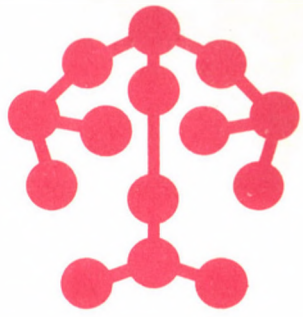


XIV

CL&CL

Computational Linguistics and Computer Languages

computer
and
automation
institute
hungarian
academy
of sciences

COMPUTER AND AUTOMATION INSTITUTE
HUNGARIAN ACADEMY OF SCIENCES

COMPUTATIONAL LINGUISTICS

AND

COMPUTER LANGUAGES

XIV.

ISSN 0324-2048

Budapest, 1980.

Responsible Publisher:

Prof. Dr. TIBOR VÁMOS
Director of the Computer and Automation
Institute, Hungarian Academy of Sciences

CL & CL

COMPUTATIONAL LINGUISTICS AND COMPUTER LANGUAGES

A scientific periodical published in English under the auspices of the
COMPUTER AND AUTOMATION INSTITUTE, HUNGARIAN ACADEMY OF SCIENCES,

Topics of the periodical:

The editorial board intends to include papers dealing with the syntactic and semantic characteristics of languages relating to mathematics and computer science, primarily those of summarizing, surveying, and evaluating, i.e. novel applications of new results and developed methods.

Papers under the heading of "Computational Linguistics" should contribute to the solution of theoretical problems on formal handling and structural relations of natural languages and to the researches on formalization of semantics problems, inspired by computer science.

Papers under the heading of "Computer Languages" should analyse problems of computer science primarily from the point of view of means of man-machine communication. For example it includes methods of mathematical logic, examining problems on formal contents and model theory of languages.

The periodical is published twice a year in December and June. Deadlines are 28 February and 31 August.

All corresponsence should be addressed to:

COMPUTER AND AUTOMATION INSTITUTE
HUNGARIAN ACADEMY OF SCIENCES
Scientific Secretariat
1502 Budapest
P.O.B. 63.

Subscription information:

Available from: JOHN BENJAMINS BV.
Periodical Trade
Amsteldijk 44 Amsterdam (Z)
HOLLAND

NOTES FOR AUTHORS

Original papers only will be considered. Manuscripts are accepted for review with the understanding that all persons listed as authors have given their approval for the submission of the paper; further, that any person cited as a source of personal communications has approved such citation.

Manuscripts should be typed in double spacing on one side of A4 (210 x 297 mm) paper, and authors are urged to aim at absolute clarity of meaning and an attractive presentation of their texts. Each paper should be preceded by a brief abstract in a form suitable for reproduction in abstracting journals.

The abstract should consist of short, direct, and complete sentences. Typically, its lenght might be 150 to 200 words. It should be informative enough to serve in some cases as a substitute for reading the paper itself. For this reason, the abstract should state the objectives of the works, summarize the results, and give the principal conclusions and recommendations. It should state clearly whether the focus is on theoretical developements or on practical questions, and whether subject matter or method is emphasized. The title need not be repeated. Work planned but not done should not be described in the abstract. Because abstracts are extracted from a paper and used separately, do not use the first person, do not display mathematics, and do not use citation reference numbers.

Number each page. Page 1 should contain the article title, author and coauthor names, and complete affiliation(s) (name of institution city, state, and zip code). At the bottom of page 1 place any footnotes to the title (indicated by superscript $^{+}, ^{+}, ^{\pm}$). Page 2 should contain a proposed running head (abbreviated form of the title) of less than 35 characters. References should be listed at the end in alphabetical order of authors and should be cited in the text in forms of author's name and date.

Diagrams should be in Indian ink on white card or on cloth. Lettering should conform to the best draughtsmanship standards, otherwise it should be in soft pencil. Captions should be typed on a separate sheet. Particular care should be taken in preparing drawings; delay in publication results if these have to be redrawn in a form suitable for reproduction. Photographs for half- tone reproduction should be in the form of highly glazed prints.

List of Symbols. Attach to the manuscripts a complete typewritten list of symbols, identified typographically, not mathematically. This list will not appear in print but is essential in order to avoid costly author',s corrections in proof (If equations are handwritten in the text then the list of symbols should also be handwritten.) Distinguisg between "oh." "zero" "el." "on": "kappa." "kay": upper and lowe case "kay"; etc. Indicate also when special type is required (German, Greek, vector, scalar, script, etc.); all other letters will be set in italic.

Authors are themselves responsible for obtaining the necessary permission to reproduce copyright material from other sources.

A,. MATHEMATICAL SEMANTICS

Computational Linguistics and Computer Languages Vol. XIV. 1980.

ADDITIONS TO SURVEY OF APPLICATIONS OF UNIVERSAL ALGEBRA, MODEL THEORY, AND CATEGORIES IN COMPUTER SCIENCE

by

Hajnal ANDRÉKA and István NÉMETI

Mathematical Institute, Hungarian Academy of Sciences
Budapest

In the original survey*/ nine main directions 1 - 9 of research were briefly outlined. Below directions 10 , 11 are added to this list.

Convention: To distinguish references to items in the original survey from references to items listed below:

Burstall-Goguen [77] refers to the original survey while

Burstall-Goguen [79]* refers to an item listed below. I.e. the asterix stands for "below".

10 THEORY MORPHISMS, STEPWISE REFINEMENT OF PROGRAM SPECIFICATIONS, REPRESENTATION OF KNOWLEDGE

To investigate connections between different theories formulated in completely different languages is a problem in many branches of computer science, e.g. in structured programming, in structuring program specifications Burstall-Goguen [77], [79]*, Goguen-Burstall [78], [79], [79]*, Dömölki [79]*,

---

*/ H.Andréka-I.Németi: Applications of Universal Algebra, Model Theory, and Categories in Computer Science. CL & CL Vol.XIII, 1979, pp. 251-282.

Mosses [79]*, in data types and semantics Blum-Estes [77], in A.I. Andréka-Gergely-Németi [72]*, McCarthy-Hayes [69]*, Andréka-Németi [79a]* etc. The "connections" between the different theories and languages can be called interpretations or "translations" but translations would suggest something much simpler than the thing we have in mind. Most often they are called "Theory Morphisms". One point to be stressed is that between two theories there are many theory morphisms usually. The subject of investigation here is actually a category consisting of theories and their morphisms. The notion of a theory morphism from the theory T into the theory T' was defined e.g. in Winkowski [78]* §1. (p.277) and there it was called a modelling μ of the theory T in the other one T'. In that paper theory morphisms are used to study "Computer Simulation".

There is a related branch of "standard" Universal Algebra called "Lattice of Varieties" see Grätzer [79]* p.389. That lattice is a special subcategory of the Category of Theories considered here. For the purposes reported here, it is too special for two reasons:

1/ The main point in the presently reviewed field is that the theories are of <u>different similarity types.</u>

2/ Between two theories there are many morphisms.

(In some Computer Science papers the Category of Theories was called "Hierarchy of Languages" to emphasize that the underlying languages, similarity types, or even logics are usually different, see Andréka-Gergely-Németi [72]*, Rattray-Rus [77], Andréka-Németi [79a]*, Sain [79b]*.)

Here we quote three approaches which are complementary (and are most useful when applied together).

(<u>10.1</u>): Let the theories Th_1 and Th_2 be equational but possibly heterogeneous (many sorted). Let $\underset{\sim}{F}_1$ and $\underset{\sim}{F}_2$ be the countably generated free algebras of the varieties $Mod(Th_1)$ and $Mod(Th_2)$ respectively. Then a suitable homomorphism $h : \underset{\sim}{F}_1 \to\to \underset{\sim}{F}_2$ could

establish a connection between the two theories i.e. between the two varieties. The problem is that $\mathfrak{F}_1$ and $\mathfrak{F}_2$ are usually of completely different similarity types! Hence the notion of a homomorphism between them is just meaningless. To help this situation, Blum-Estes [77] generalized the usual notion of homomorphism to be defined between algebras of different similarity types. One definition could be to say that $f : \underset{\sim}{A} \longrightarrow \underset{\sim}{B}$ is a generalized homomorphism if f takes every term function of $\underset{\sim}{A}$ into some term function of $\underset{\sim}{B}$. I.e. the image of an operation of $\underset{\sim}{A}$ is required to be a function definable in $\underset{\sim}{B}$. One can then make restrictions (or generalizations) on the notion of definability used e.g. "term-definable", first-order definable, implicitly definable by first order formulas etc. Then the Theory of Definability which is a branch of Model Theory, see e.g. the Chang-Keisler monograph, can be used as a guide for choosing the notion with the desired properties from a fairly broad spectrum of existing and well understood ones. For some varying choices see Blum-Estes [77], Blum-Lynch [79a]*, [79b]*, [79c]*.

(10.2): One of the main aims of Algebraic Theories of Lawvere is to deal with the present problem. I.e. the aim is to investigate the "structure" or "system" consisting of several theories of several similarity types and several possible "interpretations" (i.e. connections)between these theories and languages. Hence this is not a mere translation of Universal Algebra into category theoretical language, but instead this is an approach to a problem inherent both in Universal Algebra and in Model Theory: to a problem which has not been attacked in standard Universal Algebra or Model Theory yet. Though it should

be mentioned here that: the "Lattice of Varieties", the"Reducts" the "Clone algebras", see Grätzer [79]*, are branches in standard Universal Algebra which might be applicable to the present problem. Our reason of pointing this out is that there have been misinterpretations saying that Lawvere's Algebraic Theories were "new bottles for old wine".

Algebraic Theories have been widely applied to the quoted Computer Science problems and from these applications the theory itself benefited considerably. Some of the references are: Elgot [71], [75], Tiuryn [79a], Wand [75a], [77a]*, Goguen-Burstall [78], Goguen et al [75c], Wagner et al [77], [76], and other works of the ADJ team. The field is too extensive to make proper references here: our references are samples chosen in a random manner.

Burstall and Goguen have made a distinction here which is worth of emphasizing. Namely: Stepwise refinement of programs and stepwise refinement of Specifications are two different matters which need different tools. The former needs Rational Algebraic Theories ADJ [76]*, or iterative ones Elgot [75], while the second needs only "plain" Algebraic Theories. In program specifications, in their stepwise refinement etc no algorithms are involved. Specifications are only declarative statements. Hence in the theory (or foundations) of Specifications no algorithmic or "iterative" notions are needed.

Specifications are important in themselves. Specifications are answers to "what" while programs are answers to "how". Sometimes it is more important to understand clearly what we are trying to do than to understand how we are actually trying to do it. Consciousness of what one really does intend to do and what one does not (and what one just happens to do without really intending) is of value.

An autonomous theory of specifications in their own right is badly needed. An approach to that is Burstall-Goguen [77], [79]*, Goguen-Burstall [78], [79] , [79]*, Mosses [79]* etc.

(10.3): The category of all first order theories and their morphisms can be treated naturally by using Cylindric Algebras Németi-Sain [78]*. Here a theory corresponds to a cylindric

algebra and a theory morphism corresponds to an ordinary homomorphism between two cylindric algebras. This correspondence works both ways. A representation theorem to this effect was formulated in Németi-Sain [78]*, and Sain [79b]*.The category of theories obtained this way is complete and cocomplete, thence "Theory Procedures" of Burstall-Goguen [77]do work in this setting too. Here, the so called "Regular Cylindric Algebras" are the main tool to handle all first order theories see Németi-Sain [78]*. For Regular Cylindric Algebras see Henkin-Monk-Tarski [79]*, Andréka-Gergely-Németi [77], Németi-Sain [78]*, In the latter two references they were called "i.-finite" instead of "regular". "Base homomorphisms" of cylindric set algebras do represent theory morphisms see Sain [79b]*,Andréka-Németi [79b]*.

By this kind of Algebraic Logic one can go beyond classical first order logic and languages as was shown in Andréka-Gergely-Németi [77], and Németi-Sain [78]*. I.e. Cylindric Algebra Theory is a natural frame for doing Abstract Model Theory the importance of which stems from the fact that many of recent activities in Computer Science do belong to Abstract Model Theory Makowsky [80]*, Sain [79a]*. About other connections between cylindric or related algebras and computer science see van Emde Boas [79]*, Pratt [79b]*, Kozen [79]*,Schönfeld [78], Andréka-Németi [79c]*, Andréka-Németi-Sain [80]*, Németi [80]*.

The above quoted 3 approaches to the Category of Theory Morphisms are complementary, it is not the issue "which one is the real one" but instead: "how to use them together, how each one can profit from the others". Such connections between (10.2) and (10.3) has already been elaborated in Németi-Sain [78]* which was also intended to be an easily comprehensible introduction to (10.3) and to parts of (10.2).

11 DYNAMIC ALGEBRAS, MODEL THEORY OF DYNAMIC LOGIC, OF LOGIC FOR REASONING ABOUT PROGRAMS, ABOUT ACTIONS

If Boole's work was a breakthrough in classical logic then Dynamic Algebras Pratt [79]* are a breakthrough in dynamic logic and related systems (e.g. programverification).

In Computer Science and related fields there have been around logical systems in which reasoning about consequences of "actions" is also possible. I.e. in addition to being able to say "All humans are mortal", "Sokrates is human" etc. we are also allowed to say "After throwing the switch there will be light " or "After touching the hot stove there will be pain ". The new pattern of thought appearing in these logics are of the kind "After doing action p it will be the case that φ " where φ is a formula of classical logic. These patterns of thought are at the very core of human reasoning and hence such logics have appeared not only in Computer Science but also in "Child Psychology", "Developmental Psychology", "Linguistics", "Cybernetics", see e.g. Pask [76]. In Computer Science some of these new logics for reasoning about actions are:

- Logic for Reasoning about Programs: Floyd [67], Burstall [69], Emden-Kowalski [74], Hoare-Lauer [74], Manna [74], Janssen-van Emde Boas [77b], Harel-Pratt [77], Pnueli [77], Cook [78], Gergely-Szőts [78],Gergely-Ury [78] Andréka-Németi-Sain [79],[79a]*, [79b]*, Bergstra-Tiuryn [79]*, Németi [80a]*.
- Logic of Actions: McCarthy-Hayes [69]*, Hayes [71], E.Tóth [78] .
- Robot Logics: McCarthy-Hayes [69]*, Hayes [71],[77], Stepánková-Havel [76]*, [77]*, Stepánková [78]*.

- Logic for Representing Knowledge in A.I.: McCarthy-Hayes [69]*, Hayes [77].
- Logic for Systems which Generate Plans: McCarthy-Hayes [69]*, Warren [76]*, Bowen [79]*.
- Processlogic: Pask [76], Pratt [79c]*.
- Algorithmic Logic: Rasiowa [73], [77], Engeler [74], Mirkowska [77], [78], [80]*, van Emde Boas [78], Grabowski [78], Salwicki [78].

Dynamic logic is intended to be the common "backbone" of all these and related logics, and accordingly its aim is to find that basic structure which is common in all these logics, that basic structure which makes all of them tick Pratt [77], Parikh [78], Segerberg [77],

Just as in Boole's case, the most basic ideas again were boiled down into algebraic form, see Pratt [79a]*, [79b]*, Kozen [79]*, [79a]*, [79b]*, Andréka-Németi [79c]*, Andréka-Németi-Sain [80]*. Dynamic Algebras are intermediate between Boolean algebras and Cylindric Algebras, see Pratt [79b]*. For Cylindric Dynamic Algebras see Németi-Sain [78]*.

The Propositional Model Theory of Dynamic Logic is a clear Kripke style one Pratt [77], Segerberg [77], Pratt [79a]* which fits beautifully into the system of model theories of well understood logics. About the First Order Model Theory of Dynamic Logic there are some controversies: there is a "Standard Model Theory" school and a "Nonstandard Model Theory" school. This controversy is studied in Sántáné-Tóth-Szőts [79]* and Sain [79a]*. The latter is a deep analysis of the situation and throws some light on what is really behind all these "schools".

Behaviour of programs in nonstandard models is analogous to behaviour of sets in "non-$\in$"-models of Set Theory: There

exist models of Set Theory (ZF) in which there are infinitely descending chains, but still the Axiom of Foundation is valid in them. The reason is that one can blow up a model of ZF by e.g. ultrapowers but then the "measure" or "scale" of "being a set or not" is blown up "synchronously". Analogously, when blowing up a model of Dynamic Logic into a nonstandard one the time-scale of "program runs" is blown up synchronously and this way one obtains reasonable program behaviour.

Some references from Nonstandard (or Henkin-Type) Model Theory of Dynamic Logic are Andréka-Németi [78]*, [78a], [78], Andréka-Németi-Sain [79a]*, [79b]*, [79], Csirmaz [79]*, [79a]*, [80]*, Gergely-Ury [78], Németi [80a]*, Németi-Andréka-Sain [79a]*, Sain [79a]*. For the Standard Model Theory approach to First Order Dynamic Logic see e.g. Harel [79]*.

Remark:

In addition to the above surveyed directions 10 , 11 , the enclosed bibliography contains some new references in directions 1 - 9 too, e.g. Lehmann-Pasztor [80]* which contains results connecting seemingly different branches succesfully, and introduces new methods to investigate the "ADJ-tools of Algebraic semantics of programming".

Adámek,J. [79] : Construction of free ordered algebras. Preprint, Praha, 1979.

Adámek,J. [79a]: On the Cogeneration of Algebras. In: Mathematische Nachrichten 88 /1979/ pp.373-384.

ADJ: See Goguen et al, Thatcher et al, Wagner et al, Wright et al.

ADJ [76] : Goguen,J.A. Thatcher,J.W. Wagner,E.G. Wright,J.B.: Rational Algebraic Theories and Fixed-Point Solutions. Proc. IEEE 17th Symp. on Foundations of Computer Science /Houston, Texas/ 1976, pp.147-158.

Andréka,H. Burmeister,P. Németi,I.[80] : On Quasivarieties of Partial Algebras. /In the series: Toward a manageable model theory of Partial Algebras/ Preprint Technische Hochschule Darmstadt 1980.

Andréka,H. Gergely,T. Németi,I. [72] : Hierarchy of languages for A,I. In Hungarian. Publications of Central Res. Inst. Phys. Budapest, No. KFKI-72-46. 1972.

Andréka,H. Németi,I. [78] : A characterization of Floyd provable programs. To appear in Proc. Coll. Logic in Programming Salgótarján 1978, Colloq. Math. Soc. J.Bolyai - North-Holland.

Andréka,H. Németi,I. [79a] : Néhány magyarországi kutatás a számitástudomány matematikai alapjai terén. Proceedings of the "NJSZT Első Országos Kongresszusa", 1979. pp.5-12.

Andréka,H. Németi,I. [79b] : Base homomorphisms of generalized cylindric set algebras. Manuscript, 1979.

Andréka,H. Németi,I. [79c] : Every free algebra in the variety generated by the representable dynamic algebras is separable and representable. Preprint Math. Inst. Hung. Acad. Sci.1979. To appear in Theoretical Computer Science.

Andréka,H. Németi,I. Sain,I. [79a] : Completeness problems in verification of programs and program schemes. Proc. Coll. MFCS'79 Olomouc, Lecture Notes in Computer Science 74 pp.208-218, Springer-Verlag, 1979.

Andréka,H. Németi,I. Sain,I. [79b] : Henkin-type semantics for program-Schemes to turn negative results to positive, In L.Budach/ed/: Fundamentals of Computation Theory FCT'79, pp.18-24. Full version available from the authors as a preprint.

Andréka,H. Németi,I. Sain,I. [80] : Representable Dynamic Algebras are not first order axiomatizable and other considerations on the Representation Theory of Dynamic Algebras. Preprint, 1980.

Apt.K.R. van Emden,M.H. [80] : Contributions to the theory of Logic Programming, Preprint Dept. Comp. Sci. Univ.Waterloo Canada, 1980.

Banaschewski,B. Nelson,E. [79] : Completions of partially ordered sets as reflections. Comp. sci. techn. rep. no. 79-CS-6, McMaster Univ., Ontario, 1979.

Berman,F. [79] : A completeness technique for D-axiomatizable sematics. Proc. 11th ACM Symp. on Theory of Comp. /May 1979/ pp.160-166.

Bergstra,J. Tiuryn,J. [79] : Implicit Definability of Algebraic Structures by means of Program Properties. In L.Budach/ed/: Fundamentals of Computation Theory FCT'79. pp.58-63.

Blum,E.K. Lynch,N.A. [79a] : A difference in expressive power between flowcharts and recursion schemes. Math. Systems Theory 12, pp.205-211, 1979.

Blum,E.K. Lynch,K.A. [79b] : Relative complexity of operation sets for numeric and bit string algebras. To appear in Math. Systems Theory.

Blum,E.K. Lynch,N.A.. [79c] : Relative complexity of algebras. To appear in Math. Systems Theory.

Bowen,K. [79] : PROLOG, School of Computer and Information Syracuse Univ., 313 Link Hall, Syracuse, N.Y., 13210 USA, Preprint, 1979.

Burstall,R.M. Goguen,J.A. [79] : Some Fundamental Properties of Algebraic Theories: A Tool for Semantics of Computation. To appear in Theoretical Computer Science.

Burstall,R.M.-Goguen,J.A. 80 : The Semantics of CLEAR, a Specification Language.Abstract Software Specifications. Lecture Notes in Computer Science, Vol. 86. ,Springer,1980.

Csirmaz,L. [79] : On definability in Peano Arithmetic. Bull. Section of Logic, Wroclaw, Vol.8, No.3, pp.148-153, 1979.

Csirmaz,L. [79a] : Structure of program runs of non-standard time.Acta Cybernetica, Tom.4, Fasc.4,1980.

Csirmaz,L. [80] : Programs and Program verifications in a general setting. Preprint No 4/1980, Math.Inst.Hung.Acad. Sci. 1980.

Dömölki,B. [79] : An example of hierarchical program specification.Abstract Software Specifications. Lecture Notes in Computer Science, Vol.86. Springer, 1980.

Ehrig,H. Kreowski,H. Weber,H. [78] : Algebraic Specification Schemes for Data Base Systems. Report of the Hahn-Meister-Institute, Berlin, HMI-B 266, Febr. 1978.

van Emde Boas,P. Janssen,T.M.V. [79] : The impact of Frege's principle of compositionality for the semantics of programming and natural languages. Report 79-07 Univ. Amsterdam, 1979.

FCT'79 : L.Budach ed : Fundamentals of Computation Theory FCT'79. Band 2. Akademie-Verlag Berlin, 1979.

Goguen,J.A. Burstall,R.M. [79] : CAT, a System for the Structured Elaboration of Correct Programs from Structured Specifications. In preparation.

Goguen,J.A. Varela,F. [79] : Some Algebraic Foundations for Self-Referential System Process. Submitted to International Journal of General Systems.

Grätzer,G. [79] : Universal Algebra. Second Edition. Springer-Verlag, 1979.

Guessarian,I. [79] : About Algebraic Semantics. In the Report of the Second International Workshop on the Semantics of Programming Languages. Presented in the Rundbrief d. Fachgruppe Künstliche Intelligenz in der Gesellschaft für Informatik, No.17, 1979. Publisher: P.Raulefs, Institut für Informatik III, Univ. Bonn, Kurtfürstenstr. 74, 5300 Bonn 1.

Guessarian,I. [79a] : Program Transformation and Algebraic Semantics. Theoretical Computer Science 9 (1979) 39-65.

Harel,D. [79] : First-Order Dynamic Logic. Lecture Notes in Computer Science 68, Springer Verlag, 1979.

Henkin,L. Monk,J.D. Tarski,A. [79] : Cylindric set algebras and related structures I. Preprint Univ. of Colorado (in Boulder) , 1979.

Kozen,D. [79] : A representation theorem for models of *-free PDL. Report RC7864, IBM Research, Yorktown Heights, New York, Sept.1979.

Kozen,D. [79a] : On the duality of dynamic algebras and Kripke models. Report RC7893, IBM Research, Yorktown Heights, New York, Oct. 1979.

Kozen,D. [79b] : On the representation of dynamic algebras. Report RC7898, IBM Research, Yorktown Heights, New York, Oct. 1979.

Lehmann,D.J. [78] : On the Algebra of order. Proc. 19-th Symp. Found. Comp. Sci. pp.214-220, 1978.

Lehmann,D.J. Pasztor,A. [80] : On a conjecture of Meseguer. TECHNION, Dept. Comp. Sci., Haifa,Israel, Technical Report # 170, 1980.

Maibaum,T.S.E. [79] : The semantics of nondeterminism. Preprint Dept. Comp. Sci. Univ. Waterloo, Canada, 1979.

Maibaum,T.S.E. [79a] : IO and OI Revisited. Preprint Dept. Comp. Sci. Univ. Waterloo, Canada, 1979.

Makowsky,J.A. [80] : Measuring the expressive power of Dynamic Logics: An application of Abstract Model Theory. Preprint, Math.Inst. der Freien Universität Berlin , 1980.

McCarthy,J. Hayes,P.J. [69] : Some philosophical problems from the standpoint of artificial intelligence. Machine Intelligence 4, 1969.

Meseguer,J. [79] : Order completion monads. Preprint of the Math. Dept. Univ. of California, Berkeley, CA 94720, USA.

Meseguer,J. [80] : Varieties of chain-complete algebras. To appear in journal of Pure and Applied Algebra. Jan. 1980.

MFCS'79: Mathematical Foundations of Computer Science 1979, Proceedings, Olomouc, Czechoslovakia. Ed, J.Becvár. Springer Verlag, Lecture Notes in Computer Science 74, 1979.

Mirkowska,G. [80] : PAL - propositional algorithmic logic. To appear in Fundamenta Informaticae.

Mosses,P. [79] : Modular Denotational Semantics.In the Report of the Second International Workshop on the Semantics of Programming Languages. Presented in the Rundbrief d. Fachgruppe Künstliche Intelligenz in der Gesellschaft für Informatik, No 17, 1979 pp.50-51. Publisher: P.Raulefs, Institut für Informatik III, Univ.Bonn, Kurtfürstenstr. 74, 5300 Bonn 1.

Németi,I. [80] : Some constructions of cylindric algebra theory applied to dynamic algebras of programs. To appear in CL & CL, Budapest.

Németi,I. [80a] : Complete first order Dynamic Logic, Math. Inst.Hung.Acad.Sci. Preprint 1980. Submitted to Acta Cybernetica Szeged.

Németi,I. Andréka,H. Sain,I; [79] : Program verification within and without logic. Bull. Section of Logic. Wroclaw, Vol.8, No 3, 1979. pp. 124-129.

Németi, I. Sain,I. [78] : Connections between Algebraic Logic and Initial Algebra Semantics of CF Languages. To appear in Proc. Coll. Logic in Programming Salgótarján 1978. Colloq. Math. Soc. J.Bolyai, North Holland.

O'Donell,M.J. [77] : Computing in Systems Described by Equations. Lecture Notes in Computer Science 58, Springer Verlag 1977.

Pasztor,A. [79] : Surjections in the category of ω-continuous algebras. Preprint Univ. of Stuttgart Bericht 1/79, 1979.

Pratt,V.R. [79a] : Dynamic Algebras, MIT Preprint 1979.

Pratt,V.R. [79b] : Dynamic Logic. In the Abstracts of Invited Papers of the 6th International Congress of Logic, Methodology and Philosophy of Science, Hannover 1979.

Pratt,V.R. [79c] : Process Logic. Proc. 16th Ann. ACM Symp. on Principles of Programming Languages. January 1979.

Pratt,V.R. [79d] : Models of Program Logics. To appear in the Proc.of 20th Annual Symposium on IEEE Foundations of Computer Science.

Pratt,V.R. [80] : Dynamic algebras and the Nature of Induction. In 12th ACM Symp. on Theory of Computing, Los Angeles, CA, May, 1980.

Sain,I. [79a] : There are general rules for specifying semantics: Observations on Abstract Model Theory. Cl & CL Budapest, (Computational Linguistics and Computer Languages) Vol XIII, 1979, pp.251-282.

Sain,I. [79b] : Theories, Theory Morphisms and Cylindric Algebras. Manuscript 1979.

Sántáné-Tóth,E. Szőts,M. [79] : A report on Colloquium on Logic in Programming 10-15 September 1978, Salgótarján /Hungary/. Rundbrief der Fachgruppe Künstliche Intelligenz in der Gesselschaft für Informatik, No.17, pp.20-26, 1979. Publisher: P.Raulefs, Institut für Informatik III, Univ. Bonn, Kurtfürstenstr. 74, 5300 Bonn 1.

Smyth,M.B. Plotkin,G.D. [78] : The category theoretic solution of recursive domain equations. Dept. of Artificial Intelligence, Research Report No.60, December 1978.

Stepánková,O. Havel,I.M. [76] : A Logical Theory of Robot Problem Solving. Artificial Intelligence 7 (1976), 129-161.

Stepánková,O. Havel,I.M. [77] : Incidental and State Defendent Phenomena in Robot Problem Solving. Proceedings of AISB, Edinburgh, 1976, 266-278.

Stepánková,O. [78] : Planning in Uncertain Environments. Proceedings of AISB Conf., Hamburg, 1978, 330-339.

Thatcher,J.W. Wagner,E.G. Wright,J.B. [79] : More on Advice on Structuring Compilers and Proving them Correct. Preprint, IBM RC 7588 /#32847/ 4/2/79, 1979.

Tholen,W. [79] : General Machines and Concrete Functors. In L.Budach/ed/: Fundamentals of Computation Theory FCT'79, pp.443-461.

Tiuryn,J. [79] : Fixed points in the power set algebra of infinite trees. Schriften für Informatik und angewandten Mathematik, Bericht Nr.54, Juli 1979, Rheinisch-Westfälische Technische Hochschule Aachen.

Trnková,V. [79] :Machines and Their Behavior in a Category. In L.Budach/ed/: Fundamentals of Computation Theory FCT'79, pp.450-461.

Wagner,E.G. Wright,J.B. Thatcher,J.W. [79] : Many-sorted and ordered algebraic theories. Preprint,MIT, RC 7595 /#32868/ 4/9/79, 1979.

Warren,D. [76] : WARPLAN: a system for generating plans, DCL Memo 76, Dept. of Artificial Intelligence, Univ. of Edinburgh, 1976.

Wand,M. [77a] : Algebraic theories and tree rewriting systems. Technical Report No.66, Comp. Sci. Dept. Indiana Univ., Bloomington, Indiana 47401, 1977.

Wand,M. [77b] : First-Order Identities as a Defining Language. Technical Report No.29, Univ. Indiana. To appear in Theoretical Computer Science, 1977.

Winkowski,J. [78] : Towards an understanding of computer simulation, Annales Societatis Mathematicae Polonae,Series IV: Fundamenta Informaticae I /1978/, pp.277-289.

Wiweger,A. [79] : Pre-adjunctions and lambda-algebraic theories. Preprint Nr.184 Institute of Math. Polish Acad.Sci.,1979.

Computational Linguistics and Computer Languages Vol. XIV. 1980.

A SURVEY OF SEMANTICS OF FLOYD-HOARE DERIVABILITY

László CSIRMAZ
Mathematical Institute, Hungarian Academy of Sciences, Budapest

The aim of this paper is to give some recent results and some open problems on this subject. The reader is supposed to be familiar with the basic notation of program verification and model theory, the concepts used here are basically from [6] and [10].

1. Definitions

1.1. The syntax of programs

Here we use the expression "program" instead of "program scheme" in [10]. Let d be a similarity type, i.e. $d = \langle C,R,F,f_R,f_F \rangle$ where C,R,F are the sets of constant, relation, and function symbols respectively, and the functions f_R and f_F assign to the relation (function) symbols their arities. The set $Y = \{y_i : i \in \omega\}$ is the set of variable symbols, here ω denotes the set of natural numbers. $L = \{\ell_i : i \in \omega\}$ is the set of labels. Finally, the programs may contain the following other symbols, too:

$\leftarrow$, IF, THEN, HALT, and : .

There are three types of (labelled) commands, namely
(i) assignation of the form $\ell : y \leftarrow \tau$, here $\ell \in L$ is a label, $y \in Y$ is a variable symbol, and τ is a d-type

term [6] with variables from Y;

(ii) if-statement ℓ : IF χ THEN ℓ', here $\ell, \ell' \in L$ and χ is a d-type quantifier-free formula; finally

(iii) the halt command ℓ : HALT.

The program is a finite sequence of labelled commands, in which

- no two members have the same label,
- every label occurring in the program is a label of some command,
- the last command is a halt command, and
- (for technical reasons) this is the only halt command in the program.

The set of programs is denoted by P_d, if $p \in P_d$ then its parts are denoted as

$$p = \langle \ell_0 : u_0, \ell_1 : u_1, \ldots, \ell_{n-1} : u_{n-1}, \ell_n : \text{HALT} \rangle .$$

Moreover, the set $\{y_0, y_1, \ldots, y_{e-1}\}$ contains all the variable symbols occurring in the program p, such that (e-1) is the maximum of the indices of variables occurring in p. We call y_e the control variable of p.

1.2. The semantics of programs

While (almost) everybody agrees with our definition of program syntax, there are great deviations in the definition of semantics. Here we try to be as general as possible, and we hope that our proposal covers all the previous definitions. To define semantics, we need, first of all, the universe D of data values. We are allowed to operate on data values (by the symbols of the similarity type d), therefore D must be endowed with a structure. This structure will be denoted by $\underline{\underline{D}}$ (cf.[6,5]). The definition of the semantics of a program is a precise definition of the run of this program. But the run is not a static but a dynamic phenomenon so we should know something about the structure of "time". This structure is often

identified with ω but we need not go so far. The program run is discrete and it starts sometime. Therefore, if T denotes the set of time points, we require every $b \in T$ to have a (unique) successor time point denoted by b+1, and we require T to have a very first time point denoted by $0 \in T$. So the time structure $\underline{\underline{T}}$ is supposed to be of type t where t contains the constant symbol 0 and the function symbol "+1" of arity 1.

Finally, we should have memory registers (or locations) such that every location should be capable to contain any data value from D. The set of locations is denoted by I, this can be a finite or infinite set. And, of course, we have a function C(s,b) which assigns to the location $s \in I$ and the time point $b \in T$ the actual content $C(s,b) \in D$ of the location s at the time b. The content of location $s \in I$ may change during time, this can be expressed by $C(s,b_1) \neq C(s,b_2)$. Summarizing, the environment in which our programs run is a quadruple $\underset{\sim}{M} = \langle \underline{\underline{T}}, \underline{\underline{D}}, I, C \rangle$, where $\underline{\underline{T}}$ is the time structure, $\underline{\underline{D}}$ is the data structure, I is the set of the locations, and $C : I \times T \to D$ is the "content of... at time..." function. We refer to the quadruple $\underset{\sim}{M} = \langle \underline{\underline{T}}, \underline{\underline{D}}, I, C \rangle$ as a <u>time-model</u>, cf. [1,4,5,11,12].

Consider, e.g. the statement "$y \leftarrow y+1$" which frequently occurs in programs. Suppose that the variable symbol y corresponds to the location $s \in I$. The exact definition of the execution of this statement can be given by "$C(s,b+1) = C(s,b) + 1$".

Now, we have all the tools to define the semantics.

Let $p \in P_d$ be a program and let $\underset{\sim}{M} = \langle \underline{\underline{T}}, \underline{\underline{D}}, I, C \rangle$ be a time-model. We assume that $\underline{\underline{D}}$ is a d-type structure, and the set L of labels consists of constant terms of type d. Consequently we may associate to each labe $\ell_i \in L$ a unique member $\ell_i^* \in D$ defined by the same term, and assume that if $i \neq j$ then $\ell_i^* \neq \ell_j^*$. Let the set $\{y_0, \ldots, y_{e-1}\}$ contain all the variables occurring in the program $p = \langle \ell_0 : u_0, \ell_1 : u_1, \ldots, \ell_n : \text{HALT} \rangle$.

The (e+1)-tuple of locations $\bar{s} = \langle s_0,\ldots,s_e\rangle$ is a <u>trace</u> (of the run)of p in $\underset{\sim}{M}$, if the following statements (i), (ii) hold. Recall that y_e is the control variable.

(i) $$C(s_e,0) = \ell_0^* .$$

(ii) For every $b \in T$, $C(s_e,b)$ is an element of $\{\ell_0^*,\ell_1^*,\ldots,\ell_n^*\}$. If $C(s_e,b) = \ell_m^*$ where $m \leq n$ and the command u_m is

"$y_w \leftarrow \tau$" then $C(s_e,b+1) = \ell_{m+1}^*$,

$C(s_j,b+1) = C(s_j,b)$ for every $j < e$, $j \neq w$,

$C(s_w,b+1) = \tau[C(s_0,b),\ldots,C(s_{e-1},b)]$

"IF χ THEN ℓ" then $C(s_j,b+1) = C(s_j,b)$ for every $i < e$, and

$C(s_e,b+1)$ is either ℓ^* or ℓ_{m+1}^*

depending on whether $\chi[C(s_0,b),\ldots,C(s_{e-1},b)]$ is true or not, finally in case of

"HALT" $C(s_j,b+1) = C(s_j,b)$ for every $j \leq e$.

The data values $C(s_0,0)$, $C(s_1,0),\ldots,C(s_{e-1},0)$ form the <u>input</u> of the program, and if $C(s_e,b) = \ell_n^*$ for some $b \in T$ (i.e. the program halts) then the values $C(s_0,b),\ldots,C(s_{e-1},b)$ constitute the <u>output</u>. Now let φ_{in} and φ_{out} be two d-type first order formulas, and fix the time model $\underset{\sim}{M}$. The program p is <u>correct</u> with respect to φ_{in} and φ_{out} in $\underset{\sim}{M}$, if for every trace of p in $\underset{\sim}{M}$ whenever the input data satisfy the formula φ_{in} (in $\underset{=}{D}$), the program halts and the output data satisfy φ_{out}.

In this paper we intend to deal with Floyd-Hoare derivability which is capable to prove partial correctness only. The program p is <u>partially correct</u> if for every trace of p, whenever the input satisfies φ_{in} <u>and</u> the trace halts then the out-

put satisfies φ_{out}. We write this assertion in the form

$$\underset{\sim}{M} \models (\varphi_{in}, p, \varphi_{out}).$$

1.3. The Floyd-Hoare induction assertions method

This method introduced in R.W.Floyd and reformulated by C.A.R.Hoare is the most commonly used method for proving partial correctness from a d-type theory Th. In our case this method can be restated as follows.

Let $p \in P_d$ be a program of the form

$$p = \langle \ell_0 : u_0, \ell_1 : u_1, \ldots, \ell_n : \text{HALT} \rangle,$$

and let φ_{in} and φ_{out} be two d-type formulas. The program is Floyd-Hoare derivable from Th with respect to φ_{in} and φ_{out}, denoted by $Th \vdash (\varphi_{in}, p, \varphi_{out})$, if there are formulas Φ_m for $m \leq n$ of type d (with variables from Y) such that the following implications are derivable from Th:

(i) $\varphi_{in} \rightarrow \Phi_0$

(ii) if the command u_m is "$y_w \leftarrow \tau$" then

$$\Phi_m \rightarrow \Phi_{m+1}\,[y_w/\tau[y_0, \ldots, y_{e-1}]]$$

(iii) if the command u_m is "IF χ THEN ℓ" then

$$\Phi_m \wedge \chi[y_0, \ldots, y_{e-1}] \rightarrow \Phi_\ell$$

and

$$\Phi_m \wedge \neg\chi[y_0, \ldots, y_{e-1}] \rightarrow \Phi_{m+1}$$

(iv) $\Phi_n \rightarrow \varphi_{out}$

It is easy to see that if the time structure $\underset{\sim}{T}$ is just the set of natural numbers with the usual successor function, and $\underset{\sim}{D} \models Th$ (i.e. $\underset{\sim}{D}$ is a model for the theory Th) then $Th \vdash (\varphi_{in}, p, \varphi_{out})$ implies $\underset{\sim}{M} \models (\varphi_{in}, p, \varphi_{out})$. The converse of this claim is not true, see [4,2,9]. Here we give a new and very simple counterexample.

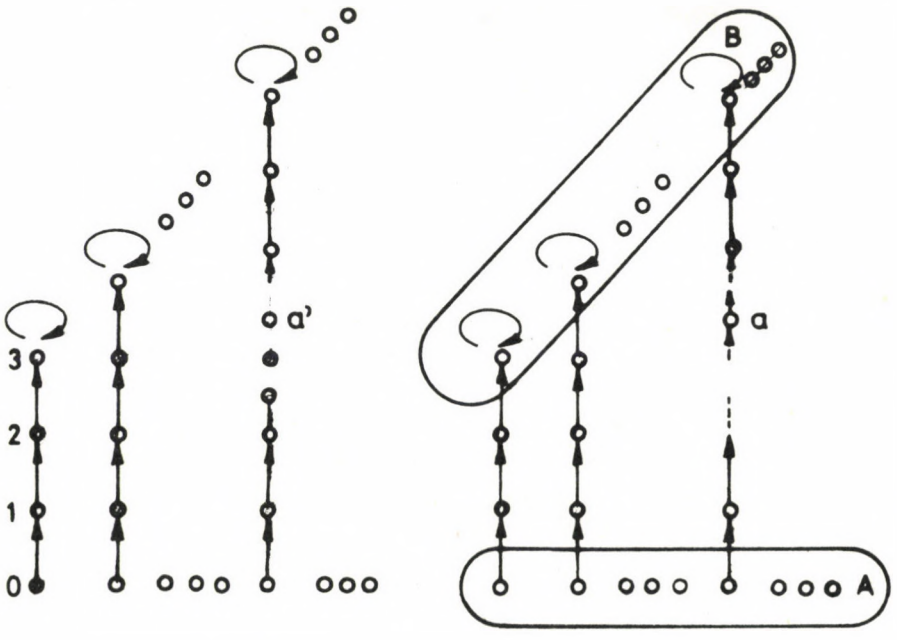
Let the similarity type d consist of the one-placed relation symbols A and B, the one-placed function symbol f, and the constants 0,1,2,3. The program in question is

$$p = \langle 0 : \text{IF } y_0 = f(y_0) \text{ THEN } 3,$$

$$1 : y_0 \leftarrow f(y_0),$$

$$2 : \text{IF } y_0 = y_0 \text{ THEN } 0,$$

$$3 : \text{HALT}\rangle.$$



Let the structure $\underline{\underline{D}}$ be as indicated on the figure. The effect of the function f is shown by the arrows. The elements satisfying the relation A(x) are just the ones which are in the cloud A. Similarly for B. Let Th be the set of formulas which are true in $\underline{\underline{D}}$ (this Th is recursively enumerable), and let $\varphi_{in} = A(y_0)$, $\varphi_{out} = B(y_0)$.

Obviously, $\underset{\sim}{M} \models (\varphi_{in}, p, \varphi_{out})$ for every time model $\underset{\sim}{M}$ the time structure of which is ω and $\underline{\underline{D}} \models \text{Th}$. But $\text{Th} \not\vdash (\varphi_{in}, p, \varphi_{out})$.

Indeed, suppose the contrary, and let $\Phi(y_0)$ be the formula attached to the first statement u_0. This formula, say, contains k symbols (including the brackets, indices, etc.) and let a be the middle element of the thread of length 10k+1

starting from A. Then, by the hypothesis, $\underline{\underline{D}} \models \Phi(a)$. But then $\underline{\underline{D}} \models \Phi(a')$ too, where a' is the middle element of the other thread of length 10k+1, because the formula Φ can speak about elements $f^i(y_0)$ with $-k < i < k$ only and they have the same properties for a and a'. So the uppermost element of the thread containing a' satisfies the formula φ_{out}, a contradiction.

2. The Results

We should like to give a semantical counterpart to the Floyd-Hoare derivability, therefore we may not restrict ourselves to time structures isomorphic to ω. A natural generalization would be allowing the time structure to be any model of some axiom system. But if there are unreachable time points (i.e. $b \in T$ such that $b \neq 0+\underbrace{1+1+\ldots+1}_{k \text{ times}}$ for $k \in \omega$) then, in general, we can say nothing about the contents of the locations at the time b. To solve this problem we require the time-models $\underset{\sim}{M}$ to satisfy some induction-type formulas. To describe the exact definition we need the following definitions.

2.1. The time-formulas

Let t be the similarity type of the time structure $\underline{\underline{T}}$, d be that of $\underline{\underline{D}}$. We have two sorts of variables. Variables of sort t are elements of $X = \{x_0, x_1, \ldots\}$ and variables of sort d are elements of $Y = \{y_0, y_1, \ldots\}$. Let the set I of locations be fixed. The elements of I will be used as constants in the time-formulas. The elements of TF, the time formulas are defined by the following (recursive) schemata.

(i) The first order formulas of t are elements of TF (with variables from X).

(ii) The first order formulas of type d are elements of TF (with variables from Y).

(iii) If $s \in I$ is a location, $y_j \in Y$, and τ is a term of type t with all variables in X, then the formula $y_j = C(s,\tau)$ is in TF

(iv) If φ and Ψ are in TF, then the formulas $\neg\varphi$, $(\varphi \wedge \Psi)$ $(\varphi \vee \Psi)$, $(\varphi \rightarrow \Psi)$ etc. and the formulas $\forall x_i \varphi$, $\exists x_i \varphi$, $\forall y_j \varphi$, $\exists y_j \varphi$ are elements of TF, too.

In the formulas TF we amalgamate time and data with the help of the content function C.

If we have a time model $\underset{\sim}{M} = \langle \underline{\underline{T}}, \underline{\underline{D}}, I, C \rangle$, and a formula $\varphi \in$ TF, then the phrase "φ is true in $\underset{\sim}{M}$ under the interpretation $b_0, b_1, \ldots, a_0, a_1, \ldots$" has its obvious meaning, and we denote it by $\underset{\sim}{M} \models \varphi[b_0, b_1, \ldots, a_0, a_1, \ldots]$, supposing that $b_i \in T$ and $a_i \in D$ for $i \in \omega$. We write $\underset{\sim}{M} \models \varphi$ if φ is true in $\underset{\sim}{M}$ under every allowed interpretation.

Now let us fix the set I of locations and the similarity types t and d. These determine uniquely the time formulas TF, let $S \subset$ TF. If in every possible time model $\underset{\sim}{M} = \langle \underline{\underline{T}}, \underline{\underline{D}}, I, C \rangle$ with the given I, $\underset{\sim}{M} \models S$ implies that the program p is partially correct in $\underset{\sim}{M}$ with respect to φ_{in} and φ_{out} (i.e. $\underset{\sim}{M} \models (\varphi_{in}, p, \varphi_{out})$) then we write this fact by $S \models (\varphi_{in}, p, \varphi_{out})$. The following important results are due to Andréka and Németi [5,1,11,14]:

Theorem 1

Suppose $S \models (\varphi_{in}, p, \varphi_{out})$. Then there is a finite subset S' of S such that $S' \models (\varphi_{in}, p, \varphi_{out})$.

Theorem 2

Let S be recursively enumerable. Then the triplets $(\varphi_{in}, p, \varphi_{out})$ for which $S \models (\varphi_{in}, p, \varphi_{out})$ holds are recursively enumerable, too.

Their proof [14,11] of the second theorem gives a calculus

which proves the partially correctness of a program if it is indeed partially correct. This calculus can be constructed effectively from the enumeration of S, i.e. this is not only an existence theorem.

But we are given a concrete calculus, namely the Floyd--Hoare method and we ask in which cases it is complete.

2.2. Axioms on the time structure

From now on we assume that the time structure satisfies a set of axioms denoted by TA (time axioms). We shall distinguish three main cases.

1. TA is just the minimal set of axioms, denoted by TA_1. It consists of formulas stating that every element has exactly one predecessor except for 0, which has no predecessor, and there is no "loop" in the time, i.e.

$$x+1 = y+1 \rightarrow x = y$$

$$x \neq 0 \rightarrow \exists y(x=y+1)$$

$$x+1 \neq 0$$

$$x + \underbrace{1 + 1 + \ldots + 1}_{k \text{ times}} \neq x \qquad \text{for every } k=1,2,\ldots$$

The models of TA_1 contain a thread isomorphic to ω, and (perhaps a lot of) threads isomorphic to the set of integer numbers.

2. The similarity type t contains the two-placed relation symbol "$\leq$" and TA is the theory of discrete linear ordering with initial element. (This theory states that $\leq$ is a linear ordering and every element has an immediate successor, and every element, except for the least one, has an immediate predecessor.) The initial element is 0, and x+1 is the smallest element greater than x. This theory, denoted by TA_2, is finitely axiomatizable. The models of TA_2 are similar to those of

TA_1 but the threads are linearly ordered.

