zieht/ziehen herab | znižuje(jo) | (8.10) □

$\models_{rel\_prim\_whole}$ relate(s)_primarily_and_on_the_ _whole | bezieh-t/en_sich primär_und_ _im_ganzen | se_nanaša(jo)_ _predvsem_in_v_celoti | (12.3) □

$\models_{reveal\_as}$ reweal(s)_as | entdeckt_in | odkriva(jo)_kot | (7.5) □

$\models_{root}$ root(s)_in | wurzel-t/n_in | temelji(jo)_v | (13.1) □

$\models_{root} \circ \models_{just\_as\_much}$ root(s)_just_as_much_in | wurzel-t/n_ebensosehr_in | temelji(jo)_pravtako_v | (13.1) □

$\models_{sake}$ inform(s)_for_the_sake_of_which | ist/sind_umwillen_dessen(_als) | informira(jo)_zaradi_tega(_kot) | () □

$\models_{say}$ say(s) | sagt/sagen | reče(jo) | (9.4) □

$\models_{say\_with}$ say(s)_with | sagt/sagen_mit | reče(jo)_s/z | (9.4) □

$\models_{see}$ see(s) | sieht/sehen | vidi(jo) | (17.2) □

$\models_{seize}$ seize(s) | ergreift/ergreifen | se_polasti(jo) | (4.3) □

$\models_{show}$ show(s) | zeigt/zeigen | kaže(jo) | (15.1) □

$\models_{sight}$ sight(s) | sichtet/sichten_»sich« | »se«_ugleda(jo) | (12.5) □

$\models_{sign}$ signifies | bedeuted | pomeni | (3.8) | □

$\models_{sketch\_out\_beforehand}$ sketch(es)_out_beforehand | zeichne-t/n_vor | skicira(jo)_vnaprej | (10.1) □

$\models_{so\_far}$ is/are_only_in_so_far_as | ist/sind_nur_sofern_in | je/so_samo_dokler | (12.5) □

$\models_{such\_t\_e\_c}$ is_such_that_in_every_case | verhält_sich_so_daß_immer | je_tak_da_vselej | discussion of (5.2) □

$\models_{supp}$ keep(s)_suppressed; informs_in_a_ _suppressed_way | hält_nieder; informiert_verdeckt | duši; informira_zakrito | (1.3) □

$\models_{sure}$ must_be_sure | muß/müssen | mora(jo)_biti_prepričan(i) | (14.1) □

$\models_{take}$ take(s) | nimmt/nehmen | vzame(jo) | (16.3) □

$\models_{take\_away}$ take(s)_away | zieht/ziehen_herab | odvzema(jo) | (8.10) □

$\models_{than}$ than(_is) | (ist_)als | (je_)kot | (3.10), (9.1) □

$\models_{that}$ inform(s)_that | informier-t/en_ _in/im | informira(jo)_v | (17.5) □

$\models_{think\_out}$ think(s)_out | denkt/denken_aus | si/se_izmišlja(jo) | (8.6) □

$\models_{through}$ is/are_through_and_through | ist/sind_durch_und_durch | je/so_skozi_in_skozi | (4.4), (9.4), (16.4) □

$\models_{throughout}$ is/are_throughout | ist/sind_durch | je/so_skozi | (12.4) □

$\models_{throughout} \circ \models_{all}$ is/are_throughout_all | ist/sind_durch_hindurch | je/so_vseskozi | (12.4) □

$\models_{throw}$ throw(s) | wirft/werfen | vrže(jo) | (10.3) □

$\models_{\text{throw\_before}}$ throw(s)_before | vorwirft/vorwerfen | vrže(jo)_v_ospredje | (8.10) □

$\models_{\text{thrown}}$ is/are_thrown | ist/sind_geworfen | je/so_vržen(i/o/e) | (8.5) □

$\models_{\text{to}}$ inform(s)_to | informiert_zu | informira(jo)_v | (8.10), (14.6), (14.7), (17.6), (18.1) □

$\models_{\text{towards}}$ inform(s)_towards | informier-t/en_zu | informira(jo)_k | (8.6), (14.6) □

$\models_{\text{trans}}$ is/are_transparent | ist/sind_durchsichtig | je_razviden/so_razvidni | (4.6) □

$\models_{\mathfrak{U}}$ understand(s); has_understood | versteh-t/en; hat_verstanden | razume(jo); je_razumelo | (5.2), (7.9), (10.2), (10.5), (10.11), (16.4) □

$\models_{\mathfrak{U}} \circ \models_{\text{only}}$ understand(s)_'only' | versteh-t/en_»nur« | razume(jo)_»samo« | (10.5) □

$\models_{\text{waive}}$ is_constantly_waiving | begibt_sich_ständig | je_vselej_na_poti_k | (4.3) | □

$\models_{\text{upon}}$ is/are_upon | ist/sind_auf | je/so_na | (16.2), (16.3) □

$\models_{\text{why}}$ why_is | warum_ist | zakaj_je | (7.9) □

$\models_{\text{with}}$ is_with | ist_mit | je_z | (5.1), (12.2), (12.5), (13.1), (14.3), (18.1) □

$\models_{\text{with}} \circ \models_{\text{equal}}$ is/are_with_equal | ist/sind_gleich_in | je/so_enak(i)_v | (12.5) □

$\models_{\text{with\_eq\_p}}$ inform(s)_with_equal_primordiality | informiert_ebenso_ursprünglich | informira(jo)_prav_tako_izvirno | (8.3) □

$\models_{\text{with\_regard}}$ is/are_with_regard_to | ist/sind_hinsichtlich | je/so_glede_na | (8.4) □

$\models_{\text{work\_out}}$ work(s)_out | arbeite-t/-n_aus | izdela(jo) | (18.1) □

$\models_{\text{yet}}$ is/are_yet | ist/sind_doch | je/so_pač | (9.3) □

$\not\models$ does/do_not_inform; does/do_not_impact; deos/do_not_make; am/are/is_not; does/do_not_maintain; etc. | informiert_nicht; beeinflußt/beeinflussen_nicht; bin/bist/ist/sind/seid_nicht; usw. | ne_informira-m/š/mo/te/jo; ne_vpliva-m/š/mo/te/jo; nisem/nisi/ni/nismo/niste/niso; itd. | (3.4), (3.8), (5.1), (7.9), (8.6), (10.7), (12.4) □

$\not\models_{\text{act}}$ is_not_yet_actual | ist_noch_nicht_Wirkliche | še_ni_aktualno | (3.8) □

$\not\models_{\text{comp}}$ has/have_not_competence | ist_kein(e) | ni_kompetenten | (3.2) □

$\not\models_{\text{conceal}}$ inform(s)_unconcealedly; do(es)_not_inform_concealedly | ist/sind_unverdeckt; ist/sind_nicht_ _verdeckt | informira(jo)_nezakrito; ne_informira(jo)_zakrito | (14.4) □

$\not\models_{\text{essen}}$ is_essentially_never | ist_als_wesenhaft_nie | ni_bistveno_nikoli | (5.1) □

$\not\models_{\text{first}}$ does/do_not_first_arise | ist/sind_nicht_erst_erwachsen | ni(so)_šele_nastal(i) | (5.4) □