3. TA is the set of Peano axioms for the type $\langle \leq, +, \cdot, 0, 1 \rangle$ this theory is denoted by TA_3.

Observe, that TA_1 and TA_2 are complete theories (TA_3 is not) and TA_i is a subtheory of TA_{i+1} (i.e. $TA_{i+1} \vdash TA_i$) for $i=1,2$.

If we speak about TA, we always suppose that TA_1 is a subtheory of TA, i.e. the formulas listed under point 1 are valid in every time structure $\underline{\underline{T}}$.

2.3. The induction axioms

Let $\varphi(x) \in TF$ be a time-formula such that x is a variable of sort t. Then $\varphi^* \in TF$ denotes the following formula:

$$[\varphi(0) \wedge \forall x(\varphi(x) \to \varphi(x+1))] \to \forall x \varphi(x).$$

It is important to stress here that $\varphi(x)$ may contain other free variables of sort t and d. All the free variables of $\varphi(x)$ are free in φ^* except for x. They are the parameters of the induction. Now the induction axioms are

$$IA_1 = \{\varphi^* : \varphi(x) \in TF\}.$$

We introduce a proper subset of IA_1, namely
$IA_0 = \{\varphi^* : \varphi(x) \in TF$ and $\varphi(x)$ does not contain quantifier on variable of sort t$\}$.

These formulas say that if a property changes during time then it must change some time, i.e. there is a time point $b \in T$ such that $\varphi(b)$is true and $\varphi(b+1)$ is not.

2.4. Completeness of Floyd-Hoare derivability

In this section we list the most important results concerning the Floyd-Hoare derivability. These results seem to mark out the border of the validity of this method. The first result states that our time models are reasonable enough, they do not contradict to the Floyd-proof rules [1,3,7,11].

Theorem 3

Let Th be any d-type theory, $p \in P_d$ and φ_{in}, φ_{out} be d--type formulas. Suppose $Th \vdash (\varphi_{in}, p, \varphi_{out})$. Then

$$(TA \cup IA_0 \cup Th) \models (\varphi_{in}, p, \varphi_{out}).$$

Here, of course, TA is any set of time axioms, for which $TA \vdash TA_1$, and IA_0 denotes the induction axioms of the restricted form. Because $\underset{\sim}{M} \models IA_1$ implies $\underset{\sim}{M} \models IA_0$, this theorem holds even if we change IA_0 to IA_1. Throughout $\vdash$ denotes Floyd provability.

In the following theorem we give two tables with 18 entries. The rows indicate the strength of the time structure, 1--1 row for TA_1, TA_2, TA_3. The left hand side table stands for the induction axioms IA_0 of restricted form, the right hand side table for IA_1. Finally, in the columns we indicate the status of the theory Th and the type d. The first column shows the case where Th is just the set of Peano axioms and d is just $(\leq, +, \cdot, 0, 1)$. In the second column d contains the symbols $\leq, +, \cdot, 0, 1$ and Th contains the set of Peano axioms where the Peano induction axioms are stated for the symbols $\leq, +, \cdot, 0, 1$ only. Finally, in the third column we have no restriction on Th and d.

The difference between the first and second columns is due not to the number of axioms but to the new symbols in the second column without induction axioms in Th for them. In the second column ($Th \supseteq PA$) d may contain symbols for which there is

no induction axiom in Th. Actually, all the results about the first column remain true if we replace the requirement "Th = PA" by "Th $\supset$ PA and for every formula φ of type d there is an inudction axiom about φ in Th".

Theorem 4

In the following table the entry of the square indicated by TA_i, IA_j and Th is Y if for every $p \in P_d$, $(TA_i \cup IA_j \cup Th) \models (\varphi_{in},p,\varphi_{out})$ implies $Th \vdash (\varphi_{in},p,\varphi_{out})$; the entry is N if this implication does not hold. The mark ? indicates that the result is not known.

the induction axioms are IA_0

TA is	the theory Th is: $=PA$	$\supset PA$	any
TA_1	Y	Y	Y
TA_2	Y	Y	Y
TA_3	Y	?	N

the induction axioms are IA_1

TA is	the theory Th is: $=PA$	$\supset PA$	any
TA_1	Y	Y	N
TA_2	Y	Y	N
TA_3	Y	?	N

Compare Theorem 4 with Theorem 3, Y means the completeness, and N means the incompleteness of the Floyd-Hoare derivability under the indicated conditions.

Proof

Part of the proofs can be found in [1-8]. Most of them are based on [7,13]. The proof of $[TA_2 \cup IA_0 \cup Th \models (\varphi_{in},p,\varphi_{out}) \rightarrow TA_2 \cup IA_0 \cup Th \vdash (\varphi_{in},p,\varphi_{out})]$ for <u>any</u> Th can be found in [13].

It is not without interest to mention the following result , which checks how our intuitive feelings agree with the strict notion of program traces. Recall that the trace $\bar{s} = \langle s_0,\ldots,s_e\rangle$ of the program p halts at the time point $b \in T$

if $C(s_e,b) = \ell_n^*$, and the output of the program is the e-tuple

$$<C(s_0,b),\ C(s_1,b),\ldots,\ C(s_{e-1},b)>$$

of the elements of D.

Theorem 5

With the same notation as above, the following table shows whether for every $p \in P_d$ and for every trace $\bar{s}$ of p, the outputs coincide or not. (I.e. whether the output is unique or not.)

	=PA	⊃PA	any
TA_1	Y	Y	N
TA_2	Y	Y	Y
TA_3	Y	Y	Y

	=PA	⊃PA	any
TA_1	Y	Y	N
TA_2	Y	Y	Y
TA_3	Y	Y	Y

Part of the proofs can be found in [1,3-5,7,8,11,12].

Theorems 4 and 5 can be used to compare some of the literature as follows. $TA_3 \cup IA_0$ was used in [2,3]; $TA_1 \cup IA_0$ was used in [7,8]; and $TA_3 \cup IA_1$ was used in [1,5,11,12] and in a way also in [9].

2.5. Reasoning about programs in large

The problems with theorems stated until now is that they require the use of non-standard time structures. Theorems 1 and 2 tell that in that case a nice completeness theorem holds, but someone may ask the following question. Is there no other possible method, some reasoning in large giving partial correctness for programs which works, in fact, well if the time structure is isomorphic to ω?

This new method should be a syntactical one, therefore by Gödel's incompleteness theorem, it cannot prove the partial correctness of every partially correct program with standard time . But still it could be possible that by this method one could prove partial correctness of programs which are out of the scope of theorems 1 and 2.

Now suppose that this "global" method can be described within the framework of set theory. Let d be a fixed type, Th be a theory of type d, $p \in P_d$ be a program and φ_{in} and φ_{out} be two formulas. Suppose that the 5-tuple $<d,Th,\varphi_{in},p,\varphi_{out}>$ is definable (by set theoretical tools). If the method gives that

"$\underset{\sim}{M} \models (\varphi_{in},p,\varphi_{out})$ for every $\underset{\sim}{M} = <\underline{\underline{T}},\underline{\underline{D}},I,C>$

where $\underline{\underline{T}}$ is isomorphic to ω and $\underline{\underline{D}} \models Th$ "

then the statement between the quote symbols holds in every model of set theory. But the "ω" in that models are, in general, not isomorphic to the real ω, which shows that the role of non-standard time structures is more than a successful but artificial tool for proving completeness. We feel, moreover, that there is no such "global" method which would go beyond the scope of Theorems 1-2, more precisely the following Theorem holds.

Theorem 6

Let Th be a set of first order formulas of type d. Assume that Th is recursively enumerable. Then there are a similarity type t, a set I of locations and a recursively enumerable set $S \subset TF$ of time formulas such that for every triple $(\varphi_{in},p,\varphi_{out})$ conditions (i) and (ii) below are equivalent.

(i) $S \models (\varphi_{in},p,\varphi_{out})$

(ii) For every model $\underline{\underline{W}}$ of ZFC we have
$\underline{\underline{W}} \models$ "for every time model $\underset{\sim}{M}$ with standard time such that $\underset{\sim}{M} \models Th$ we have $\underset{\sim}{M} \models (\varphi_{in}, P, \varphi_{out})$".

Proof

The proof is based on results in [1,11] and on Theorems 2.1 and 2.10 of [12].

3. APPENDIX (Equivalence with the Classical Formalism)

The formalism of [1,4,5,11,12] is slightly different from the one used in the present paper. In this section we show that our formalism is completely equivalent with their formalism and therefore all the results stated in the present paper can be applied to [1,4,5,11,12].

Recall that a time model $\underset{\sim}{M} = \langle \underline{\underline{T}}, \underline{\underline{D}}, I, C \rangle$ is a 3-sorted structure the sorts being t, d and i (where T, D, I are the universes of sorts t, d, i). Recall that X and Y are the sets of variables of sorts t and d respectively. Let $Z = \{z_j : j < \omega\}$ be the set of variables of sort i. (Of course, X, Y and Z are pairwise disjoint.)

Let F_{td} be the set of all first order formulas in the language of $\underset{\sim}{M}$ with variables in X, Y, Z as defined below.

DEFINITION

F_{td} and Te_d are the smallest sets satisfying conditions (i) - (vii) below.

(i) The first order formulas of type t with variables in X are in F_{td}.

(ii) $(z_i = z_j) \in F_{td}$ for all $i,j \in \omega$.

(iii) $C(z_i,\tau) \in Te_d$ for all $i \in \omega$ and for all term τ of type t with variables in X.

(iv) If f is an n-ary function symbol in d and $\tau_1,\ldots,\tau_n \in Te_d$ then $f(\tau_1,\ldots,\tau_n) \in Te_d$. Further $Y \subset Te_d$.

(v) If $\tau_1, \tau_2 \in Te_d$ then $(\tau_1 = \tau_2) \in F_{td}$.

(vi) If $\tau_1,\ldots,\tau_n \in Te_d$ and r is an n-ary relation symbol of type d then $r(\tau_1,\ldots,\tau_n) \in F_{td}$.

(vii) If $\varphi,\psi \in F_{td}$ then $\neg\varphi \in F_{td}$, $(\varphi \wedge \psi) \in F_{td}$, $\exists x_i\varphi \in F_{td}$, $\exists y_i\varphi \in F_{td}$, $\exists z_i\varphi \in F_{td}$.

Now, clearly TA_1, TA_2, $TA_3 \subset F_{td}$.

Let $\varphi(x_0) \in F_{td}$. Then φ^* is the following formula

$([\varphi(0) \wedge \forall x_0(\varphi(x_0) \to \varphi(x_0+1))] \to \forall x_0\varphi(x_0))$

Clearly $\varphi^* \in F_{td}$. We define

$Ia_1 = \{ \varphi^* : \varphi \in F_{td}\}$. We define

$Ia_0 = \{\psi^*: \psi \in F_{td}$ and ψ contains no quantifier of sort t$\}$.

I.e. let $\psi^* \in Ia_1$. Then $\psi^* \in Ia_0$ iff for all $i \in \omega$ "$\exists x_i$" does not occur in ψ.

PROPOSITION 7

Let $Th \subset F_{td}$ be such that $(\forall i \in \omega) z_i$ does not occur in Th. (Note, that then $Th \subset TF$.)

Let $p \in P_d$ and let e be as in Section 1.1. Let $\underline{\underline{T}}$ and $\underline{\underline{D}}$ be fixed. Let $s_i : T \to D$ be a function for all $i \leq e$. Then statements (i) and (ii) below are equivalent, for every $j < 2$.

(i) $\langle s_i : i \leq e\rangle$ is a trace of p in some $\langle \underline{\underline{T}},\underline{\underline{D}},I,C\rangle \models$

$\models (Th \cup Ia_j)$ such that $(\forall i \leq e)(\forall r \in T)C(s_i,r) = s_i(r)$

(ii) $\langle s_i : i \leq e\rangle$ is a trace of p in some $\langle \underline{\underline{T}},\underline{\underline{D}},I,C\rangle \models$

$\models (Th \cup IA_j)$ such that $(\forall i \leq e)(\forall r \in T)C(s_i r)=s_i(r)$.

Proof

Let $j < 2$.

(1) Assume (ii). Then $\langle \underline{\underline{T}},\underline{\underline{D}},I,C\rangle \models (Th \cup IA_j)$ and

$(\forall i \leq e)[s_i \in I$ and $C(s_i,r) = s_i(r)$ for all $r \in T]$.

Let $S = \{s_i : i \leq e\}$ and let $\underset{\sim}{M} = \langle \underline{\underline{T}},\underline{\underline{D}},S,C\rangle$ or more precisely $\underset{\sim}{M} = \langle T,D,S,S\times T\restriction C\rangle$ where $S\times T\restriction C$ is the function C domain-restricted to $S\times T$. Clearly $\langle s_i : i \leq e\rangle$ is a trace of p in $\underset{\sim}{M}$. To prove (i) it is enough to prove $\underset{\sim}{M} \models Ia_j$.

We define a function $h : F_{td} \to TF \cup F_{td}$ by recursion.
$h(z_i=z_k) \overset{d}{=} \forall x_0(C(z_i,x_0) = C(z_k,x_0))$ for all $i,k \in \omega$.
Let $\varphi \in F_{td}$ be any atomic formula such that
$\varphi \notin \{(z_i=z_k) : i,k \in \omega\}$. Then we let $h(\varphi) \overset{d}{=} \varphi$. (By these h is defined on all atomic formulas in F_{td}.)

Assume that $h(\varphi)$ has already been defined. Let $i \in \omega$.
Then $h(\exists z_i\varphi) = [\varphi(z_i/s_0) \vee \varphi(z_i/s_1) \vee \ldots \vee \varphi(z_i/s_e)]$
where $\varphi(z_i/s_k)$ is obtained from φ by replacing every free occurrence of z_i in φ by s_k.
$h(\exists x_i\varphi) \overset{d}{=} \exists x_i h(\varphi)$, $h(\exists y_i\varphi) \overset{d}{=} \exists y_i h(\varphi)$ and $h(\neg\varphi) \overset{d}{=} \neg h(\varphi)$. Assume that $h(\varphi)$, $h(\psi)$ are defined, then we let $h(\varphi \wedge \psi) \overset{d}{=} (h(\varphi) \wedge h(\psi))$.
The definition of $h : F_{td} \to TF \cup F_{td}$ is completed.

Let $\varphi \in F_{td}$ be arbitrary. Then we define $h^+(\varphi)$ to be the formula obtained from $h(\varphi)$ by replacing every subformula in $h^+(\varphi)$ of the form $\forall x_0(C(s_i,x_0) = C(s_k,x_0))$

with TRUE if $s_i = s_k$ and with FALSE if $s_i \neq s_k$. Since $s_i (i \leq e)$ are constants we have $\underset{\sim}{M} \models h(\varphi)$ iff $\underset{\sim}{M} \models h^+(\varphi)$.

Now it is easy to check that statements (*) - (***) below hold, for every $\varphi \in F_{td}$.

(*) If φ contains no free variables then

$[h(\varphi) \in TF$ and $h^+(\varphi) \in TF]$.

(**) $\underset{\sim}{M} \models \varphi$ iff $\underset{\sim}{M} \models h(\varphi)$ iff $\underset{\sim}{M} \models h^+(\varphi)$.

(***) Assume that all the free variables of φ are in $\bar{x},\bar{y},\bar{z}$ resp. Let φ' be the formula $\forall\bar{x}\ \forall\bar{y}\ \forall\bar{z}\ \varphi$. Then if $\varphi \in Ia_j$ then $h^+(\varphi') \in IA_j$.

Note that (***) holds for φ, $h(\varphi)$ and j=1 but for j=0 we need $h^+(\varphi')$.

To be precise, instead of $h^+(\varphi') \in IA_j$ we should have written $IA_j \models h^+(\varphi')$ in (***) above.

Let $\varphi \in Ia_j$ and let $\bar{x},\bar{y},\bar{z}$ contain all the free variables of φ. Then let φ' be the formula $\forall\bar{x}\ \forall\bar{y}\ \forall\bar{z}\ \varphi$. Clearly $\underset{\sim}{M} \models \varphi$ iff $\underset{\sim}{M} \models \varphi'$ iff $\underset{\sim}{M} \models h^+(\varphi')$ by statement (**). By (***) we have $h^+(\varphi') \in IA_j$ for all $\varphi \in Ia_j$. By the assumption (ii) we have $\langle \underline{\underline{T}},\underline{\underline{D}},I,C\rangle \models IA_j$ hence $\underset{\sim}{M} \models IA_j$ too and thus $\underset{\sim}{M} \models Ia_j$ by the aboves. Clearly $\langle \underline{\underline{T}},\underline{\underline{D}},I,C\rangle \models Th$ iff $\underset{\sim}{M} \models Th$ and thus $\underset{\sim}{M} \models (Th \cup Ia_j)$ by our assumption (ii). We have proved (ii) → (i).

(2) Assume (i). Then $\langle \underline{\underline{T}},\underline{\underline{D}},I,C\rangle \models (Th \cup Ia_j)$ and $S = \{s_i : i \leq e\} \subset I$ and $(\forall s \in S)(\forall r \in I)C(s,r) = s(r)$. Clearly $\langle \underline{\underline{T}},\underline{\underline{D}},S,S \times T^1 C\rangle \models Th$. Let the only "location-constants" occurring in IA_j be the elements of S. Then $\langle \underline{\underline{T}},\underline{\underline{D}},I,C\rangle \models IA_j$ and hence $\underset{\sim}{M} \stackrel{d}{=} \langle \underline{\underline{T}},\underline{\underline{D}},S,S \times T^1 C\rangle \models \models IA_j$. This completes the proof of the proposition.

The language F_{td} and the axioms Ia_j $(j < 2)$ were used in [1,4,5,11,12] . These papers were based on axiom systems of the form $(TA_{i+1} \cup IA_j)$ for various choices of $i < 3$, $j < 2$.

By Proposition 7 above our Theorems 3-6 can be applied to the above quoted papers too.

Next we turn to the so called Continuous Traces approach [2,3,7,8]. We shall show that it is a special case of our present formalism. First we quote the definition of continuous traces from the literature.

DEFINITION ([2,3,7,8])

Let $p \in P_d$ and $\underline{\underline{D}}$ be a d-type structure. Let $\{y_0,\ldots,y_{e-1}\}$ contain all the variables occurring in p. Let $s_0,\ldots,s_e \in {}^{T}D$ for some set T. Let $\bar{s} = \langle s_0,\ldots,s_e\rangle$. Then $\bar{s}$ is said to be a continuous trace of p in $\underline{\underline{D}}$ if conditions (i) -(iii) below hold for some $f : T \to T$ and $k \in T$.

Note that $\langle T,k,f\rangle$ is a structure of type t such that k is the interpretation of the constant symbol 0 and f is the interpretation of the function symbol "+1" in $\langle T,k,f\rangle$.

(i) $\langle T,k,f\rangle \models TA_1$

(ii) $\langle T,k,f\rangle$, $\underline{\underline{D}}$ and $\bar{s}$ satisfy the conditions in the definition of a trace (in Sec.1.2. of the present paper) if 0 is replaced by k, b+1 is replaced by f(b) and $C(s_i,b)$ is replaced by $s_i(b)$ everywhere.

(iii) Notation $\bar{s}(b) = \langle s_0(b),\ldots,s_e(b)\rangle$ for any $b \in T$. For any formula $\varphi(x_0,\ldots,x_e)$ of type d it is true that

$$\underline{\underline{D}} \models [(\varphi(\bar{s}(k)) \wedge \bigwedge_{b\in T} [\varphi(\bar{s}(b)) \to \varphi(\bar{s}(f(b)))]) \to \bigwedge_{b\in T} \varphi(\bar{s}(b))].$$

PROPOSITION 8

Let p, $\underline{\underline{D}}$ and e be as in the above definition. Let $s_0,\ldots,s_e \in {}^{T}D$ for some set T. Then statements (i) and (ii) below are equivalent.

(i) $\langle s_0,\ldots,s_e\rangle$ is a continuous trace of p in $\underline{\underline{D}}$.

(ii) There is $\underset{\sim}{M} = \langle \underline{\underline{T}}, \underline{\underline{D}}, I, C\rangle$ such that a.) - c.) below hold.

a.) $\underset{\sim}{M} \models (TA_1 \cup IA_0)$.

b.) $\langle s_0,\ldots,s_e\rangle$ is a trace of p in $\underset{\sim}{M}$.

c.) $(\forall i \leq e)(\forall b \in T)\ \ C(s_i,b) = s_i(b)$.

Since [2,3,7,8] were based on continuous traces the results of the present paper can be applied to these works too by Proposition 8 above. By Propositions 7-8 above we have a connection between all the quoted approaches.

Note that in (iii) of the definition of continuous traces above φ may have free variables x_j, $j > e$. These x_j are the parameters of the induction in (iii).

REFERENCES

[1] Andréka,H. Csirmaz,L. Németi,I. Sain,I.: More Complete Logics for Reasoning about Programs.Preprint Mathematical Inst.Hung.Acad.Sci. Budapest, /1980/ .

[2] Andréka,H. Németi,I.: Completeness of Floyd Logic. Bulletin of Section of Logic,Vol.7,No.3,Wroclaw, /1978/ pp.115-121.

[3] Andréka,H. Németi,I.: A Characterization of Floyd Provable Programs, Proc. Coll. Logic in Programming Salgótarján /1978/ Colloq. Math.Soc.J. Bolyai, North-Holland. To appear.

[4] Andréka,H. Németi,I. Sain,I.: Completeness Problems in Verification of Programs and Program Schemes. Proc.Coll.MFCS 79 Olomouc. Lecture Notes in Computer Science 74,Springer Verlag /1979/ pp. 208-218.

[5] Andréka,H. Németi,I. Sain,I.: Henkin-Type Semantics for Program Schemes to Turn Negative Results to Positive. Proc.Coll.FCT 79 Berlin,Akademie Verlag,Berlin /1979/ pp.18-24.

[6] Chang,C.C. Keisler,H.J.: Model Theory, North-Holland, /1973/ .

[7] Csirmaz,L.: Completeness of Floyd-Hoare Programverification submitted to the Journal of Symbolic Logic.

[8] Csirmaz,L.: Program Runs in Nonstandard Time. Acta Cybernetica,Szeged,Tom.4. /1980/ pp.325-331.

[9] Gergely,T. Ury,L.: Mathematical Theory of Programming. Budapest, /1978/ ,Manuscript.

[10] Manna,Z.: Mathematical Theory of Computation, McGraw Hill, /1974/ .

[11] Németi,I.: A Complete Dynamic Logic.Preprint Math. Inst. Hung. Acad. Sci. /1980/ .

[12] Sain,I.: There are General Rules for Specifying Semantics Observations on Abstract Model Theory, CL and CL Budapest, 13 , /1979/ pp.251-282.

[13] Csirmaz,L.: On Completeness of Proving Partial Correctness, Math. Inst.Hung.Acad.Sci. Preprint No. 19/1980, Acta Cybernetica,Szeged to appear.

[14] Andréka,H., Németi,I.,Sain,I.: A Complete Logic for Reasoning about Programs via Nonstandard Model Theory. Theoretical Computer Science, to appear.

Computational Linguistics and Computer Languages Vol. XIV. 1980.

SOME CONSTRUCTIONS OF CYLINDRIC ALGEBRA THEORY APPLIED TO DYNAMIC ALGEBRAS OF PROGRAMS

by

István NÉMETI

Mathematical Institute, Hungarian Academy of Sciences
Budapest, Hungary

Dynamic Algebras were introduced in Pratt [79] and Kozen [79] to investigate programs, program schemes, dynamic logics of programs and other subjects of Computer Science. Pratt [79a] pointed out that Dynamic Algebras are related to Cylindric Algebras. In this paper we intend to initiate directions of applying the theory of Cylindric and related algebras to Dynamic Algebras, and more generally we intend to initiate work on the connections between these fields.

Our main reference is the monograph: "Henkin-Monk-Tarski [71]: Cylindric Algebras" which will be referred to as HMT[71]. In this paper we shall use the notations of HMT[71]. E.g. throughout α is an arbitrary ordinal. Throughout t_α denotes the following fixed similarity type (of algebras)

$$t_\alpha = \{\langle +,2\rangle,\langle -,1\rangle,\langle c_i,1\rangle,\langle d_{ij},0\rangle \quad : \quad i,j \in \alpha\}$$

e.g. any Boolean algebra is of similarity type t_0 and a cylindric algebra of dimensions α is of type t_α.

BA denotes the class of all Boolean algebras and by "A is a BA" we mean to write $A \in$ BA. Similar convention will be used

for other classes of algebras, e.g. by "a Bo_α A" we understand "an α-dimensional Boolean algebra with operators" see HMT[71] and Definition 1 below.

DEFINITION 1 (HMT[71] Def.2.7.1. p 430)

By a Bo_α we understand an algebra A of type t_α such that $A = \langle A,+,-,c_i,d_{ij}\rangle_{i,j \in \alpha}$, $\langle A,+,-\rangle \in BA$ and

$(\forall i < \alpha)\ A \models (c_i(x+y) = c_ix + c_iy)$.

A is normal if $(\forall i < \alpha)\ A \models c_i0 = 0$ where $0 = -(x+-x)$.

More generally, let H be any fixed set. By a Bo_H we understand an algebra $A = \langle A,+,-,c_i,d_{ij}\rangle_{i,j \in H}$ such that $\langle A,+,-\rangle \in BA$ and $(\forall i \in H)\ A \models (c_i(x+y) = c_ix + c_iy)$. A is normal if $(\forall i \in H)\ A \models c_i0 = 0$ where 0 was defined above.

END of Definition 1

REMARK: Bo_α is a variety defined by schemes of equations in the sense of Andréka-Németi [78].

Throughout we shall use the derived operations $\cdot$, 0, 1 and the relation $\leq$. E.g. let $A = \langle A,+,-,c_i,d_{ij}\rangle_{i,j \in \alpha} \in Bo_\alpha$ and let $x,y \in A$. Then $x\cdot y = -(-x+-y)$, $0 = -(x+-x)$, $1 = x+-x$ and $x \leq y$ means $x+y = y$.

DEFINITION 2 (HMT[71] Def.2.2.1)

Let $A \in Bo_\alpha$ and $w \in A$.

$rl_w = rl_w^A = \langle w\cdot x : x \in A\rangle$. Clearly, $rl_w^A : A \to A$.

$Rl_w A = Rg\ rl_w^A$ (i.e. $Rl_wA = \{w\cdot x : x \in A\}$) .

$Rl_w A = \langle Rl_wA,\ rl_w \circ f^A : f \in Do\ t\rangle$ i.e.

$Rl_w A = \langle Rl_wA,\ +^w,\ -^w,\ c_i^w,\ d_{ij}^w\rangle_{i,j \in \alpha}$ where $(\forall i,j \in \alpha)$

$(\forall x,y \in Rl_wA)\ [x +^w y = x +^A y,\ -^w(x) = w\cdot -^A(x),\ c_i^w x = w.c_i^A x$ and

$d^{w}_{ij} = w \cdot d^{A}_{ij}]$.

END of Definition 2

Proposition 1 below says that $Rl_w A$ is a Bo_α even if rl_w is not a homomorphism.

PROPOSITION 1

Let $A \in Bo_\alpha$ and $w \in A$. Then $Rl_w A \in Bo_\alpha$. If A is normal then so is $Rl_w A$.

Proof

We have to prove that $Rl_w A \models c_i(x+y) = c_i x + c_i y$ for all $i < \alpha$, because rl_w is always a homomorphism on BA-s. Let $x,y \in Rl_w A$. Then $c^w_i x = w \cdot c^A_i x$. We shall write c_i for c^A_i.

$$c^w_i(x+y) = w \cdot c_i(x+y) = w \cdot (c_i x + c_i y) = (w \cdot c_i x) + (w \cdot c_i y) =$$

$$= c^w_i x + c^w_i y \quad .$$

Suppose, that A is normal, i.e. that $c_i 0 = 0$. Then $c^w_i 0 = w \cdot c_i 0 = w \cdot 0 = 0$, i.e. $Rl_w A$ is normal, too.

QED of Proposition 1

Recall from HMT[71] the followings. Let $h : A \to B$. Then $h \in Hom(A,B)$ iff h is a homomorphism from A into B, i.e. iff $h : A \to B$. $h \in Ho(A,B)$ iff h is onto B. Further $h \in Ho(A)$ iff $(\exists B)\ h \in Hom(A,B)$. We define $\ker(h) = \{\langle a,b\rangle : h(a) = h(b)\}$. Co A denotes the set of all congruence relations on A.

PROPOSITION 2

Let A be an algebra of type t_α. Assume $A \models x \cdot (x \cdot y) = x \cdot y$. Let $w \in A$. Then statements (i) and (ii) below are equivalent.

(i) $rl_w^A \in Ho(A)$ (i.e. $\ker(rl_w^A) \in Co\ A$)

(ii) $rl_w^A \in Ho(A, R\mathcal{l}_w\ A)$ (i.e. $rl_w^A : A \to R\mathcal{l}_w\ A$)

Proof

(ii) $\to$ (i) is obvious, hence we prove only (i) $\to$(ii). Let A be of type t_α, $w \in A$ and assume $rl_w \in Ho(A)$, i.e. assume $\ker(rl_w^A) \in Co\ A$. Let f be an arbitrary operation symbol of A such that $t_\alpha(f) = n$. Let $a_1,\ldots,a_n \in A$. Then by $(\forall i \leq n) rl_w(a_i) = rl_w(w \cdot a_i)$ we have $rl_w(f^A(a_1,\ldots,a_n)) = rl_w^A(f^A(w\cdot a_1,\ldots,w\cdot a_n))$, since $\ker(rl_w)$ is a congruence. Then $rl_w(f^A(a_1,\ldots,a_n)) = rl_w(f^A(w\cdot a_1,\ldots,w\cdot a_n)) = f^w(rl_w(a_1),\ldots,rl_w(a_n))$, where f^w is the interpretation of the operation symbol f in the algebra $R\mathcal{l}_w\ A$. This proves that $rl_w : A \to R\mathcal{l}_w\ A$. Clearly $rl_w : A \twoheadrightarrow Rl_w\ A$ by definition.

QED of Proposition 2

THEOREM 1

Let A be a Bo_α and let $w \in A$. Then (i) - (iii) below hold.

(i) Assume that $(\forall i < \alpha)\ c_i(-w) \leq -w$. Then $rl_w \in Ho(A)$.

(ii) Assume that A is normal. Then $rl_w \in Ho\ A$ iff $(\forall i < \alpha) c_i\ -w \leq -w$.

(iii) Assume that A is not normal. Then there are $v \in A$, $i < \alpha$ such that $c_i - v > -v$ and $rl_v \in Ho(A)$.

Proof

1.) Proof of (iii): Assume that A is not normal. Then there is $i < \alpha$ such that $c_i 0 > 0$. Let $v = 1^A$. Then $-v = 0^A$ and thus $c_i - v > -v$. Since $BA \models \forall x(1\cdot x = 1)$ we have $rl_v \in Ho(A)$ since $rl_v : A \twoheadrightarrow A$ is an automorphism of A since $rl_1 = A\,1\,Id$.

2.) Proof of (i): Let A be a Bo_α, $w \in A$ and let $(\forall i < \alpha)c_i -w \leq -w$ in A. Clearly $rl_w \in Ho(\langle A,+,-,d_{ij}\rangle)$ always holds. Thus, we have to prove $(\forall i < \alpha)rl_w \in Ho(\langle A,c_i\rangle)$. Let $i < \alpha$ and $x \in A$ be fixed. By Proposition 2 it is enough to prove that $rl_w^A(c_i^A x) = c_i^w rl_w(x)$ where c_i^w is the interpretation of c_i in $Rl_w\mathcal{A}$.

Fact 1: $Bo_\alpha \models \forall y \forall z \ (y \leq z \rightarrow c_i y \leq c_i z)$.
Fact 1 is true because $y \leq z \rightarrow y+z=z \rightarrow c_i y + c_i z = c_i z \rightarrow$
$\rightarrow c_i y \leq c_i z$.

By the hypothesis on w and by Fact 1 we have $-w \geq c_i -w \geq$
$\geq c_i(x\cdot -w)$ and thus $w\cdot c_i(x\cdot -w) = 0$. Since $x = (x\cdot w + x\cdot -w)$ we have $c_i x = c_i(x\cdot w) + c_i(x\cdot -w)$. Using these facts we obtain

$$rl_w(c_i x) = w\cdot c_i x = w\cdot(c_i(x\cdot w) + c_i(x\cdot -w)) = w\cdot c_i(w\cdot x) +$$
$$+ w\cdot c_i(x\cdot -w) = w\cdot c_i(w\cdot x) + 0 = w\cdot c_i(w\cdot x) = c_i^w(rl_w(x)).$$

We have seen $rl_w \in Ho(A)$.

3.) Proof of (ii): Assume that A is normal and $w \in A$, $i < \alpha$ are such that $c_i -w \neq -w$. Let $z = w\cdot c_i -w$. Then $z > 0$. Let $x = -w$. Then $rl_w(c_i x) > 0$, $rl_w(x) = 0 = rl_w(0)$, and $rl_w(c_i 0) = 0$. Thus $\langle x,0\rangle \in \ker(rl_w)$ but $\langle c_i x, c_i 0\rangle \notin \ker(rl_w)$. Hence $rl_w \notin Ho(A)$.

QED of Theorem 1

As a contrast to Theorem 1 see Proposition 7.

The condition $A \models c_i(x+y) = (c_i x + c_i y)$ is needed in Theorem 1 and cannot be replaced with the condition that c_i be monotonic, increasing and normal as Proposition 3 below shows.

PROPOSITION 3

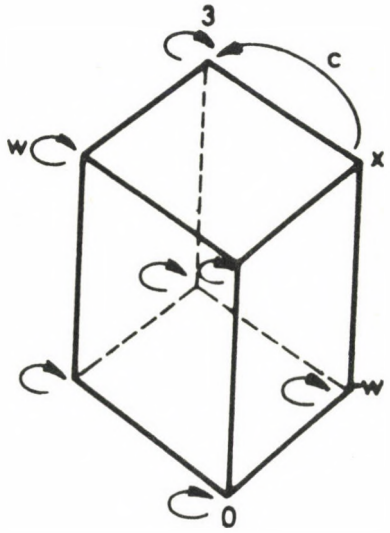
Let $\alpha > 0$.

Then there are an algebra A of type t_α and $w \in A$ such that $\langle A,+,- \rangle \in BA$ and for every $i < \alpha$

$$A \models [c_i x + c_i(x+y) = c_i(x+y) \quad \text{and}$$
$$x + c_i x = c_i x \quad \text{and}$$
$$c_i 0 = 0 \quad],$$

for every $i < \alpha$ we have $c_i - w = -w$ and $c_i w = w$ but $rl_w^A \notin Ho(A)$.

Proof



Let $A = Sb\ 3$, $w = 2$, $x = \{1,2\}$.

For every $i < \alpha$ we define $c_i = (A \uparrow Id)_3^x$.

$A = \langle A, \cup, \sim_3, c_i, 0 \rangle_{i,j \in \alpha}$. See the Figure. Then $rl_w\ c_i x = w \cdot c_i x = 2 \cap c_i x = 2 \cap 3 = 2 = w$.

$$c_i^w rl_w x = w \cdot c_i(w \cdot x) = w \cap c_i(2 \cap \{1,2\}) = w \cap c_i\{1\} =$$
$$= w \cap \{1\} = \{1\} \neq w.$$

QED of Proposition 3

Recall from HMT[71] that CA_α is the variety of α-dimensional cylindric algebras and that $Cr_\alpha = \{Rl_w\ \mathcal{A} : w \in A \text{ and } \mathcal{A} \in CA_\alpha\}$. Note that $CA_\alpha \subsetneq Cr_\alpha$.

Recall from HMT[71] that $\Delta^{\mathcal{A}} x = \Delta x = \{i < \alpha : c_i x \neq x\}$ for every $Bo_\alpha\ \mathcal{A}$ and every $x \in A$.

The following corollary is a generalization of 2.3.26. of HMT[71].

COROLLARY 4

Let $\mathcal{A} \in HSP\ Cr_\alpha$ and let $w \in A$. Then

$$rl_w \in Ho(\mathcal{A}) \quad \text{iff} \quad \Delta w = 0.$$

Proof

By Thm.2.2.3. of HMT[71] we have $HSP\ Cr_\alpha \subseteq Bo_\alpha$ and each member of $HSP\ Cr_\alpha$ is normal. By 2.2.3 of HMT[71] we also have $(\forall i < \alpha)[HSP\ Cr_\alpha \models c_i x \geq x \text{ and } HSP\ Cr_\alpha \models c_i - c_i x = -c_i x]$. Thus $(\forall i < \alpha)[HSP\ Cr_\alpha \models (c_i x = x \leftrightarrow c_i - x = -x)]$. Assume $\Delta w = 0$. Then we have $(\forall i < \alpha) c_i w = w$ and hence $(\forall i < \alpha) c_i - w = -w$. Then w and $\mathcal{A}$ satisfy the conditions of Theorem 1 which then yields $rl_w \in Ho(\mathcal{A})$. Assume $i \in \Delta w$. Then $c_i w > w$. Since $(c_i - w = -w \rightarrow c_i w = w)$ holds we have $c_i - w \neq -w$. Then by $c_i x \geq x$ we conclude $c_i - w > -w$. Then by Theorem 1, $rl_w \notin Ho(\mathcal{A})$.

QED of Corollary 4

Now we turn to applications of Theorem 1 in Dynamic Algebras.

Notations (HMT[71])

Sb U = $\{x : x \subseteq U\}$. The function $\tilde{U} : Sb\ U \rightarrow Sb\ U$ is defined as $\tilde{U} = \langle U \sim x : x \in Sb\ U \rangle$, where $U \sim x = \{u \in U : u \notin x\}$.

Let $R, S \subseteq {}^2U$ then

$R|S = \{\langle x,y \rangle : (\exists z \in U)(\langle x,z \rangle \in R \text{ and } \langle z,y \rangle \in S)\}$, and for any

$x \subseteq U$ we let $R^*x = \{y : (\exists z \in x)\langle z,y\rangle \in R\}$, and

$R^{-1} = \{\langle y,x\rangle : \langle x,y\rangle \in R\}$.

Ds stands for Dynamic set-algebra, see Definition 3 below.

DEFINITION 3 (Andréka-Németi[79], Pratt[79], Kozen[79], Andréka-Németi-Sain[80])

By a Ds with base U we understand a two-sorted algebra $D = \langle A,B,\Diamond\rangle$ such that $A \subseteq Sb\ ^2U$, $B \subseteq Sb\ U$ and

$A = \langle A,\cap,\cup,^{\oplus}\rangle$ where $(\forall a \in A)\ a^{\oplus} = \cap\{b \in Sb\ ^2U : a \subseteq b$ and b is reflexive and transitive$\}$,

$B = \langle B,\cup,\tilde{\cup}\rangle$ and

$(\forall a \in A)(\forall x \in B)\ \Diamond(a,x) = (a^{-1})^*x$.

The variety Da of Dynamic Algebras was defined in Pratt [79], by finitely many equations. We shall use here the following properties (*) - (***) of Da only

(*) Ds $\subseteq$ Da

(**) Da $\models \{\Diamond(z,x) + \Diamond(z,y) = \Diamond(z,x+y),\ \Diamond(z,0) = 0\}$

(***) If $D \in$ Da then $D = \langle A,B,\Diamond\rangle$ where $B \in$ BA and $\Diamond : A \times B \to B$.

END of Definition 3

Notations: Let $D = \langle A,B,\Diamond\rangle$ be a Da. Let $a \in A$. Then we define the function $\Diamond(a,-) : B \to B$ as follows:

$(\forall x \in B)\ \Diamond(a,-)(x) = \Diamond(a,x)$ i.e. $\Diamond(a,-) = \langle \Diamond(a,x) : x \in B\rangle$.

For any function $f \in {}^AB$ and $a \in A$ we write $f_a = f(a)$.

Let $B \in$ BA, $c \in {}^{\alpha}({}^BB)$ and $d \in {}^{\alpha\times\alpha}B$. Then

$\langle B,c_i,d_{ij}\rangle_{i,j\in\alpha} = \langle B,+,-,c_i,d_{ij}\rangle_{i,j\in\alpha}$ where $B = \langle B,+,-\rangle$.

Note that $\langle B,c_i,d_{ij}\rangle_{i,j\in\alpha}$ is an algebra of type t_α.

PROPOSITION 5

Let $D = \langle A, B, \Diamond \rangle$ be any Da and let $d \in {}^{A\times A}B$. Then $\langle B, \Diamond(a,-), d_{ab} \rangle_{a,b\in A}$ is a normal Bo_A. In addition let $n \in {}^{\alpha}A$ and $b \in {}^{\alpha\times\alpha}B$ be arbitrary. Then $\langle B, \Diamond(n_i,-), b_{ij} \rangle_{i,j\in\alpha}$ is normal Bo_α.

Proof

By Properties (**) and (***) of Da in Definition 3.

QED of Proposition 5

The theory of many-sorted algebras can be found e.g. in the monograph Lugowski[76] Chap.II.2. (pp.137-170). See also Birkhoff-Lipson[70], Matthiessen[76], ADJ[77].

Recall that a homomorphism between two-sorted algebras is not a function but a pair of functions, see Lugowski[76] p.147 Def. II.2.(1) in Sec.II.2.2. E.g. if $D_1 = \langle A_1, B_1, \Diamond^1 \rangle$ and $D_2 = \langle A_2, B_2, \Diamond^2 \rangle$ are two Da-s then $h \in \mathrm{Hom}(D_1, D_2)$ iff $[h = \langle h_A, h_B \rangle$ and $h_A \in \mathrm{Hom}(A_1, A_2)$, $h_B \in \mathrm{Hom}(B_1, B_2)$ and $(\forall a \in A_1)(\forall x \in B_1)\ h_B(\Diamond^1(a,x)) = \Diamond^2(h_A(a), h_B(x))]$.

COROLLARY 6

Let $D = \langle A, B, \Diamond \rangle$ be any Dynamic Algebra (Da), let $w \in B$ and $h \in \mathrm{Ho}(A)$. Then statements (i) and (ii) below are equivalent.

(i) $\langle h, rl_w^B \rangle \in \mathrm{Ho}(D)$

(ii) $(\forall a \in A)\ \Diamond(a,-w) \leq -w$ and

$(\forall \langle a,b \rangle \in \ker h)(\forall x \in B)\ [x \leq w \to w \cdot \Diamond(a,x) = w \cdot \Diamond(b,x)]$

Proof

Recall from Lugowski[76] p.154 Sec.II.2.3 that a congruence of a two-sorted algebra is not a single relation but is a

pair of relations.

Notation: $\langle R_A, R_B\rangle \in Co(\langle A, B, \Diamond\rangle)$ if

$[R_A \in Co(A),\ R_B \in Co(B),$ and $(\forall \langle a_0, a_1\rangle \in R_A)(\forall \langle x_o, x_1\rangle \in R_B)$

$\langle \Diamond(a_0, x_0),\ \Diamond(a_1, x_1)\rangle \in R_B$].

Clearly, $\langle f_0, f_1\rangle \in Ho(D)$ iff $\langle \ker f_0 ,\ \ker f_1\rangle \in Co(D)$.

Cf. HMT[71] 0.2.21 (ii) and Lugowski[76] p. 157 Sec.II.2.4. item (1).

We let $\equiv_A = \ker h$ and $\equiv_B = \ker rl_w^B$.

Then clearly $\equiv_A \in Co(A)$ by $h \in Ho(A)$ and $\equiv_B \in Co(B)$ since $rl_w \in Ho(B)$ by Theorem 1 since $B \in BA = Bo_0$.

Now, $\langle h, rl_w\rangle \in Ho(D)$ iff $\langle \equiv_A, \equiv_B\rangle \in Co(D)$ iff $(\forall a_0, a_1, x_0, x_1)[(a_0 \equiv_A a_1$ and $x_0 \equiv_B x_1) \to \Diamond(a_0, x_0) \equiv_B \Diamond(a_1, x_1)]$ iff both (*)and (**) below hold.

(*) $(\forall a \in A)(\forall x_0, x_1)[x_0 \equiv_B x_1 \to \Diamond(a, x_0) \equiv_B \Diamond(a, x_1)]$ and

(**) $(\forall x \in B)(\forall a_0, a_1)[a_0 \equiv_A a_1 \to \Diamond(a_0, x) \equiv_B \Diamond(a_1, x)]$.

Now by Theorem 1 and Proposition 5 (*) holds iff $rl_w \in Ho(\langle B, \Diamond(a,-)\rangle_{a\in A})$ iff $(\forall a \in A)\Diamond(a,-w) \leq -w$.

Suppose (*) holds.

By $rl_w\ rl_w x = rl_w x$ we have that $rl_w x \equiv_B x$ for all $x \in B$. Then by (*) we have that $(\forall a \in A)\Diamond(a,x) \equiv_B \Diamond(a, w\cdot x)$. Now by the aboves (**) holds iff

$(\forall x \in B)(\forall a_0, a_1)[a_0 \equiv_A a_1 \to \Diamond(a_0, w\cdot x) \equiv_B \Diamond(a_1, w\cdot x)]$ iff

$(\forall \langle a_0, a_1\rangle \in \ker h)(\forall x \in B)[x \leq w \to w\cdot\Diamond(a,x) = w\cdot\Diamond(b,x)]$.

Now, the following tautology $((p_1 \leftrightarrow q_1) \wedge (p_1 \to (p_2 \leftrightarrow q_2))) \to$ $\to((p_1 \wedge p_2) \leftrightarrow (q_1 \wedge q_2))$ of propositional logic completes the

proof (by substituting (*) for p_1, (**) for p_2 and (ii) for $(q_1 \wedge q_2)$ in an appropriate way).

QED of Corollary 6

For a deeper application of Theorem 1 in the theory of Dynamic Algebras see Andréka-Németi-Sain[80].

REMARK:

Let $A \subseteq B \in Bo_\alpha$. Let $w \in B$. Consider $A \uparrow rl_w^B$ which is either in $Ho(A)$ or not. Let $rl_w^{AB} = A \uparrow rl_w^B$. Then the condition $c_i w \leq w$ is no more necessary for rl_w^{AB} to be a homomorphism. Namely, there are $A \subseteq B \in Bo_\alpha$, $w \in B$ such that $(\forall i < \alpha) c_i w \not\leq w$ and $rl_w^{AB} \in Ho(A)$. Moreover, we can choose $B \in CA_\alpha$ if $A \in CA_\alpha$ and we can choose $B = \langle C, \Diamond(n_i,-), 0_i \rangle_{i \in \alpha}$ for some Dynamic Algebra $\langle N, C, \Diamond \rangle$ and mapping $n : \alpha \rightarrowtail N$ if A is a normal Bo_α. See Propositions 7 and 10 in this paper.

Both in the theory of Cylindric Algebras and in the theory of Dynamic Algebras some of the most useful relativization homomorphisms are of this rl_w^{AB} kind, i.e. they are such that $c_i w \not\leq w$, but $rl_w^{AB} \in Ho(A)$. Proposition 7 below says that every homomorphism is of the form rl_w^{AB} for some B. (Of course, this is not true for rl_w^A, there are $A \in BA$ and $R \in Co(A)$ such that $(\forall w \in A) R \neq \ker(rl_w^A)$.)

PROPOSITION 7

Let A be a Bo_α and let $R \in Co(A)$.
Then there are a Bo_α $B \supseteq A$ and $w \in B$ such that $R = \ker(rl_w^{AB})$. Further, if $A \in CA_\alpha$ then $B \in CA_\alpha$.

Proof

By 2.7.5., 0.2.15, and 2.7.6. of HMT[71] there is a complete $B \in Bo_\alpha$ such that $A \subseteq B$ and $(\forall X \subseteq A)[\sup^B X = 1 \rightarrow (\exists X_0 \subseteq X)(|X_0| < \omega$ and $\sup X_0 = 1)]$. (This is the so called Stone-representation of A.) By 2.7.15. of HMT[71], if $A \in CA_\alpha$ then $B \in CA_\alpha$ too.

Let $R \in Co(A)$ be arbitrary and define $I = 0^A/R$. Let $w = \sup^B I$. w exists since B is complete. Consider the function rl^B_w. Clearly $rl^B_w \in Ho(\langle B,+,-\rangle)$. Let $K = \ker rl^B_w$ and $J = 0^B/K$. Then $K \in Co(\langle B,+,-\rangle)$ and K is determined by the ideal J of $\langle B,+,-\rangle$.

Claim: $J \cap A = I$.