$\not\models_{\text{grasp\_thema}}$ does/do_not_grasp_thematically | erfaßt/erfassen_nicht_thematisch | ne_zapopade(jo)_tematsko | (8.9) □

$\not\models_{\text{lay\_aside}}$ lay(s)_aside | legt/legen_nicht_ab | ne_odloži(jo)(_vstran) | (10.9) □

$\not\models_{\text{less}}$ is/are_never_less | ist/sind_nie_weniger | ni(so)_nikoli_manj | (9.3) □

$\not\models_{\text{mean}}$ do(es)_not_mean | mein-t/en_nicht | ne_pomeni(jo) | (10.5), (14.3) □

$\not\models_{\text{mean}} \circ \models_{\text{just}}$ do(es)_not_mean_just | mein-t/en_nicht_nur | ne_pomeni(jo)_samo | (14.3) □

$\not\models_{\text{more}}$ does/do_not_become_more | wird/werden_nicht_...er | ni(so)_...ji/...ejši | (17.5) □

$\not\models_{\text{more}} \circ \models_{\text{now}}$

does/do_not_become_now_more | wird/werden_jetzt_nicht_...er | ni(so)_zdaj_...ji/...ejši | (17.5) □

$\not\models_{more\_than}$ is/are_never_more_than | ist/sind_nie_mehr_als| ni(so)_nikoli_več_kot | (9.2) □

$\not\models_{onto}$ is/are_not_ontologically | ist/sind_nicht_ontologisch | ni(so)_ontološko | (16.4) □

$\not\models_{onto} \circ \models_{conceive}$ does/do_not_ontologically_conceive | begreif-t/en_nicht_ontologisch | ne_zapopade(jo)_ontološko | (16.4) □

$\not\models_{root}$ do(es)_not_root_in | wurzelt/wurzeln_nicht_in | ne_temelji(jo)_v | (13.1) □

$\not\models_{root} \circ \models_{prim\_sol}$ do(es)_not_root_primarily_and_solely_in | wurzelt/wurzeln_nicht_einzig_und_primär_in | ne_temelji(jo)_edino_le_in_predvsem_v | (13.1) □

$\not\models_{satisfy}$ is/are_not_satifactory | ist/sind_nicht_entsprechend | ni(so)_ustrezno | (16.7) □

$\not\models_{satisfy} \circ \subset_{clarify}$ is/are_not_satisfactorily_clarified_within | entspricht/entsprechen_nicht_einer_befriedigenden_Aufklärung_in/im | ne_ustreza(jo)_zadostni_razjasnitvi_v | (16.7) □

$\not\models_{sense}$ is/are_non-sensory | ist/sind_unsinnlich | je/so_nesmiseln(i) | (14.3) □

$\not\models_{sign}$ does_not_signify | bedeuted_nicht | ne_pomeni | (4.1) □

$\not\models_{yet}$ is/sre_not_yet | ist/sind_noch_nicht | ni(so)_še | (9.3) □

$\not\models_{U}$ has_not_understood | hat_nicht_verstanden | ni_razumel(o) | (5.2') □

$=\!|$ inform(s)_alternatively | informiert_in_alternativen_Weise | informira(jo)_alternativno | (5.2') □

$=\!|_{accord}$ is/are_in_accordance_with | ist/sind_gemäß | je/so_skladno_s/z | (8.6) □

$=\!|_{as}$ as; is_alternatively_as | als; ist_alternativ_als | kot; je_alternativno_kot | (3.8) □

$=\!|_{char}$ characterize(s)_alternatively | characterisier-t/en_in_alternativen_Weise | karakterizira(jo) alternativno | (3.6) □

$\not=\!|$ inform(s)_not_alternatively | informiert_nicht_in_alternativen_Weise | ne_informira(jo)_alternativno | (5.2'), (9.4) □

$\|\models$ is_in_parallel | ist_parallel | je_paralelno | (3.11) □

$\forall$ inform(s)_for_all | informiert_für_alle | informira(jo)_za_vse | discussion of (4.2), (5.2) □

$\forall_{already}$ is/are_in_every_case_already | ist/sind_je_schon | je/so_vselej_že | (5.6) □

$\forall_{pertain}$ always_pertain(s)_to | betrifft_immer | zadeva(jo)_vselej | (7.1) □

$\Rightarrow$ implies/imply; if...then... | impliziert/implizieren; wenn...dann... | implicira(jo); če...potem... | (1.5), (2.3), (2.4), (5.5), (5.6), (7.3), (7.6), (7.9), (8.2), (9.1), (10.9), (10.10), (14.1), (15.1), (15.4), (16.5), (17.7), (18.1) □

$\Leftarrow$ implies_alternatively | impliziert_in_alternativen_Weise | implicira_alternativno | (7.9), (9.2) □

$\Leftarrow_{so\text{-}far\text{-}as}$ is_implied_so_far_as | ist_impliziert_sofern | je_implicirano_do_nadaljnega | (3.7) □

$\Leftrightarrow$ is/are_equivalent_to; mean(s) | ist/sind_equivalent; besagt/besagen | je_ekvivalentno; pomeni(jo) | (2.2), (2.4), (2.8), (4.4) □

$\Leftrightarrow_{Df}$ is/are_in_every_case; is_definitively | ist/sind_je | je/so_vselej | (3.5) □

$\subset$ is_a_part_of; is_in; has/have (read in the right to the left direction of operator $\subset$), etc. | ist_ein_Teil_von; ist_in; hat/haben (gelesen in der Richtung von rechts nach links der Operanden); usw. | je/so_del; je/so_v; ima/imajo (prebrano v smeri od desnega proti levemu operandu); itd. | (1.5), (1.6), (2.4), (2.5), (3.2), (3.3), (3.7), (3.11), (4.2), (5.1), (5.4), (5.7), (7.3), (8.2), (8.10), (8.11), (10.1), (14.3), (14.5), (14.7), (17.3), (17.4), (17.5) □

$\subset_{always}$ is_always_a_part_of; has_always ($\leftarrow$) etc. | ist_immer_ein_Teil_von; hat_je ($\leftarrow$); usw. | je/so_vselej_del; ima/imajo_vselej ($\leftarrow$) | (1.3), (1.4), (3.6) □

$\subset_{clarify}$ clarifies/clarify_in | klärt/klären_sich_auf_in/im | se_razjasni(jo)_v | (16.7) □

$\subset_{comp}$ possesses_compentence_for | besitzt_..._zu_können | ima_..._kar_razume | (3.4) □

$\subset_{discl}$ is/are_disclosed_in | ist/sind_in/im_..._erschlossen | je/sta/so_razprt(a/i)_v | (2.8), (8.1) □

$\subset_{essen}$ as_essentially_having | ist_als_wesenhaft_in | je_kot_bistveno_v | (4.2) □

$\subset_{exist}$ lies_existentially_in | liegt_existenzial(_in/im) | se_nahaja_eksistencialno_v | (3.3) □

$\subset_{ground}$ is/are_grounded_in | gründet/gründen_in/im | temelji(jo)_v | (15.3) □

$\subset_{ground} \circ \subset_{prim}$ is/are_gronded_primarily_in | gründet/gründen_primär_in/im | temelji(jo)_predvsem_v | (15.3) □

$\subset_{lie}$ lies/lie_in | liegt/liegen_in/im | leži(jo)_v | (16.2) □

$\subset_{prim}$ is/are_primarily_in | ist/sind_primär_in/im | je/so_najprej_v | (15.1), (16.1), (16.2), (16.3), (16.4), (16.6) □

$\in$ is_one_of; belongs_to | ist_eine(r)_der; gehört_zu | je_eden/ena/eno_izmed; pripada | (1.1), (1.5), (1.6), (10.5) □

$\in_{essen}$ belong(s)_essentially | gehört/gehören_wesenhaft | pripada(jo)_bistveno | (9.2) □

$\neq$ is_distinguished_from; is_different_from | ist_unterschieden_von | je_razločljivo_od | (1.6) □

$\neq_{sharply}$ is_to_be_sharply_distinguished_ _from | unterscheidet_sich_ebensosehr_von | se_razlikuje_prav_tako_ostro_od | (3.7) □

$\rightarrow_{equi\_p}$ pertains_equiprimordially | betrift_gleichursprünglich | zadeva_enakoizvorno | (2.6) □

$\circ$ operator_composition | Operatorskomposition, die | operatorska_kompozicija | (10.1), (10.2), (10.5), (10.10), (10.11), (12.1), (12.2), (12.4), (12.5), (13.1), (14.3), (14.4), (14.6), (14.7), (15.1), (16.2), (16.3), (16.4), (16.6), (16.7), (17.3), (17.5), (18.1) □

A Short (and Essential) Comment

The informational operand is a synonym for the processing, acting entity, is an informational subject which during its existence predicates (operates) the nature, constitution, being, foundation, behavior, phenomenality, impacting, impactedness, informing, communicating, etc. of the marked real entity. An informational operand operates informationally, predicates its subjectivity, behaves as a predicating subject.

Novice in zanimivosti

Sistemske znanosti in kibernetika
(Oblikovanje seznama elektronske pošte)

Seznam elektronske pošte za področje *sistemskih znanosti in kibernetike* se oblikuje na računalniškem sistemu SUNY-Binghampton (NY). Seznam je namenjen izmenjavi sporočil, ki zadevajo splošno razumevanje razvoja zapletenih, večravenskih sistemov, kot so organizmi, entitete uma (minds) in družbe, in sicer kot *informacijske entitete* (informational entities) v obliki možgočih cirkularnih procesov.

Namen elektronskega seznama je

(1) vzpodbujanje razprave na področju sistemov in kibernetike,

(2) komuniciranje v okviru raziskovalne komune med sistemskimi znanstveniki in kibernetiki,

(3) namestitev odlagališča elektronskih zbirk za splošno distribucijo na področju sistemov in kibernetike ter naposled

(4) oblikovanje osrednjega, javnega imenika za sistemske znanstvenike in kibernetike. Elektronska pošta lahko shranjuje in oddaja notice in sporočila, članke (technical papers), reference, pozive piscem člankov oz. referatov, računalniške programe, slike in diagrame.

K sodelovanju v tej razpravi so vabljeni vsi zainteresirani.

Za vpis (subskripcijo) je potreben računalniški konto z dostopom na eno od mednarodnih mrež. Pošljite najprej zbirko z eno samo vrstico:

"SUB CYBSYS-L vaše_polno_ime"

strežniku (server) na naslov

LISTSERV@BINGVMB.BITNET

Ko prejmete potrditev subskripcije, pošljite sporočilo v sam seznam na naslov

CYBSYS-L@BINGVMB.BITNET

V sporočilo zapišite vaše ime, delovno mesto in kratek opis vašega dela ali interesa na področju sistemov in kiberentike.

Anton P. Železnikar

Spreminjanje paradigem v programirni tehniki

Avstrijska računalniška družba (ÖCG) in madžarsko Društvo Johna von Neumanna za računalniške znanosti (NJSZT) prirejata v Celovcu (Avstrija), od 21. do 23. septembra 1992, skupno letno konferenco na temo »Spreminjanje paradigem v programirni tehniki« (Shifting Paradigms in Software Engineering). Namen konference je obravnava najnovejšega stanja in trendov v softverski tehniki. Bistvene spremembe te tehnike se kažejo v objektno usmerjenem razvoju programov in tehniki računalniško podprtega programiranja. Nove tehnike spreminjajo šolniško znanje in vplivajo na aplikacije, teorijo, vzgojo in prenos znanja. Praktiki, raziskovalci in učitelji majhnih držav naj bi bili še posebej zainterisirani za ta razvoj.

Naslovne teme konference so: projektno upravljanje; umetna inteligenca: vidiki modeliranja; umetna inteligenca: vidiki gradnje orodij; jezikovne značilnosti; objektno usmerjeni razvoj programov; izzivi soočenja s kompleksnostjo; metodologija in izkustvo; in poučevanje programirne tehnike. Sponzorji konference so: Zvezno ministrstvo za znanost in raziskavo, Društvo prijateljev Instituta za informatiko, Bacher Electronics, DEC in Siemens AG Österreich. Na koncu konference bo okrogla miza na temo Raziskovalna politika za informacijske tehnologije v majhnih evropskih državah. Sodelovali bodo predstavniki Komisije Evropske skupnosti, avstrijske, madžarske in slovenske vlade. Ob konferenci bo razstava CASE orodij.

Slovenija se bo na konferenci predstavila z referatoma K. Rizmana in I. Rozmana (Enhancing Reusability and Simplifying The OO Development with the Use of Events and Object Environment) ter Tatjane Welzer in Joszefa Györkösa (Teaching and Training in the CASE Tool Environment) z mariborske Tehniške fakultete. Kotizacija za pozno registracijo in nečlane znaša ÖS 3400. Naslov: Universität Klagenfurt, Institut für Informatik, Universitätsstr. 65, A-9020 Klagenfurt, Österreich Elektronska pošta pa je:

mittermeir@edvz.uni-klagenfurt.ada.at).

A.P. Železnikar

Strokovno povezovanje Slovenije na področju informatike

Definicija majhnih in velikih evropskih držav bržkone ni potrebna. Naredimo pa lahko poseben izbor, ki bi utegnil biti zanimiv za Slovenijo. V ta izbor spadajo ali bodo spadale: Avstrija, Belorusija*, Bolgarija*, Bosna in Hercegovina, Belgija, Češka, Danska*, Estonija*, Francija*, Grčija*, Hrvaška, Italija, Latvija*, Litva, Luksemburg*, Makedonija, Madžarska, Nemčija, Poljska*, Portugalska*, Romunija, Rusija, Slovaška, Srbija, Španija*, Švica*, Ukrajina*, Velika Britanija itd. Od neevropskih držav so aktualne Argentina*, Avstralija*, Izrael, Japonska, Južna Koreja*, Južnoafriška republika, Kanada, Nova Zelandija*, Singapur*, ZDA itn. Povezava preko svetovne podatkovne mreže bo prej ko slej mogoča in učinkovita, vendar z državami z zvezdico trenutno še nimamo dobrih osebnih povezav.