Proof

Suppose $y \in J \cap A$. Then $y \leq w$. Therefore $b = \sup^B(I \cup \{-y\}) = 1$ since $b \geq w \geq y$ and $b \geq -y$. Then $\sup(I_0 \cup \{-y\}) = 1$ for some finite $I_0 \subseteq I$, by the properties of B. I.e. $y \leq \sup I_0$ which implies $y \in I$ since I is an ideal. We have seen $J \cap A \subseteq I$. $I \subseteq J \cap A$ is clear.

QED of Claim

Since $\langle A,+,-\rangle \subseteq \langle B,+,-\rangle$ are BA-s and $R \in Co(\langle A,+,-\rangle)$, $K \in Co(\langle B,+,-\rangle)$, $I = 0/R$, $J = 0/K$, $I = J \cap A$, and since the ideals of BA-s determine their congruences (e.g. by 0.2.26 (i) of HMT [71]) we can conclude that $R = {}^2A \cap K$. Then $R = \ker rl^{AB}_w$ since $K = \ker rl^B_w$.

QED of Proposition 7

Notation (HMT [71] 0.3.1. (ii), 0.3.24)

B is a direct factor of A, in symbols $B \mid A$, iff $A \cong B \times C$

for some C.

pj_i is the i-th projection function of any direct product, specially let $i < 2$ then $pj_i : A_0 \times A_1 \to A_i$ is such that

$(\forall <a_0,a_1> \in A_0 \times A_1)[pj_1(<a_0,a_1>) = a_1$ and

$$pj_o(<a_o,a_1>) = a_0] \quad .$$

Hence $pj_i \in Ho(A_0 \times A_1, A_i)$ for all $i < 2$.

Is (A,B) is the set of all isomorphisms of A onto B, i.e. Is $(A,B) = \{h \in Ho(A,B) : h$ is one-to-one (i.e. injection)$\}$. Let $R \in Co(A)$. Then R is a direct factor congruence on A iff there are B_0, B_1 and $h \in Is(A, B_0 \times B_1)$ such that $R = \ker(pj_1 \circ h)$.

THEOREM 2

I. Let A and B be two normal Bo_α-s.
Then $B \mid A$ iff $B = Rl_w A$ for some $w \in A$ such that
$(\forall i < \alpha)[c_i w \leq w$ and $c_i(-w) \leq (-w)]$.

II. Let A be a normal Bo_α and let $R \in Co(A)$.
Then statements (i) and (ii) below are equivalent.
(i) R is a direct factor congruence of A.

(ii) $R = \ker(rl_w)$ for some $w \in A$ such that
$(\forall i < \alpha)[c_i w \leq w$ and $c_i(-w) \leq (-w)]$.

III. Let $\alpha > 0$.
Then there are $A \in Bo_\alpha$ and $w \in A$ such that $\ker(rl_w)$ is a direct factor congruence on A and $c_0 w > w$ and $c_0 -w > -w$.

Proof

Proof of III:

Let $\alpha > 0$ be arbitrary. Let $B \in BA$ and $d \in {}^{\alpha\times\alpha}B$ be arbitrary but assume $|B| > 2$. Let $(\forall i < \alpha)c_i = B \times \{1^B\}$. Then $A = \langle B, c_i, d_{ij}\rangle_{i,j\in\alpha}$ is a Bo_α. Let $w \in B \sim \{0^B, 1^B\}$. Exists since $|B| > 2$. Then $-w \neq 1^B$. Thus $(\forall i < \alpha)[w \leq c_i w = 1 = c_i{-w} \geq -w]$. Clearly, $rl_w \in Ho(A)$, $rl_{-w} \in Ho(A)$ and

$(\exists h \in Is(A, R\mathcal{l}_w A \times R\mathcal{l}_{-w}A))[pj_0 \circ h = rl_w$ and $pj_1 \circ h = rl_{-w}]$.

This easily follows from the fact that c_i is a constant function in A and that the equation $\forall x(c_i x = 1)$ is preserved under both products and homomorphisms.

Statement I is an immediate corollary of Statement II and therefore it is enough to prove II.

Proof of II:

Let A be a normal Bo_α and let $R \in Co(A)$.

Proof of (i) → (ii):

Suppose $A = B \times C$. Let $w = \langle 0^B, 1^C\rangle$. Since A is normal $c_i w = c_i\langle 0,1\rangle = \langle 0^B, c_i^C 1^C\rangle \leq w$ and $c_i{-w} = c_i\langle 1^B, 0\rangle = \langle c_i^B 1^B, 0^C\rangle \leq \langle 1^B, 0^C\rangle = -w$ for every $i < \alpha$. By Theorem 1, $rl_w^A \in Ho(A)$. $\{x \in A : rl_w x = 0\} = \{x : w \cdot x = 0\} = \{\langle y,z\rangle : \langle y,z\rangle \cdot \langle 0,1\rangle = 0\} = \{\langle y,0\rangle : y \in B\} = B \times \{0^C\}$.

Clearly, $\{x \in A : pj_1(x) = 0\} = \{\langle y,z\rangle : z = 0^C\} = B \times \{0^C\}$. This proves that $(\forall x \in A)[rl_w x = rl_w 0^A$ iff $pj_1(x) = pj_1(0^A)]$ which by observing that A is an expansion of a Boolean algebra $\langle A,+,-\rangle$ implies that $\ker rl_w = \ker pj_1$.

Proof of (ii) → (i):

Let $w \in A$ be such that $(\forall i < \alpha)[c_i -w \leq -w$ and $c_i w \leq w]$. Then by Theorem 1 we have $rl_w^A \in Ho(A)$ and $rl_{-w} \in Ho(A)$. Let $R = \ker(rl_w^A)$ and $S = \ker(rl_{-w}^A)$.

It is known from Boolean Algebra Theory that our R and S are a direct decomposition of the Boolean reduct $\langle A,+,-\rangle$ of A.

Fact 2 (Universal Algebra)

Let B be any algebra and let $K \in {}^{I}Co(B)$. Then K is a direct decomposition of B iff K is a direct decomposition of some reduct of B. See 0.3.22. of HMT[71].

Hence by Fact 2, R is a direct factor congruence of A.

QED of Theorem 2

Corollary 2.4.8. of HMT[71] saying that the direct factors of a $CA_\alpha A$ are exactly $\{Rl_w A : w \in A, \Delta w = 0\}$, is a corollary of Theorem 2 above. (Moreover, Theorem 2 shows that 2.4.8. of HMT [71] is true for HSP Cr_α in general.)

In connection with Corollaries 8 and 9 below we note the following: Let $D = \langle A,B,\Diamond\rangle$ be a Ds with base U. In Andréka-Németi[79], a $w \in B$ is said to be a subbase provided that $(\forall a \in A)a \subseteq {}^2w \cup {}^2(U \sim w)$, which means $(\forall a \in A)[\Diamond(a,w) \leq w$ and $\Diamond(a,-w) \leq -w]$. Hence Claims 2 and 3 in Andréka-Németi[79] are special cases of Corollaries 8 and 9 below. Again, a deeper application of Theorem 2 to Da-s can be found in Andréka-Németi--Sain[80].

COROLLARY 8

Let $D = (A,B,\Diamond)$ be a Da and let $\langle R_A, R_B\rangle \in Co(D)$ be a direct factor congruence on D.

Then there is $w \in B$ such that $R_B = \ker(rl^B_w)$ and $(\forall a \in A)[\Diamond(a,w) \leq w$ and $\Diamond(a,-w) \leq -w]$.

Proof

Let $D^+ = \langle B, \Diamond(a,-), 0\rangle_{a\in A}$ and let $D^e = \langle\langle A,a\rangle_{a\in A}, D^+, \Diamond\rangle$. Then D is a reduct of D^e and $Co(D) = Co(D^e)$. Hence by Fact 2 we have that $\langle R_A, R_B\rangle$ is a direct factor congruence of D^e. Then R_B is a direct factor congruence of D^+, by basic definitions in the theory of two-sorted algebras, see Lugowski[76] p.150. item II.2.2.(10). D^+ is a Bo_A, by Proposition 5. Now, applying Theorem 2 to D^+ and R_B completes the proof.

QED of Corollary 8

COROLLARY 9

Let $A \in Bo_\alpha$ and let $R \in {}^I Co(A)$ (i.e. $\langle R_i : i \in I\rangle$ is a system of congruences of A). Then statements (i) and (ii) below are equivalent.

(i) R is a direct decomposition of A (i.e. there is $A \cong P_{i\in I} B_i$ with projections $\langle pj_i : i \in I\rangle$ such that $R_i = \ker pj_i$ for all $i \in I$).

(ii) There is a system $w \in {}^I A$ of elements of A such that $(\forall i \in I)(\forall j < \alpha)c_j - w_i \leq -w_i$ and $(\forall i,j\in I)(i\neq j \rightarrow w_i \cdot w_j = 0)$, $\sup\{w_i : i\in I\} = 1^A$, $(\forall y \in {}^I A)\sup\{y_i \cdot w_i : i \in I\}$ exists and $(\forall i \in I)R_i = \ker rl^A_{w_i}$.

Proof

(i) → (ii) follows from observing that if R is a direct decomposition then R_i is a direct factor congruence for all $i \in I$ and then applying Theorem 2.

(ii) → (i):

Let $R_i = \ker(rl_{w_i})$ for all $i \in I$. Then for every $i \in I$ $R_i \in Co(A)$ by Theorem 2, since w_i satisfies the conditions of Theorem 2. Then we have a system $R \in {}^{I}Co(A)$ of congruences of A. From BA-theory it follows that R is a direct decomposition of the BA $\langle A,+,-\rangle$ because Fact 3 below is a theorem of BA-theory.

Fact 3

Let $B \in BA$ and $b \in {}^{I}B$ be such that $(\forall i,j \in I)(i \neq j \rightarrow b_i \cdot b_j = 0)$, $\sup b^*I = 1^B$ and $(\forall y \in {}^{I}B)$ $\sup\{y_i \cdot b_i : i \in I\}$ exists in B. Then $\langle rl^B_{b_i} : i \in I\rangle$ is a direct decomposition of B i.e. $(\exists h \in Is(B, P_{i\in I} R\!l_{b_i} B))(\forall i \in I)\, pj_i \circ h = rl^B_{b_i}$.

Now by Facts 2 and 3 we have that R is a direct decomposition of A.

QED of Corollary 9.

DEFINITION 4 (HMT[71] Def.2.7.33. p. 453)

Let N be a relational structure $N = \langle N, R_i, E_{ij}\rangle_{i,j\in\alpha}$ such that $(\forall i < \alpha)R_i \subseteq {}^{2}N$ and $(\forall i,j \in \alpha)E_{ij} \subseteq N$.

Then the complex algebra $Cm\ N$ of N is defined as $Cm\ N = \langle Sb\ N, \cup, \tilde{N}, R_i^*, E_{ij}\rangle_{i,j\in\alpha}$. Clearly, $Cm\ N$ is of type t_α.

END of Definition 4

Part (i) - (iv) of Proposition 10 below is taken from the representation theory of Cylindric Algebras, see HMT[71], chapter 2.7. We shall show that it can be used in the representation theory of Dynamic Algebras, too.

PROPOSITION 10

Let $\mathcal{C}$ be an algebra of type t_α.
Then statements (i) - (vii) below are equivalent:

(i) $\mathcal{C}$ is a normal Bo_α

(ii) $\mathcal{C} \cong \mathfrak{Nr}_\alpha \mathfrak{Cm}\, \mathcal{N}$ for some relational structure $\mathcal{N}$.

(iii) $\mathcal{C} \cong \mathfrak{Nr}_\alpha \mathfrak{Cm}\, \mathcal{N}$ for some relational structure $\mathcal{N} = \langle N, R_i, E_{ij} \rangle_{i,j\in\alpha}$ such that the same identities hold in $\langle C, +, \cdot, 0, 1, c_i, d_{ij} \rangle_{i,j\in\alpha}$ and in $\langle \mathrm{Sb}\, N, \cup, \cap, 0, N, R_i^*, E_{ij} \rangle_{i,j\in\alpha}$.

(iv) $\mathcal{C} \cong \langle B, \cup, \sim_{\tilde{N}}, R_i^*, E_{ij} \rangle_{i,j\in\alpha}$ for some relational structure $\mathcal{N} = \langle N, R_i, E_{ij} \rangle_{i,j\in\alpha}$ and for some $B \subseteq \mathrm{Sb}\, N$.

(v) $\mathcal{C} \cong \mathfrak{Nr}_\alpha \langle B, \Diamond(n_i, -), e_{ij} \rangle_{i,j\in\alpha}$ for some Ds $\mathcal{D} = \langle A, B, \Diamond \rangle$ and mappings $n : \alpha \to A$, $e : \alpha \times \alpha \to B$.

(vi) $\mathcal{C} \cong \mathfrak{Nr}_\alpha \langle B, \Diamond(n_i, -), e_{ij} \rangle_{i,j\in\alpha}$ for some full Ds $\mathcal{D} = \langle A, B, \Diamond \rangle$ and mapping $n : \alpha \to A$, $e : \alpha \times \alpha \to B$ such that the same identities hold in $\langle C, +, \cdot, 0, 1, c_i, d_{ij} \rangle_{i,j\in\alpha}$ and in $\langle B, \cup, \cap, 0, 1^B, \Diamond(n_i, -), e_{ij} \rangle_{i,j\in\alpha}$.

(vii) $\mathcal{C} \cong \mathfrak{Nr}_\alpha \langle B, \Diamond(n_i, -), e_{ij} \rangle_{i,j\in\alpha}$ for some $\bullet$-free Ds $\mathcal{D} = \langle A, B, \Diamond \rangle$ and mappings $n : \alpha \to A$, $e : \alpha \times \alpha \to B$.

Proof

The equivalence of (i) - (iv) follows from HMT[71] Def. 2.7.4., Thms 2.7.5.(i), 2.7.34, 2.7.35, and Def. 2.7.13. Each of (v) - (vii) implies (i) by Proposition 5.

Proof of (iii) → (vi):

Let $\mathcal{C} \cong \mathsf{I} \subseteq Cm\ N$ where $N = \langle N, R_i, E_{ij} \rangle_{i,j\in\alpha}$ is a relational structure, and the same identities hold in $\langle C,+,\cdot,...\rangle$ and in $\langle Sb\ N,\cup,\cap,...\rangle$. Let $D = \langle\langle Sb\ {}^2N, \mathsf{I}, \cup, {}^{\oplus}\rangle , \langle Sb\ N,\cup,\tilde{N}\rangle,\Diamond\rangle$ be the full Dynamic set algebra with base N (see Andréka-Németi [79] Def.2).

Let $n : \alpha \to Sb\ {}^2N$ and $e : \alpha\times\alpha \to Sb\ N$ be defined as $n_i = R_i^{-1}$ and $e_{ij} = E_{ij}$ for all $i,j \in \alpha$. Then $\Diamond(n_i,-) = \Diamond(R_i^{-1},-) = R_i^*$ for all $i < \alpha$. I.e. $Cm\ N = \langle Sb\ N,\cup,\tilde{N},R_i^*,E_{ij}\rangle_{i,j\in\alpha} = \langle Sb\ N,\cup,\tilde{N},\Diamond(n_i,-),e_{ij}\rangle_{i,j\in\alpha}$. The additional requirement about identities clearly holds.

(vi) implies (v).

Proof of (v) → (vii):

Let $\mathcal{C} \cong \mathsf{I} \subseteq \langle B,\Diamond(n_i,-),e_{ij}\rangle_{j\in\alpha}$ for some Ds $D = \langle A,B,\Diamond\rangle$ and mappings $n : \alpha \to A$, $e : \alpha\times\alpha \to B$. Then $\mathcal{C} \cong \langle B',\Diamond(n_i,-),e_{ij}\rangle_{i,j\in\alpha}$ where $B' \subseteq B$ and $(\forall i,j \in \alpha)$ $[\Diamond(n_i,-): B' \to B' \text{ and } e_{ij} \in B']$.

Let $A' = \{a \in A : (\forall x \in B')\Diamond(a,x) \in B'\}$. Clearly, $\{n_i : i \in \alpha\} \subseteq A'$. We show that A' is a closed subset of $\langle A,\mathsf{I},\cup\rangle$:

Let $a,b \in A'$ and let $x \in B'$. $(a\mathsf{I}b)^{-1*}x = (b^{-1}\mathsf{I}a^{-1})^*x = b^{-1*}(a^{-1*}x) \in B'$ since $a^{-1*}x \in B'$ by $a \in A'$ and then $b^{-1*}(a^{-1*}x) \in B'$ by $b \in A'$.

$(a \cup b)^{-1*}x = a^{-1*}x \cup b^{-1*}x \in B'$ since both $a^{-1*}x$ and $b^{-1*}x$ are in B' by $a,b \in A'$, and B' is closed under $\cup$.

Let $A' = \langle A', |, \cup \rangle$. Now clearly $D' = \langle A', B', \Diamond \rangle$ is a $\oplus$-free Ds, and $C \cong \langle B', \Diamond(n_i, -), e_{ij} \rangle_{i,j \in \alpha}$ where $n : \alpha \to A'$ and $e : \alpha \times \alpha \to B'$.

QED of Proposition 10

DEFINITION 5 (Cf.Pratt[79])

By a $\oplus$-free Da we understand a two-sorted algebra $D = \langle\langle A, ;, \vee\rangle, B, \Diamond\rangle$ such that $B \in BA$, and

$D \models \{\Diamond(a, x+y) = \Diamond(a,x) + \Diamond(a,y)$

(1) $\Diamond(a;b,x) = \Diamond(a, \Diamond(b,x))$

(2) $\Diamond(a \vee b, x) = \Diamond(a,x) + \Diamond(b,x)$

$\Diamond(a,0) = 0 \ \}$.

A Da $D = \langle A, B, \Diamond \rangle$ is separable iff $(\forall a,b \in A)(a \neq b \to \Diamond(a,-) \neq \Diamond(b,-))$.

END of Definition 5

The following result is due to Kozen, we show that it can be derived from the representation theory of CA-s, too.

COROLLARY 11 (Kozen)

Any separable $\oplus$-free Da is isomorphic to a $\oplus$-free Ds.

Proof

Let $D = \langle A, B, \Diamond \rangle$ be a separable $\oplus$-free Da. Let $C = \langle B, \Diamond(a,-), 0 \rangle_{a \in A}$. Then C is a normal Bo_A by Proposition 5. By Proposition 10 (i) $\to$ (vi) there are a full Ds

$D' = \langle A', B', \Diamond' \rangle$, a mapping $n : A \to A'$ and a one-to-one homomorphism $h \in \mathrm{Hom}(C, \langle B', \Diamond'(n_a, -), 0 \rangle_{a \in A})$, such that the identities (1) and (2) in Definition 5 hold in D'.

First we show that $n \in \mathrm{Hom}(A, \langle A', \mid, U \rangle)$. This will follow from the facts that D' is full and D' satisfies (1) and (2), as follows:

Let $a,b \in A$. Let the base of D' be N and let $k,m \in N$. Then $\{m\} \in B'$ since D' is full. Now by the definition of $\Diamond'$ and by identity (1) we have

$(k,m) \in n_{(a;b)}$ iff $k \in \Diamond'(n_{(a;b)}, \{m\})$ iff

$k \in \Diamond'(n_a, \Diamond'(n_b, \{m\}))$ iff $(k,m) \in n_a \mid n_b$.

We have seen $n(a;b) = n(a) \mid n(b)$. The proof of $n(a \vee b) = n(a) \cup n(b)$ is entirely analogous, the only difference is that we use (2) instead of (1):

$(k,m) \in n_{(a \vee b)}$ iff $k \in \Diamond'(n_{(a \vee b)}, \{m\})$ iff

$k \in (\Diamond'(n_a, \{m\}) \cup \Diamond'(n_b, \{m\}))$ iff $(k,m) \in n_a \cup n_b$.

By these we have seen that $n \in \mathrm{Hom}(A, \langle A', \mid, U \rangle)$.

Then $\langle n,h \rangle$ is a homomorphism from D into the $\oplus$-free reduct $\langle\langle A', \mid, U \rangle, B', \Diamond' \rangle$ of D' since $h \in \mathrm{Hom}(B, B')$ and

$(\forall a \in A)(\forall x \in B)\Diamond'(n_a, hx) = h\Diamond(a,x)$ hold by

$h \in \mathrm{Hom}(\langle B, \Diamond(a,-), 0 \rangle_{a \in A}$, $\langle B', \Diamond'(n_a, -), 0 \rangle_{a \in A})$.

By separability of D we have that n is one-to-one, since

$a \neq b \to \Diamond(a,-) \neq \Diamond(b,-) \to \Diamond'(n_a,-) \neq \Diamond'(n_b,-) \to n_a \neq n_b$.

Since both n and h are one-to-one, we have that $\langle n,h \rangle$ is an isomorphism of D into a $\oplus$-free Ds.

The fact that $\oplus$-free Ds-s are closed under subalgebras completes the proof.

QED Corollary 11

Andréka-Németi[79] exhibits a different parallelism between the representation theories of Da-s and CA_α-s. About a representation theory of CA_α-s and related algebras see Henkin--Monk-Tarski[79], Andréka-Németi[79a], Németi[78] Chapter 7, Andréka-Németi[75]. The connection between Logic, Model Theory and CA_α-s is made explicit and is elaborated in Chapter 7 of Németi[78] to a certain extent.

REFERENCES

ADJ[77]: Goguen,J.A.,Thatcher,J.W.,Wagner,E.G.,Wright,J.B., Initial Algebra Semantics and Continuous Algebras, JACM Vol. 24, No 1, 1977, pp. 68-95.

Andréka,H.,Gergely,T.,Németi,I.[77]: On Universal Algebraic Construction of Logics, Studia Logica 36 (1977), pp. 9-47.

Andréka,H.,Németi,I.[75]: A Simple, Purely Algebraic Proof of the Completeness of some First Order Logics, Algebra Universalis 5 (1975), pp. 8-15.

Andréka,H.,Németi,I.[78]: On Systems of Varieties Definable by Schemes of Equations, Math.Inst.Hung.Acad.Sci Preprint 1978. To appear in Algebra Universalis.

Andréka,H.,Németi,I.[79]: Every Free Algebra in the Variety Generated by the Representable Dynamic Algebras is Separable and Representable,Math.Inst.Hung.Acad.Sci. Preprint 1979.
To appear in Theoretical Computer Science.

Andréka,H.,Németi,I.[79a]: On Regular Cylindric Set Algebras, Math.Inst.Hung.Acad.Sci. Preprint 1979.

Andréka,H.,Németi,I.,Sain,I[80]: Representable Dynamic Algebras are not First Order Axiomatizable and other Considerations on the Representation Theory of Dynamic Algebras, Math.Inst.Hung.Acad.Sci. Preprint 1980.

Birkhoff,G.,Lipson,J.D.[70]: Heterogeneous Algebras, J.Combinat. Theory 8 (1970), pp. 115-133.

HMT[71]: Henkin,L.,Monk,J.D.,Tarski,A.[71]: Cylindric Algebras Part I, North-Holland 1971.

Henkin,L.,Monk,J.D.,Tarski,A.[79]: Cylindric Set Algebras and Related Structures I, Univ.of Colorado at Boulder, Preprint 1979.

Kozen,D.[79]: A Representation Theorem for Models of *-free PDL, RC 7864, IBM Research, Yorktown Heights, New York, Sept.1979.

Lugowski,H.[76]: Grundzüge der Universellen Algebra, Teubner Texte zur Math., Lepzig 1976.

Matthiessen,G.[79]: Theorie der heterogenen Algebren, Dissertation, Univ.Bremen, 1976. Mathematik-Arbeitspapiere Nr.3.

Németi,I.[78]: Connections between Algebraic Logic and Initial Algebraic Semantics of CF Languages, Proc.Coll.Logic in Programming, Salgotarján 1978, Colloq.Math.Soc.J. Bolyai, North Holland, to appear.

Pratt,V.R.[79]: Dynamic Algebras: Examples, Constructions, Applications, Report MIT/LCS/TM-138, July 1979.

Pratt,V.R.[79a]: Dynamic Logic, 6th International Congress of Logic, Methodology and Philosophy of Science, Hannover 1979. abstract .

Pratt,V.R.[79b]: Models of Program Logics, Proc. 20th IEEE Conf. on Foundations of Computer Science, San Juan, PR, Oct., 1979.

Computational Linguistics and Computer Languages Vol. XIV. 1980.

SOME PROBLEMS OF THE SEMANTIC TREATMENT OF THAT-CLAUSES IN MONTAGUE GRAMMAR

Miklós SÁNTHA
Institut of Linguistics, Hungarian Academy of Sciences
Budapest, Hungary

1. That-clauses are known to play several different roles in complex sentences, for instance, as illustrated by the following examples:

(1) The proposal that he would marry me caught me unprepared.
(2) That Peter is right is clear as noonday.
(3) He complained that he had to go out.

In sentence (1), the that-clause can be said to expound the content of the proposal that caught me unprepared. In (2) there is no word to sum up the content of the that-clause (as opposed to proposal in (1)) but it would be easy to find a suitable one. (3) on the one hand shows a structure parallel to (2), on the other, its that-clause can be understood as an indirect quotation.

In general, sentences with that-clauses can be approached in two different ways. On the first approach, that-clauses are sometimes directly subordinated to the matrix verb, like the other arguments, while sometimes they are subordinated to an abstract nominal. According to the second approach it is always the latter which is the case, only the abstract nominal may happen to be absent from the sentence. Here one regards this correspondence as the most characteristic property of that-clauses. Ilona Molnár [1980] also accepts this fundamental relation between the abstract nominal and the that-clause; nevertheless, she makes it explicit by means of a transformation into an attributive sentence having an abstract nominal in its

main clause; i.e. instead of (4) and (5), she relates (4) and (6) by transformation:

(4) It is well-known that Mary is pretty.

(5) Mary's prettiness is well-known.

(6) The fact that Mary is pretty is well-known.

Now it is clear that sentence (1) exemplifies the special type in which this relation is explicit. In this paper I will deal with sentences like this and demonstrate that despite its apparent simplicity, a satisfactory semantic account for the relation between the abstract nominal and the that-clause is not self-evident. (I will not treat the problem how and when sentences can be transformed into the desired form of (1).)

One more remark in advance: it is evident that an abstract nominal can be followed by infinitely many different subordinate clauses. It is already not so evident, however, that the that--clause does not have a unique 'summarizing' abstract nominal either. (Naturally, we can have only a finite number of abstract nouns.) For instance from (7) we can get either of (8 a, b, c):

(7) They published that John had escaped.

(8 a) They published the statement that John...

(8 b) They published the supposition that John...

(8 c) They published the news that John...

From now on I will call the abstract nominal related to the that-clause a 'sentential name' (cf. Molnár (1980)).

2. I will carry out my analysis in the framework of R. Montague, defined by means of intensional logic in Montague (1974). Let me begin by sketching what we can expect for the logical formulae to reflect. Consider sentence (9):

(9) The decision that John is leaving for Berlin surprises me.

We can rightly expect that the translation of (9) should express that

(i) something surprises me,

(ii) this is a decision, and, moreover,

(iii) the very decision that John is leaving for Berlin.

In the following part of this section I will review the proposal made by E.B. Delacruz (1976), where he endeavours to satisfy these conditions, although without stating them explicitly. His approach is really the most natural one and although I will try to criticize and modify his proposals it is worthwhile to have a look at them since in this inductive way we can get a better insight into the problems which render our task more difficult.

The first idea is that propositions (by this notion is understood the declarative sentence preceded by the connective that) can play a role in the building of the sentences at issue which is similar to that of entities in the building of the usual categories, the difference being that entities are represented by variables or constants of type <s,e> and propositions correspond to ones of type <s,t>. On the basis of this observation proposition-level basic categories, syntactic and translation rules can be given quite automatically. For the sake of parallelism I will use the same letters for naming entity-level and proposition-level categories except that the latter are underlined.

Categories and their basic expressions:

$B_{\underline{IV}}$: proposition-level intransitive verbs of category t/t.
E.g. happen, turn-out

$B_{\underline{CN}}$: proposition-level common nouns of categoriy t//t (in fact, the members of this set are sentential names).
E.g.: news, fact, assumption, dream

$B_{\underline{T}} = B_{t/\underline{IV}}$: proposition-level terms of category t/(t/t).
(No basic expressions)

$B_{\underline{IV}/T}$: transitive verbs taking propositions as subject, of category (t/t)/(t/(t/e)).
E.g.: surprise, trouble, amuse

$B_{IV/\underline{T}}$: transitive verbs taking propositions as object, of category (t/e)/(t/(t/t)).
E.g.: know, hear, suppose

$B_{\underline{IV}/\underline{T}}$: transitive verbs taking propositions both as subject and object, of category (t/t)/(t/(t/t)).
E.g.: be

New syntactic rules:

S_1' : If $\alpha \in P_t$, then $F_1'(\alpha) \in P_T$, where $F_1'(\alpha) =$ that α.

S_2' : If $\mu \in P_T$ and $\upsilon \in P_{CN}$, then $F_2'(\mu,\upsilon) \in P_T$, where

$$F_2'(\mu,\upsilon) = \begin{cases} \text{the } \upsilon\mu & \text{if } \mu \text{ is of the form that } \alpha \\ \mu & \text{otherwise} \end{cases}$$

The introduction of the following variables is necessary for the stating of the corresponding translation rules:

p,q,r,p',q',r'	$\in$	$Var_{<s,t>}$
P,R	$\in$	$Var_{<s,<<s,t>,t>>}$
Q,Q'	$\in$	$Var_{<s,<<s,<<s,t>,t>>,t>>}$
Z	$\in$	$Var_{<s,<<s,<<s,<<s,t>,t>>,t>>,t>>}$
C,D	$\in$	$Var_{<<s,t>,t>}$

Translation rules:

T_0' : the translation of a basic expression is a constant of the appropriate type, designated by the primed variant of the word. Exception: be translates as be' =

$$= \lambda Q.\lambda r.(^{\vee}Q)(^{\wedge}\lambda q'(r=q'))$$

T_1' : If $\alpha \in P_t$, then $F_1'(\alpha)$ translates as $\lambda P.^{\vee}P(^{\wedge}\alpha')$.

T_2' : If $\mu \in P_T$ and $\upsilon \in P_{CN}$, then

$$F_2'(\mu,\upsilon) \text{ translates as } \begin{cases} \lambda R.\exists q(\forall p((\upsilon'(p) \wedge be'(^{\wedge}\mu'))(p) <=> p=q) \wedge {}^{\vee}R(q)) \\ \mu' \end{cases}$$

Applying this latter rule we can already obtain the translations of the sentences examined. E.g., after the automatic application of PTQ-rules λ-conversion will yield the following formula corresponding to (9):

(9') $\exists q(\forall p((\underline{\text{decision}}'(p) \wedge p' = {}^{\wedge}\underline{\text{John is leaving...}}') <=>$
$<=> p=q) \wedge \text{surprise}'(^{\wedge}\underline{I}')(q))$

This form of the translation already shows that the three requirements are satisfied because

(ii) and (iii): it follows from the lefthand side of the conjunction within the scope of the existential quantifier that there is a unique q which is a decision, and from its righthand side that it is identical to that John is leaving for Berlin, and (i): this q surprises me.

3. This most natural looking construction is practically the same as what we find in Delacruz's paper (without the explication of the requirements). Nevertheless, the translation obtained in this way has two properties which cast serious doubts at its correctness. In the following parts I will try to remedy this situation by modifying the construction. Before turning to that, however, I will show that there exists a much simpler formula which is equivalent to (9') although we cannot get it by λ-conversion. (9') has the form $\exists q(F(q) \wedge G(q))$. Obviously, such a formula is true iff the sets determined by F and G have a common member i.e. iff

(10) $\{q : F(q)\} \cap \{q : G(q)\} \neq \emptyset$

Notice, however, that $\{q : F(q)\}$ may at most be a unit set (its only member being ^John is leaving...'), depending on whether decision' (^John is leaving...') is true. It is therefore evident that the two sets may have no other common member either, that is, (9') is true only if (11) is true:

(11) decision' (^John is...') $\wedge$ surprise'(^I')
(^John is...')

The problems to get rid of are already perspicuous in this formula. Let us first form the negation of (9):

(12) The decision that John is leaving for Berlin does not surprise me. The translation of (12), drawing from (11) is

(12') $\neg$decision' (^John is...') $\vee$ $\neg$surprise'(^I')
(^John is...')

This cannot be an acceptable translation for (12), however, since (12) implies that it is a decision that John is leaving for Berlin while (12') can be true even if the contrary is the

case. It is clear that this problem does not result from anything particular to (9) but it concerns all the sentences under investigation. The reason is that in all those it follows both from the sentence and its negation that the subordinate clause is conceived of as its sentential name. In other words, (9) and (12) presuppose (13), which is not reflected by the above translation:

(13) It is a decision that John is leaving...

This kind of presupposition is worth dwelling on a bit further since it can be said to be a generalization of factive presupposition. Kiparsky and Kiparsky (1970) assign a special status to predicates like regret, the that-complement of which is presupposed to be true and thus it is a defining property of so-called factive sentences that the that-complement of their predicate can always be prefixed with fact without changing their meaning. It is certainly true that the truth of the subordinate clause follows both from those sentences and their negation; if however we define the constant fact' so that the intension of a sentence is a member of it iff the sentence itself is true,

(14) fact' (α') is true iff α' is true

then this natural definition renders factive presuppositions as a special case of the above stated kind. Let us take a complex sentence with a factive predicate. In view of the factivity of the predicate the that-clause may be prefixed with fact. In view of the generally valid presupposition (cf. (13)) the intension of the that-clause must be a member of fact' and, in view of (14), it must be true. In other words, factive presuppositions constitute a special case insofar as the sentential name fact has a special semantics among sentential names in general. Consequently, the question which predicates are factive in the sense of which predicates may be related to the sentential name fact will also become a special case of the question which predicates may be related to any given sentential name.

The treatment of presuppositions requires an extension of Montague's intensional logic. Here I will only indicate the outlines, referring the reader to Ruzsa (1978). To the set {0,1} of truth values we add a new element 2 to designate the truth value gap, which comes into play if, for some reasons, we cannot or do not want to assign a truth value to the sentence (e.g. because one of its presuppositions fails). We define such a degenerate element (null-entity), not only for the set of truth values but for all types. The key idea of the definition is that given the null-entity of domain D_b, the null-entity of domain $D_b^{D_a}$ is that function which, for every element of D_a, takes the null-entity of D_b as its value.

In syntax we define the iota-operator. Let ME_a be the set of meaningful expressions of type a.

A.1. If $F \in ME_t$ and contains a free occurrence of the variable $x \in ME_a$, then $\iota x(F) \in ME_a$.

B.1. $V(\iota x(F)) = \begin{cases} \text{that element of } D_a \text{ of which F is true} \\ \text{if } D_a \text{ has exactly one such element} \\ \\ \text{the null-entity of } D_a \text{ otherwise} \end{cases}$

The new proposal for the translation of (9) makes use of the iota-operator:

(9'') $\underline{\text{surprise}}'(\hat{}\underline{\text{I}}')(\iota p(\underline{\text{decision}}'(p) \wedge p = \hat{}\underline{\text{John is...}}'))$

That this translation really corrects the first mistake can be easily seen from the table of its values:

(15) $V(9'') = \begin{cases} 0, \text{ iff } \underline{\text{decision}}'(\hat{}\underline{\text{John...}}') \wedge \neg \underline{\text{surprise}}'(\hat{}\underline{\text{I}}')(\hat{}\underline{\text{John...}}') \\ 1, \text{ iff } \underline{\text{decision}}'(\hat{}\underline{\text{John...}}') \wedge \underline{\text{surprise}}'(\hat{}\underline{\text{I}}')(\hat{}\underline{\text{John...}}') \\ 2, \text{ iff } \neg \underline{\text{decision}}'(\hat{}\underline{\text{John...}}') \end{cases}$

4. Now I turn to the point which I regards as the second essential mistake in (9'). As it can be seen from (11), (16') is a trivial consequence of (9'):

(16) That John is leaving for Berlin surprises me.
(16′) surprise′ (^I′)(^John is...′)

Notice, however, that (16) is not necessarily a consequence of (9). It may well be the case that John goes to Berlin every week-end and therefore the fact itself does not surprise me but this being a decision (or, with other sentential names, a dream, an urgent message etc.) it does, indeed. To put it in another way, the present construction would make (17) a contradiction - since both (16′) and its negation would follow from its translation - while it is not:

(17) I am not surprised by the fact that John is leaving for Berlin but I am surprised by the decision that John is leaving for Berlin.

The table in (15) also makes it clear that the new translation (9′′) has not corrected this mistake, either, that is, further corrections are needed.

The source of the mistake in (9′′) is that the relation between the sentential name the the that-clause is indicated by means of equating the variable bound by the iota-operator with the embedded proposition. This problem can be resolved if, instead of an equation, we say that the iota-bound variable contains the proposition as its member. In this way we introduce an intermediate category between propositions and sentential names (or, predicates) applicable to them. The new category will have the type <<s,t>,t> (so far reserved for proposition-level intransitive verbs and common nouns) and all the other categories become one level more complex, i.e. are built up from this new category in the same way as they have so far been built up from t. The new category has mainly logical significance. No direct translation will yield a constant of this type and the meaning of its elements cannot be described in natural language any more precisely than 'the way/the kind of thing as the subordinate clause can be conceived of'.

5. Before giving an exact definition of the new categories, syntactic and translation rules I will demonstrate this final translation proposal on the example discussed above and show that the two crucial objections no longer apply to it. The final translation of (9) is (9'''):

(9''') surprise' (^I')(^¬C(decision'(^C)∧ C(^John is...')))

Compare with the final translations of (16), (13), and (18):

(16) That John is leaving for Berlin surprises me.

(16''') surprise' (^I')(^¬C(C(^John is...')))

(13) It is a decision that John is leaving for Berlin.

(13''') ∃C[decision' (^C) ∧ C(^John is...')]

(18) The decision is that John is leaving for Berlin.

(18''') ∃C[∀D[decision'(^D)⇔D=C] ∧ C(^John is...')]

(16''') is no longer a logical consequence of (9''') since for the former to be true it is necessary that there be exactly one element of $D_{<<s,t>,t>}$ that ^John is leaving for Berlin is a member of. This is not a necessary condition of the truth of (9''') however, since it can be true even if there are several such members of $D_{<<s,t>,t>}$ but only one of them is an element of decision'. That is, we have got rid of the second mistake. As for the first one: (13''') remains a consequence of both (9''') and its negation since these sentences now presuppose that

(19) ∃!C[decision'(^C) ∧ C(^John is...')]

which obviously implies (13'''). Note, that from (9''') one cannot infer (18'''), which is quite right, however, since the former does not say that there is only one decision.

The definition of syntactic rules and their translations is somewhat complicated by the fact that in sentences like (9''') and (16''') the iota-operator binds the variable characterizing the embedded proposition and this proposition cannot be retrieved. Therefore the structure of (9''') cannot be obtained from that of (16'''). A simple solution seems to be to provide that with various translations (which is also justified by the variety of roles this word may play in syntax). Consequently, the same proposition may belong to various types

depending on the role it plays in the sentence and the syntactic rules applicable to it vary with those categories.

The proposition-level categories redefined (members remain unchanged):

$B_{\underline{IV}} = B_{t/(t/t)}$

$B_{\underline{CN}} = B_{t//(t/t)}$

$B_{\underline{T}} = B_{t/\underline{IV}} = B_{t/(t/(t/t))}$

$B_{\underline{IV}/T} = B_{(t/(t/t))/(t/(t/e))}$

$B_{IV/\underline{T}} = B_{(t/e)/(t/(t//(t/t)))}$

New categories and basic expressions:

$B_K = B_{(t/t)/\underline{T}}$	be
$B_{H1} = B_{(t///(t/t))/t}$	$\underline{\text{that}}_1$
$B_{H2} = B_{\underline{T}/t}$	$\underline{\text{that}}_2$
$B_{H3} = B_{(\underline{T}/\underline{CN})/t}$	$\underline{\text{that}}_3$

Syntactic rules:

S_3' : If $\alpha \in P_{\underline{CN}}$ then $F_{3,1}'(\alpha)$, $F_{3,2}'(\alpha) \in P_{\underline{T}}$ where $F_{3,1}'(\alpha) = \underline{\text{a(n)}}\alpha$ and $F_{3,2}'(\alpha) = \underline{\text{the}}\ \alpha$.

S_4' : If $\alpha \in B_{H1}$ and $\beta \in P_t$ then $F_4'(\alpha,\beta) \in P_{t///(t/t)}$ where $F_4'(\alpha,\beta) = \underline{\text{it}}\ {}^*\alpha\beta$.[1]

S_5' : If $\alpha \in B_{H2}$ and $\beta \in P_t$ then $F_5'(\alpha,\beta) \in P_{\underline{T}}$ where $F_5'(\alpha,\beta) = \underline{\text{it}}\ {}^*\alpha\beta$.

S_6' : If $\alpha \in B_{H3}$ and $\beta \in P_t$ then $F_6'(\alpha,\beta) \in P_{\underline{T}/\underline{CN}}$ where $F_6'(\alpha,\beta) = \alpha\beta$.

[1] Since the present paper focuses on the semantic problems of that-clauses, the syntactic rules I give here are merely meant to illustrate the working of my proposals. Therefore, I ignored the intricacies of the alternation of bare that-clauses and their dummy-it version in subject position and provided only for the generally applicable it-version. It appears that the only case in which this procedure is not viable is the type (18) since the sentence It is the decision that... is not synonymous with The decision is that... although my fragment treats it as if it were.

S_7' : If $\alpha \in P_{\underline{T}/\underline{CN}}$ and $\beta \in P_{\underline{CN}}$ then $F_7'(\alpha,\beta) \in P_{\underline{T}}$ where $F_7'(\alpha,\beta) = \underline{\text{the}}\ \beta\alpha$.

S_8' : If $\alpha \in P_K$ and $\beta \in P_{\underline{T}}$ then $F_8'(\alpha,\beta) \in P_{t/t}$ where $F_8'(\alpha,\beta) = \alpha\beta$.

S_9' : If $\alpha \in P_{t///(t/t)}$ and $\beta \in P_{t/t}$ then $F_9'(\alpha,\beta) \in P_t$ where $F_9'(\alpha,\beta) = \alpha\ [*/\beta]$.[2]

S_{10}' : If $\alpha \in P_{\underline{T}}$ and $\beta \in P_{\underline{IV}}$ then $F_{10}'(\alpha,\beta) \in P_t$ where

$F_{10}'(\alpha,\beta) = \alpha\beta$ if $\alpha = F_7'(\gamma,\delta)$ and

$F_{10}'(\alpha,\beta) = \alpha[*/\beta]$ if $\alpha = F_5'(\gamma,\delta)$.

The new basic expressions translate as:

$\underline{be}' = \lambda Z.\lambda p.{}^{\vee}Z({}^{\wedge}\lambda R.{}^{\vee}R(p))$

$\underline{that}_1 = \lambda p.\lambda P{}^{\vee}P(p)$

$\underline{that}_2 = \lambda p.\lambda Q.{}^{\vee}Q({}^{\wedge}\neg C(C(p)))$

$\underline{that}_3 = \lambda p.\lambda Q'.\lambda Q.{}^{\vee}Q({}^{\wedge}\neg C({}^{\vee}Q'({}^{\wedge}C) \wedge C(p)))$

Translation rules:

$T_{3,1}'$: $F_{3,1}'(\alpha)$ translates as $\lambda Q.\exists C[\alpha'({}^{\wedge}C) \wedge {}^{\vee}Q(C))]$

$T_{3,2}'$: $F_{3,2}'(\alpha)$ translates as $\lambda Q.\exists C[\forall D[\alpha'({}^{\wedge}D) <=> C=D] \wedge {}^{\vee}Q(C)]$

$T_i'(i=4,5,6,7,8,9,10)$: $F_i'(\alpha,\beta)$ translates as $\alpha'({}^{\wedge}\beta')$.

To derive sentence (9) I used S_6', S_7' and S_{10}', for (16) S_5' and S_{10}', and for (13) and (18) S_3', S_4', S_8' and S_9'.

[2] That is, $F_9'(\alpha,\beta)$ is obtained by replacing * by β in α.

REFERENCES

Delacruz, E.(1976): Factives and Proposition Level Construction in Montague Grammar. In: Partee, B.(ed.), Montague Grammar. Academic Press, New York

Kiparsky,P.,C.Kiparsky(1970): Fact. In: Bierwisch,M.-K.E.Heidolph(eds.), Progress in Linguistics. The Hague

Molnár,I.(1980): Existential Relations in "hogy"-sentences (Sentences Containing a that-clause) in Hungarian. In: Kiefer, E.(ed.), Hungarian Contributions to General Linguistics. John Benjamins, Amsterdam

Montague,R.(1974): The Proper Treatment of Quantification in Ordinary English. In: Thomason, R.(ed.), Formal Philosophy, Selected Papers of R. Montague. Yale Univ. Press, New Haven and London

Ruzsa, I.(1978): A New Approach to Modal Logic, Computational Linguistics and Computer Languages XII

B. SYSTEM ARCHITECTURE

Computational Linguistics and Computer Languages Vol. XIV. 1980.

ON THE BASIC CONCEPTS OF SDS /SYSTEM DEVELOPMENT SYSTEM/ PART I.

by

Gábor DÁVID

Computer and Automation Institute of the
Hungarian Academy of Sciences, Budapest,
Hungary

ABSTRACT

The problem of programming technology is embedded here in the more general problem of system development technology and presented here in this environment. The reason to do so is that the level of system development is rather architectural than the level of programming, hence the underlying question can be formulated: "which type of architecture is suitable for system development"? Architecture is meant here as both hardware and software and it is interfaced with users via the operating system, specially designed to this problem. This operating system consists of two pairs of /pairwise "orthogonal"/ notions of processors and processes and of specifications and implementations. The flow of the system development /or as a special case: of the program-development/ is an appropriate sequence of the /verified/ steps through the specification - implementation and allocating and activating processors - processes. The common notion for these is the frame, which can be qualified by the user as process or processor and specification and implementation.

System Development System SDS is being designed as an experimental operating system with specification and implementa-

tion languages although the available languages will not be described here in details.

1. INTRODUCTION, MOTIVATIONS

System development methods require an iterative process from the formulation of problem through the description of solution, until the realization of the solution and of the problem. This process includes the questions on:

- the adequacy of the description of the problem /together with its environment/
- the completeness of the description /unambiguity, etc./
- verified /or proved/ realization of the problem
- a comparison of the real-life problem and the realized model.

From technological point of view, these problems can be reformulated as

- design methodology
- implementation and realization methodology
- documentation
- verification.

They are interconnected. The adequacy of the description of the problem means that given a language /a formal or natural one/, in which the problem is described. The system designer should know that the description reflects the real-life problem with or without restrictions and if the description of the problem caused some restrictions, then the distance of the "reduced" model and problem is significant or not. Our approach is that a model is adequate if every question which is relevant in the problem, can be answered by the system designed. This is rather philosophical than theoretical or practical requirement because the relevant questions should be formulated in the same language in which the system had been described.

The completeness of the description is a little bit opposite: if given a description, whether this model covers the application or not. If the description is ambiguous, the model is either empty or non-deterministic. We may assume that the problem to be solved, the system is deterministic. The model should not contain contradiction.

The verification of the problem-realization includes the question of the implemented model /the system itself/ and the description in the design language.

The process of system-development can be represented as shown in Fig.1,

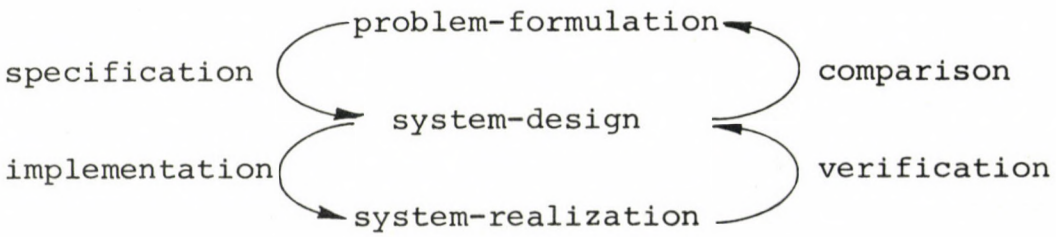
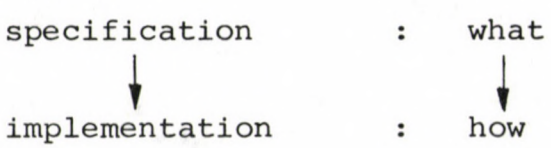
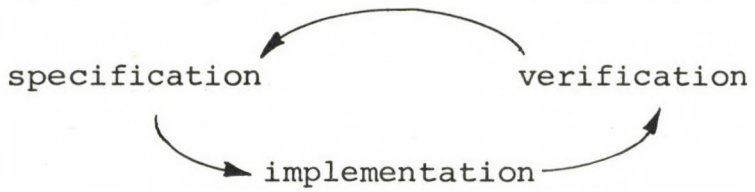


Fig.1.
System development

where the arrows on the left side lead us to the solution and those on the right side do the quality control of the solution. We used the words specification and implementation to distinguish the actions of description of what the system will do /the specification/ and of how it will do /the implementation/. Due to the structured approach, these actions can be repeated on the lower level again, describing the more elementary subsystems by specification and by implementation. Hence the scheme can be reduced to



The other side: the comparison is made between the specified system and the problem or it is made between the implementation and specification. Assuming that the problem formulated in a formal language, then the comparison between the system designed and problem, is a special case of verification, and the development cycle is completed:



This cycle is repeated until the verification states the correspondence between the implementation and specification, and on the lower-level subsystems can be repeated this cycle again.