V naslednjem letu bo časopis Informatica razširil svoj uredniški odbor, v katerem so že zdaj zastopniki novih balkanskih držav, še na urednike iz Avstrije, Italije, Japonske, Madžarske, Romunije in ZDA. Časopis bo dosledno uporabljal angleški jezik. Vsebinski spekter časopisa bo ostal ohranjen, ojačila pa se bodo področja umetne inteligence, informacijske sistemske znanosti, robotske in kibernetske informatike. Kriteriji za objavo se bodo poostrevali skladno s ponudbo piscev, vendar bo Slovensko društvo Informatika (SDI) kot založnik obdržalo specifične kakovostne kriterije in strategijo znanstvenega in tehnološkega razvoja regionalnih prostorov.

Povezava s strokovnimi društvi in združenji omenjenih držav se bo sistematično ojačevala. Izhodišče SDI in drugih slovenskih društev bo, da bodo strokovne manifestacije (konference, seminarji, publikacije, projekti) s področja računalništva in informatike vselej mednarodna, tudi tako, da bodo slovenski strokovnjaki dejavno udeleženi na tujih prireditvah, pri tujih projektih in organizaciji, vendar s skupnimi cilji in dogovori. V tem okviru bo omogočena tudi strokovna razprava o ekonomskih vidikih in strategiji razvoja informacijskih znanosti in informacijske tehnologije v Sloveniji in v drugih državah. Tako bo mogoče postavljati vprašanja: Kaj proizvajati na področju informacijske tehnologije v Sloveniji? Kako organizirati racionalno obliko informacijskih raziskav? Kje pospeševati priliv tujega kapitala in investicij v domačo informacijsko industrijo, komunikacije in majhna računalniška tehnološka in visokotehnološka podjetja?

A.P. Železnikar

Popravek

V članku »An Introduction to Structured Systems Analysis«, objavljenem v časopisu Informatica **16** (1992), št. 2, stran 1, je bil v naslovu napačno odtisnjen priimek avtorja članka, in sicer Dimij namesto pravilno Damij. Avtorju *Talibu Damiju* se uredništvo opravičuje.

Nekateri zanimivi regionalni časopisi

Revue internationale de sistemique (Revue éditée par l'AFCET; M.R. Vallée, AFCET, 156, Boulevard Péreire, F-75017, France) objavlja prispevke v angleščini in francoščini.

Cybernetics and Systems (Robert Trappl, Department of Medical Cybernetics and Artificial Intelligence, University of Vienna, Freyung 6, A-1010 Vienna, Austria).

Computers and Artificial Intelligence (Institute of Technical Cybernetics, Slovak Academy of Sciences, Dubravskà cesta 9, 842 37 Bratislava, Republika Češka in Slovaška).

Communication and Cognition (Fernand Vandamme, Blandijnberg, 2, B-9000 Gent, Belgium).

Metalogicon (Rivista internazionale di logica pura e applicata, di linguistica e di filosofia, Corso Umberto I, 70, 80034 Marigliano, Italy) izhaja v angleščini.

Cybernetica (Review of the International Association for Cybernetics, Palais des Expositions, Place A. Rijckmans, B-5000 Namur, Belgium) izhaja štirikrat na leto in objavlja prispevke v angleščini in francoščini.

XIII Magazine (Directorate General for Telecommunications, Information Industries and Innovation of the Commission of the European Communities) izhaja v evropskih jezikih.

A.P. Železnikar

Nada Lavrač

Compulog NoE: Network of Excellence in Computational Logic

Given is an overview of the ESPRIT III Network of Excellence in Computational Logic (Compulog NoE) and the report on the Second General Meeting of the Compulog Network of Excellence held in Rome, May 11-14, 1992. Since 1991, two Ljubljana Artificial Intelligence Laboratories (at the J. Stefan Institute and the Faculty of Electrical Engineering and Computer Science) act as a node in this network.

1 Introduction

Compulog Net is the ESPRIT Network of Excellence (NoE) in Computational Logic. It currently consists of over 60 nodes in leading European universities, research institutes and industries. The NoE aims at coordinating research in computational logic, coordinating training and mobility, and promoting the exploitation by European industry of the research results. The network is structured around six areas: constraint logic programming, knowledge bases, knowledge representation and reasoning, program development, programming languages, and parallelism and implementation techniques. The first General Meeting was in May 1991 in Rome. I participated at this meeting as the manager of the Ljubljana node of the Compulog NoE. Since the first meeting, the network has set up email facilities (ECRC, Munich) and a publication server (KU Leuven). Furthermore, a summerschool in Logic Programming will be held in Zurich (September 7-11, 1992), with a particular emphasis on industrial applications. For the near future, attention will be focussed on improving the structure of the NoE, improving the contacts with industry, and the implementation of a training and mobility scheme.

2 Area overviews

The network is structured around six areas. At the Rome meeting, the area overviews consisted mostly of an overview of the papers presented at the various area workshops.

Constraint Logic Programming (Alain Colmerauer, Marseille) deals with extensions of the logic programming paradigm by means of algorithms and methods for constraint satisfaction. Work was reported on solving Boolean constraints, constraints on finite domains, linear equations on integers, non-linear constraints, and Gaussian elimination.

Knowledge Bases (Sury Sripada, ECRC Munich) topic deals with methods and techniques for building, querying, updating and checking the integrity of large deductive databases. The report was focussed on knowledge representation in KBs (semantics, different kinds of knowledge, structuring knowledge), querying KBs (conditional and intensional answers, simplifying queries by integrity constraints, complexity classes of queries), updating KBs (intensional updates, integrity checking), and building KBs (verification and validation, conceptual modeling, architectures). The proceedings of this workshop are available at the J. Stefan Institute.

Knowledge Representation and Reasoning (Bob Kowalski, London) dealswith the computer modeling of knowledge and mechanised reasoning procedures to use that knowledge for problem solving. Work was reported on abduction, nonmonotonic reasoning, inductive logic programming, temporal reasoning, and meta-level reasoning. Work on the first three areas is advancing significantly, while especially meta-level reasoning is "wide-open for future development", says Kowalski. He furthermore claimed that "work in Europe is presently influencing work in North-America". J. Stefan Institute was active in this workshop: Sašo Džeroski (co-author Ivan Bratko) has presented the paper "Using the m-estimate in inductive logic programming". The proceedings of this workshop are available at the J. Stefan Institute.

Program Development (Maurice Bruynooghe, Leuven) deals with methods and techniques for the automatic development of computer programs together with techniques for improving their efficiency. Work is being done on program analysis (mainly termination analysis), program transformation (with a renewed interest in fold/unfold transformations), program synthesis (proofs as programs, derivation of programs from specification, program synthesis from examples), and abstract interpretation. Bruynooghe signalled a trend from Logic Programming in general to LP languages, and from syntax to semantics.

Programming Languages (Franco Turini, Pisa) topic deals with the design of innovative programming languages and environments based on the logic programming paradigm. Turini gave an overview of the state of the art. Research aims at bridging the gap between the conceptual power of computational logic, and practical programming, without re-inventing the wheel, and maintaining the basic ideas. Work is done on semantics (negation, from ground Herbrand models to richer models, algebraic properties of program composition and backtracking, metalogic), pushing existing techniques to their limits (e.g., metalogic, negation as failure), generalisations of existing techniques (e.g., constraint LP), importing linguistic features of other programming paradigms (e.g., modules, object orientation), and software engineering for LP (e.g., debugging).

Parallelism and Implementation Techniques (Manuel Hermenegildo, Madrid): This network area has only recently been established. One of its aims is improving the status of Computational Logic in the marketplace, by exploiting concurrent computers, and by formalisation of concurrent languages.

Invited talks were given by **U. Montanari** (Pisa): True concurrency for concurrent constraints and **Krysztof Apt** (CWI): Why the occur-check is not a problem. The occur-check (which should prevent the construction of circular bindings) is omitted from Prolog for efficiency reasons, thus rendering Prolog unsound. A solution to this problem is to let the interpreter decide where the occur check is needed. Apt presented conditions for occur-check free programs.

3 Software demonstrations

The following systems were demonstrated: UNIF, an environment for solving symbolic constraints (Claude Kirchner, INRIA); CMCAI CLP, a Prolog system featuring extensible unification for CLP (Christian Holzbaur, Vienna); FOLON, an environment for declarative construction of logic programs (Namur); and a generic abstract interpreter for Prolog (Namur). These systems are available from the authors.

4 The COMPULOG project

The COMPULOG BRA (Basic Research Action, not to be confused with the NoE, and therefore written in capitals) has just finished; it will be continued as a basic research project in ESPRIT III. Partners presented their achievements and future plans. The following topics are/have been addressed: Constraint Logic Programming (Marseille; actually they will not be in COMPULOG II, but have submitted their own project), types and objects (Kaiserslautern, Rome, Lisbon, Aachen, ECRC), non-monotonic reasoning (Lisbon, Cyprus, Amsterdam, Leuven), the meta-programming language Godel (Bristol), meta-reasoning (Pisa, London, Rome), legal reasoning (London, Uppsala), program development (Leuven, Pisa, Uppsala, Amsterdam, Edinburgh, Rome).

5 Overview of other approved ESPRIT III projects

MEDLAR (Wolfgang Bibel, Darmstadt): Mechanizing deduction in the logics of practical reasoning. This is a continuation of an ESPRIT II BRA, aiming at providing a parametrised logical framework for different kinds of practical reasoning. The framework proposed is that of labeled deduction systems, advocated by Dov Gabbay (Imperial College). Bibel also proudly announced that the German government is funding a big national project on Deduction.

PARFORCE (Manuel Hermenegildo, Madrid): Parallel formal computing environment. This project aims at building, integrating and assessing tools for global analysis and program transformation.

ACCLAIM (Janson, SICS): Advanced concurrent constraint languages: application, implementation, methodology.

ILP (Maurice Bruynooghe, Leuven): Inductive Logic Programming ESPRIT III Basic Research project no. 6020. Artificial Intelligence Laboratories of the J. Stefan Institute and Faculty of Electrical Engineering and Computer Science, Ljubljana are a partner in this project.

LAC (Antonio Porto, Lisbon): Logical account of change. This is a working group.

6 The future of Compulog NoE and ESPRIT Basic Research

Simon Bensasson (CEC, Brussels) mentioned the following two main goals of NoEs: they are funded for coordination (not for e.g., research), and they are the vehicles to make research industrially interesting, leaving Basic Research basic. On the topic of the new BRA round, Bensasson could not guarantee a starting date, which he said was due to the new procedures entered after Maastricht: the new contracts are now to be signed by the Commission, supposedly causing a considerable delay. The really bad news was not made public by Bensasson, but was spread later: the amount of money spent on Esprit III will be seriously reduced (supposedly because of necessary aid to Eastern Europe), and contracts will be signed only for one and a half year. This bad news significantly influenced a general discussion on the future of the Compulog NoE. The most optimistic remarks came from Kowalski, who said: "ESPRIT has been bad before, and they will improve again." He added that one should treat Brussels a bit less seriously, and just proceed with the things one thinks should be done. Anyway, "you shouldn't be in Compulog NoE if you hadn't joined it, had it not been funded". He conceives a role for the NoE as a sort of European chapter of the Association of Logic Programming. Others added that the NoE should focuss on activities for which one needs a network of groups rather than an association of individuals, such as preparing proposals, managing communication, contacts with industry and other networks, and acting as a pressure group. With respect to this last point, it was decided to protest firmly against the cuts in ESPRIT III, possibly seeking publicity.

Later in June this year, NoE members got more optimistic news about the starting date of ESPRIT III projects and the financing of them. The J. Stefan Institute is in particular interested in the beginning of the ILP basic research project no. 6020 which should start in September this year. ILP stands for Inductive Logic Programming, an emerging research area spawned by machine learning and logic programming whose main goal is to upgrade the classical empirical learning paradigm to a logic programming framework. There are 7 main partners and two associate partners in the project (coordinated by Maurice Bruynooghe and Luc De Raedt, Katholieke Universiteit Leuven). Ljubljana AI laboratories act as one of the main partners (principal researcher Ivan Bratko, project manager Nada Lavrač).

7 Our participation in Compulog NoE

Compulog NoE facilitates comminication, mobility and technology transfer by means of regular scientific meetings, access to electronic communication, coordination of training of graduate and postgraduate researchers and dissemination of information about publications.

Regular scientific meetings provide a firm basis for building a strong research community with shared goals and common techniques. Two types of meetings are organized each year: a general meeting covering the whole field and an area meeting for each of the subgroups. In addition, NoE organizes annual summer schools for training students and for technology transfer to industry.

Since June this year, NoE is being reorganized: network nodes are now classified as managing nodes, full nodes and associated nodes. As the emphasis of the work at the J. Stefan Institute is on the use of logic programming and machine learning rather than computational logic itself, we got classified as an associated node. We are part of the subgroup area Knowledge Representation and Reasoning (coordinated by Robert Kowalski, London). Although the participation in the NoE does not bring any financial funding, our involvement in this NoE is more than useful - we got access to many useful informations, we take part in the Compulog NoE database of scientific publications, etc. Since NoE will coordinate European research in different subareas and allocate funds which will be available through the new ESPRIT III initiative named Human capital and mobility, our involvement in this NoE is invaluable.

Acknowledgements

My participation at the Second General Meeting of the Compulog Network of Excellence was financed in part by the Slovenian Ministry of Science and Technology and by the J. Stefan Institute. The report is partially based on the text written by Peter Flach, ITK, Tilbugh University.

STATUT SLOVENSKEGA DRUŠTVA INFORMATIKA

1. SPLOŠNE DOLOČBE

Čl. 1

Ime društva je »Slovensko društvo INFORMATIKA« (v nadaljnjem besedilu: društvo). Sedež društva je v Ljubljani, Vožarski pot 12.

Čl. 2

Društvo je prostovoljno združenje vseh, ki so zainteresirani za razvoj in uporabo informatike in jih združuje, vključuje in povezuje na profesionalni podlagi. Dejavnosti, ki določajo delo društva, se določijo s klasifikacijo dejavnosti NACE REV.1. — so za društvo O/91.12. Društvo deluje predvsem na področjih informacijske tehnnologije in informacijske storitve ali informatike.