This scheme is an oversimplified one. System development is a process not only in time, but also in the tools applied. In order to illustrate this, a list of cases is given as follows:

- in some phases the designer wants to use only the specifications in a structured way /hence the implementation part is empty/,
- designer could modify, delete or rewrite the specifications,
- the implementation-part may vary in time: for example, at the first step the designer wants to simulate the subsystem in question and later he will realize it in an algorithmic language suitable for his purposes, or there will be a hardware component realizing the subsystem,
- the notions of specification and implementation are interchangeable: the implementation may consist of a set of lower-level specifications and sometimes the implementation describes the problem itself,
- the specification language should reflect the nature of the problem, hence we used a set of languages in which the systems can be specified /and on different levels they may vary/.

Hence we need a set of implementation-languages, a set of specification-languages /not necessarily disjoint ones/, and a set of mechanism by which one can verify the described versions.

Until now we discussed mostly the requirements /but it is not an exhaustive discussion/ to illustrate the main reasons for research System Development System, SDS. This project started with automatic programming /1-5/ in 1975, an independent research was made in the line of description of control structures /7, 8/, and for architecture-description /6/, /14/.

SDS is an experimental operating system, integrating the classical procedural and non-procedural /new principled/ languages, like Structure Logic Language SL /1-5/, PROLOG /9/, following the line of Kowalski /lo/. SDS has the same philosophy as described by R.M.Burstall and J.A. Goguen /11/, R. Nahijama et al. /12/, and W.A. Wulf et al. /13/.

In SDS the basic notion is the frame: frames are the elementary units which can be manipulated by both SDS and the user.

A frame consists of:
- specification-part and
- implementation-part.

In the specification-part the user expresses what this frame will do: on which data-structures and which transformations will be executed on data structures, which assertions will hold. The implementation-part is a realization of the frame, written in an implementation language.

A frame may contain locally declared subframes and in the implementation-part they can be activated. Also it may use externally described frames; hence frames can be structured.

Before we would go further, let us analyse a typical action in present-day's operating systems. The program consists of:
- a set of declaration D
- a program-body B

and the system will execute it. We had assumed that the program written in a language L either had been translated into an executable form or L had an interpreter. In a program like

```
        integer i,j                                        } D
        integer array x(0:10), y (0:10);
1       begin read x;
2          for i:= 1 step 1 until 10 do
3              y (i):= x (i);                              } B
4          print y;
        end,
```

The declaration-part describes a processor P /a hypothetical one/ in which the data structures i, j, x, y would be implemented and this processor has those instructions what L has:

$$P : \{L, D\}$$

The body between begin and end uses this processor only. Let us introduce a programcounter C varying on the statements labelled with 1,...4, initially 1.

Process F here is meant as an execution of the program-body by the processor

$$F : \{ P, C, B \}$$

The counter C is shared between P and B, we may assume that it is a part of the processor, but we want to separate it.

In this model the first significant step was made by SIMULA '67. The concept of the class means that the processor P is described not only by the users' data structures but the interpreter of the language L is extended by special "instructions", i.e. transformations defined on user-declared data. Hence instead of D we have a declaration D', in which the set of transformations T has been added

$$D' = \{ D, T \}$$

and the processor P' will be able to execute not only its instructions, but the new ones also belonging to T

$$P' = \{ L, D' \}$$

The process F remains the same /although in the SIMULATION class the user may use sophisticated control-structures, manipulating C/:

$$F: \{P', C, B\}$$

In the body B the user may use all instructions of P'. This step made by SIMULA '67 can be treated as a dynamic extendibility of processors. In the model of SIMULA '67, the process remains the excution.

The second step is due to those which can be represented here by Alphard /13/ introducing the forms the concept of the process is extended: not only the execution, but the verification, abstraction, implementation are also introduced as typical processes and they are embedded in the language itself. /This project has many other main results, too, but here from our point of view we are interested in this step./ In this approach one can build up processes with other processors not only with the language-processor but a processor P_V "executing" reasoning, i.e. the verification mechanism, P_V and processes

$$F_V : \{P_V, \{D, B\}\}$$

The body B contains the appropriate "instructions", i.e. logical statements for invariants, preconditions and post-assertions, etc., and also the declaration-part is included in this process. /Here the counters C have been omitted./ The notion of processes extended by Alphard and the same "body" B can form different processes with different processors.

From this we may summarize the results quickly. In this system of notions the language of control structures made possible to change the control dinamically /based on Petri-nets, this result is due to Kotov /7/ and Dávid /8/, /14//, realizing the notion of "computed control". The logic programming or new principled programming teams /1-5, 9, 10/ proved that the body may be empty if the "declaration-part" written in a logical language having correct semantics and calculi. From this time the declaration-part is named by specification-part.

Burstall, Goguen /11/ and Nahijama et al. /12/ showed that with an appropriate set H of transformations defined on the specifications /represented by D/ one can form a process of processes

$$F_v^{h_i} : \{P_v, h_i(D)\} \quad h_i = h_0, h_1, h_2 \ldots h_{i-1}, h_j \varepsilon H$$

Having these results, what we want to do is not else but to try to extend all of the notions of processors, processes, specification, implementation, control- structure, etc. more or less systematically and apply them for system-development purposes. SDS will do this, and it will support design-methods like structured design, non-deterministic programming, etc. also.

In our approach a frame F is a pair of specification S_F and implementation I_F

$$F = \{ S_F, I_F \}$$

and the control-structure C_F is described in S_F as an extension of the notion of the "counter".

Let be given a processor P then

$$P_F = P \cup S_F$$

forms an extension of processor P. P_F is called abstract processor and is meant as a set of abstract data-types and the transformations defined on them.

The ordered pair

$$R_F = < P_F, I_F >$$

is a process, /where P_F - an abstract processor, I_F -implementation/.

The implementation I_F is a transformation defined on abstract data-types of S_F hence R_F can be treated as a processor again.

Hence one can define a frame

$$F_1 = \{S_{F_1}, I_{F_1}\}$$

with a processor

$$P_{F_1} = P_F \cup S_{F_1}$$

and resulting the process

$$R_{F_1} =< P_{F_1}, I_{F_1} >$$

in that case the frame F is activated by F_1.

The typical batched-job run can illustrate this. In a language, for example FORTRAN, the program F is written. The program is the frame F, consisting of declarations and the algorithm as the body B.

$$F = \{ D, B \}$$

The FORTRAN translator FT is a processor, hence

$$P = FT \cup D$$

declares those data on which the body B will be executed. During translation the process

$$R = < FT \cup D, B >$$

will be formed. If we have data A in this actual run, the

$$R_1 \underset{*}{<<FT \cup D,} \underset{*}{B>,A} > \qquad /1/$$

is again a process, where R stands as a processor. This reasoning is true, regardless of the fact that the R as an abstract processor had been defined as a process previously and had been translated into another processor /the existing hardware processor/ of assembly or of lower-language. This means, that in a deeper level of investigations /that of the system, namely/ not the R itself will be an abstract processor, but the processor of the machine-language MACHL, and R_1 can be rewritten as

$$R_1 = <<\text{MACHL}, \underset{*}{<FT \cup D,} \underset{*}{B>>,A}> \qquad /2/$$

From the point of view of the user - and his command language - the version of /1/ is of great interest because he sees that level and not the internal one of /2/.

The stars below commas in /1/ and /2/ should be explained.

These sign just the actions on operating systems, controlled by the users' commands, read left to right as translate and execute. This is the everydays interpretation of the followings: In

$$R = < P, I >$$

the user says:
- qualify P as a processor,
- define with I the process R

Qualification means that /already defined/ P will be allocated regardless of the role played by P previously /i.e. whether it is a process or a processor/.

The qualification-mechanism assumes that a process, like R can be used as an /abstract/ processor. Recalling the definition, R should be formulated as a set of data-types and the transformations defined on them. But the transformation is described by the algorithms, implemented by I_F . Hence we need the description of what I_F will do. This means that in the same language in which the specification-part S_F of the frame F has been described, one has to formulate I_F also. Let us denote this description by S_{I_F} and the algorithms realized by I_F by A_{I_F},

$$I_F = (S_{I_F}, A_{I_F})$$

$$F = \{ S_F, (S_{I_F}, A_{I_F})\} \qquad /3/$$

If this frame will be used later as a processor, then the processor

$$P' = P_F \cup S_{I_F}$$

declares that one, which is equivalent with R_F from the users' point of view. The frame F in /3/ assumes a processor P_{S_F},

$$P_{\{S_F,(S_{I_F},A_{I_F})\}} = P_{S_F} \cup S_F$$

and this processor will interpret A_{I_F} as it is described in S_{I_F}:

is the process, for which

and one can form a new frame

$$F_1 = \{ S_{I_F}, (S_{I_{F1}}, A_{I_{F1}})\}$$

and its processor:

$$P_{F1} = P_{SI_F} \cup S_{I_{F1}}$$

In the remaining parts of Part II, the informal description of the frames will be given and the architecture required by SDS will be discussed. In this early version the author wants to explain those tools which are necessary during systems' design and development and integrated into the operating system SDS.

REFERENCES

[1] Dávid, G.: Structured Automatized Design of Microprograms in Large Scale Integration,H.W.Lawson et al. /eds/,North-Holland /1978/

[2] Dávid,G., Keresztély,S. and Sárközy,A.: Microprogram Synthesis by Theorem Proving. Proceedings of the II. Hungarian Comp.Sci. Conf. /1977/, Part 1, 291-310.

[3] Dávid,G., Keresztély, S., Losonczi,I. and Sárközy,A.: Logic-Based Description of Microcomputers /1978/ MTA SZTAKI Közlemények, Budapest, Hungary

[4] Dávid,G., Keresztély,S., Losonczi,I. and Sárközy,A.: Microprogram Synthesis /1978/ MTA SZTAKI Közlemények, Budapest, Hungary

[5] Dávid, G.: Proving Correctness and Automatic Synthesis of Paralel Programs. In Algorithms,Software and Hardware of Parallel Programs. 1980. VEDA, Bratislava, Academic Press /in print/

[6] Dávid, G.: Architecture Language. MTA SZTAKI Közlemények, Budapest, 1979 /in print/.

[7] Kotov,V.E.: Concurrent Programming with Control Types. In: Constructing Qality Software,North-Holland,1978.

[8] Dávid,G.: Description of Dynamic Control Structures, Algorithms'79. Proceedings.

[9] Warren,D. and Pereira,L.: PROLOG-The Language and its Implementation Compared with LISP.Proc. of ACM SIGART-SIGPLAN Conf. on "AI and Programming Languages", Rochester,N.Y. August,1977.

[10] Kowalski,R.: Predicate Logic as Programming Language, Proc.IFIP Congress,1974, North-Holland.

[11] Burstall,R.M. and Gougen,J.A.: Putting Theories together to make Specifications.Proc. of the 5th IJCAI, Cambridge, Mass.1977.

[12] Nahijama,K., Honda,M. and Nakahara,H.: Describing and Verifying Programs with Abstract Data Types. Proc. of "Formal Description of Programming Concepts", E.J. Neuhold /ed./ North-Holland, 1978.

[13] Wulf,W.A., London,R.L. and Shaw,M.: Abstraction and Verification in Alphard: Introduction to Language and Methodology.Techn. Rep. Carnegie-Mellon University, Pittsburgh, PA.1976.

[14] Dávid,G.: Restructurability-a Tool for Systems Development in "Microprogramming,Firmware and Restructurable Hardware", Lorth-Holland, /1980/ Ed. G. G. Chroust /in print/

Computational Linguistics and Computer Languages Vol. XIV. 1980.

S I L I C E A

A SIMULATOR FOR 'REALIZABLE' CELLULAR AUTOMATA*

W.O. HÖLLERER
Lehrstuhl C für Informatik
Technische Universität Braunschweig

Braunschweig
FRG

Abstract

The topic of this paper is the demonstration of a simulation software for cellular processors which are composed of MEALY-type cells. Although such processors are also of theoretical interest, here they are treated as models for (LSI-) realizable cellular structures. This approach is motivated by the manifold works of LEGENDI. The simulation software was developed for a highly interactive graphic system and makes extensive use of its capabilities concerning the input of transition functions, the design of configurations, and the control of simulation runs. Whenever it is appropriate and efficient the dialogue with the system is based on light pen interactions. We made efforts to design the system as self-explanatory as possible. By our experiences, unpracticed students can start to develop their first own cellular algorithms after a short oral introduction of ca. one hour.

* This work was partly supported by the Deutsche Forschungsgemeinschaft, Grant Vo 287/1.

0.0 A SUMMARY OUTLINE OF SOME EXISTING SIMULATION SYSTEMS FOR CELLULAR STRUCTURES

In the following we give a short overview of existing simulation systems, as far as they have been published and the publications have been available to us. Surely, there may exist a lot of further simulation programs; but in general these have been developed to test hypotheses concerning a special and small class of problems. A typical example is the multitude of simulation programs for CONWAY's game of life existing all over the world. In this case transition function, neighbourhood and state set are fixed and the only varying parameter is the initial configuration. As the underlying cell is a two-state automaton input and (representation of) output of configurations are no sophisticated tasks and the overall data management is simple. The simulation systems discussed below were all designed with respect to special problem classes, but they are 'universal' in that sense that they apply to a variety of application fields which are different from those they were originally intended for. A further criterion is that the discussed systems provide higher level features for (1) definition of the space topology (including local neighbourhood(s)), (2) definition of the basic cell type(s), (3) (re)definition of the local transition function(s), (4) definition of (initial) configurations, (5) formatted output of (intermediate) results, (6) control of the simulation steps, if possible by interactive intervention, (7) flexible control of data- and information-flow.

BAKER and HERMANN(1970,1972) have developed a modular programming system (called CELIA, in FORTRAN) for the simulation of linear arrays of identical MOORE-type cells which are connected according to the von NEUMANN template. As the authors' main application field is that of biology - especially the developmental behaviour of organisms - during one global state transition a cell may divide into two or more cells, or may disappear: This program is therefore a simulator for LINDENMAYER systems. The transition function is either read in as a table

by a standard subroutine, or, it is itself defined as a subroutine in terms of (in general relatively few) logical instructions. Besides, the system allows to test for cycles - concerning the overall state of the array - and to tabulate certain (standard) phenomena as well as histograms. Special user defined statistical modules may be incorporated. The simulation runs can be controlled in that sense that they can be stopped upon the occurence of (the states of) certain kinds of cells. The authors demonstrate the capabilities of their system by verifying LINDENMAYER's famous example of the development of the red alga Callithamnion Rosemium Harvey. In HERMANN and LIU(1973) a 'daughter' of CELIA, an improved version of CELIA, is introduced. Some improvements concern a more efficient memory management making possible the simulation of longer arrays. States are now represented by structures of (different) data types and lists which can be accessed by appropriate selectors (for function definition and outputs). A simple interpretative control language is added to control the execution of CELIA in the batch environment by the user. This makes possible the simulation of more than one system at a time (and to compare the results).

BRENDER(1969) describes (a compiler for) CESSL, a simulation language for two-dimensional cellular spaces. The system is implemented on a configuration of two closely (100 KHz core-to-core) coupled small computers, an IBM 1800 and a PDP-7 with a modified 338 interactive graphic display. CESSL is a procedural higher level language.The specifications of the cell state structure, neighbourhood relationship, size of the cell space and the initial states are included in the language itself as well as the definition of the necessary subprograms that must be supplied. The latter must define: The local transition function (in terms of logical and arithmetical expressions), (up to ten) so-called mapping functions (mapping the actual (selected field of a) state to one of 128 different graphic symbols), user defined commands (for additional interactive control of the simulation run), and input functions (to influence cells

by additional time-dependent inputs from the 'outer world'). During run time the system is controlled either from keyboard or via light buttons on a display menue. This is facilitated by the fact that one can only simulate arrays of up to 32x32 cells which can be represented (by one of the mapping functions) on one screen. The run time system permits, for example, a redefinition of the neighbourhood (provided the number of neighbours remains unchanged), the specification of inputs from the 'outer world', the change of the actual mapping function and the redefinition of (selected components of the structure representing) the state of single cells. BRENDER's simulation examples refer on the first place to biological models. Keeping in mind that a cell state can be especially a real number, it is not surprising that a further example demonstrates the solution of the LAPLACE equation with relaxation methods. It should be mentioned that BRENDER gives a lot of hints concerning the development of cellular simulation languages that exceed the frame of his specific implementation. BRENDER and FRANTZ(1971) give an expanded description of CESSL and its extensions since 1969.

LEGENDI(1976) describes CELLAS, a batch oriented, command-type language which is implemented as an interpreter. There exist several versions of CELLAS for different computers (in fact, LEGENDI therefore speaks of a whole family of simulators). A very interesting variant within this family is INTERCELLAS - see LEGENDI(1977) - which again stands for a whole 'subfamily' of simulators. Roughly speaking, concerning the instruction set INTERCELLAS is a shrunken version of CELLAS which is additionally supplied with some interactive capabilities. Therefore, it is well suited for small real-time systems like the PDP-8 or the MITRA-15. There exists cross-software that permits to create versions of INTERCELLAS that are adapted to the specific machine and configuration on which they are to be run. We don't want to go into the differences between the two programming systems and mention only some of the common main features.

One of the most distinctive characteristics are the vari-

ous ways to define the local transition function of the cellular space. This can be done either by tables, by lists of terms, by definition of terms in tree form, or by so-called feature definition (for examples of these types see LEGENDI(1977)).The space to be simulated may be inhomogeneous in such a way that different cells may have different (predefined) transition functions. The presetting of the space is facilitated by some simple geometrically oriented instructions and copying features for (initial sub-) configurations. The presentation of intermediate and final results is basically controlled by three parameters which specify (1) at which time (i.e. after how many global transitions), (2) what subconfiguration has to be displayed, and (3) how this should be done, i.e. there is a conversion table defining which (external) characters correspond to the cells' states. The simulation software was designed for the development of cellular algorithms and higher level languages (for e. g. 2- and 16-state) cellular processors. A lot of yet unpublished algorithms have been developed, e.g. concerning integer- and vector-arithmetics, sorting and further non-numerical applications. In LEGENDI(1978) TRANSCELL, a language for the definition and minimization of transition functions is given. The main goals are (1) to provide a tool for an easy and impressive formulation of transition tables, (2) to generate from this information very compact (optimized) presentations. As the function definition is performed in terms of so-called properties this optimization must be done (3) concerning a specific set of basic properties, i.e. those that are realized by the hardware of a given cellular processor.

SIBICA has been developed by PECHT(1977). It was designed for the interactive simulation of two-dimensional cellular spaces with two-state MOORE-type cells and MOORE-neighbourhood and is implemented in RT-11 assembler for our GT-42 graphic system. Roughly speaking, SIBICA is an interactive, stack-oriented interpreter for operators which are defined concerning the following operands: Integers, transition functions, configurations, stacksymbols and (recursively defined) lists of such operands.

Typical operators are: Arithmetical operators on integers; definition and calls of (sub-) programs; definition of transition functions which, again, may be combined with already existing ones by (componentwise) logical operations to form new ones. Configurations can be input from keyboard as 4x4-subconfigurations, or in an obvious way by light pen interaction. During run time the local transition function, (sub-) configurations and the control program may be modified in various ways (concerning the sources for that changes). As the structure of the original system is very complex, for an easier handling of the basic operations a block oriented control language - BISICA - was integrated. A typical task for SIBICA was the search for all stable 4x4-configurations in CONWAY's game of life. We mention this example to illustrate that SIBICA provides the mechanisms for the algorithmic generation of (in our example, all 4x4-) configurations (and, as the transition function is symmetric, to omit the symmetric ones).

VOLLMAR(1973) has developed SICELA, a simulation system for two-dimensional cellular spaces. It is implemented in FORTRAN as an interpreter. Therefore, the system is really portable and easily adaptable to special user demands. For that reason SICELA exists in a lot of versions, and, as far as we know it is the most spread out simulation software for cellular spaces. In contrast to all simulators mentioned until now, SICELA allows to define totally inhomogeneous cellular spaces. I.e. the cells themselves as well as their neighbourhood connexion scheme may vary from point to point of the underlying lattice.A further unique feature is that to each cell type there can be explicitely allocated a (fixed) number of counters that are updated during each transition step dependent in the neighbours' states. Note, that (1) implicitely this is possible e.g. in BRENDER's system, too, (2) these cells are specifically organized MEALY-type automata. These principles were originally introduced by the aim to keep the transition tables small and simple concerning special biologically oriented simulations (see e.g. VOLLMAR 1973)). Originally, SICELA was designed for batch-ori-

ented systems; therefore,all definitions fixing a cellular space (e.g. the transition table or initial configurations) must be defined in the system itself or be read in as data. BREDE and SZWERINSKI(1979) have added to SICELA a lot of new features which make the system more comfortable to the programmer and provide interactive capabilities; but as far as these additions concern graphic capabilities they are only applicable to our present machine configuration (a PRIME 300 as host computer and a GT-42 as graphic terminal). Local transition functions can be computed now, i.e. the basic types 'logical' and 'integer' with the full set of operations are introduced as well as 'if-then-else' and 'while' constructions for computing the next state by expressions. Configurations can be generated by repeated setting of a subconfiguration which, again, may be mirrored or rotated. The output of intermediate results can be formatted in various ways; concerning the graphic display states can be represented by user-defined graphic symbols and such figured configurations can be copied by the (HP 7245A) plotter. During run time the simulation can be controlled from the screen by light pen sensitive buttons which correspond to (sequences of) SICELA commands. Naturally, cell states can be redefined by light pen interaction in this context. It should be mentioned that the interactive capabilities of SICELA - as far as they are not specific for the graphic terminal - are available with any alpha-numerical terminal, too. The interactive version of SICELA is therefore adaptable to any (virtual) multi-user system (provided with FORTRAN). The improvements resulted from the aim to make the handling of the system more comfortable with respect to research that had been done by our chair concerning the 'Computing Space' of K. ZUSE(1969). The object of this work was the modelling of physical laws in a cellular automaton which, again, serves as a model for the physical space. A further variant of SICELA is SICELA-H. This adaption was done by LEGENDI and DIÓSLAKI(1979). Mainly they improved the original system in such a way that the originally format driven input is abolished by a preprocessor.

0.1 MOTIVATIONS AND THE STRUCTURE TO BE MODELLED

In the rich literature on cellular automata (e.g. see the surveys of SMITH(1976) and VOLLMAR(1977)) most of the authors concentrate on such, where the basic cell is a MOORE-type automaton with identity output function. At least with respect to (LSI) hardware realizations this is not very realistic, because in this case we have to keep small the complexity of the neighbourhood connexions (LEGENDI(1976)). Three consequence are:

- the choice of a simple neighbourhood (e.g. the canonical von NEUMANN neighbourhood),
- a distinction between internal states and outputs,
- to keep the number of output lines small.

On account of the last two points it seems to be natural to choose a MEALY-type automaton as the basic cell for the models of 'realizable cellular automata'. But there are still some other reasons for this choice.

Consider a MEALY-type cell $M = (I, Q, O, \delta, \omega)$, where O is the output alphabet and $I = O_N \times O_E \times O_S \times O_W \times O_C$ is the input alphabet, and $O_i = O$, $i \varepsilon \{N,E,S,W,C\}$, and Q is the alphabet of internal states.

$\delta : I \times Q \to Q$ is the next state function, and

$\omega : I \times Q \to O$ is the output function.

The cell works in the usual way: dependent on its own (Centre) and its (North, East, South, West) neighbours' outputs it calculates its next state (by means of δ) and the next output (by means of ω).

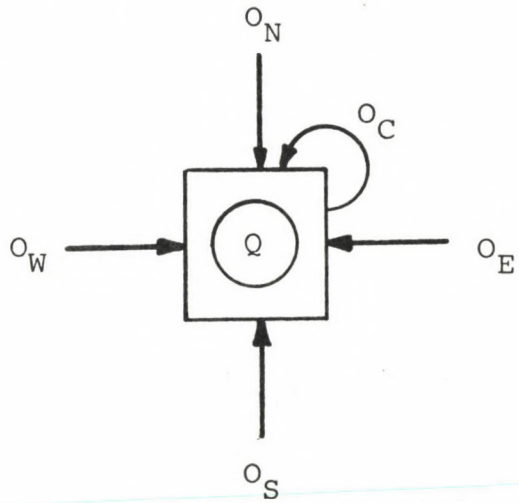


Fig. Ø.1

Then M can be decomposed into two MOORE-automata M_δ and M_ω (the latter at the first glance non-deterministic), where M_δ can be imagined as an automaton performing a microprogram which indicates to the automaton M_ω the (index q of the partial) function ω_q to be applied (Fig.$\emptyset$.2).

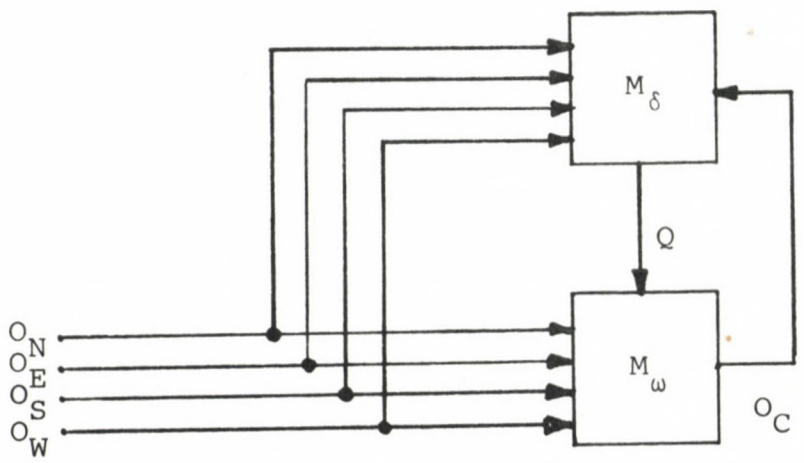


Fig. $\emptyset$.2

Formally, we have

- $M_\delta = (I, Q, Q, \delta, id)$, where
  δ: $I\ Q \to Q$ is the next state function from above, and
  id: $Q \to Q$ is the identity output function on Q.

- $M_\omega = (O_N \times O_E \times O_S \times O_W,\ O_C,\ O,\ \{\omega_q \mid q \in Q\},\ id')$, where
  $\{\omega_q \mid q \in Q\}$ is the set of transition functions, and
  $id' : O_C \to O$ is the identity output function on $O_C = O$.

And the connexion between M_δ and M_ω is given, setting $\omega_q((o_N, o_E, o_S, o_W), o_C) := \omega((o_N, o_E, o_S, o_W, o_C), q)$ for all inputs and for all states.

At a first glance, this decomposition seems to be a very formal and artificial action. On the other hand it illustrates that it is admissible to associate with a cell at any instant

the present state and the present (index of the) function to be applied. Besides, such a decomposition (introduced by LYAPUNOV and YANOV) was used by GLUSKOW as a model for 'real' microprogrammed computers. With respect to an LSI realization it is desirable to pack as many cells as possible on one chip (LEGENDI (1976) claims 10^2 to 10^3 cells/chip). This can only be reached by a reduction of the cell size itself. On the other hand there are experiences from cellular programming advising us not to choose the cell to small concerning the number of functions to be implemented. Bearing in mind that the cell size is to a large extent determined by the microprogrammed control unit M_δ - i.e. the transition table complexity - from these two aims obviously results a conflict.

To escape from this dilemma, LEGENDI(1976) therefore proposes to emulate cellular automata according to the following scheme (Fig.$\emptyset$.3):

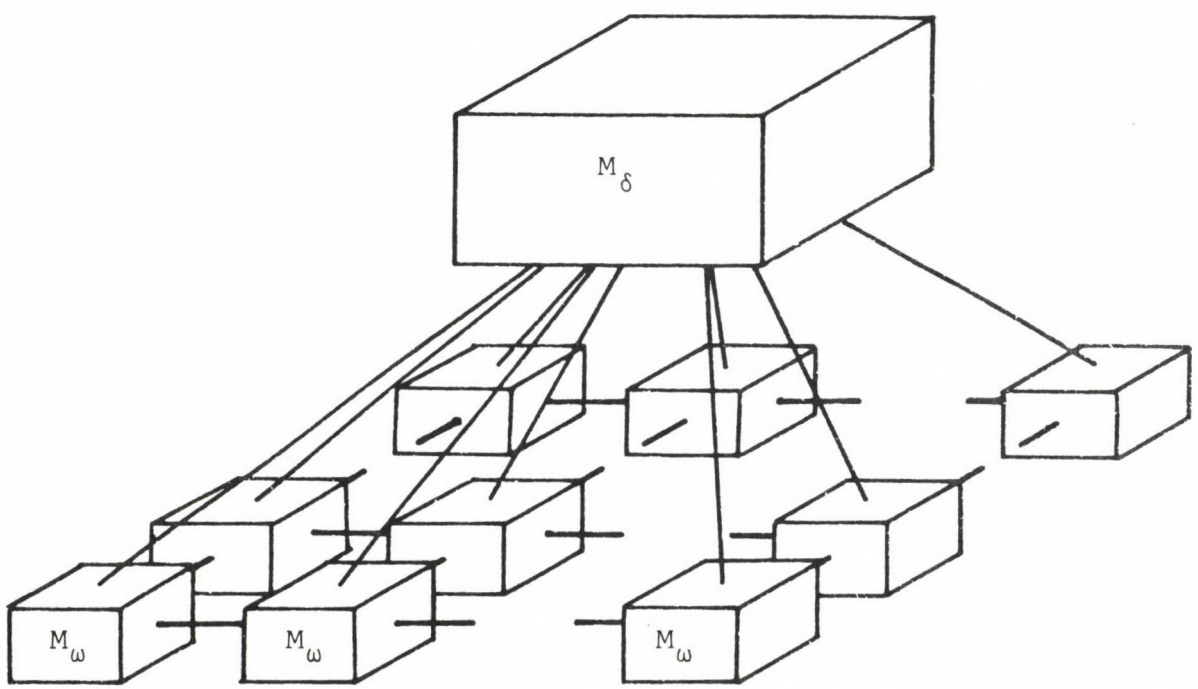


Fig. $\emptyset$.3

- The cellular structure itself is built from the less complex operational units M_ω.
- The generally much more complex control unit M_δ is only existing once (or 'few times') and shared by the operational units in order to calculate their next output and the index of the transition function to be applied next.

If we understand 'sharing a common microprogram' as a serial process this concept could be filled out with a structure, sketched in Fig. ∅.4.

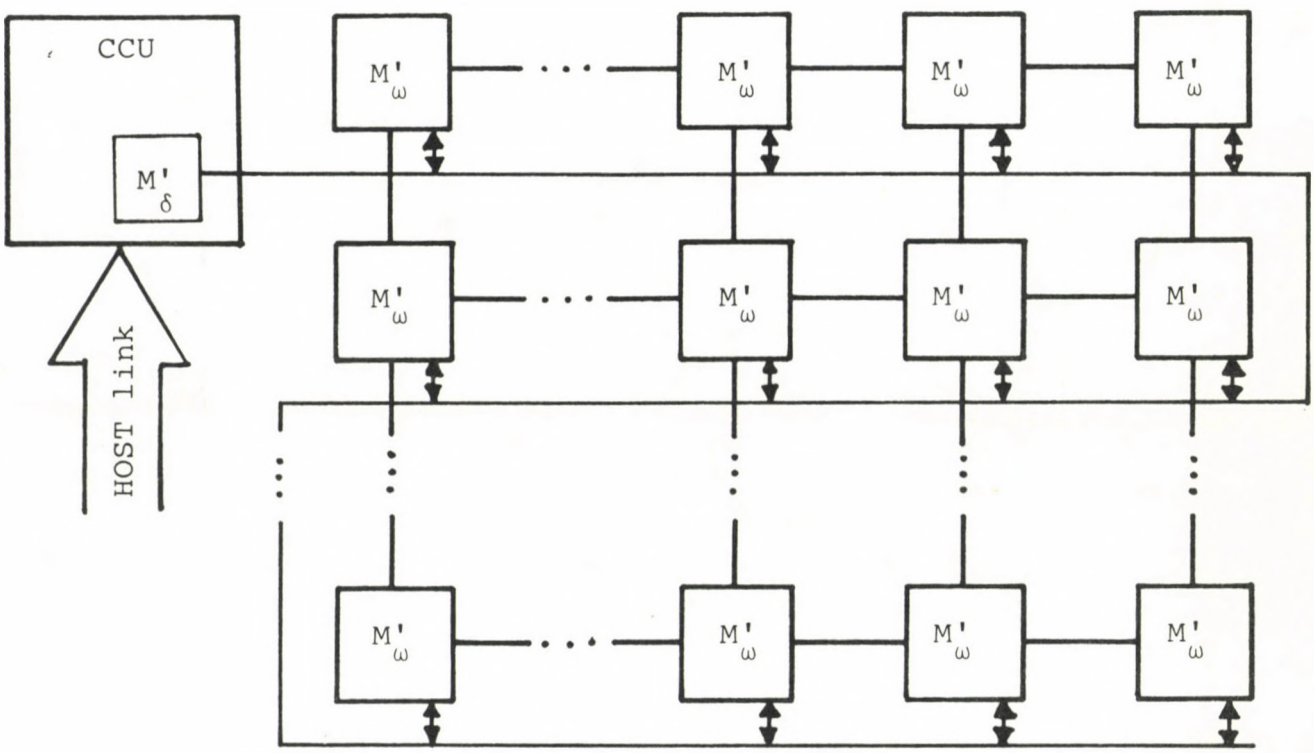


Fig. ∅.4

The cellular structure is built from uniformely connected 'pseudo-cells' M'_ω which are functionally realizing the operational units M_ω. In order to perform this task the 'pseudo-cells' are linked to a common bus system controlled by a unit, called CCU. Among other things the CCU functionally realizes the control unit M_δ. This is done in the following way:

Initially the CCU is loaded from the host computer with the needed - not necessarily complete - transition tables representing the partial functions ω_q. To emulate one transition of the underlying cellular automaton the CCU steps through these tables and sends them line by line $((o_N^t, o_E^t, o_S^t, o_W^t, o_C^t, q^t, o^{t+1}, q^{t+1}))$ consecutively to the bus. The 'pseudo-cells' are equipped with a logic enabling them to perform the basic tasks roughly sketched in Fig.$\emptyset$.5:

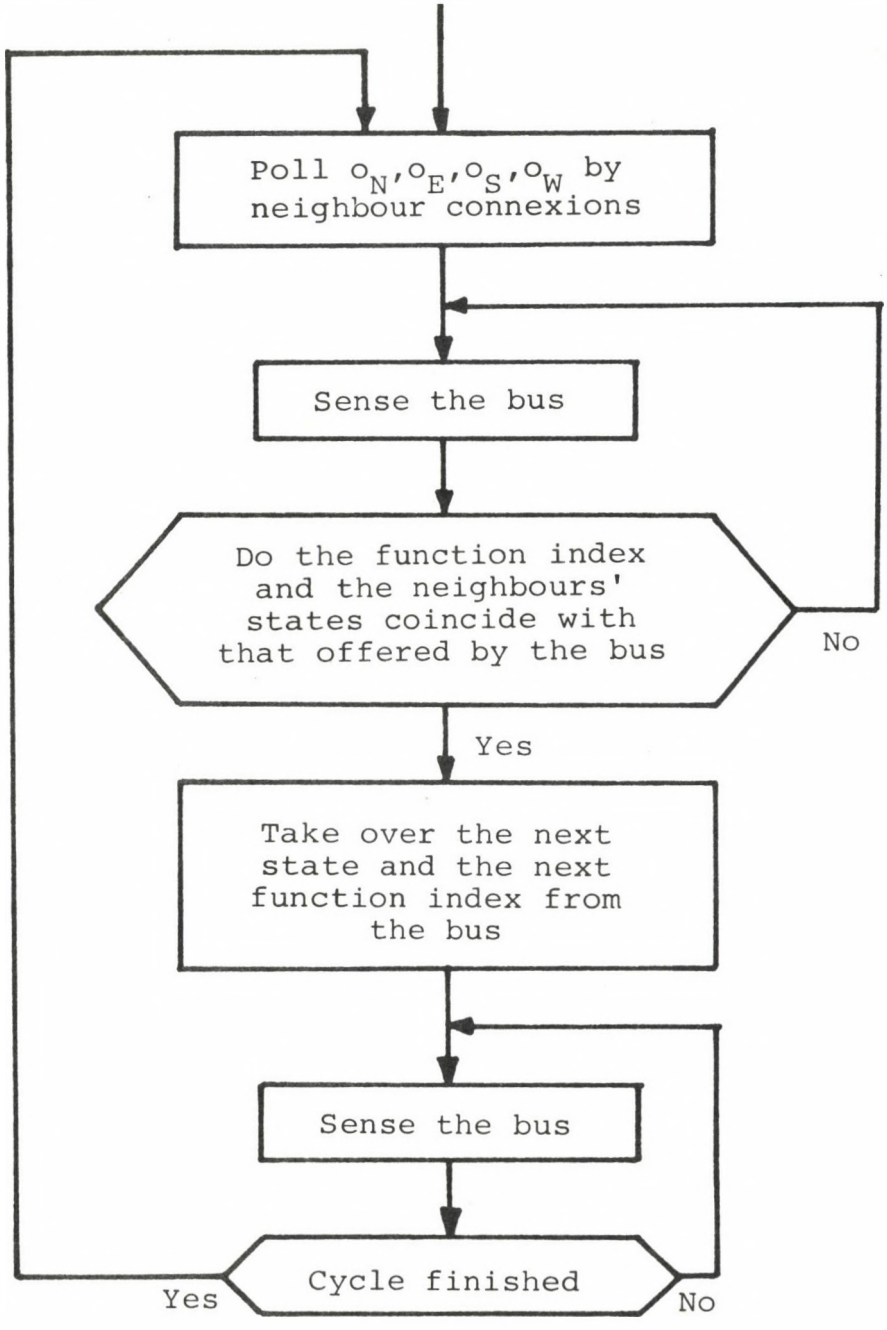


Fig. $\emptyset$.5

Thus, after one run through the set of transition tables by the CCU each cell has assumed its next output as well as its next function index, and one global transition has been performed. From this results that one emulation step directly corresponds to the overall number of entries in the transition tables, and, one consequence is that the transition tables should be specified as little as possible. Concerning the layout of the bus this means that it must be equipped with a care/don't care logic permitting to 'mask out' the outputs of irrelevant neighbours. As in many cellular algorithms (e.g. arithmetics and sorting) a cell is only influenced by few neighbours from this design principle there results an enormous increasing of the time/performance ratio of cellular algorithms by orders of magnitude.

It should be pointed out that LEGENDI proposes this 'table-driven' structure of 'pseudo-cells' as 'the organic part of a computer architecture'. This fact is indicated in Fig.Ø.4 by the link to the host computer.

The structure is much more flexible than the structure (an array of MEALY-type cells) we started from. As the transition table is centralized it can be easily exchanged, which means that (O and Q fixed) any pair of functions δ and ω can be implemented. The transition table can even be swapped dynamically after each global transition. The proposed structure is therefore comparable with the advantages of microprogramming to the architecture of conventional computers, and, on account of its high flexibility, it is more than an emulator for cellular spaces, but a self-contained conception of a cellular structure.

0.2 THE DESIGN OF A SIMULATOR FOR A LEGENDI-TYPE CELLULAR STRUCTURE: S I L I C E A

The simulator is implemented on a system GT-42 which is roughly speaking a PDP 11/10 additionally equipped with a graphic processor and a vectorized working screen with a resolution of 1024x1024 points and light pen interaction capabilities. The structure of the actual configuration is sketched below (Fig.0.6):

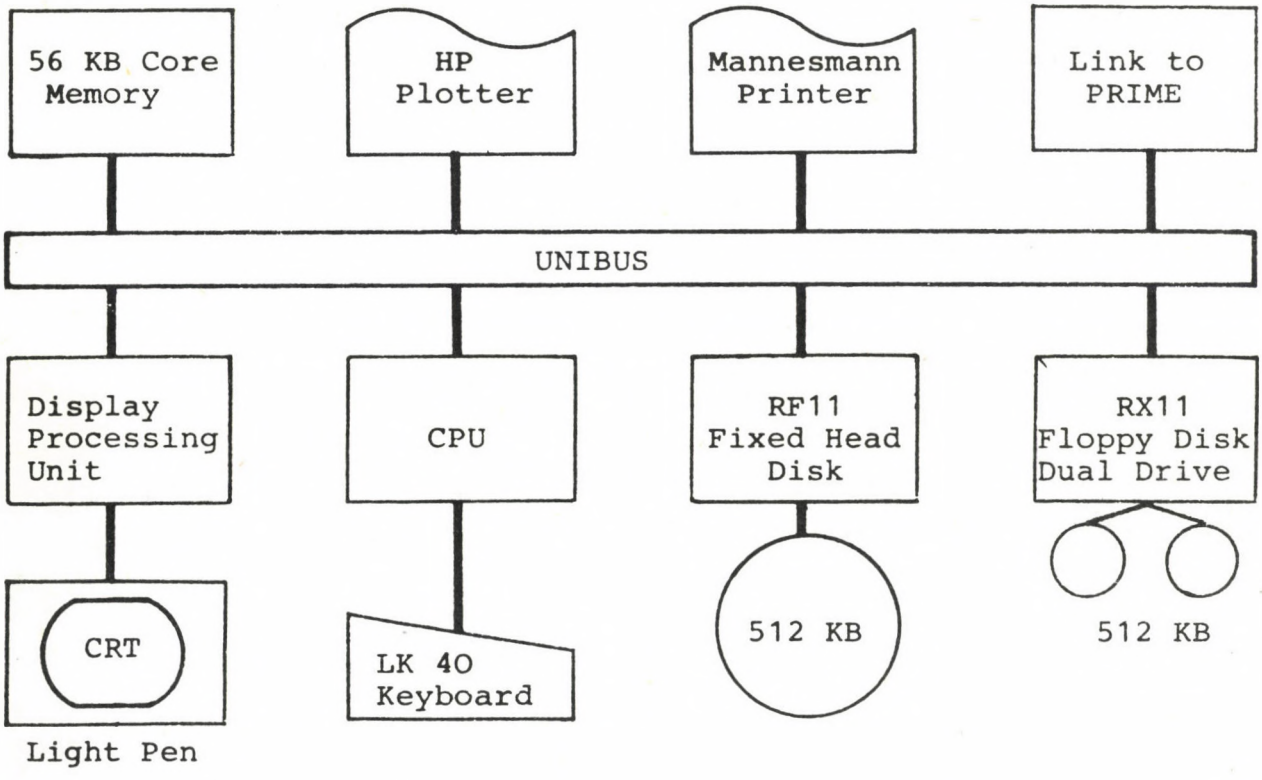


Fig.0.6

The main advantages of the system are obviously its graphic and interactive capabilities. Its disadvantages are obviously the small primary- and secondary-storage capabilities. It should be mentioned that during the implementation period the configuration neither contained the fixed head disk (enabling the efficient handling of overlay structures) nor the plotter (which is in

the first place used to procedure hard copies of the screen). This had consequences concerning the software architecture.

SILICEA allows to simulate arrays of up to 128_x256 cells. Concerning the basic cells the set of external states (i.e. the output alphabet) is $O=\{\emptyset,1\}$ (1 Bit); the maximal number of qualifiers - i.e. the number of different function indices, or internal states - is 128 (7 Bits). The choice of O is motivated by the aim to reduce the neighbourhood connexions to the minimum. From the theoretical point of view, we want to find out whether it is possible to develop efficient (concerning the time/space complexity) algorithms with a two-state cell. The choice of the array size and of the number of qualifiers is obviously determined by the machine's size and structure and the aim to avoid overlay structures whenever this is possible. In order to attain efficient run time characteristics (e.g. it is hard to keep a packed screen flicker free by programming it in a higher level language) SILICEA was implemented in RT-11 Macro Assembler.

From our point of view the development of cellular algorithms is in many stages strongly related to the interactive manipulation of geometrical objects. For example, starting with some basic cells, these are geometrically arranged to form a subconfiguration which again may be combined with another subconfiguration, and so on, until a configuration is established that performs some cellular algorithm. Bearing this is mind we tried to make extensive use of our system's graphic and interactive capabilities whenever this seemed to be advantageous and to meet the requirements listed below:

- Simple and impressive input of transition tables. Easy access to the latter for the purposes of editing, renaming, copying (into different libraries), etc..
- Qualifiers can be associated with free definable graphic symbols in order to allow an impressive identification of cells conerning

  - their location in libraries,
  - the arrangement of cells to bigger units during the MOLECULE phase,
  - the interactive control of the real simulation during the EXEC phase, where the dynamic behaviour of single cells, i.e. the actual state and qualifier and their change to the next state and qualifier, are of interest,
  - classification: different qualifiers can be associated with the same symbol,
  - selection: cells can be located on the screen without knowledge of their absolute coordinates.

- Easy and comfortable arrangement of cells (referenced by atoms here) and already existing configurations (which we call molecules here) to form new configurations. As these geometrical operations are performed without explicit knowledge of absolute coordinates, here seems to be one of the most powerful applications of the graphic system's interactive capabilities.
- Comfortable presetting of molecules with initial states including editing functions.
- Implementation of a simple but powerful control language making possible

  - an efficient supervision of the real simulation,
  - an easy development of cellular algorithms by interactions during program execution (break in and trace concepts!),
  - flexible format-controlled layout for the output of (intermediate-) results to all RT-11 devices in order to reduce the amount of information,

  and, producing good run time characteristics.

It should be emphasized that the purpose of the simulator is not to simulate the structure of a 'table driven cellular processor' but to develop cellular algorithms that can be run on such a structure. In some sense any simulator for a cellular

automaton is a table driven device, which means that in general the transition table only exists once, and, concerning the single cells only as much information is held as necessary to compute their next (overall) state. Thus, SILICEA can be regarded upon as to be a simulator for cellular automata, where the basic cell is a MEALY-type automaton with the above mentioned restictions concerning the cardinalities of the sets of external and internal states.

When building a simulator, one of the most important problems is always how to represent and to input the transition table: In our case the latter may have up to $|O^5| \cdot |Q| = 2^5 \cdot 2^7 = 4K$ entries. From existing cellular algorithms we can derive that in practice the underlying transition table is not an unstructured list but can be disassembled in subtables defining subautomata (in the usual sense). Thus, it is an obvious advance to compose the transition table successively by the transition tables of subautomata which, again, are composed from socalled atoms. By an atom we understand a single qualifier $q_i (0 \leq q_i \leq 127)$ associated with a transition table τ_i which may be obviously identified with a mapping $\tau_i : \{\emptyset,1\}^5 \times \{q_i\} \rightarrow \{\emptyset,1\} \times \{\emptyset,1,\ldots,127\}$. A local partial function is defined by a set of qualifiers $Q' = \{q_o, q_1, \ldots, q_n\}$, $(\emptyset < n \leq 127)$, such as to the induced mappings $\tau_i, (\emptyset < i \leq n)$, for all inputs $p \varepsilon \{\emptyset,1\}^5$ it holds that $\tau_i(p,q_i) \varepsilon \{\emptyset,1\} \times Q'$. Thus we have defined a subautomaton in the usual sense, and, returning to the notion we started with $\delta|_{I \times Q'}$ and $\omega|_{I \times Q'}$ are well-defined.

In the following we give a short description of the user interface of SILICEA, whereas we do not go into the details how it was implemented. Basically, concerning the figures it should be mentioned that ■ is always a light pen sensitive switch, and, that light pen interactions are possible and can be combined whenever this gives sense. In its entirety the simulator as well as some utility routines are described in the german user handbook, SILICEA(1979).