Društvo je nepolitična organizacija, sestavina civilne družbe in deluje na neprofitnih principih.

Čl. 3

Društvo deluje na območju Republike Slovenije.

Čl. 4

Društvo je pravna oseba.

Čl. 5

Društvo ima svoj znak in pečat. Znak društva je oblikovan napis INFORMATIKA. Pečat društva je krog s premerom 3 cm, v sredini katerega je znak društva, ob robu pa napis »Slovensko društvo, Ljubljana«.

Čl. 6

Društvo lahko ustanovi območne podružnice zaradi učinkovitejšega delovanja povsod, kjer je za to interes članov, in sekcije, kjer se je potrebno organizirati po interesnih področjih.

Podružnice in sekcije niso pravne osebe.

Čl. 7

Društvo se lahko včlani v sorodno tujo ali mednarodno društveno organizacijo s podobnimi nameni in cilji, predpisanimi s temi pravili pod pogojem, da dejavnost te organizacije ni v nasprotju z interesi Slovenije.

Čl. 8

Društvo obvešča o svojem delovanju širšo in ožjo javnost.

1. Ožjo javnost obvešča:
   - z razpošiljanjem vabil in zapisnikov občnega zbora članom društva;
   - z občasnimi obvestili društva;
   - s tem da so zapisniki vseh organov društva dostopni na vpogled članom društva.
2. Širšo javnost obvešča:
   - s tem, da so seje vseh organov društva javne in da se nanje vabijo osebe, ki kažejo tak interes;
   - z izdajo vabil, strokovnih publikacij in obvestil ter preko drugih sredstev javnega obveščanja.

Za zagotovitev javnosti dela je odgovoren predsednik ali član odbora, ki je za to pooblaščen.

Čl. 9

Društvo ima naslednje namene in cilje predvsem na področjih, določenih v členu 2 :

- da organizirano širi zavest in krepi pomen informacijske tehnologije in storitev za razvoj slovenske družbe in njeno vključevanje v svetovno in evropsko delitev dela;
- da pospešuje razvoj znanj med članstvom in drugimi strokovnjaki razvija medsebojno sporazumevanje, strokovno kritiko, tekmovalni duh in strokovnost;
- da pomaga in pospešuje razvoj tehnologije in znanosti;
- da spremlja, proučuje in pomaga pri razvoju, vpeljevanju in uporabi informatike;
- da skrbi za popularizacijo in pospeševanje strokovnega usposabljanja kadrov na svojem področju in na sorodnih področjih;
- da povezuje delo svojega članstva z delom ostalih strok;
- da omogoča svojemu članstvu in širši javnosti seznanjanje z najnovejšimi dosežki iz svojega področja doma in v tujini;
- da sodeljuje s podobnimi tujimi in mednarodnimi društvenimi organizacijami;
- da sodeluje z organi oblasti, političnimi in gospodarskimi organizacijami in drugimi asociacijami;
- da sodeluje pri izdelavi zakonskih predpisov in standardov, ki se nanašajo na razvoj in uporabo računalniške tehnike in informatike

v Sloveniji;
- da pri svojih članih razvija in vzpodbuja profesionalno etiko;
- da pomaga ali samo ustanovi druge organizacije civilne družbe na svojem področju, kot so na primer programerska ali podobne zbornice, in za to kandidira in prevzema koncesije za določene državne pristojnosti.

Čl. 10

Društvo uresničuje svoje namene in cilje z naslednjimi dejavnostmi:
- s prirejanjem strokovno-znanstvenih sestankov, simpozijev, seminarjev, tečajev, razstav, tekmovanj;
- z izdajanjem zbornikov, referatov in člankov, strokovnih časopisov, internega glasila, posebnih obvestil in strokovne literature;
- s posredovanjem podatkov o aktualnih problemih in najnovejših dosežkih na področju informatike doma in v svetu javnosti prek javnega obveščanja;
- s povezovanjem in vključevanjem v podobne domače, tuje in mednarodne društvene organizacije;
- z organizacijo posebnih oblik funkcionalnega izobraževanja v svoji organizaciji ali v sodelovanju z drugimi;
- s smotrnim gospodarjenjem z lastnimi sredstvi in premoženjem;
- z iniciranjem raziskovalnih in razvojnih pobud in njihovim izvajanjem;
- z iniciranjem pobud za funkcioniranje državne uprave, ki so povezane s področjem informatike, oziroma upravne tehnologije.

Čl. 11

Društvo dela strokovno prek sekcij in komisij ter podružnic:
- Sekcije so praviloma odprte vsem članom društva, ki so za delo pokazali interes. Vodi jih predsednik, nadomešča pa namestnik. Po potrebi lahko Izvršni odbor društva ustanovi nove sekcije.
- Po potrebi se oblikujejo komisije za specifične dejavnosti. Komisije delajo praviloma v trajnem sestavu in so njihove naloge bolj ekspertne. Vodi jih načelnik in ima namestnika. Člani so imenovani s strani Izvršnega odbora.
- Podružnice se lahko ustanovijo po potrebi po krajevnem principu. Pravila za volitev vodstev in način upravljanja dela podružnic določa izvršni odbor. Verificira jih naknadno. Predsednika podružnice voli njen občni zbor. Podružnice se praviloma oblikujejo po enotah lokalne uprave v Sloveniji. Podružnice samostojno določajo svoj program dela in lahko vodijo svoj račun v okviru skupnega računovodstva društva.
- Sekcije, komisije in podružnice vodijo praviloma člani izvršnega odbora ali tisti, ki jih za to pooblasti izvršni odbor.

II. ČLANSTVO

Čl. 12

Član društva lahko postane vsak državljan Republike Slovenije, ki je zainteresiran za ravoj in uporabo informatike oz. informacijske tehnologije in storitev in sprejema ta statut ter se ravna po njem.

Član društva lahko postane tudi tujec, če je njegovo delovanje v skladu z načeli naše družbe in nameni društva, navedenimi v tem statutu in ga sprejme s posebnin sklepom Izvršni odbor.

Čl. 13

Pravice članov društva so:
- da volijo in so izvoljeni v organe društva;
- da sodelujejo pri delu organov društva;
- da dajejo predloge in sugestije organom društva o delu in reševanju nalog;
- da imajo vpogled v delo organov društva in dajejo o njem pripombe.

Čl. 14

Dolžnosti članov društva so:
- da volijo v organe društva;
- da sodelujejo pri delu organov društva;
- da se ravnajo po statutu in sklepih organov društva;
- da z osebnim prizadevanjem in vzorom pripomorejo k uresničitvi delovnega programa društva;
- da redno plačujejo članarino;
- da se ravnajo po kodeksih in drugih etičnih normah, sprejetih v društvu

Člani društva in voljeni organi društva so osebno odgovorni za vestno opravljanje sprejetih nalog in

funkcij.