1. ATOMS

An atom is uniquely defined by a qualifier and is (not necessarily is a unique way) referenced by a name and a graphic figure or a pair of hardware-generated special symbols. In the first instance an atom exists or is to be defined with respect to the library at the moment being used. Fig.1.Ø shows the layout of the screen the system initially responds with

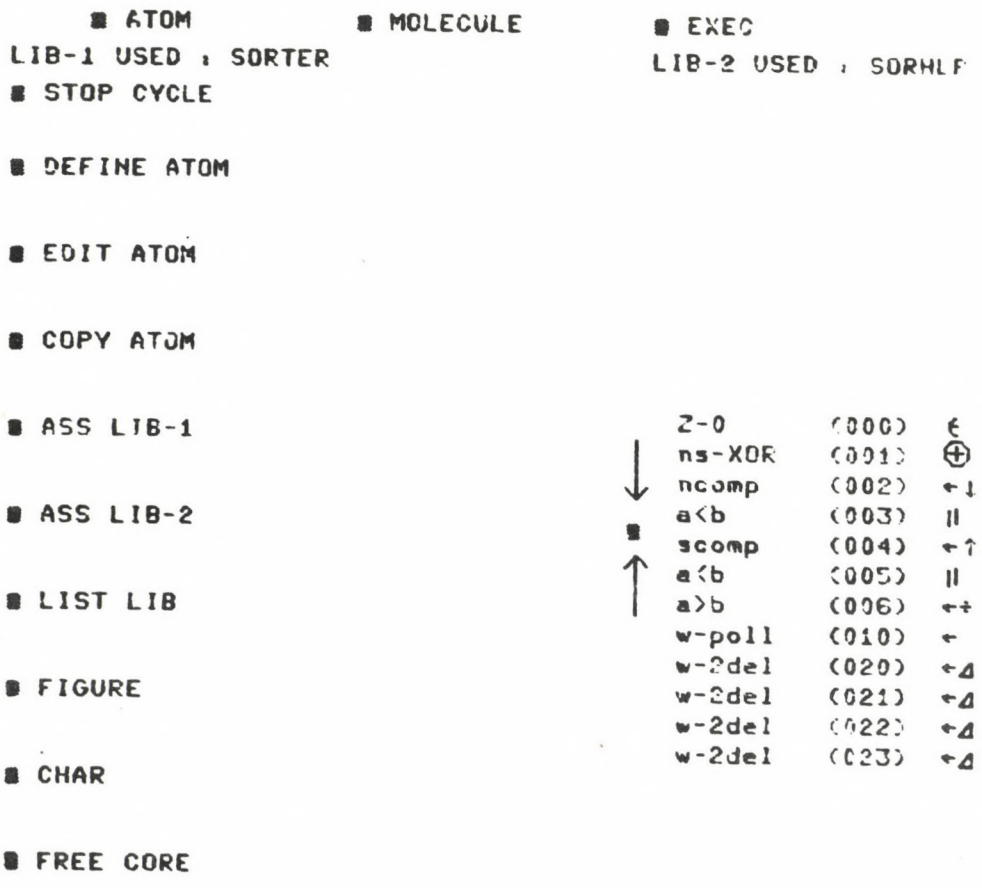


Fig. 1.Ø

And, in general, one of the first actions is to assign the libraries we want to work with. This is done by marking the appropriate switches (ASS LIB-1, ASS LIB-2) and entering the appertaining standard RT-11 names from keyboard. The library in present use (LIB-1 USED, LIB-2 USED) is blinking and we can change

over to the other one by marking it with the light pen. Marking the switch LIST LIBrary results in the representation of the already existing atoms in a special scrolling area of the screen. In Fig.1.Ø we see this listing: E.g. there exists an atom ns-XOR with qualifier ØØ1 and the user-defined symbol ⊕ ; the atom ncomp is associated with the pair of hardware-generated symbols ←↓. The advantage of user-defined symbols is that they can be formed very impressive concerning the local partial function of the cell they are associated with; their disadvantage is that their drawing is very time-consuming, and, thereof may result a flickering screen during the MOLECULE- and EXEC-phase when we have to deal with bigger structures built from cells. This can be avoided by the use of pairs of special symbols (so-called shift-out characters, including the 'blank' symbol (see the atom w-poll)). Examining the atoms named w-2del we see that the unique identification of an atom is its qualifier; besides, these four atoms define a partial local transition function such that the cell - initialized with qualifier 2Ø - delays the west neighbour's output by two time-steps. It is obvious that one needs four qualifiers for internal storage to perform this task. This is only mentioned to make clear that, in some sense, in many applications we cannot make such a clear distinction between 'data' (i.e. outputs or external states) and 'function' (i.e. to comprehend the qualifier only as a function index) as it would be desirable with respect to the clarity and transparency of (the development of) cellular algorithms.

1.1. DEFINE ATOM

If we have marked this switch an initial dialogue is started, and, to begin with, we are asked for the symbolic name of the atom we want to define. Afterwards we are asked for the qualifier and the input qualifier is checked whether it already exists in the library. If this is true an error message occurs and the dialogue restarts. This difficulty can be avoided by generating the qualifier automatically (see Fig.1.1). If we mark this switch the first unoccupied qualifier is offered. We

can confirm this choice, or, marking the switch again the next unoccupied qualifier is offered.

```
-ENTER SYMBOLIC NAME : na-XOR

-ENTER QUALIFIER : 001

 GENERATE QUALIFIER
```

Fig.1.1

Eventually this initial dialogue will be successfully ended and it is turned over to the next display (Fig.1.2):

```
   ■ ATOM          ■ MOLECULE          ■ EXEC
LIB-1 USED : SORTER                   LIB-2 USED : SORHLP
■ STOP CYCLE

                                      S W N E   OWN   NEW   QUALIF
  DEFINE ATOM : na-XOR        (001)   ■ ■ ■ ■         ■       ■
                                      ■ 0 * 0 *   0     0     001
■ LIST LIB                            ■ 0 * 1 *   0     1     001
                                      ■ 1 * 0 *   0     1     001
                                      ■ 1 * 1 *   0     0     001
■ COMPLETE NOW                        ■ 0 * 0 *   1     0     001
                                      ■ 0 * 1 *   1     1     001
   Z-0        (000)  ←               ■ 1 * 0 *   1     1     001
   ncomp      (002)  ←↓              ■ 1 * 1 *   1     0     001
   a<b        (003)  ||
   scomp      (004)  ←↑
■  a<b        (005)  ||
   a>b        (006)  ←+
   w-poll     (010)  ←
   w-2del     (020)  ←⊿
   w-2del     (021)  ←⊿
   w-2del     (022)  ←⊿
   w-2del     (023)  ←⊿
```

Fig.1.2

LIST LIB performs the same function as mentioned above. STOP CYCLE is the exit from this section 'forgetting' all activities having occured until now. The normal exit saving the result of

all activities (i.e. the established transition table) is COMPLETE NOW. Fig.1.2 shows the end of a session during which we have defined an atom ns-XOR having occupied the qualifier ØØ1. The right side of the screen shows the established transition table which may have up to 32 entries. In our case it has 8 entries because we left the east and the west neighbours' states unspecified; this is indicated by the * symbol (i.e. don't care) in the table. Fig.1.3 shows that initially the system responds with a totally unspecified table. Concerning the old state (OWN) the table is preset with the two alternatives Ø and 1; the next state NEW is in both cases preset with Ø, whereas the next qualifier (QUALIF) is preset with that we are starting with, namely ØØ1.

```
    ■ ATOM          ■ MOLECULE          ■ EXEC
LIB-1 USED : SORTER                     LIB-2 USED : SORHLP
■ STOP CYCLE

                                    S W N E   OWN    NEW    QUALIF
  DEFINE ATOM : ns-XOR      (ØØ1)   ■ ■ ■ ■          ■      ■
                                    ■ * * * *   0      0      001
                                    ■ * * * *   1      0      001
■ LIST LIB

■ COMPLETE NOW
```

Fig.1.3

Starting from this rudimentary entries the transition table is successively developed. This is done by three different operations:

- Expand don't care. Marking one of the neighbours N,E,S, W and an appropriate line in the transition table one position is uniquely determined. If this position is occupied by a don't care symbol * the marked line is doubled and the don't care is expanded to the two alter-

natives $\emptyset$ and 1. If this position is occupied by a 1 or a $\emptyset$ it is searched for a line that differs from the marked one only in this position. If this is true the two lines are fused to one line by inserting a don't care symbol at the marked position and deleting the second line; thus we have an operation Retract which may be considered as the inverse of Expand. If this is not true an error message occurs and the command is ignored.

- Invert NEW. Marking NEW and an appropriate line of the transition table results in inverting ($\emptyset \rightarrow 1$ and $1 \rightarrow \emptyset$) the so selected state. Naturally, Invert is invariant against twice repeated application.
- Change QUALIF. Marking QUALIF and an appropriate line the system is prepared for replacing the selected qualifier. This can be done in three different ways:
  . First, we can enter the new one from the keyboard.
  . Secondly, we can mark an existing qualifier in the scrolling list of the atoms.
  . Thirdly, we can mark (the line of) a qualifier already existing in the transition table.

Obviously, any atom (and therefore any local transition function) can be easily established and even edited during the input by these operations.

1.2. EDIT ATOM

After some initial dialogue (the simplest way to specify the atom to be edited is to mark it in the scrolling area) the system responds with the screen which is shown in Fig.1.4:

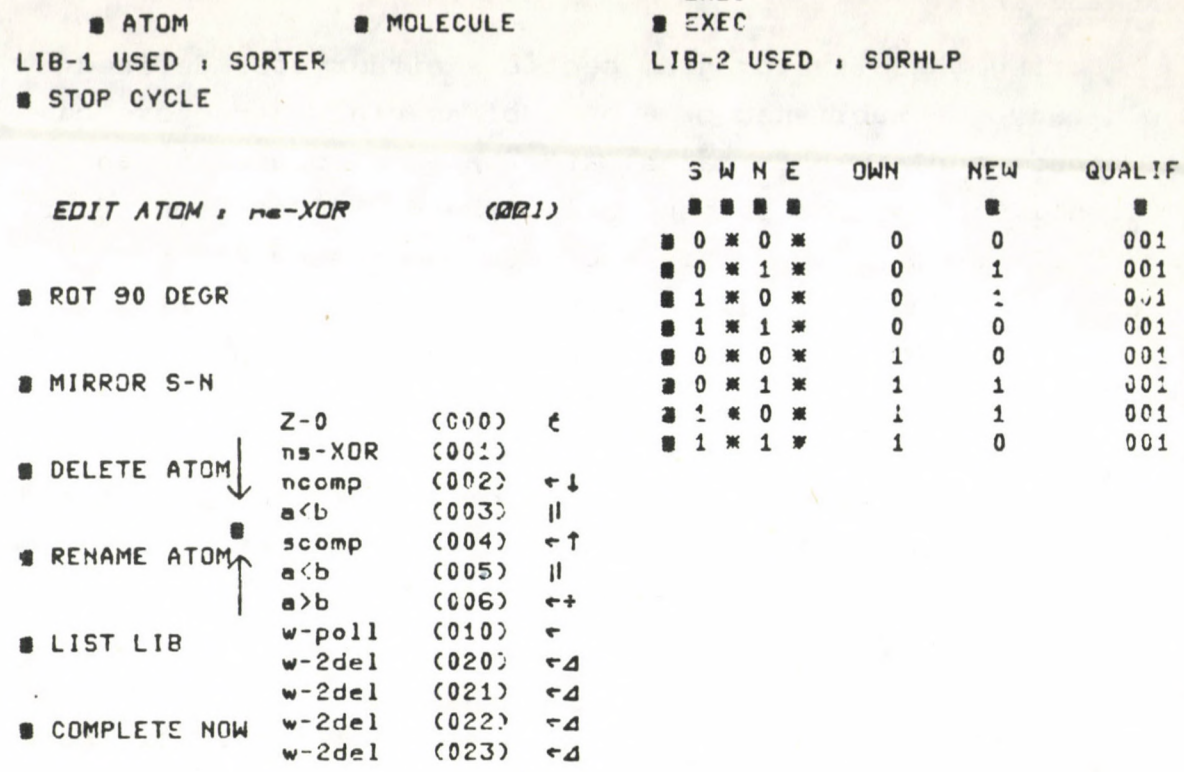


Fig.1.4

Naturally, all operations (Expand/Retract, Invert, Change) that have been described above are applicable to the displayed (here ns-XOR) atom, too. And, there are some further operations. Marking ROT 90 DEGR results in a cyclic shift of the neighbours' states: E→N→W→S→E. By MIRROR S-N the states E and W are exchanged. The effect of these operations is immediately applied and therefore visible. In connexion with the COPY ATOM feature (see Fig.1.Ø) it is easy to derive similar atoms from an already existing one. The functions DELETE ATOM and RENAME ATOM are applied and initiated, respectively, by marking the appropriate atom in the scrolling area; the complete dialogue (which can be held from keyboard, too) is not demonstrated.

FIGURE and CHAR

During these two program sections either user defined symbols can be established or a pair of special characters can be arranged, and, in both cases, afterwards allocated to an atom. Fig.1.5 shows the screen during the FIGURE section after we have defined a figure. This had been done using the two switches POINT and LINE. If we mark one point in the grid and afterwards mark POINT we have fixed an absolute point; if we afterwards mark LINE we draw a vector from the last marked point in the grid to this point. CLEAR retracts this step. If we mark ALLOCATE FIGURE the scroller for the figures are turned on. In Fig.1.5 the latter only contains the just defined figure which is referenced by the pointer. By marking ns-XOR the allocation takes place.

The allocation of figures consisting of a pair of special symbols is very similar to the above mentioned procedure.Fig. 1.6 shows the screen during this program section: By moving the bar under the list of special symbols we select a symbol, and, marking SET FIRST or SET SECOND we position it at the corresponding place of the pair to be defined. Repeated application of this procedure overrides previous set symbols.

■ ATOM ■ MOLECULE ■ EXEC

LIB-1 USED : SORTER LIB-2 USED : SORHLP

■ STOP CYCLE

FIGURE-CHANGE

■ LIST FIGURE-DIR

LISTING OF FIGURES

■ ALLOCATE FIGURE

■ DELETE

ORIGINAL SIZE :

■ COMPLETE DEFINITION

Z-0	(000)	ξ
ns-XOR	(001)	⊕
ncomp	(002)	←↓
a<b	(003)	‖
scomp	(004)	←↑
a<b	(005)	‖
a>b	(006)	←+
w-poll	(010)	←
w-2del	(020)	←◿
w-2del	(021)	←◿
w-2del	(022)	←◿
w-2del	(023)	←◿

■ LINE

■ POINT

■ CLEAR

Fig. 1.5

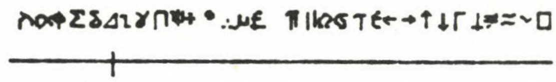
■ ATOM ■ MOLECULE ■ EXEC

LIB-1 USED : SORTER LIB-2 USED : SORHLP

■ STOP CYCLE

CHAR-CHANGE

■ ALLOCATE



■ SET FIRST

■ SET SECOND

FIGURE : ←Δ

Z-0	(000)	ϵ
ns-XOR	(001)	⊕
ncomp	(002)	←↓
a<b	(003)	||
scomp	(004)	←↑
a<b	(005)	||
a>b	(006)	←+
w-poll	(010)	←
w-2del	(020)	←Δ
w-2del	(021)	←Δ
w-2del	(022)	←Δ
w-2del	(023)	←Δ

Fig. 1.6

2. MOLECULES

Concerning the usual terminology in the theory of automata a molecule is, roughly speaking, corresponding to a (initial-) configuration, i.e. a function c_M: $\{\emptyset,\ldots,255\}\times\{\emptyset,\ldots,127\} \rightarrow O\times Q$. Thus, every automaton within the finite integer grid to be realized is at any time step defined by a state and a qualifier. At least the behaviour of an automaton is defined by the (default) qualifier $\emptyset$, i.e. the transition table which assigns to an automaton independently from the neighbours' states the next state $\emptyset$ and the next qualifier $\emptyset$. In the following we identify a molecule with the set of automata having not occupied the qualifier $\emptyset$, i.e. with its support
$\sup(c_M) = \{(i,j) \mid c_M((i,j)) \neq (o,\emptyset),\ o\in O,\ \emptyset\leq i\leq 255,\ \emptyset\leq j\leq 127\}$.
If we reference a molecule as a geometrical object, we reference it by the south-east corner of the minimal circumscribing rectangle of its support. The purpose of the MOLECULE phase is to create more complex molecules from less complex molecules and from atoms. Besides, after a different initial dialogue the switch EDIT MOLECULE results in the same screen as DEFINE MOLECULE. This screen is demonstrated in Fig.2.$\emptyset$.

The screen is essentially subdivided into four areas being of basic interest. The upper left area is occupied by the switches. Under that there is the scrolling area which contains - dependent on the marked switch - the directory of the so selected objects, i.e. ATOMS, MOLECULEs, or LABELs, respectively. In Fig.2.$\emptyset$ our well known atom directory is switched on. Thus, we are performing a SET ATOM operation. The upper right region contains a rectangle which represents the total grid of 128×256 automata to be simulated. Within this rectangle a window (in Fig.2.$\emptyset$ located in the left down corner) can be roughly positioned by marking appertaining positions at the light pen sensitive edges. The fine positioning of the window is done with the bars of the cross at the right side of this area. By marking one of the bars the window moves stepwise (incrementing/decrementing the x/y-coordinates) into the marked direction.

This process is controlled by the contents of the window which is displayed in the square under that. Within this area the automata under consideration are represented by their allocated figures. The window contains 2Ø×2Ø automata, and, the actions to be performed on these automata are controlled by a cursor which, again, is set by the light pen sensitive edges. As, for instance, the identification of an atom is in general not unique, we can mark this atom with the cursor. Marking afterwards NAME the name and the qualifier of this atom are displayed. Fig.2.Ø shows the result if we had marked the atom scomp with the cursor in position (2,Ø). If we afterwards mark FIX the allocation of this atom is definite in that sense that it can only be retracted by FORGET. A simple repeated structure can be generated in the following way: Firstly, we position the cursor and allocate an atom by marking it in the atom library. Afterwards we position the cursor again. By the first and the second position there is uniquely determined a rectangle with edges in parallel to those of the window. If now is marked REPEAT this rectangle is filled out with that atom (and can be FIXed!).

Molecules are allocated by a similar procedure: mark SET MOLECULE → select molecule (from directory) → mark FIX. But, in contrast to atoms a new situation may arise; namely, the (supports of) two molecules may superpose. In this case we have to make things definite. By AND the intersection of both supports is displayed and is viewed in the (appropriately positioned or moving) window. By marking OLD the atoms of the first loaded molecule are displayed, by marking OLD again those of the secondly loaded one are displayed, etc. The FIXed new molecule is built from the displayed constellation. The process whether the OLD or the new molecule should define the intersection is supported by turning the switch OR on or off in order to display the symmetric difference of both supports or not, dependent on the fact whether this information is helpful or confusing. Thus, by pairwise well defined combination we are able - starting from simple atoms and molecules - to establish very complex molecules. It should be mentioned that this process is

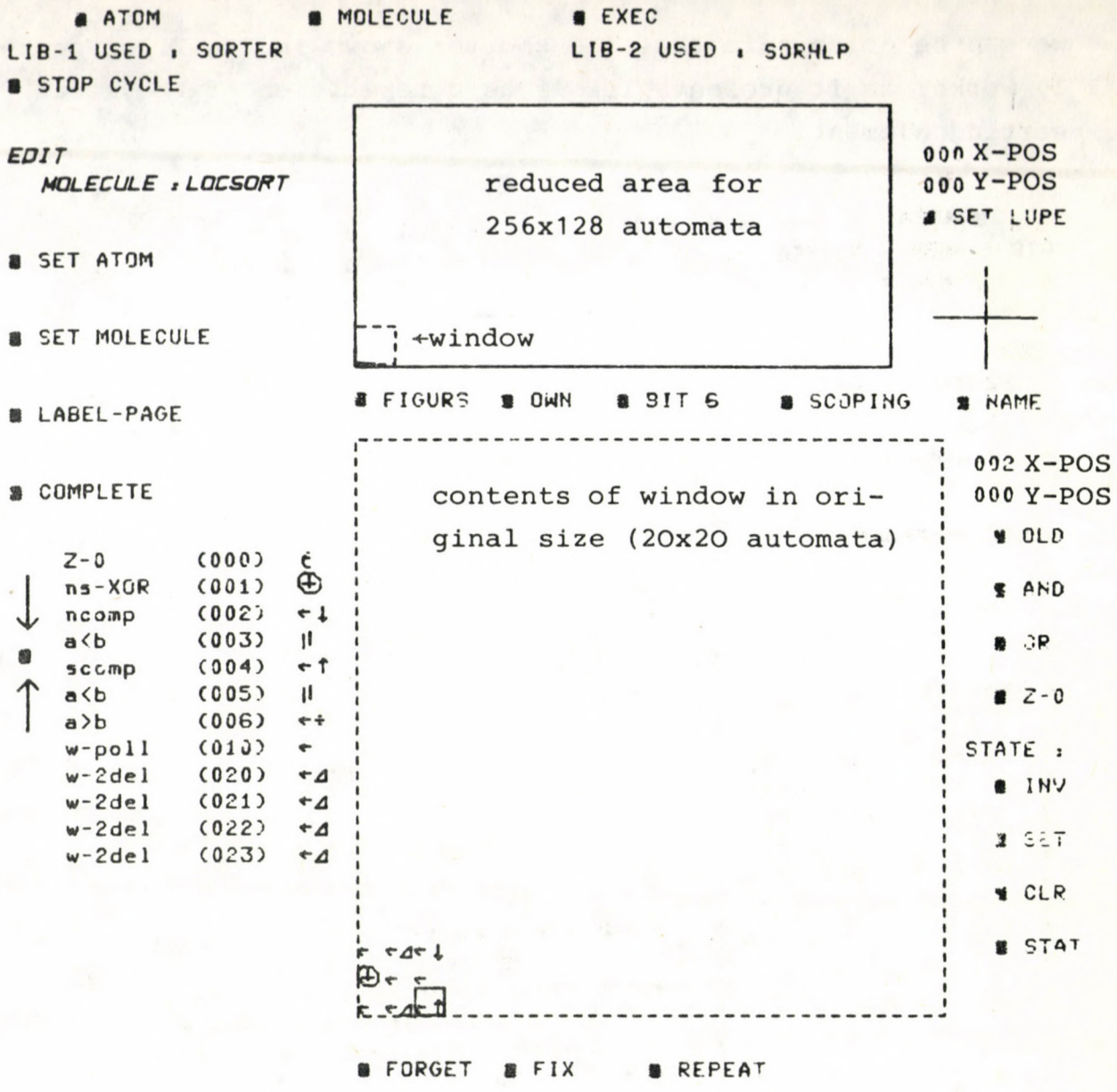


Fig.2.Ø

supported by the interactive execution language. It is always possible to interrupt the design process (COMPLETE), to switch over to the EXEC phase, and to test the so established configuration whether it really functions. E.g. Fig.2.Ø shows a molecule LOCSORT which is intended to work as a local sorter for binary strings. This simple structure is easy to verify by some commands. Afterwards we switch back to the MOLECULE phase and

we can be quite sure that the shorter shown in Fig.2.1 properly works; as it presents itself as a repetition of the local sorting element.

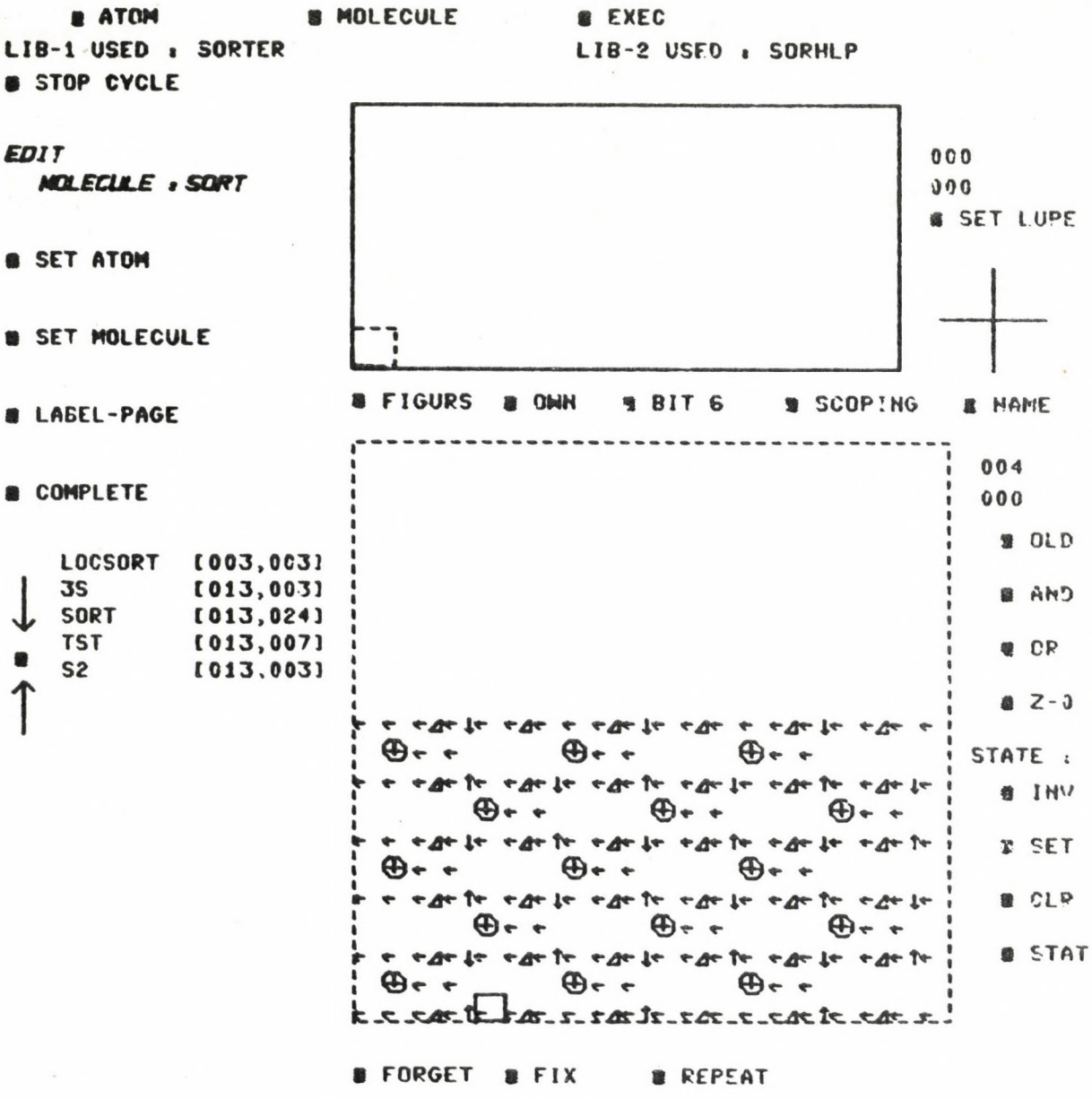


Fig.2.1

3. STATES AND DATA

A molecule (more precisely the whole simulation area) is preset with state ϕ. By marking STATE and afterwards SET the automaton which is referenced by the cursor indicates by a small bar that it is preset with state 1 if we FIX this action.Again the repeat function can be used. Let us assume that we had run through a sequence position cursor → STATE → <action> → position cursor → REPEAT → FIX. Concerning the rectangle determined by the two cursor positions the effect depends on the type of <action>. SET: all automata are set to state 1. CLR: all automata are reset to ϕ. INV: all states are inverted. Naturally, all functions (e.g. AND, OLD) apply to the states, too. With one exception: if two molecules superpose and STAT is marked then the atoms of the first loaded molecule remain unchanged, while the switches OLD, AND and OR affect the states of both molecules as described concerning qualifiers. Thus, we have the facility to carry over the whole 'pattern of states' from one molecule to another without altering the qualifiers of the latter.

One of the most powerful application fields for cellular processors might be seen in the implementation of pipeline structures. In these applications we understand a configuration as a (less or more complex) functional unit that transforms a steadily fed in stream of data to an output stream of data. For the simulation of such structures we must provide means to connect single automata to an input library as well as to an output library, repectively. Besides, the capability to influence a cellular automaton by inputs from the 'outer world' is of pure theoretical interest, too. Marking LABEL-PAGE results in the screen which is shown in Fig.2.2. A sequence mark DEF → position cursor → enter symbolic name for label from keyboard → mark GET NEW LABEL causes the so defined label to be appended to the label list. Fig.2.2 shows the situation after the last definition of label OUT5 with obsolute coordinates (ϕ12,ϕ23). If two molecules superpose, dependent on how we define the resulting structure, by GET NEW LAB labels of the second loaded molecule (with recalculated coordinates) are

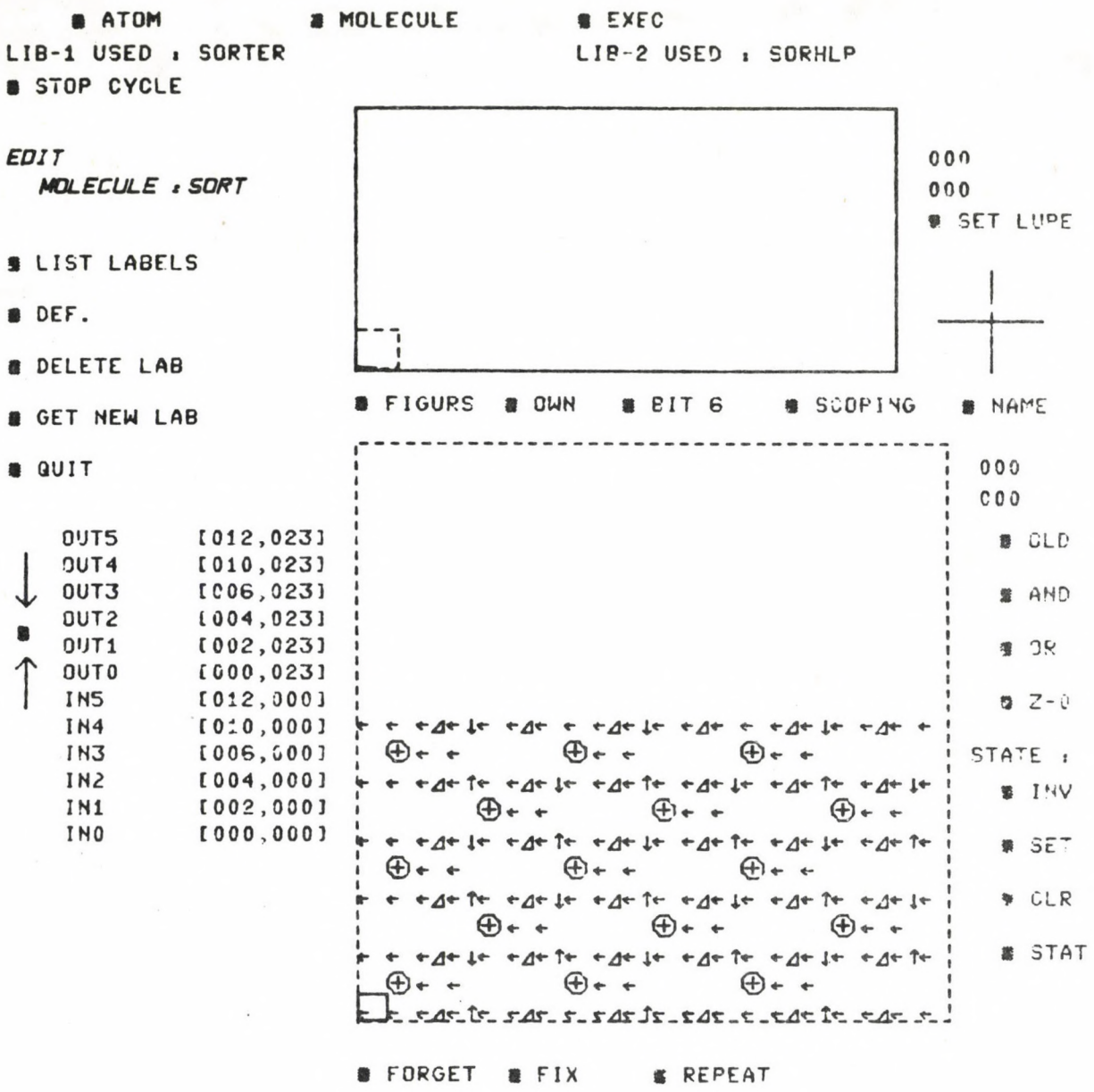

ATOM
MOLECULE
EXEC
LIB-1 USED : SORTER
LIB-2 USED : SORHLP
STOP CYCLE
EDIT
MOLECULE : SORT
LIST LABELS
DEF.
DELETE LAB
GET NEW LAB
QUIT
000
000
SET LUPE
FIGURS
OWN
BIT 6
SCOPING
NAME
000
000
OLD
AND
OR
Z-0
STATE :
INV
SET
CLR
STAT
OUT5 [012,023]
OUT4 [010,023]
OUT3 [006,023]
OUT2 [004,023]
OUT1 [002,023]
OUT0 [000,023]
IN5 [012,000]
IN4 [010,000]
IN3 [006,000]
IN2 [004,000]
IN1 [002,000]
IN0 [000,000]
FORGET
FIX
REPEAT

Fig.2.2.

appended to the label list of the resulting molecule. The dynamic supply of labelled automata is controlled by an EXEC-program (see 4.2.2). Fig.2.3 shows the initial screen from which all activities concerning molecules are started:

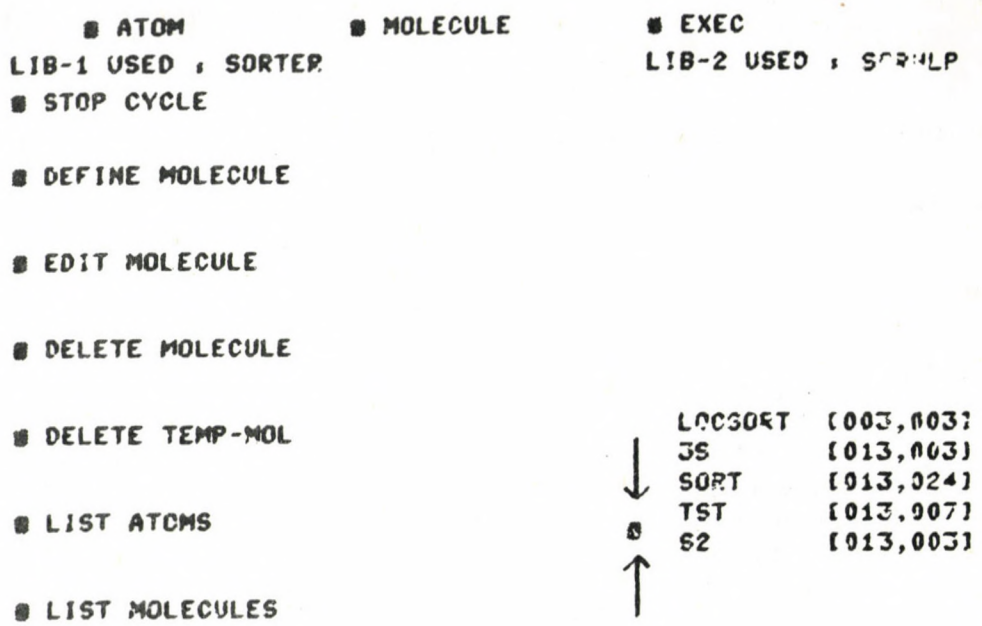


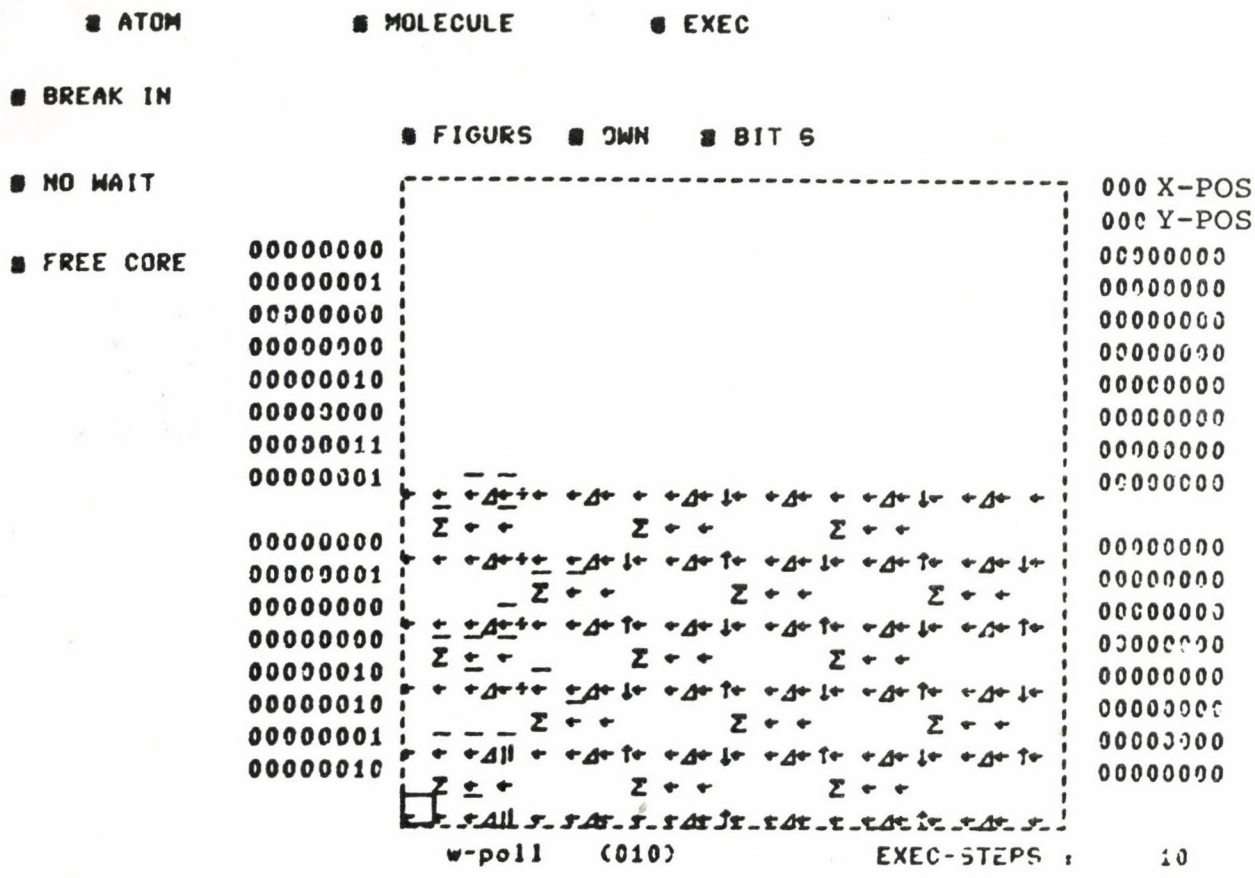
Fig.2.3

4. PROGRAM DEVELOPMENT AND PROGRAM EXECUTION

The real simulation (i.e. the transitions of defined configurations) is supervised by so-called EXEC-programs. Naturally the language must have more capabilities than a 'naive' advancing step by step through the transition table to perform a given number of global transitions. At least, this is true for the period where cellular algorithms are developed and tested out, and, for the representation of intermediate and result configurations. The basic properties of the control language

defines the states of the input automata. Analogously, a WRITE statement causes the states of the output automata to define the next word of the output file.

4.2.3 Concerning outputs, the screen plays a special role. By the statement WRC DISPLAY ('y-pos', 'x-pos'),the output is directed to the screen in a special performatted way:




```
(* ALL INPUT - LINE BY LINE - FROM KEYBOARD *)
-LIB'SORHLP.LIB' (* SELECT LIBRARY *)
-RDC'SORT'       (* SELECT CONFIGURATION *)
-EXTERNAL IN0,IN1,IN2,IN3,IN4,IN5 (* DECLARE INPUT CELLS *)
-EXTERNAL OUT0,OUT1,OUT2,OUT3,OUT4,OUT5 (* DECLARE OUTPUT CELLS *)
-RDD'SRT.DIN' (IN0,IN1,IN2,IN3,IN4,IN5) (* CONNECT INPUT STREAM *)
-WRD'RES.OUT' (OUT0,OUT1,OUT2,OUT3,OUT4,OUT5)
-(* THE OUTPUT STREAM IS NOW CONNECTED, TOO *)
-(* THE NEXT SEQUENCE CONTROLLS THE REAL SIMULATION *)
-LOOP 100 IN WRC DISPLAY; WAIT;READ(*INPUT*);EXEC;WRITE(*OUTPUT*) POOL□
```

Fig. 4.Ø

are listed and shortly discussed below.

4.1 The control language is block structured, where 'block' is defined recursively.

4.2 Configurations can be input from any file-structured RT-11 device. Outputs can be directed to all RT-11 devices:

4.2.1 Concerning the output of configurations it can be dynamically specified in what subconfiguration of the actual configuration we are interested in. Furtheron,this information can be formatted and therefore reduced to the amount we are really interested in. E.g. we can specify that only the first three characters of the name, the state, and bits seven and three of the qualifier of the automata should be printed.

4.2.2 In order to simulate pipeline-configurations, the following features are provided:

- Up to sixteen arbitrary automata of a configuration can be selected in each case to work as input and output automata. This can be done during the molecule phase by lightpen interaction, and, in this case they must be declared as EXTERNALs in the EXEC program. The other way is to declare labels in an EXEC program itself explicitly as variables of the type LABEL, or, to use them there implicitly as constants. In this case we must have knowledge (even if we compute them by 'label-expressions') of absolute coordinates.

- By declaration of a data input file (RDD 'filename') and a data output file (WRD 'filename') the selected automata are connected to these inputs and outputs. More precisely, the up to sixteen input automata are identified with the corresponding bits of a sixteen bit input buffer which is connected to the input file. The output automata are treated analogously.

- Whenever in the EXEC program a READ statement is encountered, the next sixteen bit word of the input file

- The actual configuration is visible within a window of 20×20 automata starting from ('y-pos','x-pos') defining the leftmost down corner. Changing to BREAK-IN context (which is explained later) we can modify ('y-pos', 'x-pos') and move the window to make visible any subconfiguration of interest.
- At the left and the right margin of the window the eight most significant sixteen bit words of the input stream and the output stream, respectively, are visible. This does not depend on whether the input or output automata are visible in the window.
- Some further useful functions are:
  . State (OWN) and bit 6 of all automata can be made visible.
  . Positioning the cursor to any automaton within the window its name and qualifier are indicated.
  . The number of EXEC steps performed until now is displayed.
- Concerning program control a BREAK IN can be performed by lightpen interaction and a WAIT statement of the actual EXEC program can be suspended as long as NO WAIT is sensified.

4.3 Program execution (and therefore the real simulation) can be controlled by the EXEC program itself in various ways at different levels.

4.3.1 Almost the simplest way to do this is by loops of type LOOP 'expression' IN 'block' POOL. Initially 'expression' is evaluated to an integer determining how often 'block' is to be performed (if 'block' does not contain further program modifications!). In simple applications 'block' is the following type:
READ; WRC DISPLAY; WAIT; WRITE; EXEC.
This sequence is executed as follows:

- First, the next 16 bit word of the connected input

data file defines the states of the input automata.

- Next, the window mentioned in 4.2.3 (fixed at the default position (0,0)) is displayed.
- Now the program waits for the permission to continue. WAIT is suspended either by an arbitrary keyboard action (except ^C) or (in this context) by sensifying NO WAIT at the screen.
- The output buffer is written to the connected output data file.
- By EXEC one global transition is performed. Besides, the states of the output automata define the new contents of the output buffer.

4.3.2 A more powerful construction is the if-then-else clause: IF 'condition' THEN 'block' / ELSE 'block' / FI . As 'block' can contain a GOTO statement the simulation can be efficiently controlled if we keep in mind the various ways 'condition' can look like. E.g. it can be a logical AND or OR over the states of a predetermined set of automata and values gained in this way can be combined by boolean operators to form new ones. Besides, statements like TRACE and TORUS are considered as boolean variables, i.e. they may be dynamically switched on.

4.3.3 A further useful statement is the TRACE statement. It can be disabled and enabled dynamically by any boolean expression and is used to produce runtime information to the screen like programlabels, the present value of integer variables, the contents of the input and output buffers, respectively, etc.

4.4 Another way to control the simulation is provided by the implementation of a BREAK IN concept. A BREAK IN causes the current EXEC program to halt due to one of the following conditions:

- from program, by a BREAK statement,
- from keyboard, by entering ^C,

- by lightpen, sensifying BREAK IN,
- from switchregister.

In BREAK IN context program control is transferred to keyboard. The interrupted program can be continued (CONTINUE) or abandoned (QUIT), or, the values of some variables can be examined or reassigned etc. But the most powerful feature of the language in conjunction with this concept are:

- Any program can be executed now, without destroying the interrupted program or its data.
- Programs can be entered from keyboard, line by line, too, and executed.
- A BREAK IN will also occur in case of a non-fatal error which can be corrected now.

In any case afterwards (entering continue) the interrupted program can be continued at the point where it had been stopped.

It should be mentioned that most of the features of the EXEC language have not been implemented to support the real simulation, but to support the development of cellular algorithms. Once, a cellular algorithm is developed, the EXEC program will be very simply structured.

Besides, calling back Fig.Ø.4 in our mind, the EXEC language has to cover functions of the CCU (Cellular Control Unit) and of the Host computer. It depends on the specific realization how autonomous the CCU is tailored and how close the connexion between these two components is. And, from these design principles results in which component these functions are implemented. Some examples for such functions are: initialization of the cellular structure, initial loading (and possibly time dependent swaping) of the transition table, initial loading of the single cells with the appertaining states and qualifiers, dynamic supply with data and output of (intermediate) results. Obviously, the simulator should not depend on a specific reali-

zation, and has to provide these functions at a higher level.

5. CONCLUSIONS AND OUTVIEWS

At the beginning of this paper we gave motivations for our attempt to add a further simulator for cellular structures to the broad spectrum of already existing ones. The simulation software is basically characterized by three features.

- There are simulated MEALY-type cells with a fixed neighbourhood and fixed output- and state-alphabets. From this results an easy interactive development of transition tables which, again, is supported by the qualifier concept (due to LEGENDI).
- The design of configurations - and thus of cellular algorithms - is facilitated by modular and orthogonal construction principles as well as by the interactive (in general not depending on absolute coordinates) manipulation of subconfigurations in the plane.
- The real simulation runs are controlled by a high level programming language. The run time system is capable to enter into dialogs with the programmer and provides break in and trace concepts.

At the present time we implement the time varying case; i.e. the dynamic, time-dependent swaping of the transition tables. Moreover, we are in the implementation phase of a higher cellular programming language which, roughly speaking, allows the algorithmic generation of configurations.

ACKNOWLEDGEMENTS: In the first place I would like to thank Mr. V. Henkel who has done the hard labour of the implementation. At any time he was an imaginitive and cooperative partner. Many thanks to Prof. R. Vollmar and Dr. T. Legendi that encouraged me to this work. And, last not least, many thanks to my dear colleague Mr. J. G. Pecht to whom I am indebted for many stimulating discussions.

REFERENCES:

Baker R. W., Herman, G. T.: CELIA - a cellular linear array simulator, Proceedings of the Fourth Conference on Applications of Simulation, 64-73, New York, 1970.

Baker R. W., Herman, G. T.: Simulation of organisms using a developmental model, Part 1: Basic description, International Journal of Bio-Medical Computing 3, 201-215, 1972.

Brede, H. J., Szwerinski, H.: SICELA-I - Benutzerhandbuch, Stand: 1. Sept. 1979, internal doc., Braunschweig, 1979.

Brender, R. F.: A Programming System for the Simulation of Cellular Spaces, Ph.D.diss., Univ.of Michigan, 1969.

Brender, R. F., Frantz, D. R.: The CESSL Programming Language, Technical Report, No. 012520-5-T. Univ. of Michigan, 1971.

Herman, G. T., Liu, W. H.: The daughter of CELIA, the French flag and the firing squad, Simulation 21, 33-42, 1973.

Legendi, T.: Cellprocessors in computer architecture, Computational Linguistics and Computer Languages XI, 147-167, 1976.

Legendi, T.: INTERCELLAS - an interactive cellular space simulation language, Acta Cybernetica 3, 261-267, 1977.

Legendi, T.: TRANSCELL - a cellular automata transition function definition and minimization language for cellular microprogramming, Computational Linguistics and Computer Languages XII, 55-62, 1978.

Legendi, T., Dióslaki, F.: Benutzerhandbuch für SICELA-H, internal doc., Szeged, 1979.

Pecht, J. G.: Erfahrungen mit SIBICA einem Programmsystem zum interaktiven Studium von 2-dimensionalen binären zellularen Netzen, in: GOLZE, U., VOLLMAR, R. (Hrgs.): Beiträge zur Theorie der Polyautomaten, Informatik-Berichte, Nr.7703, 77-86, Braunschweig, 1977.

SILICEA, Benutzerhandbuch, internal doc., Braunschweig, 1979.

Smith III, A. R.: Introduction to and Survey of Polyautomata Theory, in: LINDENMAYER, A., ROZENBERG, G. (eds.): Automata, Languages, Development, 405-422, Amsterdam, New York, Oxford, 1976.

Vollmar, R.: Über einen Interpretierer zur Simulation zellularer Automaten, Angewandte Informatik 6, 249-256, 1973.

Vollmar, R.: Cellular Spaces and Parallel Algorithms - An Introductory Survey -, in: FEILMEIER, M. (ed.): Parallel Computers - Parallel Mathematics, 49-58, IMACS, Amsterdam, New York, Oxford, 1977.

Zuse, K.: Rechnender Raum, Vieweg, Braunschweig, 1969.

Computational Linguistics and Computer Languages Vol. XIV. 1980.

AN EXPERIMENTAL LANGUAGE ARCHITECTURE DESIGN AND IMPLEMENTATION*

Gábor SIMOR
Institute for Coordination of Computer Techniques
Budapest, Hungary

ABSTRACT

The subject of the paper is an experimental language architecture, which is aimed at optimized machine code size and execution time for programs written in COBOL. During design and implementation of this architecture many programming tools - generally applicable in high-level language architecture developments - will be used for evaluation and realization of different alternative solutions. "One-to-one correspondence property", locality-based field size minimization, frequency-based encoding are used as optimization methods. Most of these optimization measures promote autonomous phase processors to be relevant for overlapped instruction execution, while the price/performance ratio also depends on the complexity of the actual operations included in the instructions. Different instruction formats and representation codes are proposed to be evaluated and implemented by programmed tools which transform the formal description into encoding and decoding modules, FPLA programs. Development flexibility is promoted by using the abstract data type concept when structuring the machine language interpreter program. A technological language is used for the construction of the interpreter to be implemented by hardware firmware tools, applying more levels of interpretation.

* This work has been sponsored by the State Office of Technical Development

1. INTRODUCTION

1.1 Machine Language Optimization Methods

The extensive use of microprogramming technics for implementation of machine architectures and the demand on dedicated language processors in distributed systems has promoted numerous research works which are aimed at the elaboration of a methodology for defining optimal "execution-oriented"* instruction sets /providing minimal machine code-size and execution time for programs written in a high-level language:[1], [2]. These methods propose "1: correspondence" between operations, data objects of the source language and those of the machine language; locality-based minimization of instruction field sizes [1]; frequency-based compact encoding [2].

1.2 Flexibility of the Development Process

Using the above mentioned design methods, various concrete instruction sets may be defined and various implementation solutions may be applied. For the final design decision, the trade-off between several environment characteristics should be investigated taking into consideration the complexity of operations and other properties of the source language, the usage statistics of various language elements, the cost of certain microarchitecture features /e.g. see Chapt. 3 for the discussion of the relevance of autonomous phase processors for overlapped instruction execution, or Chapt. 2 for the illustration of a possible use of high-speed local stores for "locality descriptions"/. Therefore, simulators are needed for the evaluation of different alternative architectures. The growing number of various problem-oriented high-level languages - needing architectural support - and the great variety of al-

*Architecture support considerations for the program development and compilation process are not discussed in this paper

ternative architectures to be evaluated* demand, that simulators should be created and modified in a very flexible way. On the other hand, it is desireable to avoid double coding of the instruction set interpreter /once for the simulation and again for the implementation/. Therefore the simulator has to be used as a "development-interpreter", which initially serves for the verification and evaluation of the instruction set, it is also the object of a subsequent performance tuning for the selection of the interpreter modules to be implemented directly by HW-FW means or to be executed parallelly, and finally the whole simulator is turned into the implementation using powerful translators, program tools for generating and synthetizing. This approach also promotes the application of standard implementation elements to different source languages and architecture solutions.

In connection with the above described requirements, numerous researches have also been carried out: formal architecture descriptions have been used for generating evaluator-simulators [3], microprogram sequences [4], and even for the synthesis of hardware logical design [5].

1.3 An Experimental Project

In our experimental language architecture project we intended to apply a development methodology with the aim of both, previously discussed research directions /i.e. machine language optimization methods and development flexibility/. The steps of this kind of development process are illustrated in Fig.1 and discussed in certain perspectives in the following chapters.

1.3.1 A COBOL-Oriented Machine Language for Experimental Purposes

*Works described in [1], [2] have pointed out, that an appropriate machine language may increase the main efficiency parameters by a factor of 3÷5.

A COBOL subset has been chosen as the source high--level language to be supported in a first experiment, first of all for its simplicity and wide-spread use. The architecture levele data structures and an instruction set skeleton has been specified for the COBOL machine, applying the "Canonic Interpretation Form" definition rules [1]. The proposed architecture has explicit access to the tables for the description of data objects, paragraphs, addresses; contains stacks for the expression evaluation and for the nested paragraph calls; performs dynamic /locality-dependent/ instruction field selection and decoding functions. More details are presented in Chapt.2.

1.3.2 Development Tools

At the "development-interpreter" creation the flexibility and the automation of the development are promoted in the following ways:

a., The interpreter has been modularized by the abstract data type concept, therefore generally applicable optimized architecture component types are being defined for various target languages and implementation environments /see 4.1/;

b., Certain modules /which are to be frequently replaced during the development: e.g. format-interpretation, representation-decoding/ are intended to be generated /both for simulation and HW-FW implementation purposes/ from a formal, non-procedural specification of the functions, easily executable by logical arrays /see 4.2/;

c., A procedure-oriented language /CDL2/ is used - as a "Technological High-Level Language" for the development of the interpreter /see Chapt.5/. Hence:

- an efficient programming technology may be applied;
- the prevailing part of the CDL2 code of the interpreter may be used also for the generation of an implementation without significant performance losses, because of the following properties of the CDL2:
  - the language itself may be directly interpreted in a

relatively simple way /demanding limited microprogram storage capacity/;

- it provides efficient performance tuning possibilities, and the "performance-critical" functions may be easily substituted by microprogram sequences;

d., The function distribution among the phase processors was the base for the structure of the interpreter /see Chapt.3/, therefore:

- the load-balance of the phase processors may be previously estimated;
- the communication procedures between the phase processors may be experimentally verified.

2. OPTIMAL INSTRUCTION SET DESIGN

A design methodology - called "Canonic Interpretation Form" /CIF/ - has been published in [1] based upon a criteria set, independent of language usage statistics and hardware implementation /Phase 5 in Fig.1/. Instruction sets designed in this way may be characterized by the "1:1 property" and "locality-based" field size minimization.

Application of "1:1 property" of the CIF in our COBOL architecture* may be illustrated by the COBOL source statement PERFORM THRU UNTIL, implemented in a single CIF machine instruction: PTU /see Fig.2.a/. Executing this instruction the paragraph /a procedure without explicit parameters/ p_1 will be called, and the subsequent paragraphs performed up to the paragraph p_2. After the execution of the paragraph p_2 control is returned to the calling paragraph and this cycle is repeated depending on the logical expression evaluation performed by the code sequence between "addr" and the PTU instruction. The top element of the paragraph stack identifies the currently exe-

*The CIF was originally applied in a FORTRAN emulation research

cuted paragraph, the element under the top identifies the last saved paragraph /p_i/, the last paragraph to be subsequently performed /p_j/, the saved address in paragraph p_i /$addr_i$/, and the number of PERFORM executions /N/ for the case of the TIMES option in the source PERFORM statement. The implementation of source operations with similar complexity would demand 5÷2o machine instructions on a conventional, general-purpose architecture.

The "1:1 property" pertains not only to the relation between the number of operations in a source statement and the number of corresponding machine instructions, but also to the relation between the number of data objects to be distinguished in a source statement and the number of operand fields in the corresponding machine instructions /see Fig.2.b/. For this purpose additional format fields are used in the machine instructions /e.g. $F_{2,1}$, $F_{2,5}$/, which define the role of the operand fields. Format codes /e.g. $F_{2,6}$, $F_{2,12}$/ may also be used for the selection of implicit operands from one of the two top elements /"T" and "U"/ of the expression stack. Although in the case of source statements with more sophisticated syntax /e.g. Fig.2.c/, keeping "1:1 property" would suppose more complex, syntax-oriented formats to be used /than applied in [1]/*. The "1:1 property" corresponding to the data objects also means that operands in the same machine instruction may have different descriptions /e.g. different positions of the decimal point/, the conversion functions required will be included in the execution of the same instruction, too.

Solution for the "locality-based" field size minimization is also illustrated in Fig.2. Operand fields will not identify immediately the operand address in the main memory, but merely contain the sequence number of the object description inside the actual program locality /the current paragraph/. The orig-

*a more detailed analysis see in [6].

inal concept in [1] - aimed only at the minimization of data reference field sizes - has been extended for the case of many--sorted operand decoding, when different descriptions are provided for data objects /DDT/, address pointer objects /IPAT/ and paragraph objects /PDT/. Paragraph descriptions are valid for the whole program, address pointers only inside a paragraph, data objects may be local or global and distinguished by the operand field code itself /namely by its first bit/. On the other hand in [1] operations were also coded in fields with "locality dependent", minimized size; in our architecture this solution has been omitted due to the relatively small number of different operations in COBOL.

A "frequency-based" encoding method has been published in [2] /see Fig.5.a and Phase 7 in Fig.1/. Applying this method we consider that e.g. statements IF, GOTO, MOVE make up statically more than 6o% of all statements [7], therefore quite short bit patterns are used for the corresponding operation codes in machine instructions. Certain operands, e.g. "ZERO" and "SPACE" figurative constants, numeric literal "1", the most frequently referred paragraphs /e.g. those for error message output/ and variables for a given paragraph, have also a relative high frequency, so an "integrated" encoding method [2] is expedient to be used for them without explicitly separated operand and operation and/or format fields in these cases.

3. PHASE PROCESSORS FOR OVERLAPPED INSTRUCTION EXECUTION

The CIF approach [1] is directed by "absolute" optimality criteria /minimal length of machine code, minimal number of main memory references during the execution/ supposing "ideal" hardware implementation environment /e.g. large capacity of local storage with high-speed access, ideally efficient "field--extracting" capabilities/. In real cases compromises are chosen, both, in the design of the instruction set and in that one of the implementation structure. The variable instruction and instruction field sizes, the indirect accesses to the oper-

and objects in the previously described instruction set demand a more sophisticated instruction fetching, decoding and operand selection functions during its interpretation. These two kinds of functions are included in those instruction execution phases which are often overlapped with each other and also with the actual operation execution phase, even in the case of conventional, general-purpose architectures. Realizing the "1:1 property" of CIF means that the complexity of the functions of the execution also increases, therefore extensions in the previous two phases may be "covered" in time. Furthermore, the larger the complexity is for all phases, the more efficiently distinct autonomous processors can be used for these phases parallelly working either on both of the microinstruction level and machine instruction level in synchronous systems /e.g. EMMY [1], or UNIVAC 1100/80/, or on the machine instruction level in asynchronous systems /e.g. FCPU [8], COMBAT [9]. The latter solution may provide a better load-balance between the phase processors but is more expensive due to the synchronization tools.

The relevance of the phase processors is illustrated in Fig.3. In the case of low operation complexity, the relatively sophisticated instruction decoding and operand selection functions increase execution time. In the case of higher operation, complexity optimizations in the previous two phases have less significance. A final design decision on the application of phase processors has to be based upon simulation measurements. In order to enable the COBOL architecture simulator for this evaluation and the possible implementation on distinct phase processors, it is structured according to the functions of the three phases. The structure of the interpreter and the "inter-phase" procedures - which form the base for the communication protocol between the phase processors - are roughly illustrated in Fig.4.

According to this structure the instruction fetching and decoding processor sends the operation code to the execution

processor after the decoding of each instruction. Similarly operand names /which are relative to the current locality/ and their sort /which is the base for the selection of the relevant object description information/ are sent to the operand processor. When an instruction for locality change occurs /e.g. a PERFORM for a paragraph call/, the name of the new locality is passed to the operand processor in order to determine the new area /corresponding to the new locality/ in the object description tables. Otherwise the instructions for locality change are performed by the first phase processors itself /change of the locality description information, update of the locality stack, definition of a new instruction address in the code memory/. In the case of instructions for conditional locality change, the change is performed /the probable alternative/. The conditional branches /e.g. ON SIZE ERROR condition/ inside a locality and the erroneous locality changes are recognized by the operation execution processor. In this case a branch command is sent by the 3rd processor to the 1st one. The 1st processor responses by stop commands in order to prevent processing of information being active, or waiting in the FIFO-queues of the two other processors /except the store of the result of the last successfully executed instruction/. The communication between the 2nd and the 3rd processor is needed for sending data descriptions /or actual values in the case of short data lengths/.

4. CHANGE OF THE IMPLEMENTATION MODULES

4.1 Modularization by Abstract Data Types

Development flexibility has been promoted by using the abstract data type concept in the specification of the notions, realized by the modules shown in Fig.4. Here, abstractions are used for the development of modules with a higher degree of independence, both, on the implementation and on the language particularities. A data type /e.g. instruction, format, queue, selector, etc./ is specified by the effect of the procedures

allowed for access to an object with given type. For example the format type is specified and implemented by two procedures shown in Fig.5. The specified procedures are implemented by subsequent, alternative, cyclic or recursive invocation of lower level procedures /e.g. procedures defined by Code Memory module for instruction code fetching, Locality Description module for operand field extractions, Code Representation module for decoding format and operation fields/.

4.2 Automatic Generation of Certain Modules

As has already been mentioned, the choice of used instruction formats and code representation has a significant impact on the instruction set efficiency, and might be finally justified by simulation measurements, only. Therefore, a unique notation has to be used for the formal definition of the format and the operation code representation /Fig.6.b/ which might be applied as source for the automatic generation of code representation decoding module /for the interpreter, Fig.6.c/ and encoding module /for the code generation in compilers, Fig.6.d/. These modules - written in a procedure-oriented higher level language, suitable for the well structured, portable programming of compilers and simulators - provide a very low efficiency /a distinct procedure call sequence for each case in the decoding rule/. Therefore translators are also relevant for the generation of assembly instruction sequences and even logic formulas as symbolic FPLA programs /Fig.6.e/ for the more efficient lower level implementation based upon the same formal specification /Phase lo in Fig.l/. Similar tools are relevant also for format handling.

5. USE OF A TECHNOLOGICAL HIGH LEVEL LANGUAGE /THLL/

As it has been shown by the illustrations referred above, a procedure-oriented, well-structured high level language is suitable for writing the architecture simulators. In order to

promote easiness of implementation and possibilities of application of standard hardware-firmware architecture implementation modules, it would be advantageous to use the same simulator also for implementation; minimizing in this way the need for algorithm reformulations. For this purpose we can make use of the below-described properties of a relevant language, applied as a THLL to language-oriented architecture development. We have decided to use the CDL2 language [1o] as a THLL, since it was the only available one bearing these properties.

5.1 Compactness of the THLL

The main disadvantage of the higher level languages is usually the waste of the microprogram storage when translating the HLL programs into microcode*. The most obvious way of writing compact programs is the intensive use of subroutines /procedures/. In a procedure oriented language like CDL2, the only /built-in/ "available" operation is the procedure invocation, so control structure of the language itself provides a very compact code. The implementations of CDL2 [1o] contain many of automated optimization facilities /eliminating duplicated sequences, simplified translation of nonrecursive procedures, use of "closed" procedure definitions instead of open ones, etc./. Furthermore, instead of translation, an interpretation of the "CIF"-form of the THLL may be used - at least for the prevailing part of the code.

5.2 Performance Tuning Possibilities of the THLL

The use of THLL interpretation instead of translation causes losses in execution time; therefore the "performance--critical" parts of the target HLL architecture interpreter

* Although - as stated in [13] - the assembly code generated from CDL programs written by "skilful" programmers, do not exceed those ones, originally written in assembler, by more than 2o-3o %.

should not be executed by this kind of "double interpretation". Experimental measurements have stated [13] that more than 8o% of the execution time is spent by running merely 3o% of the program code. The hierarchical procedure invocation structure of the THLL enables /by incorporating frequency and time measurements into the procedure invocation and return functions/ a very accurate and easy separation of the heavily used part from the rest of the code.

5.3 "Open-ended" Language as a THLL

The THLL itself does not contain actual operations. At the bottom of the procedure hierarchy of a program such primitives are called which are not refined further by invocation of lower level procedures, but are defined by the programmer in the "host" language /i.e. in the microcode, in the case of microprogrammed interpreters/. A procedure is decided to be directly microprogrammed /or hardware implemented/ primitive, either if its function is very close to the microarchitecture, or when the above mentioned performance-tuning analysis qualifies it to be heavily used /e.g. instruction format decoding/. The latter case corresponds to the "traditional" problem-oriented instruction set design method: replacement of frequent instruction sequences by one new instruction [14] , although, here it is supported by the procedure-oriented structure and the performance tuning facilities of the THLL. This method is aimed at the realization of the optimal proportion between translation and levels of interpretation in the case of a microprogrammable machine /Fig.7,8 and [15] /. As the functions of the primitives are relatively simple and implementable by short microprogram sequences, the "machine state resolution--based" microprogram synthesis methods /[4]; Phase lo in Fig. 1/, assuming relevant specification of these functions, seem to be of practical applicability.

5.4 Architectural Supportability of the THLL

A THLL may be supported efficiently by hardware-firm-

ware tools, in two respects.

A HLL may be supported efficiently if "performance-critical" built-in functions are unambiguously definable. In the case of the CDL2 the only functions to be supported, may be connected with the procedure invocation: save and restore the actual control point, load and store the actual parameters, select the called procedure, set and restore the procedure stack pointer, etc.

Optimization methods /discussed in Chapt. 2/ are mainly based upon separability of localities inside the program and therefore they may be used efficiently for interpretation of procedure-oriented languages as they provide as narrow localities, as possible by procedures. Furthermore, additional optimization may be yielded here by making use of the fact that the only THLL program to be supported is the interpreter of the canonic target architecture. Thus, e.g. the minimal size or integrated encoding for procedure identifiers, globally and locally variable, parameter identifiers may be strictly based upon simple deterministic measurement data, instead of probabilistic ones, which would require leaving a large tolerance, too.

6. STAGE OF DEVELOPMENT

An instruction set "skeleton" has been specified for COBOL, using CIF-rules /[6], Phase 5 in Fig.1/. The machine code size of a benchmark COBOL program has been compared, manually coding with the proposed instruction set and using a COBOL compiler for the SIEMENS BS 2ooo operating system /4/. A 1:5 proportion has been reached /for the procedure division of the COBOL program only/. Elementary data structures of the proposed COBOL-architecture have been specified as abstract data types, using the syntax of the ALPHARD language [11]. On the base of this description, a simulator has been written[16] for a subset of the proposed instructions in a logic-based lan-

guage /PROLOG: [12] with a resolution-based execution mechanism. Currently a COBOL - COBOL CIF compiler /6/ and a COBOL CIF is under development using the CDL2 language. A bit-slice microprocessor-based implementation is planned.

The applicability of the PROLOG language has also been explored for instruction set design / 5, 7 : [17]/ and benchmark synthesis purposes /3, [18] /.

REFERENCES

[1] Hoevel,L.W., Flynn,M.J.: "The Structure of Directly Executed Languages: A New Theory of Interpretiv System Design", DSL Technical Report 13o, Stanford University, March 1977.

[2] Tannenbaum,A.S.: "Implication of Structured Programming for Machine Architecture", Comm. of ACM, March 1978, p. 237-246.

[3] Barbacci, Burr, Fuller, Sieworek: "Evaluation of Alternative Computer Architectures", Carnegie-Mellon University, 1977.

[4] Dávid,G.: "Structured Automated Design of Microprograms", EUROMICRO '78, München, p.241-245.

[5] Parker,A.C., Hafer,L.: "The Application of a Hardware Descriptive Language for Design Automation" Information Technology, North Holland P.C., 1978 p. 349-355.

[6] Simor,G.: "An Instruction Set Design Approach for HLL-Oriented Microprogrammed Machines" SZKI, Budapest, 1978.

[7] Chevance,R.J., Heidet,T.: "Static Profile and Dynamic Behaviour of COBOL Programs". SIGPLAN Notices, March, 1978, p.44-57.

[8] Lawson,H.W.: "The DATASAAB Flexible Central Processing Unit /FCPU/, Background, Concepts and Basic Design", DATASAAB Report, 1972.

[9] Yamamoto,M., Hakozaki,K.: "A COBOL-Oriented High Level Language Machine", EUROMICRO

[1o] Jahn,S.: "CDL2 im BS2ooo", SIEMENS DV-Technische Berichtung FWO 121, November 1976.

[11] Wulf,W.A., London,R.L., Show,M.: "Abstraction and Verification in ALPHARD:Introduction to Language and Methodology", Technical Report Carnegie-Mellon University, Pittsburgh, 1976.

[12] Warren,D., Pereira,L.M., Pereira,F.: "Prolog - the Language and its Implementation", SIGPLAN Notices Vol. 12., no.8. 1977.

[13] Bolgár,G.: "Optimization and Measurement of the Performance of a Compiler, Written in a High Level Language", Információ és Elektronika 1977/5. p.286-290 /in Hungarian/.

[14] Abd-alla,A.M., Karlgaard,D.C.: "Heuristic Synthesis of Microprogrammed Computer Architecture", IEEE Trans. on Computers, Vol.C-23, No.8., Aug.1974, p.802-807.

[15] Dömölki,B., Rajki,P.: "Microprogram Implementation of High Level Languages", INFELOR 1973, Budapest-Székesfehérvár.

[16] Kiss,V., Simor,G.: "Evaluator-Simulator for the Design and Experimental Verification of a High Level Language Oriented Architecture", SZKI, Budapest, 1979 /in Hungarian/.

[17] Simor,G.: "Illustrations for Certain Phases of Computer Aided Computer Architecture Design", SZKI, Budapest 1978.

[18] Kiss,V., Simor,G.: "The Preliminar Specification of an Architecture Design Environment and Its Programming Aids", SZKI, Budapest, 1978. /in Hungarian/.

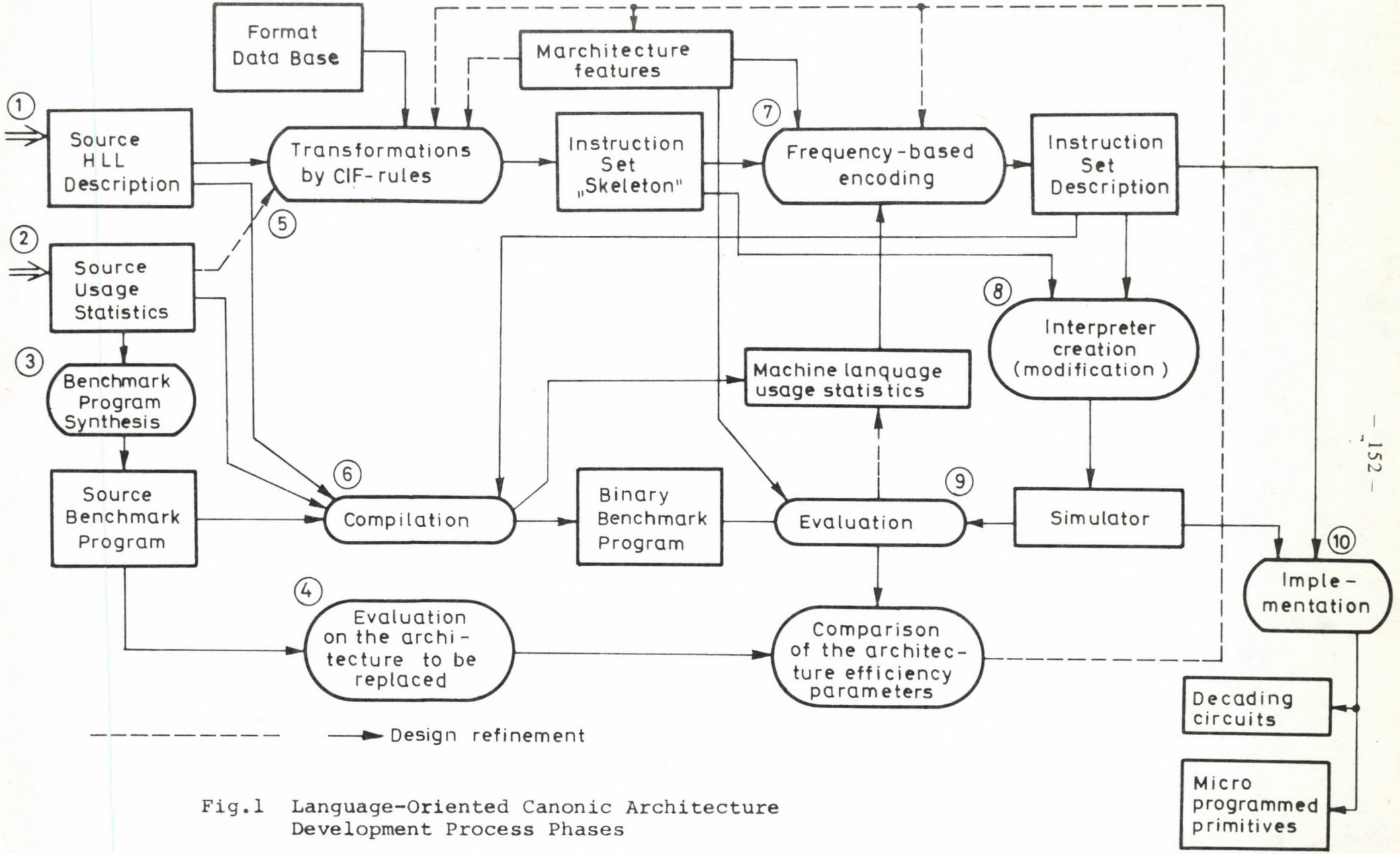


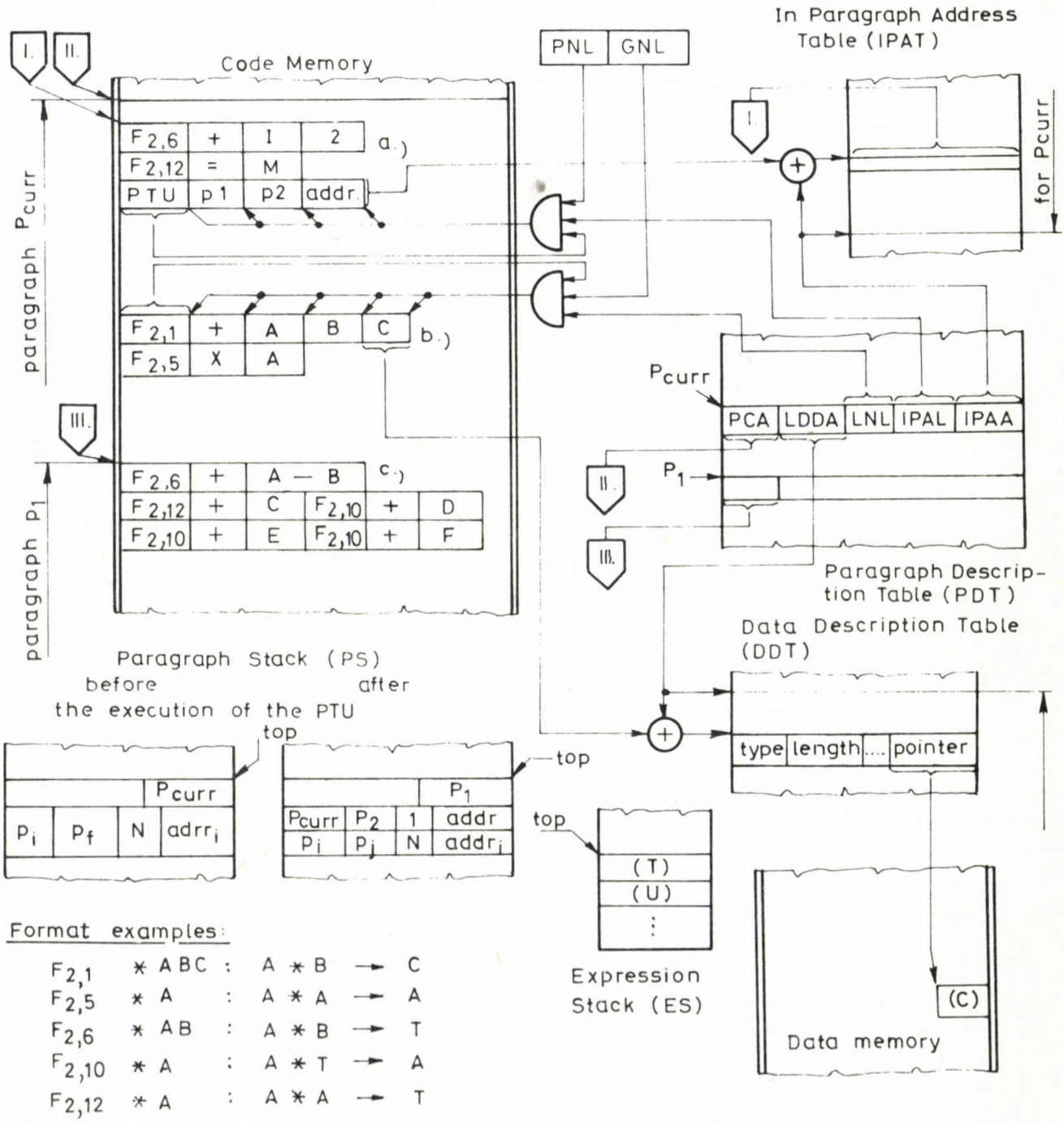
Fig.1 Language-Oriented Canonic Architecture Development Process Phases

Source Statements:

a., PERFORM PARAGR1 THRUU PARAGR2 UNTIL I+2=M
b., ADD A B GIVING C
MULTIPLY A BY A GIVING A
c., PARAGR1
ADD A B C...TO D E E...

Abbreviations:

PNL - Paragraph Name Length
GNL - Global Name Length
LNL - Local Name Length
IPAL- In-Paragraph Address Length
PCA - Paragraph Code Area
IPAA - In-Paragraph Address Area
LDDA - Local Data Description Area



Format examples:

$F_{2,1}$	* ABC	:	A * B	→ C
$F_{2,5}$	* A	:	A * A	→ A
$F_{2,6}$	* AB	:	A * B	→ T
$F_{2,10}$	* A	:	A * T	→ A
$F_{2,12}$	* A	:	A * A	→ T

Fig.2 Canonic COBOL Instruction Examples

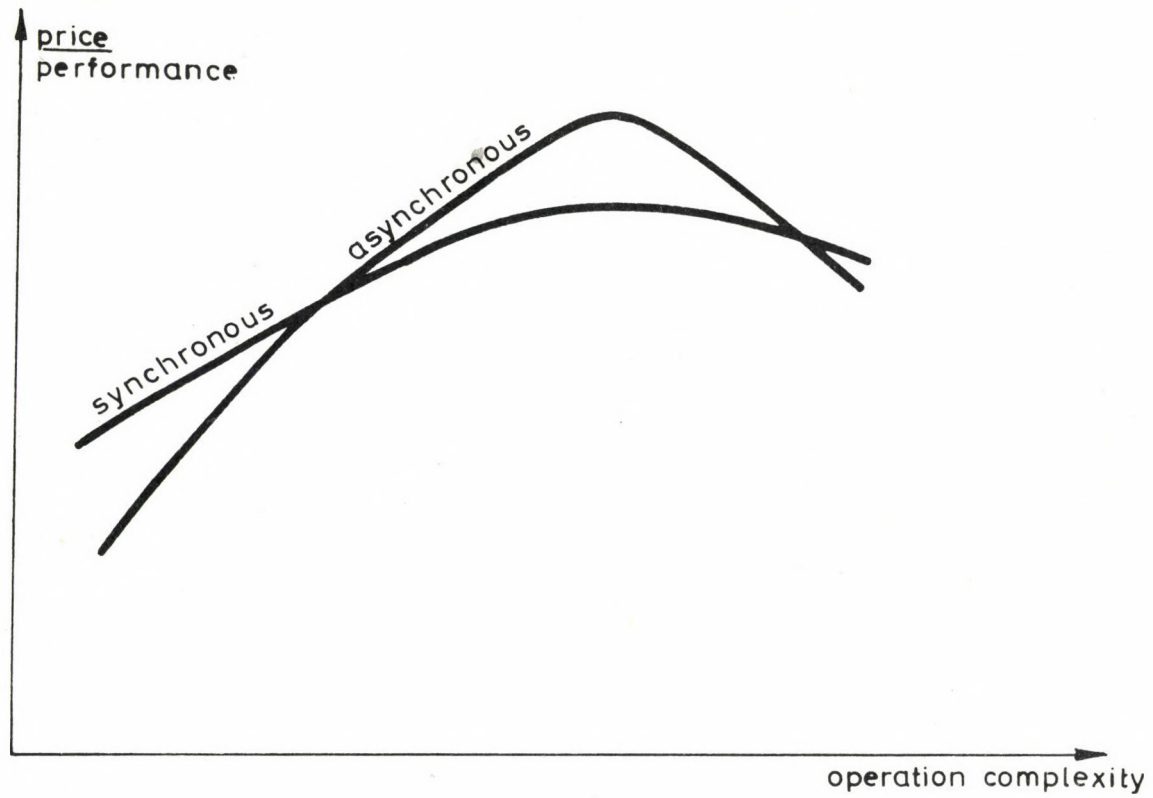
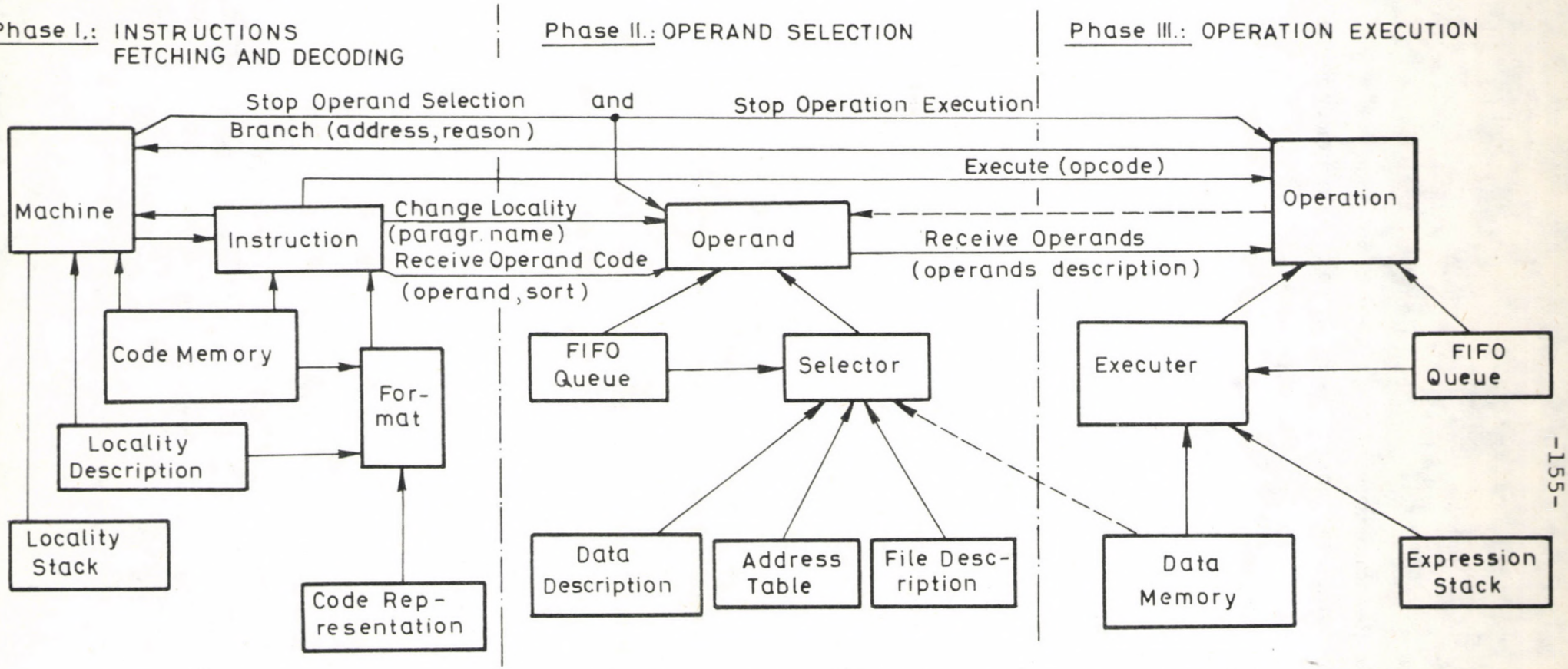


Fig.3 Expediency of Phase Processors



Connections with data structures in Fig.2:

Locality Description: PNL, GNL, PDT /PCA, LNL, IPAL/
Data Description: DDT, PDT, /LDDA/
Address Table: IPAT, PDT /IPAA/
Locality Stack: PS

Fig.4 Canonic Language Architecture Structure
/supposing asynchronous phase processors/

format-decode (> code mem.addr., operation code >)*
format-readoperand (operand>, sort>, nex code mem.addr>)

sort** values may be:
- global data item identifier;
- local data item identifier;
- address identifier;
- paragraph identifier;
- file identifier

next code mem. addr.:
Has value /code memory address for the next instruction to be executed/ only in the case of the last operand /after which the "format-decode" procedure has to be involved again/

* notation: >par - input parameter
par> - output parameter

**in the case of the canonic COBOL architecture discussed here

Fig.5 Procedures Specifying FORMAT Type

- a., Encoding Example

Ø				3	4	7	8	9	1Ø	11	
Ø	Ø	X	X	→	IF T=true THEN <addr>						
Ø	1	X	X	→	GOTO <paragr.>						
1	Ø	X	X	→	MOVE <var 1> TC <var 2>						
1	1	Ø	Ø	→	format code	ADD					
1	1	Ø	1	→	* opcode	format:A*B→A					→(f 2,5)
1	1	1	Ø	→	format code	opcode					
1	1	1	1	→	procedural or I/O operation (integrated code)						

- b., Description of Encoding by a Formal Notation

Constants:

Masks: bitØ1=&CØØ, bitØ3=&FØØ, bitØ9=&FFC, bit47=&ØFØ,....

Patterns: if=Ø, goto=1, move=2, add=12, f25=13, perform=&FØ8,...

Opcodes: add=1, sub=2, if=25, got=27,...

Formats: f11=1, f12=2,...f24=9, f25=1Ø, ...fparagr=22,faddr=23,...

Encoding

∧ mask - bitØ1: (patt-if → if-code, format - faddr;
patt-goto→ goto-code, format - fparagr;
patt-move→ move-code, format - f11);

∧ mask - bitØ3: (patt-add →add-code, format:∧ mask bit47
patt-f25→ opcode: ∧mask-bit47, format-f25;
patt-form→ opcode: ∧mask-bit811, format: ∧ mask-bit47);

∧ mask - bitØ9: patt-perform →perform-code, format-fparagr;

.
.
.
.

Fig.6 Format and Operation Code Encoding Specification and Implementation

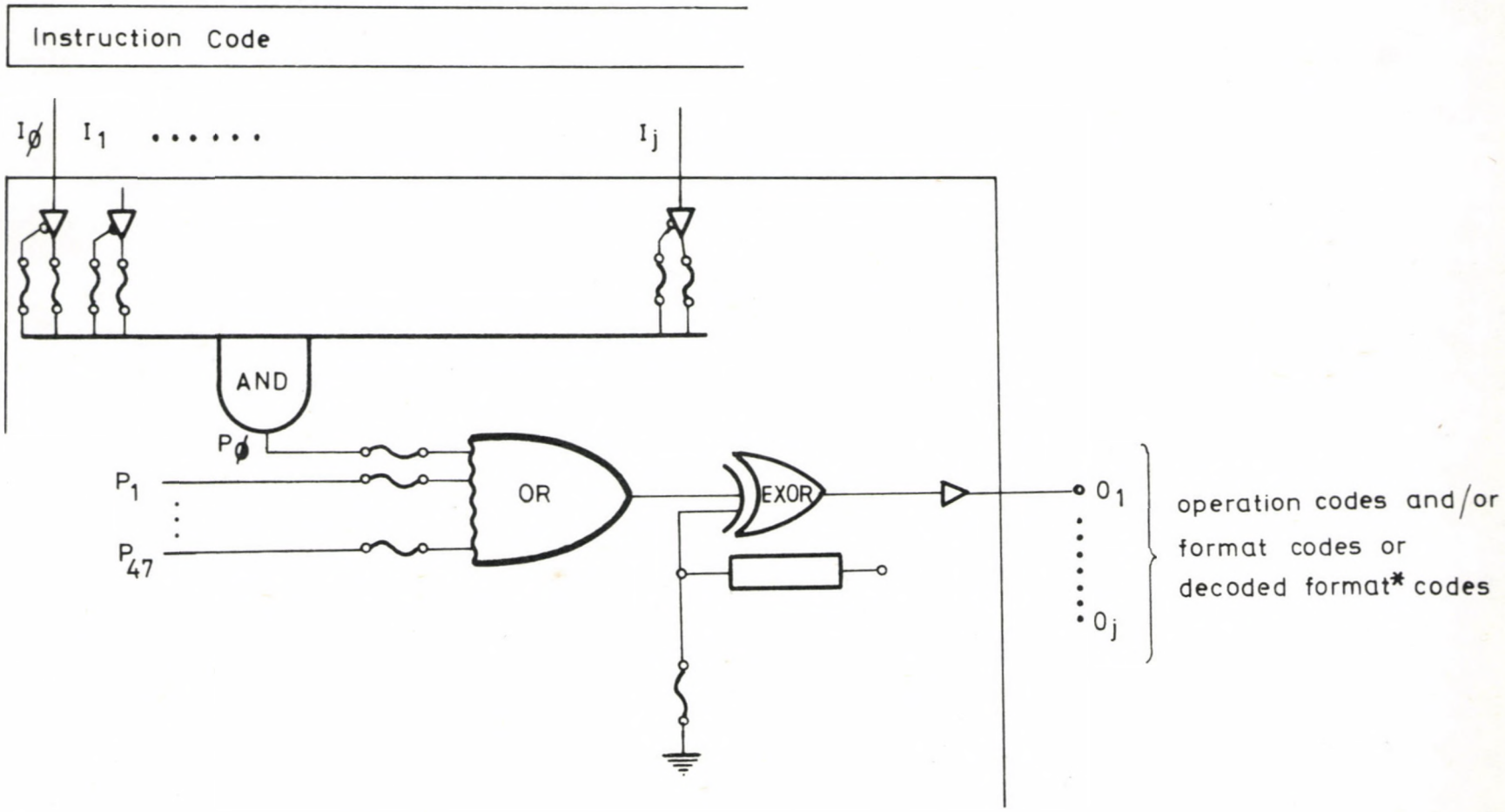
c., Decoding Procedure Declaration in the Code Representation Module of a Canonic Language Interpreter

```
Coderepr.-decode (>instr. code, format>, opcode>, nextfield >):
extract (>instr.code, >mask-bitØ1, pattern>),
 [equal (>pattern, >patt.if),let (opcode>,>if-code),let(format>,> faddr), let(nextfield>,>two);
  equal (>pattern, >patt-goto),let(opcode>,>goto-code),let(format>,> f paragr), let(nextfield>,>two);
  equal (>pattern, >patt-move), let (opcode>,> move-code),let (format>,>f 11), let(nextfield> ,>two) ],
extract (>instr.code, >mask-bit Ø3, pattern>) ,
 [equal (>pattern,>patt-add), let(opcode>,> add-code), extract(>instr.code,>mask-bit47,format>,let(nextfield>,>eight.
```

d., Encoding Procedure: Declaration in the Code Representation Module of a Compiler Code Generator

```
Coderepr.-encode (>format,>opcode, instr.code>, nextfield >):
  equal (>opcode, >if-code), equal(>format,>faddr), and(>maskbitØ1,>patt-if, instr.code>),let(>nextfield>two);
  equal (>opcode,>goto-code), .....
    .
    .
    .
    .
    .
```

e.) Use of FPLA circuits for decoding format and operation codes



*format decoding functions are included in the "Format" module of interpreter /see Fig.4.,5/

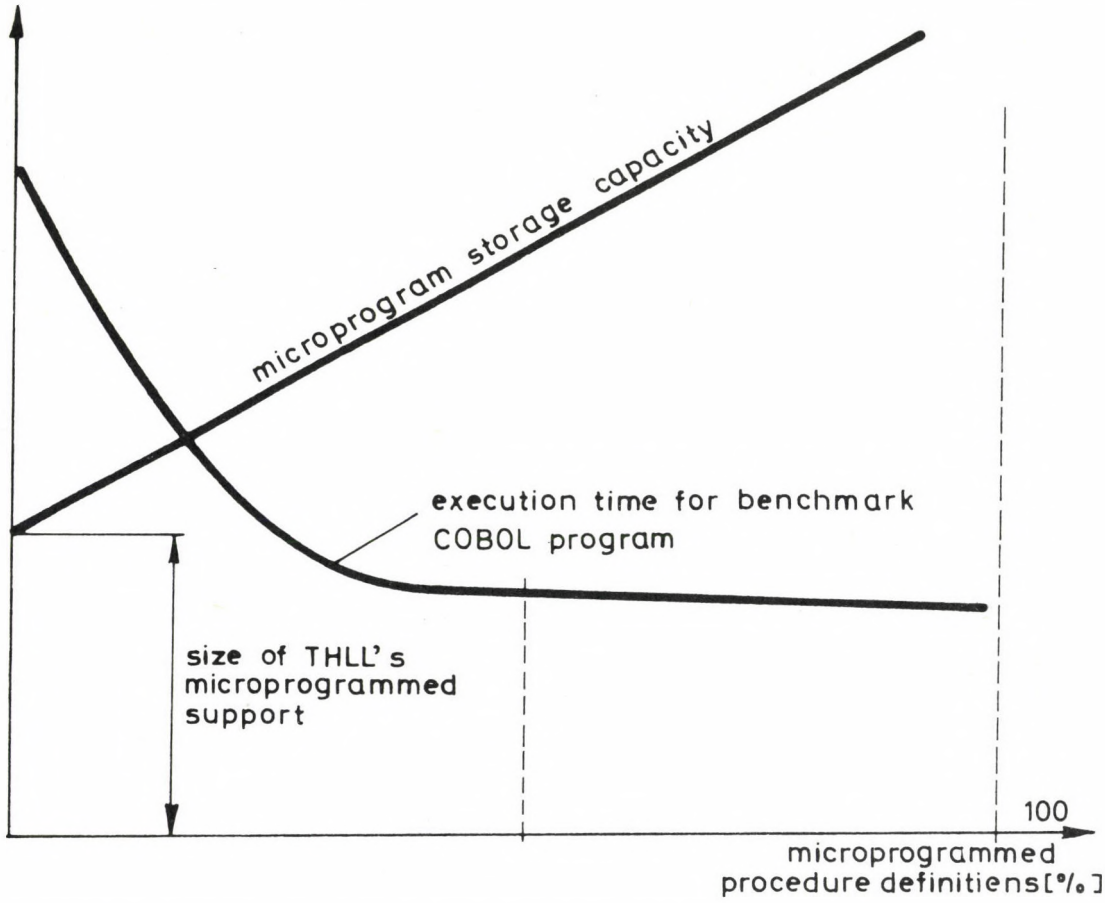


Fig.7 Performance Tuning of a Canonic Interpreter, Written in a THLL

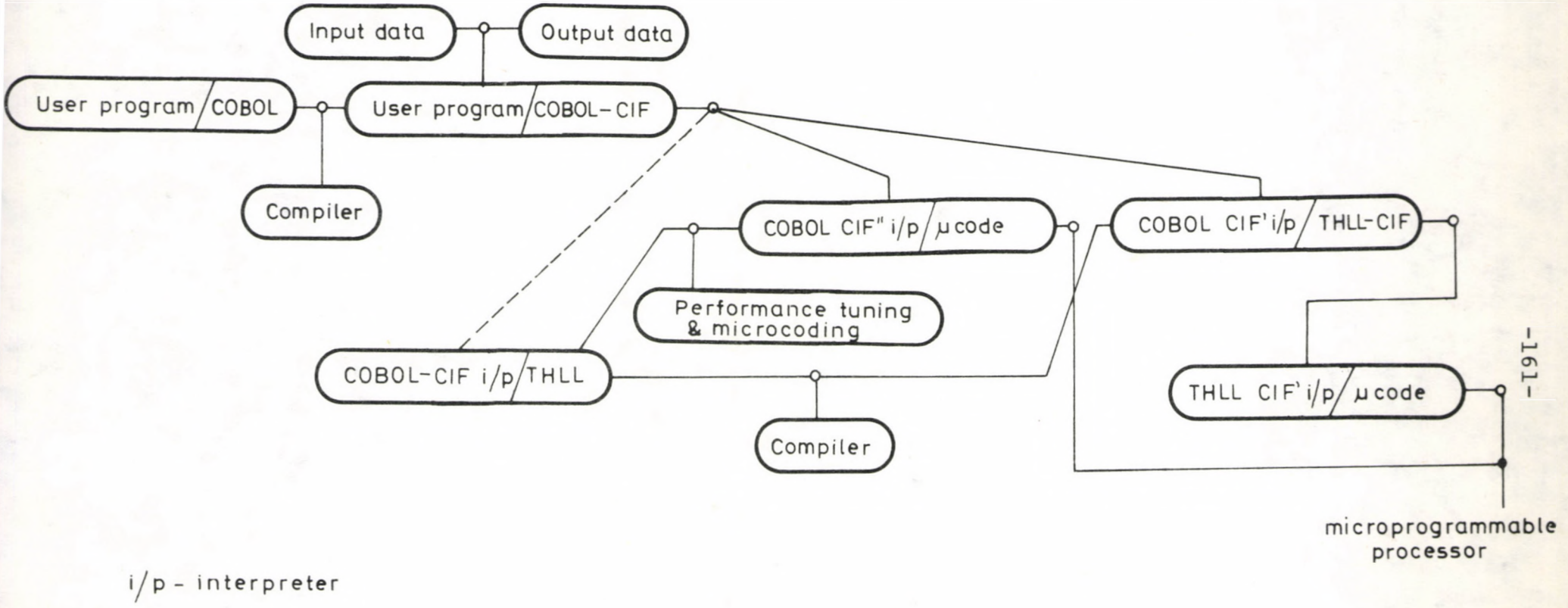


Fig.8 Mixed Double Interpretation Scheme for a COBOL-CIF Architecture

C. MATHEMATICAL ASPECTS OF PROGRAMMING

Computational Linguistics and Computer Languages Vol. XIV. 1980.