Čl. 15

Društvo lahko izvoli častnega predsednika, izmed članov, ki so se posebno uveljavili pri razvoju društva in njenih dejavnosti. Častni predsednik je vabljen na seje organov društva, lahko pa mu občni zbor ali izvršilni odbor poveri tudi konkretne zadolžitve.

Častni član društva lahko postane oseba, ki se je posebno uveljavila z delom pri razvoju področja dejavnosti društva in sprejema to priznanje.

Častne člane imenuje občni zbor na predlog Izvršnega odbora društva.

Čl. 16

Pravice in dolžnosti članov v organih društva so častne. Za svoje delo v organih društva člani ne prejmejo plačila. Izjema so operativno-administrativne funkcije, za katere se prizna honorar po pogodbi o delu.

Čl. 17

Članstvo v društvu preneha:
- z izstopom;
- s črtanjem;
- z izključitvijo na podlagi odločbe disciplinskega odbora društva;
- s smrtjo.

Čl. 18

Član izstopi iz društva prostovoljno, kadar poda Izvršnemu odboru društva pisno izjavo o izstopu.

Čl. 19

Član se izključi iz društva, če grobo krši pravice in dolžnosti, naštete v 13. in 14. členu statuta, če zavestno ravna proti interesom in ciljem društva in če ga disciplinski odbor spozna za odgovornega.

III. ORGANIZACIJA DRUŠTVA

Čl. 20

Organi društva so:
- občni zbor;
- izvršni odbor;
- nadzorni odbor;
- disciplinski odbor.

Mandat vseh organov društva traja 3 leta.

Občni zbor

Čl. 21

Občni zbor je najvišji organ društva in voli druge njegove organe. Sestavljajo ga vsi člani društva.

Čl. 22

Občni zbor je lahko reden ali izreden. Redni občni zbor sklicuje izvršni odbor vsako tretje leto. Izredni občni zbor se skliče po potrebi. Skliče ga lahko izvršni odbor na svojo pobudo, na zahtevo nadzornega odbora ali na zahtevo 1/4 članov. Izredni občni zbor sklepa samo o stvari, za katero je sklican.

Izvršni odbor je dolžan sklicati izredni občni zbor najkasneje v roku enega meseca po tem, ko je prejel tako zahtevo. V nasprotnem primeru lahko skliče izredni občni zbor 1/3 članov društva ali predsednik nadzornega odbora.

Sklic občnega zbora z dnevnim redom mora biti objavljen najmanj 10 dni pred občnim zborom.

Čl. 23

Občni zbor sprejema svoje sklepe z večino glasov navzočih članov. Način glasovanja določi občni zbor. Ko se glasuje o razrešnici članov organov društva, ti ne morejo glasovati.

Volitve članov organov društva so tajne.

Čl. 24

Občni zbor je sklepčen, če je ob predvidenem začetku navzočih več kot polovica članov.

Če ob predvidenem začetku občni zbor ni sklepčen, se začetek odloži za 1 uro, nakar občni zbor veljavno sklepa, če je prisotnih vsaj 10 članov, ki niso izvoljeni v organe društva.

Čl. 25

Občni zbor odpre predsednik društva in ga vodi, dokler občni zbor ne izvoli delovnega predsedstva. Poleg tega, izvoli občni zbor še zapisnikarja in dva overovatelja zapisnika, volilno komisijo, kandidacijsko komisijo, in druge delovne organe, vse iz navzočih članov društva.

Čl. 26

Občni zbor:
- sklepa o dnevnem redu;
- razpravlja o delu in poročilih izvršnega in nadzornega odbora ter sklepa o njem;

- sprejema delovni program društva;
- odloča o pritožbah proti sklepom izvršnega odbora ali odločbam disciplinskega odbora;
- sklepa o finančnem načrtu za obdobje trajanja mandata ter potrjuje zaključni račun za preteklo obdobje;
- sprejema, spreminja in dopolnjuje statut ter druge splošne akte društva ter potrjuje statutarne sklepe;
- z glasovanjem neposredno voli predsednika, podpredsednika in sekretarja društva ter člane izvršnega, nadzornega in disciplinskega odbora;
- odloča o prenehanju in včlanjevanju društva v druge organizacije doma ali v tujini;
- imenuje delegate in delegacije, ki zastopajo društvo pri drugih organizacijah, oziroma daje pooblastila za taka imenovanja;
- odloča o višini članarine ter o olajšavah, ki se priznajo članom društva ob uporabi uslug društva, ter določa valorizacijo obveznosti iz članarine;
- imenuje častne člane društva na predlog izvršnega odbora;
- daje in odvzema pooblastila Izvršnemu odboru.

Čl. 27

O delu občnega zbora se piše zapisnik, ki ga podpišejo predsednik delovnega predsedstva in oba overovatelja zapisnika.

Čl. 28

Društvo lahko za potrebe društva in njegove dejavnosti ustanovi profitno d.o.o, delniško družbo ali zadrugo in lahko ustanovi ali sodeljuje v delu katerekoli gospodarske družbe, vendar mora za to Izvršni odbor dobiti soglasje Občnega zbora.

Izvršni odbor

Čl. 29

Izvršni odbor je izvršilni organ občnega zbora in opravlja zadeve, ki mu jih naloži ta organ in druge zadeve, ki po naravi spadajo v njegovo delovno področje.

Izvršni odbor opravlja organizacijske, upravne, administrativne in strokovno tehnične zadeve in izvršuje sklepe občnega zbora in programa društva.

Čl. 30

Izvršni odbor je za svoje delo odgovoren občnemu zboru društva.

Čl. 31

Izvršni odbor sestavljajo: predsednik, podpredsednik(a) in sekretar društva, 4 izvoljeni člani odbora, predsedniki sekcij ali podružnic. Namestniki predsednikov sekcij in podružnic imajo v času nadomeščanja iste pravice in dolžnosti, kot tisti, ki jih nadomeščajo.

Predsednik društva je istočasno predsednik izvršnega odbora. Izvršni odbor se sestane po potrebi, vendar najmanj 4-krat letno.

Čl. 32

Društvo ima predsednika in podpredsednika. Predsednik (v primeru njegove odsotnosti pa podpredsednik) društva zastopa društvo pred državnimi organi in drugimi organizacijami ali tretjimi osebami po navodilih izvršnega odbora. Podpredsednika in člani si razdelijo delo, eden od njih pa po dogovoru nadomešča predsednika.

Predsednik, podpredsedniki, tajnik in dva člana Izvršnega odbora oblikujejo sekretariat Izvršnega odbora kot obliko dela, ki tekoče izvršuje naloge tega odbora.

Čl. 33

Izvršni odbor upravlja društvo v času med dvema občnima zboroma in po sklepih in programu, sprejetih na občnem zboru.

Sekcije se na osnovi potrjenaga programna praviloma sestajajo dvakrat letno, po potrebi tudi bolj pogosto. Podružnice se sestanejo najmanj dvakrat na leto.

Seje izvršnega odbora sklicuje predsednik društva. Sklic seje mora biti objavljen najmanj 7 dni pred dnevom, za katerega je seja sklicana. Izvršni odbor lahko za svoje delo in delo sekcij ter podružnic in komisij določi poslovnik.