THE NOTION OF CONSEQUENCE IN MANY-VALUED LOGIC

by

Peter ALBERT

Technical University for Heavy Industry

Miskolc, Hungary

Preliminaries

The general theory of inference in many-valued logic was not elaborated up to the present. According to classical two-valued logic, B follows from A /or, B is a consequence of A/ if and only if the truth of A logically excludes the falsity of B. Using 0 and 1 for the truth values, falsity and truth, respectively, and denoting the truth value of X by "|X|", one can say that B follows from A if and only if

$$|A| \leq |B|$$

holds /in all interpretations/; i.e. inference may be conceived as a minoration of /the truth value of/ the conclusion by /the truth value of/ the premise.

This approach may be extended to multi-valued logic as well. In two-valued logic, only the cases $|A| = 1$ are important /since $0 \leq |B|$ holds always/; whereas in many-valued systems the cases $|A| = c$ where c is any intermediate value, are to be taken into consideration as well.

In a certain sense, a deduction in the classical first-order calculus may be conceived as a minoration, too. If B is

deducible from the axioms $\underline{A}_1, \ldots, \underline{A}_n$, then $|\underline{A}_1 \& \ldots \& \underline{A}_n| \leq |\underline{B}|$ holds trivially /since $\underline{A}_1, \ldots, \underline{A}_n, \underline{B}$ are universally true formulas/. However, it would be wrong to identify deducibility and consequence.

In classical propositional logic, $\underline{B}$ follows from /is a consequence of/ $\underline{A}$ if and only if "$\underline{A} \supset \underline{B}$" is a truth-functional tautology. This seduces to say that $\underline{B}$ follows from $\underline{A}$ if and only if "$\underline{A} \supset \underline{B}$" is true. However, this is unacceptable, and shows how misleading it is to call '$\supset$' /material/ implication.

In classical logic, the notion of consequence, deducibility and material implication are closely related. /This is the source of a lot of misunderstandings./ In the present paper I argue /in Sections 1 and 2/ that these notions are to be treated rigorously separately, especially in many-valued logic.

In Section 3 I outline a formal system frame incorporating the general theory of consequence-relation for many-valued systems of logic. It is a two-level system where the second level is to be interpreted as the metalogic of the first level. Here '$\supset$' /the sign of material implication/ belongs to the first level; the sign '$\leq$' connects first-level formulas, yielding a second-level formula where "$\underline{A} \leq \underline{B}$" is to be interpreted as "$\underline{B}$ follows from $\underline{A}$"; and '$\vdash$' is the sign of deducibility belonging to the metalanguage of our system /if α is a second-level formula, "$\vdash\alpha$" means that α is deducible in the formal system/.

1 Deduction and Inference

Deductions /or formal proofs/ play a central role in the so-called syntactico-deductive /or "axiomatic"/ systems of logic. In such a system, certain formulas are called axioms and a /finite and nonempty/ sequence of formulas is said to be a deduction /or a formal proof/ if each term of the sequence is either an axiom or follows from previous members by an elemen-

tary syntactical rule /fixed previously/. A formula A is said to be deducible if and only if there exists a deduction terminated by A. The metastatement "A is deducible" is denoted by "$\vdash$A". In many cases, the only elementary rule of deduction is the modus ponens which may be formulated as follows: "$(\vdash A \,\&\, \vdash\ A \supset B) \rightarrow \vdash B$". This sentence expresses a metastatement about the calculus and, hence, it belongs to the metalogic of the calculus. Consequently, the expressions

$$'\vdash', \&', \rightarrow'$$

are symbols of the metalogic. Here '$\vdash$' is a unary functor forming atomic metastatements from formulas of the object system whereas '&' and '$\rightarrow$' are binary connectives forming compound sentences from sentences of the metalogic.

In most cases, the metalogic of a calculus is not defined formally. In the case of classical calculi, it is tacitly presupposed that the laws of metalogic are contained in the laws of the object system. However, the deducibility of "$(A \,\&\, (A \supset B)) \supset B$" or "$(A \,\&\, B) \supset\subset (B \,\&\, A)$" in a calculus, need not have any theoretical link to the rule modus ponens and to the commutativity of 'and' in the metalogic, respectively. Such a theoretical link holds only in the classical systems where the hierarchy between object logic and metalogic might remain hidden.

In many cases, the connectives of a propositional calculus may be interpreted as truth functions, and its variables as truth variables ranging on a fixed set of truth values. Given such an interpretation and a set of designated values as a proper subset of the truth values, a formula is called tautology if it takes only designated values by all possible assignments of its variables. An interpretation is said to be sound if every deducible formula is a tautology; and an interpretation is said to be complete if all tautologies are deducible. Interpretations which are both sound and complete, are called adequate ones with respect to the calculus.

Thus, in the case of a calculus with an adequate interpretation, all deducible formulas are tautologies and, hence, from the view-point of the interpretation, all are of the same semantical rank. None of them "follows" from the other ones in a nontrivial sense of inference. Hence, deduction must not be interpreted as inference from the axioms. In fact, another set of axioms /and another set of primitive rules/ might yield the same set of tautologies. Deducibility is only a technical notion without semantical significance.

The sign '$\vdash$' is used sometimes as a two-place functor of the metalanguage: if α is a set of formulas, then "$\alpha \vdash \underline{A}$" means that $\underline{A}$ is deducible from α in the sense that there exists a sequence $\underline{S}$ of formulas terminated by $\underline{A}$, such that each term of $\underline{S}$ is either an axiom or a member of α, or follows from previous terms by an elementary rule. If α is empty, one gets the notion of deducibility in the previous, narrower, sense.

In the case of classical calculi, if modus ponens is the only elementary rule, then "$\alpha \vdash \underline{A}$" may be interpreted as "$\underline{A}$ follows from α" in the sense that A is true in all models of α. The possibility of this interpretation is based on the fact that modus ponens is truth-preserving /in contrast, e.g. to the quantificational rule "$\underline{A} \supset \underline{B}(c) \rightarrow \underline{A} \supset \forall \underline{x}.\underline{B}(\underline{x})$" where $\underline{c}$ does not occur in $\underline{A}$; this rule is only validity-preserving/; and the latter follows from the truth-functional character of material implication.

If the mentioned conditions are not fulfilled, the interpretation of "$\alpha \vdash \underline{A}$" as "$\underline{A}$ follows from α" fails. Thus, in general, the notions "deducible from α" and "consequence of α" need not coincide. So much the more, deducibility and consequence must be treated separately in many-valued logic.

Let us see now a quite natural generalization of the consequence-relation /of two-valued logic/ in many-valued logic. In two-valued logic, $\underline{B}$ is a consequence of a set α /of pre-

mises/ if and only if $\underline{B}$ is true whenever all members of α are true. If α is finite, say, $\alpha = \{\underline{A}_1, \ldots, \underline{A}_n\}$, then this condition may be formulated as follows: "$\underline{B}$ is true whenever $\underline{A}_1$ is true and ... and $\underline{A}_n$ is true". Assuming that the conjuction of the object language and that of the metalanguage coincide, one might go further by saying that $\underline{B}$ follows from $\{\underline{A}_1, \ldots, \underline{A}_n\}$ if and only if

$$|\underline{A}_1 \bullet \ldots \bullet \underline{A}_n| \leq |B|$$

holds for all assignments of the variables occurring in $\underline{A}_1, \ldots, \underline{A}_n$, $\underline{B}$. Here '$\bullet$' denotes the conjunction of the object language, and $|\underline{X}|$ denotes the truth-value of the formula $\underline{X}$, where truth and falsity are represented by the numbers 1 and 0, respectively. From this view-point, consequence-relation is a minoration of the conclusion by a certain truth-function /namely: conjunction/ of the premises.

Given a multi-valued logic, and a fixed ordering $\leqslant$ of its truth-values /assuming that $\leq$ is reflexive, antisymmetric, transitive and connected/, the scheme

$$|\underline{f}(\underline{A},\underline{B},\ldots)| \leq |\underline{K}|$$

expresses that a truth-function $\underline{f}$ of the premises $\underline{A}$, $\underline{B}$, ... minorates the conclusion $\underline{K}$. Here the conjunction of the premises is replaced by an arbitrary truth-function of the premises. This is the general formulation of the consequence-relation of multi-valued logic. If

$$|\underline{f}(\underline{A},\underline{B},\ldots)| \leq |\underline{K}|, \qquad |\underline{g}(\underline{A},\underline{B},\ldots)| \leq |\underline{K}|$$

and

$$|\underline{f}(\underline{A},\underline{B},\ldots)| \leq |\underline{g}(\underline{A},\underline{B},\ldots)|$$

but

$$|\underline{g}(\underline{A},\underline{B},\ldots)| \nleq |\underline{f}(\underline{A},\underline{B},\ldots)|$$

then $\underline{g}$ gives a stronger minoration /of $\underline{K}$/ than $\underline{f}$ does.

As a further step, we can enlarge the formal system by a partial formalization of its metalogic. Then, one gets a two-level formal system. Formulas of the first level are that of

the primary system, whereas atomic formulas of the second level might have the form "$\underline{A} \leqq \underline{B}$" where $\underline{A}$ and $\underline{B}$ are formulas of the first level and '$\leqq$' represents the minoration relation outlined above. /This step will be treated in Section 3./ By this, deducibility and consequence will be clearly distinguished.

One could get a seemingly natural adaptation of the two-valued consequence-relation in many-valued logic, by replacing "designated value" for "true" in its definition, as follows: $\underline{B}$ is a consequence of $\{\underline{A}_1,\ldots,\underline{A}_n\}$ if and only if $\underline{B}$ assumes a designated value whenever each of $\underline{A}_1,\ldots,\underline{A}_n$ assumes a designated value. By this, the consequence-relation remains, in essence, two-valued, even in the case, the object logic doest not admit two-valued interpretations. Our approach outlined above evades this disastrous conclusion.

2 Material Implication and Inference

According to the deduction theorem, "$\underline{A} \vdash \underline{B}$" and "$\underline{A} \supset \underline{B}$" are equivalent metastatements about the classical propositional calculus (PC). However, the replacement of '$\supset$' by '$\vdash$' in a tautology gives, in some cases, unacceptable metastatements. /The same holds if '$\supset$' is replaced by the word 'implies'./ These cases are well-known as the paradoxes of material implication. An attempt to avoid these paradoxes is the introduction of a more rigorous /non-truth-functional/ connective for expressing implication /strict implication, entailment, relevant implication etc./. None of the various attempts have got a general appreciation. Some systems are criticised for excluding non-paradoxical formulas, some for not excluding all paradoxical formulas and some for both.

To avoid the paradoxes, it seems to be necessary to distinguish the occurrences of the sign of implication in the deducible formulas. In fact, if $\underline{A}$ is a PC-tautology, then the main occurrence of '$\supset$' in $\underline{A}$ is always substitutable by '$\vdash$' /or by 'implies'/ without paradoxical appearance. However, this

condition is not necessary. The tautology "$\underline{A} \bullet (\underline{A} \supset \underline{B}) \supset \underline{B}$" admits the metalogical interpretation: From $\underline{A}$ and from the fact that $\underline{B}$ follows from $\underline{A}$, follows $\underline{B}$. In contrast, the analogous interpretation of the tautology "$\underline{A} \supset (\underline{B} \supset \underline{A})$" would lead to the unacceptable metastatement: From $\underline{A}$ it then follows that $\underline{A}$ follows from $\underline{B}$.

In the light of our analysis given in the preceding section, the best way to avoid the so-called paradoxes of material implication consists in treating separately the three notions: deducibility, inference and material implication. In the case of PC, the following metastatements are equivalent:

$\underline{B}$ follows /semantically/ from $\underline{A}$,

$\underline{A} \vdash \underline{B}$

$\vdash \underline{A} \supset \underline{B}$.

But these equivalences are peculiarities of PC and,are not universal requirements of formal logical theories; still less are they laws of many-valued systems; and the least are they giving any reason to interpret "$\underline{A} \supset \underline{B}$" as "$\underline{A}$ implies $\underline{B}$". Material implication is a truth-function of two sentences, and does not express any logical relation between two statements. /In short: material implication is not implication./

Some systems of many-valued logic contain the following semantical rule for material implication:

(1) $|\underline{A} \supset \underline{B}|$ is a designated value $\rightleftharpoons |\underline{A}| \leq |\underline{B}|$.

/Here, '$\leq$' denotes an ordering of the truth-values; see, e.g., [2] and [3]./ This rule may be considered as the first step toward the multi-valued generalization of the consequence-relation. For, if "$\underline{A} \supset \underline{B}$" is deducible, then it assumes only designated values. Then, by (1), the value of $\underline{A}$ minorates the value of $\underline{B}$ /for all assignments of their variables/, and, hence, $\underline{B}$ follows from $\underline{A}$ /in our sense of "follows"/. Thus, in such a system, "$\underline{A}$ implies $\underline{B}$" means not only that "if $\underline{A}$ takes a designated value, then so does $\underline{B}$" but that "the value of $\underline{B}$ is never smaller than that of $\underline{A}$".

The following rules for $|\underline{A} \supset \underline{B}|$ satisfy condition (1):

(2) $$|\underline{A} \underset{Ł}{\supset} \underline{B}| = \min(1, 1-|\underline{A}| + |\underline{B}|)$$

(3) $$|\underline{A} \underset{P}{\supset} \underline{B}| = \min(1, |\underline{B}| : |\underline{A}|)$$

'+', '-', ':' are symbols of arithmetic.

In (2), the truth-values are the fractions $\underline{k}/\underline{n}$ where $\underline{k}$ and $\underline{n}$ are integers, $0 \leq k \leq n$, and $n \geq 1$ is fixed. /This is taken from Lukasiewicz's $\underline{n}$+1-valued logic./ In (3), the truth-values are the real numbers of the closed interval [0,1] /this example comes from the theory of conditional probability/. In both cases, the only designated value is 1. Another example /for a four-valued logic/ may be found in [3].

However, (1) does not make possible to use an implication in an inference, unless its truth-value is a designated one. According to our programme outlined in Section 1, we should like to infer $\underline{B}$ from a truth-function of $\underline{A}$ and "$\underline{A} \supset \underline{B}$". In the case of (2) and (3) above, a minoration of $\underline{B}$ by an arithmetical function of $|\underline{A}|$ and $|\underline{A} \supset \underline{B}|$ is possible:

(2') $$|\underline{A} \underset{Ł}{\supset} \underline{B}| + |\underline{A}| - 1 \leq |\underline{B}|,$$

(3') $$|A \underset{P}{\supset} B| \cdot |\underline{A}| \leq |\underline{B}|.$$

'.' is symbol of arithmetic.

A sufficient condition of such a minoration of $\underline{B}$ is, in general, the existence of the inverse function of the material implication /i.e., the possibility of calculating the value of $\underline{B}$ from the values of $\underline{A}$ and "$\underline{A} \supset \underline{B}$"/. Of course, not all implications fulfilling (1) are invertible.

In the different systems of many-valued logic, the truth-functions corresponding to the two-valued ones /negation, conjunction, disjunction, material-implication/ are defined somewhat arbitrarily. Moreover, the same system may contain two or more types of negations, conjunctions, implications etc. In contrast, the concept of consequence-relation is not in the least explained in these systems. Hence, there is no difficulty to introduce it in the way outlined in Section 1. In order

to have a uniform formulation of the consequence-relation in all kinds of multi-valued logic, it seems to be appropriate to separate it from material-implication; i.e., the minoration $|\underline{A}| \leq |\underline{B}|$ need not to be interpreted in the sense of (1). Rather, we have to introduce the second-level formula "$\underline{A} \leqq \underline{B}$" where $\underline{A}$ and $\underline{B}$ are formulas of the first level /i.e., formulas of an object system of multi-valued logic/ and we have to formalize the logical laws governing the consequence-relation expressed by "$\underline{A} \leq \underline{B}$

In doing so, it seems, however, to be appropriate, to introduce a certain form of modus ponens in the generalized theory of inference. There are two ways for solving this problem. The first one is to postulate axiomatically the deducibility of the second-level scheme "$\underline{A}.(\underline{A} \supset \underline{B}) \leq \underline{B}$". The second approach consists in postulating the unique solution of the equation

$$\underline{a} \dot{\supset} \underline{b} = \underline{c}$$

for $\underline{b}$, given the truth-values $\underline{a}$ and $\underline{c}$. /Here '$\dot{\supset}$' denotes the truth-function corresponding to the object language sign '$\supset$' in the multi-valued interpretation of the formal system./; i.e. we assume here the invertibility of '$\supset$' /mentioned above/. In general, a weaker condition, namely, the existence of a unique lower bound for $\underline{b}$, is sufficient. Of course, the invertibility of '$\supset$' might assume certain conditions, e.g. that $\underline{a}$ should not have the minimal value /be not "absolutely false"/; but these conditions depend from the peculiarities of the particular multi-valued calculi. Also, the invertibility of other truth--functions might be postulated.

If modus ponens is conceived as the inverse of material implication, and if "$\underline{A} \leq \underline{B}$" is deducible only in the case, all atomic components of $\underline{B}$ occur in $\underline{A}$, then the counterparts of the paradoxes of material-implication are avoidable. E.g.

$$\text{"}\underline{A}.\overline{\underline{A}} \leq \underline{B}\text{"} \qquad \text{"}\underline{A} \leq \overline{\underline{B}.\underline{B}}\text{"}$$

are not deducible. /Here '$-$' denotes negation./

These remarks refer to the role of the object language implication in the consequence-relation represented by '≤', and are absolutely independent from the rule modus ponens of the metacalculus.

3 The Formal System

In most cases, the atomic statements of a metasystem are of the form "⊢A" where A is a formula of the object system and the meaning of "⊢A" is that A is a tautology. However, the Boolean type algebraic formulation of the classical propositional logic does not belong to this type of systems. Here, if A and B are object-language formulas, then the identity of their values expressed by "A = B" is a metastatement; and"⊢A=B" which expresses that the identity of A and B holds tautologically, is a meta-metastatement. Hereinafter two types of systems will be called "system of tautologies" and /"system of identities".

A formalization of a many-valued logic as a system of tautologies does not reflect the fine structure of its many-valued semantics. It only represents the rough dichotomy of designated and non-designated truth-values. Apparently, the many-valued interpretation of such a formal system is solely and ingenious device to explain why the set of deducible formulas is not the same as in two-valued logic.

This method of formalization seems to be highly inadequate, especially in the case of infinitely many-valued systems and particularly in the case, the scale of truth-values is not well-ordered /e.g. if the scale of values is a closed interval of real numbers/.

The dichotomy of designated and non-designated truth-values is fully avoidable in the formulation of logical laws by means of identities, or, rather, by means of inequalities. Given an ordering of the truth-values, the atomic metastatements might have the form "A ≤ B" where A and B are object-language formulas. Then, the logical laws /expressed in the

form of compound formalized metastatements/ do not govern the truth-values of the object-language formulas. Instead, they govern the ordering relation holding between truth-values of object-language formulas.

In what follows, I shall introduce a two-level formal system frame which reflects the mentioned principles. Formulas of the first and the second level will be called object formulas and metaformulas, respectively. Object formulas are built up from atomic ones, by means of three truth-functors /'$^-$', '+', '$\bullet$'; these may be called negation, disjunction and conjunction, respectively/, and quantifiers /'Π' and 'Σ'/; further details of the system of object formulas is left open. Atomic metaformulas are of the form "$\underline{A} \leqq \underline{B}$" where $\underline{A}$ and $\underline{B}$ are object formulas. Compound metaformulas are built up from atomic ones, similarly as in classical logic.

The logical laws of the system govern the system of metaformulas; they will be formulated axiomatically. The first group of the axioms regulates the ordering properties of the sign '$\leqq$' /e.g. transitivity/, while the second one might characterize the peculiarities of the object formulas. /In our general frame, we give only two axioms regulating the functors '$\bullet$' and '+'. The particular structure of the object logic may be expressed by additional axioms belonging to this group./ Finally, the third group of axioms - together with the rules of deduction - formulates the notion of tautology in the sphere of metaformulas.

The rules of deduction include <u>modus ponens</u>. This rule might be formulated in the meta-metalogic as follows:

$$(\vdash \alpha \;\&\; \vdash(\alpha \rightarrow \beta)) \rightarrow \vdash \beta$$

where α and β stand for metaformulas. Another rule of deduction is based on the transitivity of '$\leqq$'; illustrated as follows. /Here $\underline{A}$, $\underline{B}$, $\underline{C}$ are object formulas./

1. $\vdash(\underline{A} \leq \underline{B})$
2. $\vdash(\underline{B} \leq \underline{C})$
3. $\vdash((\underline{A} \leq \underline{B} \ \& \ \underline{B} \leq \underline{C}) \rightarrow \underline{A} \leq \underline{C})$ /transitivity/
4. $\vdash(\underline{A} \leq \underline{C})$ /from 1,2,3/

It might be a tempting idea to formalize the metalogic on the basis of the same multi-valued logic which is assumed in the object-language. /By this, the used multi-valued logic would be proved to be self-contained./ However, a series of arguments supports the application of two-valued logic in the metalanguage. Using any metalogic, it is to be based on a meta-metalogic, and so on, and finally, one has to use a fragment of everyday-language based on two-valued logic. Moreover, the use of a multi-valued language would imply the /theoretical/ undecidability of atomic metastatements of the form "$\underline{A} \leq \underline{B}$". By these, the simplest way is to use two-valued logic in the metalanguage.

The detailed description of our formal system is as follows.

- Definition of object formulas

We assume a denumerably infinite supply of each of the following type of symbols: /a/ individual constants, /b/ individual variables, /c/ for all natural numbers $\underline{n}$ /$\underline{n}$ = 0 included/, $\underline{n}$-place predicate constants, and /d/ for all $\underline{n} \geq 0$, $\underline{n}$-place predicate variables.

/1/ For $\underline{n} \geq 0$, a sequence consisting of an $\underline{n}$-place predicate constant, followed by $\underline{n}$ individual constant is an /atomic/ object formula.

/2/ If $\underline{A}$, $\underline{B}$ are object formulas, then so are "$\overline{\underline{A}}$", "$(\underline{A} \bullet \underline{B})$" and "$(\underline{A}+\underline{B})$".

/3/ If $\underline{A}$ is an object formula, $\underline{x}$ is an individual variable, $\underline{c}$ is an individual constant, occurring in $\underline{A}$ /in the usual sense/, and "$\underline{A}^{\underline{c}}_{\underline{x}}$" is the expression obtained from $\underline{A}$, by substituting $\underline{x}$ for all occurrences of $\underline{c}$ in $\underline{A}$; then "$\Pi\underline{x}\ \underline{A}^{\underline{c}}_{\underline{x}}$" and "$\Sigma\underline{x}\ \underline{A}^{\underline{c}}_{\underline{x}}$" are object formulas.

/4/ An expression is an object formula only if it is composed according to the rules /1/.../3/ and perhaps some other additional rules stated explicitly.

- Definition of metaformulas

/1/ If $\underline{A}$, $\underline{B}$ are object formulas, then "$(\underline{A} \leq \underline{B})$" is an /atomic/ metaformula.

/2/ If α,β are metaformulas, then so are "$\sim\alpha$", "$(\alpha \,\&\, \beta)$", "$(\alpha \vee \beta)$" and "$(\alpha \to \beta)$".

/3/ If α is a metaformula, $\underline{c}$ is a constant /of any type/ occurring in α and $\underline{x}$ is a variable of the same type as $\underline{c}$, then "$\forall\underline{x}\ \alpha\frac{\underline{c}}{\underline{x}}$" and "$\exists\underline{x}\ \alpha\frac{\underline{c}}{\underline{x}}$" are metaformulas. /Concerning the notation "$\alpha\frac{\underline{c}}{\underline{x}}$", vide /3/ of the preceding definition./

/4/ An expression is a metaformula only if it is composed according to the rules /1/.../3/.

- Abbreviations

If $\underline{A}$, $\underline{B}$ are object formulas, then:
"$(\underline{A} = \underline{B})$" abbreviates "$((\underline{A} \leqq \underline{B}) \,\&\, (\underline{B} \leq \underline{A}))$" and
"$(\underline{A} \supset \underline{B})$" abbreviates "$(\overline{\underline{A}} + \underline{B})$".

- Definition of the "classical counterpart of a metaformula"

For all metaformulas α, we define /by induction/ a formula α^* of classical first-order logic, called the classical counterpart of α, as follows.

/1/ Let $\langle\delta_{\underline{n}}\rangle_{\underline{n} \geqq o}$ be an enumeration of all atomic metaformulas. If $\underline{c}_1,\ldots,\underline{c}_{\underline{k}}$ $(\underline{k} \geqq 1)$ is the list of the different constants /of any type/ occurring in $\delta_{\underline{n}}$ /say, in the order of their first occurrence in $\delta_{\underline{n}}$, then we let $\delta^*_{\underline{n}}$ be "$\underline{p}_{\underline{n}}(\underline{c}_1,\ldots,\underline{c}_{\underline{k}})$" where $\underline{p}_{\underline{n}}$ is a $\underline{k}$-place predicate constant /of first-order logic/, and $\underline{c}_1,\ldots,\underline{c}_{\underline{k}}$ are to be considered as individual constants of first-order logic. /If $\underline{m} \neq \underline{n}$, then $\underline{p}_{\underline{m}}$ and $\underline{p}_{\underline{n}}$ must be different./ By this, "$\underline{p}_{\underline{n}}(\underline{c}_1,\ldots,\underline{c}_{\underline{k}})$" is an atomic formula of first-order logic.

/2/If α and β are metaformulas, then

$(\sim\alpha)^* = \sim(\alpha^*)$ $\qquad$ $(\alpha \,\&\, \beta)^* = (\alpha^* \,\&\, \beta^*)$

and similarly for '$\vee$' and '$\rightarrow$';

$$(\forall_{\underline{x}}\,\alpha\,\underline{\underline{C}}_{\underline{x}})^* = \forall\underline{x}(\alpha^*)\underline{\underline{C}}_{\underline{x}}$$

and similarly for '$\exists$'; here, $\underline{x}$ is to be considered as an individual variable of first-order logic.

- Axioms

(ACL) Given a fixed standard system of classical first-order calculus, a metaformula α is an axiom if its classical counterpart, α^* is an axiom of the first-order calculus.

(AS) All metaformulas obtained from one of the five schemata below, by replacing object formulas for $\underline{A}$, $\underline{B}$, $\underline{C}$ and $\underline{D}$ and a 0-place predicate variable for $\underline{X}$, are axioms.

(A:$\leqq$) $\quad (\underline{A} \leqq \underline{B}) \vee (\underline{B} \leqq \underline{A})$

(A:$\leqq$R) $\quad \underline{A} \leqq \underline{A}$

(A:$\leqq$T) $\quad ((\underline{A} \leqq \underline{B}) \,\&\, (\underline{B} \leqq \underline{C})) \rightarrow (\underline{A} \leqq \underline{C})$

(A:I+) $\quad (((\underline{A}+\underline{B}) \leqq (\underline{C}+\underline{D})) \,\&\, (\underline{C} \leqq \underline{A}) \,\&\, \sim\forall\underline{X}(\underline{X} \leqq \underline{A})) \rightarrow (\underline{B} \leqq \underline{D})$

(A:I$\bullet$) $\quad (((\underline{A}\bullet\underline{B}) \leqq (\underline{C}\bullet\underline{D})) \,\&\, (\underline{C} \leqq \underline{A}) \,\&\, \sim\forall\underline{X}(\underline{C} \leqq \underline{X})) \rightarrow (\underline{B} \leqq \underline{D})$

/Outermost parentheses are omitted./

- Rules of deduction

The same as in the first-order calculus referred to in (ACL) above.

Remarks. Axioms (A:$\leqq$), (A:$\leqq$R) and (A:$\leqq$T) ensure that '$\leqq$' represents a linear ordering relation. (A:I+) and (A:I$\bullet$) are necessary for the invertibility of the functors '+' and '$\bullet$', respectively. For, let us substitute $\underline{C}$ by $\underline{A}$ in (A:I+); from this and A:$\leqq$R one gets /by first-order logic/:

(1) $\quad \vdash(((\underline{A}+\underline{B}) \leqq (\underline{A}+\underline{D})) \,\&\, \sim\forall\underline{X}(\underline{X} \leqq \underline{A})) \rightarrow (\underline{B} \leqq \underline{D})$.

By changing the role of $\underline{B}$ and $\underline{D}$ in /1/, we have:

(2) $\quad \vdash(((\underline{A}+\underline{D}) \leqq (\underline{A}+\underline{B})) \,\&\, \sim\forall\underline{X}(\underline{X} \leqq \underline{A})) \rightarrow (\underline{D} \leqq \underline{B})$.

From (1) and (2) it then follows /again, by first-order logic/:

(3) $\quad \vdash \sim\forall\underline{X}(\underline{X} \leqq \underline{A}) \rightarrow (((\underline{A}+\underline{B}) = (\underline{A}+\underline{D})) \rightarrow (\underline{B} = \underline{D}))$.

This means, intuitively, that if $\underline{A}$ is not maximal, then the equality "$\underline{A}+\underline{X} = \underline{C}$" has at most one solution for $\underline{X}$. The dual

law for '•' is deducible from (A:I):

/4/ $\quad \vdash_{\sim} \forall \underline{X}(\underline{A} \leq \underline{X}) \quad \rightarrow \quad (((\underline{A} \bullet \underline{B}) = (\underline{A} \bullet \underline{D})) \rightarrow (\underline{B} = \underline{D}))$.

4 Applications

It is evident that our axioms do not determine uniquely the structure of a multi-valued object logic. The complete characterization of a particular object logic might be given by additional axiom-schemata formulated as /schemata of/ metaformulas. Of course, new functors and operators of the object-language might be introduced as well. A related system /R-fuzzy algebra/ was investigated, e.g. in [1].

Atomic metaformulas $(\underline{A} \leq \underline{B})$ are to be interpreted immediately as inferences of object logic /in accordance with the introductory motivations of Sections 1, 2/. Now, let us show how modus ponens related to the object-language is expressible in particular cases /due to the axiom-scheme (A:I+)/.

As a fragment of an interpretation of our object logic, let us consider a structure consisting of a set of truth-values together with an ordering relation $\dot{\leq}$, a one-place function $\underline{n}(\underline{x})$ and two two-place functions $\underline{a}(\underline{x},\underline{y})$ and $\underline{m}(\underline{x},\underline{y})$. Let us assume that $\dot{\leq}$, $\underline{n}$, $\underline{a}$ and $\underline{m}$ are interpretations of $\leq$, $\bar{}$, + and •, respectively. Furthermore, we assume that an assignment $\underline{V}$ determines the value of all atomic object formulas, and $\underline{V}$ is extended for compound object formulas by the evident rules:

$$\underline{V}(\bar{\underline{A}}) = \underline{n}(\underline{V}(\underline{A})), \quad \underline{V}(\underline{A}+\underline{B}) = \underline{a}(\underline{V}(\underline{A}),\underline{V}(\underline{B})),$$

$$\underline{V}(\underline{A} \bullet \underline{B}) = \underline{m}(\underline{V}(\underline{A}), \underline{V}(\underline{B})).$$

Now, the atomic metaformula "$\underline{A} \leq \underline{B}$" is said to be true /according to the given interpretation/ if and only if $\underline{V}(\underline{A}) \dot{\leq} \underline{V}(\underline{B})$ holds. Finally, we assume that all axioms AS are true /in the obvious sense/, according to our interpretation.

On these assumptions, if "$\sim \forall \underline{X}(\underline{X} \leq \bar{\underline{A}})$" is true and if $\underline{V}(\underline{A})$ and $\underline{V}(\underline{A} > \underline{B})$ /i.e. $\underline{V}(\bar{\underline{A}}+\underline{B})$/ are known, then $\underline{V}(\underline{B})$ is reckonable

by

$$\underline{V}(\underline{B}) = \underline{a}^{-1}(\underline{V}(\underline{A} \supset \underline{B}), \underline{V}(\overline{\underline{A}}))$$

where "$\underline{a}^{-1}$" denotes the inverse of the function $\underline{a}$. /Cf. (3) at the end of the preceding Section./ Let us note that an object-language functor corresponding to $\underline{a}^{-1}$ /i.e. the inverse of the functor '+'/ is not definable syntactically in our general frame, although the function $\underline{a}^{-1}$ may exist in several interpretations. /See, e.g., in [1]./

If "$(\underline{A} \bullet \underline{B}) \leq \underline{B}$" is deducible /due to the additional axioms characterizing a particular object logic/, then "$\underline{A} \bullet \underline{B}$" minorates $\underline{B}$ immediately. Given $\underline{V}(\underline{A})$, a stronger minoration is possible, namely:

$$\underline{V}(\underline{B}) = \underline{m}^{-1}(\underline{V}(\underline{A} \bullet \underline{B}), \underline{V}(\underline{A})),$$

provided "$\sim\forall\underline{X}(\underline{A} \leq \underline{X})$" is true. /Cf. (4) at the end of the preceding Section./ Here, '$\underline{m}^{-1}$' denotes the inverse of the function $\underline{m}$.

If both "$\overline{\overline{\underline{A}}} = \underline{A}$" and "$(\underline{A}+\underline{B}) \leq (\underline{B}+\underline{A})$" are deducible, then the antecedent of an implication might minorate a compound involving the implication and its consequent. Namely:

/5/ $$\underline{V}(\underline{A}) = \underline{n}(\underline{a}^{-1}(\underline{V}(\underline{A} \supset \underline{B}), \underline{V}(\underline{B})),$$

provided "$\sim\forall\underline{X}(\underline{X} \leq \underline{B})$" holds true. This inference form has no classical counterpart. - Assuming that "$\exists\underline{X}\forall\underline{Y}(\underline{Y} \leq \underline{X})$" is deducible, the truth of "$\forall\underline{X}(\underline{X} \leq (\underline{A} \supset \underline{B})$ " implies "$\forall\underline{X}(\underline{A} \not\leq \underline{X}) \vee \forall\underline{X}(\underline{X} \leq \underline{B})$". This means that /5/ is nontrivial only in the case, none of the premises /"$\underline{A} \supset \underline{B}$" and $\underline{B}$ are "absolutely true" /i.e. none of them assumes the maximal value/. /Cf. the motivations of [1]./

Similarly, if "$\sim\forall\underline{X}(\underline{X} \leq \underline{A})$" holds true, then

$$\underline{V}(\underline{B}) = \underline{a}^{-1}(\underline{V}(\underline{A}+\underline{B}), \underline{V}(\underline{A}))$$

represents an inference from a disjunction and one of its members to its other member.

Intuitive motivations of the inference patterns mentioned in this section are minutely discussed in [1].

Acknowledgement. The author is indebted to I. Ruzsa for his helpful criticism.

REFERENCES

[1] Albert, P.: The algebra of fuzzy logic. Journal of Fuzzy Sets and Systems 1 /1978/, 203-230

[2] Gaines, B.R.: Foundations of fuzzy reasonings. International Journal of Man-Machine Studies 8 /1978/, 623-668

[3] Belnap, N.D.; Jr., A useful four-valued logic. In: Modern Uses of Multiple-Valued Logic /Ed. by J.M.Dunn and G. Epstein/. D. Reidel, Dordrecht, 1977.

Computational Linguistics and Computer Languages Vol. XIV. 1980.

IDENTITIES IN ITERATIVE AND RATIONAL ALGEBRAIC THEORIES

by

Zoltán ÉSIK

Department of Computer Science, University of Szeged

Szeged, Hungary

ABSTRACT

In this paper we present a basis of identities of rational algebraic theories. It is conjectured that this basis forms a basis of identities of iterative algebraic theories, as well. It is shown as a result that free rational theories coincide with the free theories over the equational class corresponding to the basis.

1. ALGEBRAIC THEORIES

An algebraic theory T is a special many-sorted algebra whose sorting set is the set of all ordered pairs (n,p) of non-negative integers. Let us denote by T(n,p) the carrier of sort (n,p) of T for each n,p. The operations in T are: composition, source-tupling and injections. For each n,p,q, composition (denoted by $\cdot$ or juxtaposition) maps $T(n,p) \times T(p,q)$ into $T(n,q)$. For each n,p, source-tupling associates with $f_i \in T(1,p)$ $(i=1,\ldots,n)$ a unique element $\langle f_1,\ldots,f_n\rangle \in T(n,p)$. Particularly, if n=0, source-tupling picks out an element $0_p \in T(0,p)$. Finally, injections are nullary operations; there is a corresponding injection $\pi_n^i \in T(1,n)$ to each i and n such that $1 \leq i \leq n$. The operations are required to satisfy the following conditions

(cf. [1]):

(i) $(fg)h = f(gh)$ if $f \in T(n,p)$, $g \in T(p,q)$, $h \in T(q,r)$;

(ii) $f\langle \pi^1_p, \ldots, \pi^p_p \rangle = f$, $f \in T(n,p)$;

(iii) $\pi^i_n \langle f_1, \ldots, f_n \rangle = f_i$, $1 \leq i \leq n$, $f_j \in T(1,p)$

$(j=1,\ldots,n)$;

(iv) $\langle \pi^1_n f, \ldots, \pi^n_n f \rangle = f$, $f \in T(n,p)$.

In particular, if n=0, the last condition asserts that $T(0,p)$ is singleton.

Under these assumptions T becomes a category whose objects are the non-negative integers and in which each object n is the n-th copower of object 1. In fact, it was the original definition of algebraic theories (cf. [7]). In this category the identities are the elements $1_n = \langle \pi^1_n, \ldots, \pi^n_n \rangle$ $(n \geq 0)$. According to the categorical analogy, the elements of T are called morphisms and $f \in T(n,p)$ is written as $f : n \to p$.

It seems convenient to extend source-tupling as follows. Let $f : n \to p$, $g : m \to p$. Then $\langle f,g \rangle = \langle \pi^1_n f, \ldots, \pi^n_n f, \pi^1_m g, \ldots, \pi^m_m g \rangle$. Evidently, this derived operation is associative. Hence, we may write $\langle f,g,h \rangle$ to denote either $\langle f, \langle g,h \rangle \rangle$ or $\langle \langle f,g \rangle, h \rangle$. Another derived operation is the separated sum. First, let us consider 1_n and 0_p. Then, $1_n + 0_p = \langle \pi^1_{n+p}, \ldots, \pi^n_{n+p} \rangle$, while $0_n + 1_p = \langle \pi^{n+1}_{n+p}, \ldots, \pi^{n+p}_{n+p} \rangle$.

In general, if $f : n \to p$ and $g : m \to q$, then $f + g = \langle f(1_p + 0_q), g(0_p + 1_q) \rangle$. The separated sum is associative, too. Concerning other identities the reader is referred to [3].

The algebraic theory T is called non-degenerate, provided $\pi^1_2 \neq \pi^2_2$. A morphism $f : n \to p$ is said to be ideal if none of the morphisms $\pi^1_n f, \ldots, \pi^n_n f$ is an injection. Finally, T is called ideal if it is non-degenerate and for arbitrary f and ideal g, gf is ideal.

One can introduce homomorphisms - called theory maps - between two theories. These are exactly homomorphisms of algebraic theories considered as many-sorted algebras. Let T and T' be ideal theories and take a theory map $F : T \to T'$. If F preserves ideal morphisms, then it is called ideal as well.

Algebraic theories, as they were introduced, have an equational presentation. Hence, for every ranked alphabet or type $\Sigma = \bigcup_{n=0}^{\infty} \Sigma_n$ there exists a free theory generated by Σ. This is denoted by T_Σ and has the following property. There is a ranked alphabet map $\eta : \Sigma \to T_\Sigma$ such that any ranked alphabet map $F : \Sigma \to T$ into an algebraic theory T has a unique homomorphic extension $\overline{F} : T_\Sigma \to T$; i.e. a theory map $\overline{F}$ which satisfies $F = \eta\overline{F}$. Here, by a ranked alphabet map we mean any mapping $F : \Sigma \to T$ such that $F(\Sigma_n) \subseteq T(1,n)$.

T_Σ can be described as the theory of finite Σ-trees on the variables $\{x_1, x_2, \ldots\}$ (cf. [4], [6]). η can be chosen as the mapping $f \mapsto f(x_1, \ldots, x_n)$ $(f \in \Sigma_n, n \geq 0)$. Since η is injective, we can consider Σ as a subset (more precisely as a subsystem) of T_Σ. In this way $F = \eta\overline{F}$ corresponds to $\overline{F}|_\Sigma = F$.

In particular, if Σ is the void alphabet, T_Σ becomes the initial theory. This is isomorphic to the theory Θ, in which $\Theta(n,p)$ is the set of all mappings of $[n] = \{1, \ldots, n\}$ into $[p]$, composition is composition of mappings, source-tupling is source-tupling of mappings, finally, the injection $\pi^i_n : 1 \to n$ is

the mapping which picks out the integer i from [n]. Since Θ is initial in the category of all theories and theory maps, each theory T has exactly one subtheory which is the homomorphic image of Θ. Further on this subtheory will be denoted by Θ_T or, simply, Θ. If T is non-degenerate, Θ_T is isomorphic to Θ, otherwise both Θ and Θ_T is isomorphic to the terminal theory, a theory whose each carrier is either void or singleton. The elements of Θ and Θ_T are called base morphisms and, in the sequal, they are identified. Lower case Greek letters, except Θ, always denote base morphisms. For arbitrary $\rho : n \to p \in \Theta$, $i\rho$ stands for the image of $i \in [n]$ under ρ. A base morphism is called surjective, injective etc. if it is surjective or injective, resp. as a mapping.

We distinguish a subset (or subsystem) from T_Σ. This will be denoted by $\tilde{T}_\Sigma \cdot f \in \tilde{T}_\Sigma (n,p)$ if and only if the frontier of f, i.e. the sequence of variables appearing in the leaves of f, is exactly $x_1 \ldots x_p$. $\tilde{T}_\Sigma$ has the following important property: Every element of T_Σ can be uniquely written in the form $\tilde{f}\rho$, where $\tilde{f} \in \tilde{T}_\Sigma$ and $\rho \in \Theta$.

The morphisms $f : n \to p$ $(n > 0)$ which can be obtained as $(\sum_{i=1}^{n} f_i)\rho$, where $f_i \in \Sigma (i=1,..,n)$ and $\rho \in \Theta$, constitute the subset $\Sigma\Theta$.

Now we are ready to prove:

Lemma 1.1.

Let $f : n \to n+p$, $g : m \to m+p$ and $\rho : m \to n$ be morphisms in a free algebraic theory T. Assume that ρ is surjective and $g(\rho+1_p) = \rho f$. Then, there exists a morphism $h : \ell \to \ell+p$ such that for some surjective $\sigma : \ell \to m$ we have

(i) $h(\sigma + 1_p) = \sigma g$,

(ii) if β is a left inverse of σ, i.e. $\beta\sigma = 1_m$ then $\sigma\beta h = h$,

(iii) for every left inverse γ of τ there are base morphisms $\tau_1,\ldots,\tau_\ell : \ell \to \ell$ satisfying both $\tau_i\tau = \tau$ and $\pi^i_\ell \tau\gamma h(\tau_i + 1_p) = \pi^i_\ell h$ $(i=1,\ldots,\ell)$, where τ denotes the composition $\sigma\rho$.

Proof

Since every free algebraic theory is freely generated by a ranked alphabet, it is enough to verify the statement of the lemma for theories $T = T_\Sigma$, the free algebraic theory generated by a ranked alphabet Σ.

Let g_i denote the i-th component of g, i.e. $g_i = \pi^i_m g$ $(i=1,\ldots,m)$. It can be written in the form $g_i = \tilde{g}_i\alpha_i(\beta_i + \beta'_i)$, where $\tilde{g}_i \in \tilde{T}(1,k_i + k'_i)$, $\beta_i : k_i \to m$, $\beta'_i : k'_i \to p$ and finally, $\alpha_i : k_i+k'_i \to k_i+k'_i$ is bijective and satisfies that both $\alpha_i|_{N_i}$ and $\alpha_i|_{N'_i}$, the restrictions of α_i to $N_i = \{j\alpha_i^{-1} \mid 1 \leq j \leq k_i\}$ and $N'_i = \{j\alpha_i^{-1} \mid k_i < j \leq k_i+k'_i\}$, are monoton mappings.

Assume that $i\rho = j\rho$ $(i,j \in [m])$. Then, also $g_i(\rho+1_p) = g_j(\rho+1_p)$, i.e. $\tilde{g}_i\alpha_i(\beta_i+\beta'_i)(\rho+1_p) = \tilde{g}_j\alpha_j(\beta_j+\beta'_j)(\rho+1_p)$. But there is a unique way to get a morphism of T_Σ as the composition of an element of $\tilde{T}_\Sigma$ and a base morphism.

Thus, we can conclude that $k_i+k'_i = k_j+k'_j$, $\tilde{g}_i = \tilde{g}_j$, $\alpha_i(\beta_i + \beta'_i)(\rho+1_p) = \alpha_j(\beta_j+\beta'_j)(\rho+1_p)$. Suppose that $t \in N_i$. Then, $t\alpha_i(\beta_i+\beta'_i)(\rho+1_p) \leq n$ and hence, $t\alpha_j(\beta_j+\beta'_j)(\rho+1_p) \leq n$. Therefore $t\alpha_j \leq k_j$, i.e. $t \in N_j$. The converse inclusion is similar. This proves the equalities $N_i = N_j$, $N'_i = N'_j$, $k_i = k_j$ and $k'_i = k'_j$.

Or even, since the mappings $\alpha_i|_{N_i}$, $\alpha_i|_{N_i'}$, $\alpha_j|_{N_j}$ and $\alpha_j|_{N_j'}$ are equally monoton, $\alpha_i = \alpha_j$; and it results from this that $\beta_i\rho = \beta_j\rho$ and $\beta_i' = \beta_j'$.

Define $\ell' = \sum_{i=1}^{m} k_i$, $\ell = m+\ell'$. For every $i \in [m]$ let $\bar{h}_i$ denote the morphism $\bar{h}_i = 0_m + \tilde{g}_i\alpha_i(\sum_{t=1}^{i-1} 0_{k_t} + 1_{k_i} + \sum_{t=i+1}^{m} 0_{k_t} + \beta_i')$.

Let $\sigma = \langle 1_m, \beta_1, \ldots, \beta_m \rangle$, $\bar{h} = \langle \bar{h}_1, \ldots, \bar{h}_m \rangle$. A simple computation shows that $\bar{h}_i(\sigma + 1_p) = g_i$ for each $i \in [m]$.