Čl. 34

Člane Izvršnega odbora voli občni zbor za tri leta in so lahko ponovno izvoljeni. Izvršni odbor lahko kooptira največ tretjino novih članov — praviloma vodij novoustanovljenih sekcij ali podružnic. Izvršno odbor lahko na osnovi ostavke zamenja največ tri voljene člane. Zamenjavo lahko izvrši tudi, če se član tega telesa ni odzval na vabilo dvakrat zaporedoma in to neupravičeno.

Izvršni odbor imenuje tudi predsednika in člane organizacijskih in programskih odborov posameznih strokovno znanstvenih sestankov oziroma člane uredniških odborov posameznih publikacij ali glasil društva.

Čl. 35

V okviru svojega delovnega področja opravlja izvršni odbor tele zadeve:
- sklicuje občni zbor in pripravlja poročila o delu ter predloge za občni zbor;
- pripravlja predloge za splošne akte društva;
- pripravlja in sestavlja predlog za finančni načrt in zaključni račun;
- vodi evidenco članov;
- imenuje iz vrst članstva komisije, organizacijske in programske odbore ter uredništva;
- skrbi za materialno-finančno poslovanje in sredstva društva;
- neposredno skrbi za uresničevanje ciljev in nalog, ki jih določajo 8., 9. in 10. člen statuta;
- koordinira delo sekcij, organizacijskih in programskih odborov ter uredništev;
- sklepa pogodbe o delu z osebami, ki opravljajo tehnično-administrativne posle društva ter usmerja in nadzoruje njihovo delo.

Čl. 36

Izvršni odbor sprejema sklepe, če seji prisostvuje večina članov. Sklepi so sprejeti, če zanje glasuje večina navzočih članov.

Po potrebi lahko Izvršni odbor organizira dopisne seje, kjer se odloča za posamezne predloge z da ali ne. Vedno mora biti istočasno izvedeno glasovanje o potrebi dopisnega izjasnjevanja.

Nadzorni odbor

Čl. 37

Nadzorni odbor je sestavljen iz šestih članov, ki jih izvoli občni zbor za tri leta. Nadzorni odbor izvoli iz svoje srede predsednika, dva člana in dva namestnika.

Čl. 38

Naloga nadzornega odbora je, da spremlja delo izvršnega odbora in opravlja stalni nadzor nad finančnim poslovanjem ter izvajanjem statuta in drugih aktov društva. Nadzorni odbor je odgovoren občnemu zboru in mora o delu Izvršnega odbora pisno poročati ob koncu leta v javnem glasilu, če v tem letu ni občnega zbora.

Čl. 39

Nadzorni odbor sprejema veljavne sklepe, če so prisotni trije člani in če zanje glasujeta vsaj dva člana.

Člani nadzornega odbora ne morejo biti hkrati člani izvršnega odbora, vendar se praviloma vabijo na vse seje izvršnega odbora brez pravice odločanja.

Disciplinski odbor

Čl. 40

Disciplinski odbor sestavljajo trije člani, ki jih izvoli občni zbor za tri leta.

Disciplinski odbor izvoli iz svoje srede predsednika. Disciplinski odbor vodi disciplinski postopek in izreka ukrepe po disciplinskem pravilniku.

Za svoje delo je odgovoren občnemu zboru društva.

IV. MATERIALNO-FINANČNO POSLOVANJE DRUŠTVA

Čl. 41

Dohodki društva so:
- članarina;
- dohodki od prijavnin za strokovno znanstvena srečanja in prodaje publikacij;
- dotacije in subvencije;
- darila in volila;
- prihodki iz sponzorskih pogodbenih odnosov;
- drugi dohodki.

Društvo razpolaga s finančnimi sredstvi v mejah odobrenega finančnega načrta in običajih ter normah, določenih za neprofitni sektor.

Čl. 42

Premoženje društva predstavljajo vse premičnine in nepremičnine, ki so last društva in so kot take vpisane v inventarno knjigo. S premoženjem društva upravlja Izvršni odbor.

Premičnine se lahko odstopijo ali odtujijo tretjim osebam le na podlagi sklepa seje Izvršnega odbora. O nakupu in odtujitvi nepremičnin odloča Občni zbor društva.

Čl. 43

Materialno in finančno poslovanje mora biti v skladu z načeli, ki veljajo za društva in z veljavnimi predpisi. Pri določanju položaja zaposlenih, njihovega nagrajevanja, honoriranju in plačevanju materialnih nadomestil za društvo veljajo določila, ki veljajo za državno upravo.

Čl. 44

Finančno poslovanje društva se odvija v skladu z zakonom.

Blagajnik vodi finančno poslovanje društva po pravilniku materialno-finančnem poslovanju oziroma standardih, ki so za to določeni.

Denarna sredstva društva vodi blagajnik v blagajniški knjigi. Blagajnik poroča o finančnem poslovanju Izvršnemu odboru.

Čl. 45

Finančne in materialne listine podpisuje predsednik ali sekretar društva oziroma določeni pooblaščeni podpisniki.

Odredbodajalec je predsednik društva, v njegovi odsotnosti pa podpredsenik.

V. KONČNE IN PREHODNE DOLOČBE

Čl. 46

Društvo preneha obstajati:

- s sklepom občnega zbora, če zanj glasuje dvotretjinska večina navzočih članov;
- z odločbo pristojnega upravnega organa o prepovedi dela;
- če pade število članov pod 10.

Čl. 47

V primeru prenehanja društva pripadejo njegova sredstva ustrezni organizaciji, ki nadaljuje dejavnost društva v Republiki Sloveniji, ali Rdečemu križu Slovenije, če take organizacije niso ustanovili eno leto po prenehanju dela društva.

Čl. 48

V skladu s tem statutom ima društvo naslednje splošne akte:

- Disciplinski pravilnik, oziroma kodeks društva;
- Pravilnik o materialno-finančnem poslovanju;
- Poslovnik o poslovanju komisij, organizacijskih, programskih in uredniških odborov.

Splošni akti iz prejšnjega odstavka morajo biti sprejeti na prvem občnem zboru po ustanovitvi društva. Pravilnike sprejema občni zbor, pripravlja pa izvršni odbor društva.

Prehodne določbe

Čl. 49

Zaradi konsolidacije organov društva se izjemoma določi rok za nov redni občni zbor društva najkasneje eno leto po sprejemu statuta.

Čl. 50

Izvršni odbor se pooblasti za kooptiranje, poleg izvoljenih, največ štirih novih članov, ker še niso določene vse sekcije in komisije.

Čl. 51

Na prihodnjem občnem zboru naj Izvršni odbor predloži sestav sekcij in komisij kot statutarno določbo.

Čl. 52

Ta statut je sprejel občni zbor društva, dne 2. julija 1992 v Ljubljani in stopi v veljavo, ko ga overovi pristojni upravni organ. S tem dnem preneha veljati statut, sprejet na ustanovnem občnem zboru, dne 19.marca 1981.

Sekretar društva:
Erna Rupnik, l.r.

Predsednik društva:
Tomaž Banovec, l.r.

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 3 August 1992 ISSN 0350 – 5596

Informatica

A Journal of Computing and Informatics

CONTENTS