Indeed, $\bar{h}_i(\sigma+1_p) =$

$$= (0_m + \tilde{g}_i\alpha_i(\sum_{t=1}^{i-1} 0_{k_t} + 1_{k_i} + \sum_{t=i+1}^{m} 0_{k_t} + \beta_i'))(\langle 1_m, \beta_1, \ldots, \beta_m \rangle + 1_p) =$$

$$= (0_m + \tilde{g}_i\alpha_i(\sum_{t=1}^{i-1} 0_{k_t} + 1_{k_i} + \sum_{t=i+1}^{m} 0_{k_t} + \beta_i'))\langle 1_m + 0_p, \langle \beta_1, \ldots, \beta_m \rangle + 0_p, 0_m + 1_p \rangle =$$

$$= \tilde{g}_i\alpha_i(\sum_{t=1}^{i-1} 0_{k_t} + 1_{k_i} + \sum_{t=i+1}^{m} 0_{k_t} + \beta_i')(\langle \beta_1, \ldots, \beta_m \rangle + 1_p) =$$

$$= \tilde{g}_i\alpha_i((\sum_{t=1}^{i-1} 0_{k_t} + 1_{k_i} + \sum_{t=i+1}^{m} 0_{k_t})\langle \beta_1, \ldots, \beta_m \rangle + \beta_i') =$$

$$= \tilde{g}_i\alpha_i(\beta_i + \beta_i') = g_i \quad .$$

This proves $\bar{h}(\sigma + 1_p) = g$.

Assume again, that $i\rho = j\rho$ $(i,j \in [m])$. Define $\rho_{i,j} : \ell \to \ell$ by $\rho_{i,j} = 1_m + \rho_{i,j}'$ where $\rho_{i,j}'$ denotes the base morphism

$$\langle \sum_{t=1}^{i-1} 1_{k_t} + \sum_{t=i}^{m} 0_{k_t} \; , \; \sum_{t=1}^{j-1} 0_{k_t} + 1_{k_j} + \sum_{t=j+1}^{m} 0_{k_t} \; , \; \sum_{t=1}^{i} 0_{k_t} + \sum_{t=i+1}^{m} 1_{k_t} \rangle \; .$$

It is easy to check that

$$(\sum_{t=1}^{i-1} 0_{k_t} + 1_{k_i} + \sum_{t=i+1}^{m} 0_{k_t})\rho_{i,j}' = \sum_{t=1}^{j-1} 0_{k_t} + 1_{k_j} + \sum_{t=j+1}^{m} 0_{k_t} \; . \text{ Thus,}$$

$\bar{h}_i(\rho_{i,j} + 1_p) =$

$$= (0_m + \tilde{g}_i \alpha_i (\sum_{t=1}^{i-1} 0_{k_t} + 1_{k_i} + \sum_{t=i+1}^{m} 0_{k_t} + \beta_i'))(1_m + \rho'_{i,j} + 1_p) =$$

$$= 0_m + \tilde{g}_j \alpha_j (\sum_{t=1}^{j-1} 0_{k_t} + 1_{k_j} + \sum_{t=j+1}^{m} 0_{k_t} + \beta_j') = \bar{h}_j \ ,$$

i.e. $\bar{h}_i(\rho_{i,j} + 1_p) = \bar{h}_j$. Furthermore, if τ denotes composition $\sigma\rho$, we have $\rho_{i,j}\tau = \tau$. Indeed,

$$\rho_{i,j}\tau = (1_m + \rho'_{i,j})\langle 1_m, \beta_1, \ldots, \beta_m\rangle\rho =$$

$$= \langle 1_m, \rho'_{i,j}\langle \beta_1, \ldots, \beta_m\rangle\rangle\rho = \langle 1_m, \beta_1, \ldots, \beta_{i-1}, \beta_j, \beta_{i+1}, \ldots, \beta_m\rangle\rho =$$

$$= \langle \rho, \beta_1\rho, \ldots, \beta_{i-1}\rho, \beta_j\rho, \beta_{i+1}\rho, \ldots, \beta_m\rho\rangle = \langle \rho, \beta_1\rho, \ldots, \beta_m\rho\rangle =$$

$$= \langle 1_m, \beta_1, \ldots, \beta_m\rangle\rho = \sigma\rho = \tau.$$

Let $h = \sigma\bar{h}$ and denote by h_i the i-th component of h. For each (i,j) such that $i\tau = j\tau$ let $\tau_{i,j} = \rho_{i\sigma,j\sigma}$. Then , we have $h_i(\tau_{i,j} + 1_p) = h_j$ and $\tau_{i,j}\tau = \tau$.

$h(\sigma + 1_p) = \sigma\bar{h}(\sigma + 1_p) = \sigma g$, this proves part (i) of Lemma 1.1. In order to verify (ii), take β an arbitrary left inverse of σ. Obviously, $\sigma\beta h = \sigma\beta\sigma\bar{h} = \sigma\bar{h} = h$. Finally, let $\gamma : n \to \ell$ be a left inverse of τ. For each $i \in [\ell]$, define τ_i by $\tau_i = \tau_{i\tau\gamma,i}$. This can be done by $i\tau\gamma\tau = i\tau$. Evidently, $\tau_i\tau = \tau$ and $\pi^i_\ell \tau\gamma h(\tau_i + 1_p) = \pi^i_\ell \tau\gamma h(\tau_{i\tau\gamma,i} + 1_p) = \pi^i_\ell h$, ending the proof of the lemma.

2. ITERATIVE AND RATIONAL THEORIES, IDENTITIES

By an algebraic theory with iteration we mean a theory T equipped with a new operation $^+$, called iteration, which, with each $f : n \to n+p$, associates a morphism $f^+ : n \to p$. An iterative algebraic theory T (cf. [3]) is an ideal theory with iteration, except that the iteration is partial. For $f \in T(n,n+p)$

f^+ exists if and only if f is ideal in T, considered as an algebraic theory. Furthermore, f^+ is required to be the unique fixed point of f, i.e. f^+ is the unique morphism $g \in T(n,p)$ auch that $g = f\langle g, 1_p\rangle$.

Homomorphisms between two iterative theories are the ideal theory maps. Observe that ideal theory maps preserve iteration.

Rational algebraic theories were introduced in [9]. A rational algebraic theory T is a theory with iteration. Each of the carriers of T is ordered, f^+ is the least fixed point of f, and ordering subject to some other conditions (cf. [9]). Homomorphisms of rational theories, as they were defined in [9], are certain theory maps, but, likewise in case of iterative theories, they preserve the iteration as well.

In a sense, iterative and rational theories have a common generalization which will be introduced here. Consider an arbitrary theory with iteration. It will be called generalized iterative theory, provided it satisfies the following identities, (A) to (E):

(A) $f^+ = f\langle f^+, 1_p\rangle$, where $f : n \to n+p$,

(B) $\langle f,g\rangle^+ = \langle h^+, (g\rho)^+\langle h^+, 1_p\rangle\rangle$, where $f : n \to n+m+p$, $g : m \to n+m+p$, $h = f\langle 1_n+0_p, (g\rho)^+, 0_n+1_p\rangle$ and $\rho = \langle 0_m+1_n, 1_m+0_n\rangle + 1_p$,

(C) $(0_n+f)^+ = f$, where $f : n \to p$,

(D) $(f+0_q)^+ = f^++0_q$, where $f : n \to n+p$,

(E) $\langle \pi^1_m\rho g(\rho_1+1_p), \ldots, \pi^m_m\rho g(\rho_m+1_p)\rangle^+ = \rho f^+$ if $f : n \to n+p$, $g : n \to m+p$, $\rho : m \to n$ is surjective, $\rho_1, \ldots, \rho_m : m \to m$ are base, furthermore, $\rho_1\rho = \ldots = \rho_m\rho = \rho$, as well as

$f = g(\rho+1_p)$ is satisfied.

In the above mentioned identities f and g are treated as variables of the given sort.

Theorem 2.1

Every rational theory is a generalized iterative theory. Every iterative algebraic theory satisfies identities (A) to (E) if ideal morphisms are substituted for f and g.

We do not present a complete proof of this theorem here. The reason is that most of these identities, except possibly the last one, were already discovered in papers [3], [9]. For (B) cf. [2], too.

Let us remark, however, that it would be enough to prove the theorem for free rational and free iterative theories. And what is more, since free iterative theories can be viewed as weak subalgebras - subtheories closed under the iteration of ideal morphisms - of free rational theories, it would be enough to consider free rational theories only.

We have already mentioned that all free iterative theories exist. This fact was first shown in [1]. Another proof can be found in [5].

I_Σ, the free iterative theory generated by the alphabet Σ can be obtained as follows (cf. [4], [5]). First consider T^∞_Σ, the algebraic theory of all, possibly infinite, Σ-trees on the variables $\{x_1,x_2,\ldots\}$. T^∞_Σ is an ideal theory, even an iterative theory. Then, construct the smallest subtheory of T^∞_Σ containing T_Σ and closed under iteration of ideal morphisms. This will be the iterative theory I_Σ, called "free" because every ranked alphabet map $F : \Sigma \to T$ into an iterative theory T, such that $F(\Sigma)$ contains ideal morphisms only, has a unique homomorphic

extension $\bar{F}$, i.e. an ideal theory map $\bar{F} : I_\Sigma \to T$, satisfying $\bar{F}|_\Sigma = F$. Recall that by $\Sigma \subseteq T_\Sigma$ and $T_\Sigma \subseteq I_\Sigma$, $\Sigma \subseteq I_\Sigma$ holds as well.

R_Σ, the free rational theory generated by Σ, has a similar description. Take $I_{\Sigma_{\underline{!}}}$, where $\Sigma_{\underline{!}}$ is Σ except $(\Sigma_{\underline{!}})_0 = \Sigma_0 \cup \{\underline{!}\}$, and $\underline{!}$ is a new symbol. There is exactly one way to extend $I_{\Sigma_{\underline{!}}}$ to a generalized iterative algebraic theory in such a manner that we have $\pi_1^{1+} = \underline{!}$. When forgetting orderings, R_Σ becomes this theory $I_{\Sigma_{\underline{!}}}$. R_Σ is free in the following sense. For any ranked alphabet map $F : \Sigma \to T$ into a rational theory T, there is exactly one homomorphism (of rational theories) $\bar{F} : R_\Sigma \to T$ extending F, i.e. such that $\bar{F}|_\Sigma = F$. This was proved in [9]. Actually, this theorem remains valid even if $\bar{F}$ is required to be an iteration preserving theory map, i.e. a homomorphism of theories with iteration.

Further on, let us consider R_Σ as an unordered theory. Do not forget that $I_{\Sigma_{\underline{!}}}$, and hereby I_Σ as well, is a weak subalgebra of R_Σ and the carriers of $I_{\Sigma_{\underline{!}}}$ and R_Σ coincide.

We now proceed by stating some consequences of the identities (A) to (E). In these statements, if $(A_1),\ldots,(A_{n+1})$ are sentences of first order, expressed in the language of theories with iteration, we write $(A_1),\ldots,(A_n) \models (A_{n+1})$ to mean the fact that every theory with iteration which satisfies $(A_1),\ldots,(A_n)$, satisfies (A_{n+1}) as well.

(X) $g^+ = \rho f^+$ if $f : n \to n+p$, $g : m \to m+p$, $\rho : m \to n$ is surjective and $g(\rho+1_p) = \rho f$, moreover for any left

inverse α of ρ there exists base morphism $\rho_i : m \to m$ $(i \in [m])$ with $\rho_i\rho = \rho$ and $\pi^i_m \rho\alpha g(\rho_i + 1_p) = \pi^i_m g$.

Lemma 2.2.

$(E) \models (X)$.

Proof

Assume that (E) is satisfied by the algebraic theory with iteration T. Let $f: n \to n+p$, $g: m \to m+p$ and $\rho: m \to n$ content the assumptions of (X) and fix an arbitrary left inverse α of ρ. Define g' by $g' = \alpha g$. Then $g'(\rho + 1_p) = f$. But there exist morphisms $\rho_1, \ldots, \rho_m : m \to m$ satisfying both $\rho_1\rho = \ldots = \rho_m\rho = \rho$ and

$$\langle \pi^1_m \rho g'(\rho_1 + 1_p), \ldots, \pi^m_m \rho g'(\rho_m + 1_p) \rangle = g.$$

Thus, by (E) we obtain $g^+ = \rho f^+$.

Further on the following special case of (X) will be used.

(X') $g^+ = \rho f^+$ if $f: n \to n+p$, $g: m \to m+p$, $\rho: m \to n$ is surjective, $g(\rho + 1_p) = \rho f$, moreover there is a base morphism $\alpha: n \to m$ with $\alpha\rho = 1_p$ and $g = \rho\alpha g$.

The next identity can be derived from (X') and from (E), too. Indeed, if in identity (E) we have $n = m$ as well as $g = f(\rho^{-1} + 1_p)$, where ρ^{-1} denotes the inverse of ρ, furthermore $\rho_i = 1_m$ is satisfied for each $i \in [m]$, then we

obtain identity

(F) $(\rho f(\rho^{-1}+1_p))^+ = \rho f^+$

and the following lemma:

Lemma 2.3. (E) $\models$ (F).

The next identity is the dual of (B).

(B′) $\langle f,g\rangle^+ = \langle f^+\langle h^+,1_p\rangle,h^+\rangle$ if $f : n \to n+m+p$, $g : m \to n+m+p$ and $h = g\langle f^+,1_{m+p}\rangle$.

Lemma 2.4. (B), (E) $\models$ (B′).

Proof

We prove that (B), (F) $\models$ (B′). For this purpose let T be an arbitrary theory with iteration which satisfies both (B) and (F). Take f and g as in (B′) and define ρ by $\rho = \langle 0_n+1_m,\ 1_n+0_m\rangle$. The inverse of ρ is $\rho^{-1} = \langle 0_m+1_n,\ 1_m+0_n\rangle$. Let $f_1 = f(\rho^{-1}+1_p)$, $g_1 = g(\rho^{-1}+1_p)$. Obviously $\rho\langle f,g\rangle(\rho^{-1}+1_p) = \langle g_1,f_1\rangle$.

By (B) we have $\langle g_1,f_1\rangle^+ = \langle h_1^+,(f_1\rho_1)^+\langle h_1^+,1_p\rangle\rangle$, where $\rho_1 = \rho+1_p$, $h_1 = g_1\langle 1_m+0_p,(f_1\rho_1)^+,\ 0_m+1_p\rangle$.

But, $f_1\rho_1 = f(\rho^{-1}+1_p)(\rho+1_p) = f$ and thus

$h_1 = g_1\langle 1_m+0_p,f^+,0_m+1_p\rangle = g(\rho^{-1}+1_p)\langle 1_m+0_p,f^+,0_m+1_p\rangle =$

$= g\langle 0_m+1_n+0_p,1_m+0_{n+p},0_{n+m}+1_p\rangle\langle 1_m+0_p,f^+,0_m+1_p\rangle =$

$= g\langle f^+,1_m+0_p,0_m+1_p\rangle = g\langle f^+,1_{m+p}\rangle = h.$

Thus, $\langle g_1,f_1\rangle^+ = \langle h^+,f^+\langle h^+,1_p\rangle\rangle$. By (F), $(\rho^{-1}\langle g_1,f_1\rangle(\rho+1_p))^+ = \rho^{-1}\langle g_1,f_1\rangle^+ = \langle f^+\langle h^+,1_p\rangle,h^+\rangle$.

It results from this that $\langle f,g\rangle^+ = \langle f^+\langle h^+, 1_p\rangle, h^+\rangle$.

Another identity is:

(G) $(f(1_n+g))^+ = f^+g$, where $f : n \to n+m$, $g : m \to p$.

Lemma 2.5. (B), (C), (D), (E) $\models$ (G).

Proof

We show that (B), (B'), (C),(D) $\models$ (G). Hence, the proof follows by Lemma 2.4.

Let $f_1 = f+0_p : n \to n+m+p$, $g_1 = 0_{n+m}+g : m \to n+m+p$ in an algebraic theory with iteration satisfying (B), (B'), (C) and (D).

By (B) we have $\langle f_1,g_1\rangle^+ = \langle h^+,(g_1\rho)^+\langle h^+,1_p\rangle\rangle$, where $\rho = \langle 0_m+1_n,\ 1_m+0_n\rangle +1_p$, $h = f_1\langle 1_n+0_p,\ (g_1\rho)^+,\ 0_n+1_p\rangle$. But, $g_1\rho = (0_{n+m}+g)(\langle 0_m+1_n,\ 1_m+0_n\rangle +1_p)=0_{n+m}+g$, hence, by (C), $(g_1\rho)^+ = 0_n+g$. Therefore, $h = f_1\langle 1_n+0_p,(g_1\rho)^+,\ 0_n+1_p\rangle =$ $= (f+0_p)\langle 1_n+0_p,\ 0_n+g,\ 0_n+1_p\rangle = f\langle 1_n+0_p,\ 0_n+g\rangle = f(1_n+g)$. We have already seen that $\langle f_1,g_1\rangle^+ = \langle (f(1_n+g))^+,(0_n+g)\langle h^+,1_p\rangle\rangle =$ $= \langle (f(1_n+g))^+,g\rangle$. On the other hand, by (B'), $\langle f_1,g_1\rangle^+ = \langle f_1^+\ \langle h^+,1_p\rangle, h^+\rangle$, where $h = g_1\langle f_1^+,1_{m+p}\rangle$, now.

By $g_1\langle f_1^+,\ 1_{m+p}\rangle = (0_{n+m}+g)\langle f_1^+,1_{m+p}\rangle = 0_m+g$ and (C), $h^+ = g$. It results from this and by (D) that $\langle f_1,g_1\rangle^+ = \langle f_1^+\langle g,1_p\rangle,g\rangle \ = \langle (f^+0_p)\langle g,1_p\rangle,\ g\rangle = \langle f^+g,\ g\rangle$.

If we put the above mentioned two facts together, we get $(f(1_n+g))^+ = f^+g$.

The next identity contains (G) as a special case.

(H) $\langle f_1, g_1 \rangle^+ = \langle f^+ \langle g^+, h \rangle, g^+ \rangle$, where $f : n \to n+m+p$, $g : m \to m+q$, $h : p \to q$, $f_1 = f(1_{n+m}+h)$ and $g_1 = 0_n+g$.

Lemma 2.6. (B), (C), (D), (E) $\models$ (H).

Proof

Instead of this we prove that (B), (G) $\models$ (H). Suppose that T is an algebraic theory with iteration satisfying both (B) and (G). Take the morphisms f,g and h and let $f_1 = f(1_{n+m}+h)$, $g_1 = 0_n+g$.

By (B) we have $\langle f_1, g_1 \rangle^+ = \langle h_1^+, (g_1\rho)^+ \langle h_1^+, 1_q \rangle \rangle$, where $h_1 = f_1 \langle 1_n+0_q, (g_1\rho)^+, 0_n+1_q \rangle$, $\rho = \langle 0_m+1_n, 1_m+0_n \rangle + 1_q$.

$g_1\rho = (0_n+g)\langle 0_m+1_n+0_q, 1_m+0_{n+q}, 0_{n+m}+1_q \rangle =$
$= g\langle 1_m+0_{n+q}, 0_{n+m}+1_q \rangle = g(1_m+0_n+1_q)$. Applying (G) we get $(g_1\rho)^+ = g^+(0_n+1_q) = 0_n+g^+$. Therefore $h_1 =$
$= f(1_{n+m}+h)\langle 1_n+0_q, 0_n+g^+, 0_n+1_q \rangle = f(1_n+\langle g^+, h \rangle)$. Again by (G) we get $h_1^+ = f^+ \langle g^+, h \rangle$.

Hence $\langle f_1, g_1 \rangle^+ = \langle h_1^+, (0_n+g^+)\langle h_1^+, 1_q \rangle \rangle = \langle f^+ \langle g^+, h \rangle, g^+ \rangle \rangle$.

Further on, we shall use the following consequence of (H):

(H′) $\langle f_1, g_1 \rangle^+ = \langle f^+ \langle \pi_m^1 g^+, h \rangle, g^+ \rangle$ if $f : n \to n+1+p$, $g : m \to m+q$ $(m \geq 1)$, $h : p \to q$ and $f_1 = f(1_{n+1}+0_{m-1}+h)$, $g_1 = 0_n+g$.

Lemma 2.7. (H) $\models$ (H′).

Proof

Assume that T satisfies (H), $f : n \to n+1+p$, $g : m \to m+q$ $(m \geq 1)$ and $h : p \to q$ are morphisms in T. Let

$f_1 = f(1_{n+1}+0_{m-1}+h)$, $g_1 = 0_n+g$. Furthermore, let $f' = f(1_{n+1}+0_{m-1}+1_p)$, $f'_1 = f'(1_{n+m}+h)$. It is easy to check that $f_1 = f'_1$. It follows from this and by (H) that $\langle f_1,g_1\rangle^+ = \langle f'^+\langle g^+,h\rangle, g^+\rangle$. By (G), a consequence of (H), $f'^+ = (f(1_n+1_1+0_{m-1}+1_p))^+ = f^+(1_1+0_{m-1}+1_p)$.

Therefore, $f'^+\langle g^+,h\rangle = f^+\langle \pi^1_m g^+ ,h\rangle$ ending the proof of Lemma 2.7.

Finally, we prove a consequence of (B), as well as that one of (A) and (B).

(I) $(1_n+0_m)\langle f_1,g\rangle^+ = f^+$ if $f : n \to n+p$, $g : m \to n+m+p$, $f_1 = f(1_n+0_m+1_p)$.

Lemma 2.8. (B) $\models$ (I).

Proof

By (B) we have $\langle f_1,g\rangle^+ = \langle h^+,(g\rho)^+\langle h^+,1_p\rangle\rangle$, where $h = f_1\langle 1_n+0_p, (g\rho)^+, 0_n+1_p\rangle$, $\rho = \langle 0_m+1_n, 1_m+0_n\rangle + 1_p$. We must prove that $h = f$. But this can be immediately seen since $h = f_1\langle 1_n+0_p, (g\rho)^+, 0_n+1_p\rangle =$ $= f(1_n+0_m+1_p)\langle 1_n+0_p, (g\rho)^+, 0_n+1_p\rangle = f\langle 1_n+0_p,0_n+1_p\rangle = f$.

(J) $\langle f'_1,f'\rangle^+ = \langle f_1^{++},f^+\langle f_1^{++},1_p\rangle\rangle$, where $f = \langle f_1,\ldots,f_n\rangle$: $: n \to n+1+p$, $f^+ = \langle f_1^+,\ldots,f_n^+\rangle$, $f' = \langle f'_1,\ldots,f'_n\rangle = f(\langle 0_1+1_n, 1_1+0_n\rangle+1_p)$, $n \geq 1$.

Lemma 2.9. (A),(B) $\models$ (J).

Proof

Assume that T is an algebraic theory with iteration satis-

fying (A) and (B), and assume that the variables appearing in (J) are interpreted in T. By application of (B) we get $\langle f_1', f' \rangle^+ = \langle h^+, (f'\rho)^+ \langle h^+, 1_p \rangle \rangle$, where

$h = f_1' \langle 1_1 + 0_p, (f'\rho)^+, 0_1 + 1_p \rangle$, $\rho = \langle 0_n + 1_1, 1_n + 0_1 \rangle + 1_p$.

$f'\rho = f(\langle 0_1 + 1_n, 1_n + 0_1 \rangle + 1_p)(\langle 0_n + 1_1, 1_n + 0_1 \rangle + 1_p) =$

$= f(\langle 1_n + 0_1, 0_n + 1_1 \rangle + 1_p) = f$. Therefore, $h = f(\langle 0_1 + 1_n, 1_1 + 0_n \rangle + 1_p)$

$\cdot \langle 1_1 + 0_p, f^+, 0_1 + 1_p \rangle = f_1 \langle 0_1 + 1_n + 0_p, 1_1 + 0_{n+p}, 0_{1+n} + 1_p \rangle \langle 1_1 + 0_p, f^+, 0_1 + 1_p \rangle =$

$= f_1 \langle f^+, 1_1 + 0_p, 0_1 + 1_p \rangle = f_1 \langle f^+, 1_{1+p} \rangle = f_1^+$. The last equality is obtained by application of (A).

Thus we get $(f_1', f')^+ = \langle f_1^{++}, f^+ \langle f_1^{++}, 1_p \rangle \rangle$.

Summarizing the results of this section, we have proved that any generalized iterative theory satisfies the identities (B'), (F), (G), (H), (H'), (I) and (J), as well as the implication (X). In fact, the same proofs can be used to show that all these sentences are valid in iterative theories, too.

3. THE MAIN RESULTS

We now turn to prove that the identities (A) to (E) form a basis of identities of rational theories. This is accomplished by verifying that free rational theories are exactly the free generalized iterative theories. As an intermediate step, we also show that every ranked alphabet map $F : \Sigma \to T$ into a generalized iterative theory T has a unique homomorphic extension (a theory map, preserving iteration of ideal morphism) $\bar{F} : I_\Sigma \to T$. In fact, the proof of the last mentioned theorem is based upon the observation that all considerations in [5] can be carried out under weaker assumptions, i.e. by using the identities (A) to (E) and their consequences only.

For the rest of this section, Σ is taken as an arbitrary fixed alphabet. With the exception of the last two theorems all

statements relate to theory I_Σ.

Lemma 3.1.

Let $f : n \to n+p$, $g : m \to m+p \in \Sigma\Theta$. Assume that $\{\pi_n^i f^+ \mid i \in [n]\} = \{\pi_m^i g^+ \mid i \in [m]\}$. Then, there exist surjective base morphisms $\rho : n \to \ell$ and $\sigma : m \to \ell$, as well as a morphism $h : \ell \to \ell+p$ such that both $f(\rho+1_p) = \rho h$ and $g(\sigma+1_p) = \sigma h$ hold.

Proof

Let ℓ denote the number of distinct components of f^+. We can choose the base morphisms $\alpha_0 : \ell \to n$, $\beta_0 : \ell \to m$, $\rho : n \to \ell$ and $\sigma : m \to \ell$ in such a way that each of the following conditions is satisfied, i.e.
$\alpha_0\rho = 1_\ell$, $\beta_0\sigma = 1_\ell$, $\rho\alpha_0 f^+ = f^+$, $\sigma\beta_0 g^+ = g^+$ and $\alpha_0 f^+ = \beta_0 g^+$.

For an arbitrary $\alpha : \ell \to n$, if $\alpha\rho = 1_\ell$, let f_α denote the composition $f_\alpha = \alpha f(\rho+1_p)$. Similarly, $\beta g(\sigma+1_p)$ is denoted by g_β, provided $\beta\sigma = 1_\ell$. It is easy to check that both $\rho\alpha f^+ = f^+$ and $\sigma\beta g^+ = g^+$ hold. Thus, $f_\alpha\langle\alpha f^+, 1_p\rangle = \alpha f^+$ and $g_\beta\langle\beta g^+, 1_p\rangle = \beta g^+$ showing that $f_\alpha^+ = \alpha f^+$ and $g_\beta^+ = \beta g^+$. But we have $\alpha f^+ = \alpha_0 f^+ = \beta_0 g^+ = \beta g^+$ for every choice of α and β, therefore the morphisms f_α and g_β have the same iteration. Hence, by Lemma 3.5. in [5], it follows that there exists an ideal element $h \in T_\Sigma$ such that f_α and g_β are the partial unwindings of h, for any α and β. But both, f_α and g_β are in $\Sigma\Theta$ resulting that $f_\alpha = h = g_\beta$.

We have shown that for every $\alpha : \ell \to n$ and $\beta : \ell \to m$ if $\alpha\rho = 1_\ell$ and $\beta\sigma = 1_\ell$ are satisfied then so is $f_\alpha = f_{\alpha_0} = g_{\beta_0} = g_\beta$ and by definition, this morphism was chosen as h. Now, we have to verify the equality $f(\rho+1_p) = \rho h$. Let $i \in [n]$ be arbitrary. Choose α in such a way that both $\alpha\rho = 1_\ell$ and $i\rho\alpha = i$

are valid. For this α we have

$$\pi_n^i f(\rho+1_p) = \pi_n^{i\rho\alpha} f(\rho+1_p) = \pi_n^i \rho\alpha f(\rho+1_p) = \pi_n^i \rho f_\alpha = \pi_n^i \rho h.$$

Since $i \in [n]$ was arbitrary, this proves that $f(\rho+1_p) = \rho h$. The proof of $g(\sigma+1_p) = \sigma h$ is similar.

At this point recall a definition from [5]. Let $f : n \to n+p$ be in T_Σ, $i,j \in [n]$. The j-th component of f is said to be **reachable** from the i-th one if there exists a non-negative integer m such that $\pi_n^i f^m$ contains an occurrence of variable x_j. Here, f^m is defined by induction on m : $f^0 = 1_n + 0_p$, $f^{m+1} = f\langle f^m, 0_n+1_p\rangle$. Furthermore, the j-th component of f is called "superfluous" if it is unreachable from the first component of f and $j \neq 1$.

Lemma 3.2.

Let $f : n \to n+p$, $g : m \to m+p \in \Sigma\Theta$. Assume that neither f nor g contains superfluous components. Furthermore, let $F : T_\Sigma \to T$ be an arbitrary theory map into the generalized iterative theory T. Then $\pi_n^1(F(f))^+ = \pi_m^1(F(g))^+$, provided that $\pi_n^1 f^+ = \pi_m^1 g^+$.

Proof

It follows under the assumption of the lemma that $\{\pi_n^i f^+ \mid i \in [n]\} = \{\pi_m^i g^+ \mid i \in [m]\}$. By virtue of Lemma 3.1. we have $f(\rho+1_p) = \rho h$ and $g(\sigma+1_p) = \sigma h$ for some surjective base morphisms $\rho : n \to \ell$ and $\sigma : m \to \ell$ and a morphism $h : \ell \to \ell+p \in \Sigma\Theta$.

Without loss of generality, we may assume ρ and σ to be such that $1\rho = 1\sigma = 1$.

By virtue of Lemma 1.1. we obtain that there exist $f' : n' \to n'+p$ in $\Sigma\Theta$ and surjective $\rho' : n' \to n$ satisfying

(1) $\rho' f = f'(\rho'+1_p)$; (2) $f' = \rho'\alpha' f'$ if $\alpha'\rho' = 1_n$; (3) for arbitrary

$\alpha : \ell \to n'$, if $\alpha\rho'\rho = 1$, then there are base morphism $\rho_1, \ldots, \rho_n$ $n' \to n'$ with $\rho_i\rho'\rho = \rho'\rho$ and $\pi^i_n, \rho'\rho\alpha f'(\rho_i + 1_p) = \pi^i_n f'$ $(i \in [n'])$.
From this, and using the fact that F is a theory map, by (X) we get $F(f')^+ = \rho'\rho F(h)^+$ and $F(f')^+ = \rho F(f)^+$. Hence, $F(f)^+ = \rho F(h)^+$.
The proof of $F(g)^+ = \sigma F(h)^+$ is similar.
Hence, $\pi^1_n(F(f))^+ = \pi^1_n\rho(F(h))^+ = \pi^1_m\sigma(F(h))^+ = \pi^1_m(F(g))^+$ is obtained.

The next statement is analogous to Lemma 3.10. in [5].

Lemma 3.3.

Let $f : n \to n+p \in \Sigma\Theta$. There exists one $g : m \to m+p \in \Sigma\Theta$ which has no superfluous component and satisfies the condition $\pi^1_n(F(f))^+ = \pi^1_m(F(g))^+$ for any generalized iterative theory T and theory map $F : T_\Sigma \to T$.

Proof

First, assume that those components of f which are not superfluous, are exactly the first components m. In this case, $(1_m + 0_{n-m})f$ can be written as $g(1_m + 0_{n-m} + 1_p)$, where $g : m \to m+p \in \Sigma\Theta$. Since F is a theory map it follows that $(1_m + 0_{n-m})F(f) = F(g)(1_m + 0_{n-m} + 1_p)$, furthermore, $F(f) = \langle F(g)(1_m + 0_{n-m} + 1_p), (0_m + 1_{n-m})F(f)\rangle$.

It results from this by (I) that $(1_m + 0_{n-m})(F(f))^+ = F(g)^+$. This implies $\pi^1_n(F(f))^+ = \pi^1_m(F(g))^+$.

In the general case, let $i_1, \ldots, i_m$ be all different indices such that $\{\pi^{i_t}_m f \mid t \in [m]\}$ is exactly the set of not superfluous components of f. We may assume that $i_1 = 1$. Let the bijection $\rho : n \to n$ satisfy $i_t\rho = t$ for each $t \in [m]$. Applying the first case for $\rho^{-1}f(\rho + 1_p)$, we get a morphism $g : m \to m+p$

in $\Sigma\Theta$ which does not contain superfluous components and satisfies $\pi^1_n(F(\rho^{-1}f(\rho+1_p)))^+ = \pi^1_m(F(g))^+$ for any theory map $F : T_\Sigma \to T$. Since F is a theory map, by the identity (F), this implies $\pi^1_n \rho^{-1}(F(f))^+ = \pi^1_m(F(g))^+$, i.e. by $1\rho = 1$, $\pi^1_n(F(f))^+ = \pi^1_m(F(g))^+$.

We are now ready to state

Theorem 3.4.

Let $F : \Sigma \to T$ be an arbitrary ranked alphabet map into a generalized iterative theory T. There exists exactly one homomorphism $\bar{F} : I_\Sigma \to T$ extending F, i.e. such that $\bar{F}|_\Sigma = F$.

Proof

Since Σ generates T_Σ and T_Σ generates I_Σ, there can be at most one $\bar{F}$ extending F. Thus, we have to show the existence of $\bar{F}$ only.

We know that there is a theory map from T_Σ into T (considered as an algebraic theory) which extends F. Let us denote this theory map by F, too.

Define $\bar{F}$ as follows:

(i) $\bar{F}(\pi^i_n) = \pi^i_n$ if $n \geq 1$, $i \in [n]$,

(ii) $\bar{F}(f) = \pi^1_n(F(a))^+$ if $a : n \to n+p \in \Sigma\Theta$ and $f = \pi^1_n a^+$,

(iii) $\bar{F}(\langle f_1,\ldots,f_n\rangle) = \langle \bar{F}(f_1),\ldots,\bar{F}(f_n)\rangle$ if $n \neq 1$, $f_i : 1 \to p$.

By Theorem 4.1.1 of [4] and lemmas 3.2. and 3.3, $\bar{F}$ is a mapping of I_Σ into T. By (i) and (iii) $\bar{F}|_\Theta = F|_\Theta$.

Take an arbitrary element $f : 1 \to p \in \Sigma$. Since $f = (0_1+f)^+$ and $0_1+f \in \Sigma\Theta$, we have $\bar{F}(f) = \pi_1^1(F(0_1+f))^+ =$ $= \pi_1^1(0_1+F(f))^+ = \pi_1^1 F(f) = F(f)$. Observe that we have used identity (C). By (iii) this results $\bar{F}|_\Sigma = F$.

By virtue of (iii), $\bar{F}$ preserves source-tupling. We now prove that $\bar{F}$ preserves composition. Since it preserves source--tupling, it is enough to show that for any morphism $f : 1 \to p$ and $g : p \to q$ $\bar{F}(fg) = \bar{F}(f)\bar{F}(g)$. This is obvious if f is base, hence we may assume that f is ideal. Or even, by a note in [5], we may confine ourselves to the case that g is base, or its first component is ideal and all other are base.

First, assume that g is base, $g = \rho$. We know that $f = \pi_n^1 a^+$, where $a : n \to n+p \in \Sigma\Theta$. By (G) $f\rho = \pi_n^1(a(1_n+\rho))^+$. Therefore, $\bar{F}(f\rho) = \pi_n^1(F(a(1_n+\rho)))^+ = \pi_n^1(F(a)(1_n+\rho))^+$. On the other hand $\bar{F}(f)\rho = \pi_n^1(F(a))^+\rho$, and this, by an application of (G), results that $\bar{F}(f)\rho = \pi_n^1(F(a)(1_n+\rho))^+$. Hence, $\bar{F}(f\rho) = \bar{F}(f)\rho$.

The proof of $\bar{F}(fg) = \bar{F}(f)\bar{F}(g)$ in the second case, i.e. the first component of g is ideal and the others are base, is similar, only apply identity (H') instead of (G).

Finally, we prove that for ideal $f : n \to n+p$ we have $(\bar{F}(f))^+ = \bar{F}(f^+)$. Since $\bar{F}$ is a theory map and by identity (B) it is enough to deal with the case: $n = 1$.

Since f is ideal there exists an $a : m \to m+1+p \in \Sigma\Theta$ such that $f = \pi_m^1 a^+$. Let $b = a(\langle 0_1+1_m, 1_1+0_m\rangle +1_p)$, $c = \langle \pi_m^1 b, b\rangle$. By (J) we get $f^+ = \pi_{m+1}^1 c^+$. Since $c \in \Sigma\Theta$, it follows that $\bar{F}(f^+) = \pi_{m+1}^1(F(c))^+$. Similarly, a repeated application of (J) yields $(\bar{F}(f))^+ = (\pi_m^1(F(a))^+)^+ = \pi_{m+1}^1\langle\pi_m^1 F(b), F(b)\rangle^+ = \pi_{m+1}^1(F(c))^+$. This ends the proof of Theorem 3.4.

Corollary

Theorem 3.4 holds under certain weaker assumptions, too. In fact the iteration need not be defined for arbitrary morphisms in the theory T. But we require F to be such that it being considered as a theory map $F : T_\Sigma \to T$ should satisfy $(F(f))^+$ to exist in T that whenever $f \in \Sigma\theta$, or $f = 0_p$ for some p. Furthermore, likewise in Theorem 3.4, we have to require the identities (A) to (E) to be satisfied in T in the strong sense: for every evaluation the left hand side of an identity exists if and only if the right hand side exists, and if both of them exist, they are equal. This is always the case if T is an iterative theory and F(f) is ideal for every $f \in \Sigma$.

Theorem 3.5.

R_Σ is the generalized iterative theory freely generated by Σ.

Proof

By virtue of Theorem 3.4 and since $I_{\Sigma_{\underline{!}}}$ is a weak subalgebra of R_Σ, moreover, the carriers of R_Σ and $I_{\Sigma_{\underline{!}}}$ coincide, it is enough to prove the following statement: for every ranked alphabet map $F : \Sigma_{\underline{!}} \to T$ such that $F(\underline{!}) = (\pi_1^1)^+$, remember that $\underline{!} = \pi_1^{1+}$ holds in R_Σ, the free extension $\bar{F} : I_{\Sigma_{\underline{!}}} \to T$ constructed in the proof of Theorem 3.4 is a homomorphism (of generalized iterative theories) from R_Σ into T.

We know that $\bar{F}$ preserves theory operations, i.e. composition, source-tupling and injections. Hence, we have to show that $\bar{F}$ preserves (arbitrary) iteration. By identity (B) and since $\bar{F}$ is a theory map, it is enough to deal with scalar morphisms.

Take an arbitrary morphism $f : l \to l+p$. If f is ideal then, by Theorem 3.4, $\bar{F}(f^+) = (\bar{F}(f))^+$. Otherwise f is an injection π^i_{l+p}. If $i = 1$ then $\bar{F}(\pi^{1^+}_{l+p}) = \bar{F}((\pi^1_1+0_p)^+) = \bar{F}(\underline{\perp}+0_p) = \bar{F}(\underline{\perp})+ 0_p =$ $= \pi^{1^+}_1 + 0_p$. On the other hand $(\bar{F}(\pi^1_{l+p}))^+ = \pi^{1^+}_{l+p} = (\pi^1_1 + 0_p)^+ =$ $= \pi^{1^+}_1 + 0_p$. Observe that identity (D) was used. Assume now that $i > 1$. Then $\pi^i_{l+p} = 0_1 + \pi^{i-1}_p$. Therefore, by (C), $\bar{F}(\pi^{i^+}_{l+p}) = \bar{F}(\pi^{i-1}_p) = \pi^{i-1}_p = \pi^{i^+}_{l+p} = (\bar{F}(\pi^i_{l+p}))^+$.

We are now able to prove the main result:

Theorem 3.6.

Identities (A) to (E) together with those defining algebraic theories, form a basis of identities of rational theories.

Proof

We have to prove that the equational class of all generalized iterative theories coincides with the equational class generated by the class of rational theories (considered as unordered theories). But this can be done immediately by Theorem 2.1 and Theorem 3.5.

Corollary

ω-continuous algebraic theories were also examined in [8] and [9]. These are special rational theories. It was proved by [8] that the free ω-continuous algebraic theory generated by Σ exactly is the theory $T^\infty_{\Sigma_{\underline{\perp}}}$ with a certain ordering.

What is important for us from this fact is that R_Σ is a subalgebra of $T^\infty_{\Sigma_{\underline{\perp}}}$. It results from this that the equational class generated by the rational theories exactly is that one generated by the class of all ω-continuous theories. Therefore, Theorem 3.6 remains valid even if rational theories are replaced

by ω-continuous theories. The same holds for some other types of continuity (cf. Δ -continuity), too.

At the beginning of this paper we have mentioned that by the author's conjecture, identities (A) to (E) together with the defining identities of algebraic theories form a basis of identities of iterative theories, too. This conjecture is based on Theorem 3.4 and its corollary. Unfortunately, we do not know any definition of validity of an identity in a class of partial algebras by which we could prove Theorem 3.6 for iterative theories, and which is accepted by mathematicians working in partial algebras.

4. FURTHER REMARKS

We know that identities listed in (A) to (E) are not completely independent; e.g. it would be sufficient to require (A) in case $n = 1$, etc.

On the other hand we conjecture that all of the identities grouped in (A) or in (B) etc. cannot be omitted. A simplification of the basis will probably be introduced in a forthcoming paper.

Another note concerns with the connection of iterative and and generalized iterative theories. We have actually verified in the proof of Theorem 3.5 that R_Σ is the free generalized iterative theory generated by I_Σ. Roughly speaking, R_Σ can be obtained by adjoining a new element $\perp$ to I_Σ. It can be seen that this remains valid in the general case, too: for every iterative theory T there exists a free generalized iterative theory generated by T and this free theory can be obtained by adjoining a new element to T. This helps us to prove another interesting statement. Let T be an iterative theory and assume that $T(1,0)$ is nonvoid, say $\perp \in T(1,0)$. Then, there is exactly one way to

extend T to a generalized iterative theory such that we have $\underline{i} = \pi_1^{1^+}$.

REFERENCES

[1] Bloom, S.L. and Elgot, C.C.: The Existence and Construction of Free Iterative Theories, J.Comput. System Sci. 12/1976/, 3o5-318

[2] Bloom, S.L.,Ginali, S. and Rutledge, J.D.: Scalar and Vector Iteration, J.Comput. System Sci. 14/1977/, 251-256

[3] Elgot, C.C.: Monadic Computation and Iterative Algebraic Theories, Logic Colloquium'73, Rose, H.E. and Shepherdson, J.C. Eds., Vol.8o, Studies in Logic, North-Holland, Amsterdam, 1975, 175-23o

[4] Elgot, C.C.,Bloom, S.L. and Tindell, R.: On the Algebraic Structure of Rooted Trees, J.Comput. System Sci. 16/1978/, 362-399

[5] Ginali, S.: Regular Trees and the Free Iterative Theory, J.Comput. System Sci. 18/1979/, 228-242

[6] Goguen, J.A.,Thatcher, J.W.,Wagner, E.G. and Wright, J.B.: Initial Algebraic Semantics and Continuous Algebras, J. Assoc. Comput. Mach. 24/1977/, 68-95

[7] Lawvere, F.W.: Functorial Semantics of Algebraic Theories, Proc. Nat.Acad.Sci. USA 5o /1963/, 869-872

[8] Wagner, E.G.,Wright, J.B.,Goguen,J.A. and Thatcher,J.W.: Some Fundamentals of Order-Algebraic Semantics, Mathematical Foundations of Computer Science, 1976, Mazurkiewicz, A. Ed., Lecture Notes in Computer Science 45, 151-168

[9] Wright,J.B.,Thatcher,J.W.,Wagner,E.G. and Goguen,J.A.: Rational Algebraic Theories and Fixed-Point Solutions, 17th IEEE Symposium on Foundations of Computing, Houston, 1976, 147-158

Computational Linguistics and Computer Languages Vol. XIV. 1980.

SIMPLE DETERMINISTIC MACHINES AND THEIR LANGUAGES

Ngo The Khanh
Socialist Republic of Vietnam*

1. INTRODUCTION

The relationship between automata and formal languages has a very deep influence on the development of their theories. Automata-theoretic aspects in studying formal languages are also related to the theory and practice of compiler-construction for programming languages. From the practical point of view it is quite natural to impose some restrictions on the way of working of language processors in order to increase their efficiency. A very important restriction of this kind is determinism which is ment to avoid the need for back-tracking while processing an input string. Deterministic context-free languages introduced by S. Ginsburg and S. Greibach [2] in 1966 have influenced a number of investigations which have still left open some very interesting problems. Deterministic context-free languages form a proper subclass of the family of context-free languages. In the paper by E.P. Friedman [1] a further restriction has been introduced, namely the number of internal states of deterministic pushdown automata has been limited to one which gave rise to the family of simple context--free languages. These languages bear a number of interesting properties established in [1] and [3].

* Presently working with the Computer and Automation Institute, Hungarian Academy of Sciences

In the present paper we extend the ideas of E.P. Friedman and develop a complete hierarchy of simple deterministic machines and that of the corresponding language-families. First, we define simple finite deterministic machines and the related subclass of regular languages which is, at the same time, a subclass of simple context-free languages. A direct generalization of the simple finite deterministic machine leads us to the concepts of the simple deterministic pushdown machine and the two-memory simple machine. The corresponding classes of languages are compared with those defined by E.P. Friedman and some of their properties are established. A common feature of the languages in our hierarchy is their prefix-free property which means that no initial substring of a word belonging to some language may belong to the same language. This is due to the fact, that whenever a simple machine accepts some input string it cannot move any further since no transition is defined in an accepting state. The same property is true also for E.P. Fiedman's classes of languages but it is not their characteritic property. In our case, however, we show that it is a characteristic property, i.e.,every prefix-free, deterministic context--free languages is accepted by some simple deterministic pushdown machine and similarly, every prefix-free regular language must be accepted by some simple finite deterministic machine. For two-memory simple languages which are not included in the context-free family, we are not sure but have a homomorphic characterization for them.

2. SIMPLE FINITE DETERMINISTIC MACHINE

Let us consider the standard definition of the finite deterministic automaton (abbreviated as fd-machine) [4]; i.e. Let $M = (K,\Sigma,\delta, q_o,H)$ be a fd-machine, where

K is the nonempty set of states,

Σ is the nonempty set of inputs,

q_o is an element of K (the initial state),

H is a subset of K (the set of final states),

δ is a mapping from $K\times\Sigma$ to K .

Notation. Given a fd-machine M, let $\vdash_M^*$, or $\vdash$ when M is understood, be the relation on $K \times \Sigma^*$ defined as follows.

For $a \in \Sigma$ and $w \in \Sigma^*$. Let $qaw \vdash pw$ if $\delta(q,a) = p$.

Let $\vdash^*$ denote the transitive closure of $\vdash$. Finally, we define the language accepted by M as:

$$L(M) = \{w \in \Sigma^* / q_o w \vdash^* p \text{ for some } p \in H\}$$

Definition (2.1.)

a./ A fd-machine $M = (K,\Sigma,\delta,q_o,H)$ is called "simple" (abbreviated sfd-machine) if and only if the function δ is a mapping from $(K-H)\times\Sigma$ to K .

b./ A language is said to be a simple finite deterministic language (sfd-language) if $L = L(M)$ for some sfd-machine M.

The family of all sfd-languages is denoted by $\mathcal{L}_{sd3}$

Notice that a sfd-machine is defined such that once the momentary state is a final state, no further input can be processed. Thus, if L is a sfd-language, then L must be prefix-free. Hence, if $x \in L$ and $xy \in L$, then $y = \lambda$ (where λ is the empty string); and in this way, if $\lambda \in L$ then $L = \{\lambda\}$.

By the prefix-free property of a sfd-language, we can easily see the following:

Lemma (2.2)

If L is a sfd-language, then $\bar{L}$ is not a sfd-language. Hence, the family $\mathcal{L}_{sd3}$ is not closed under complementation.

Theorem (2.3)

The family $\mathcal{L}_{sd3}$ is closed under concatenation.

Proof

Assume that $L = L(M)$ and $L' = L(M')$, where: $M = (K,\Sigma,\delta,q_o,H)$, $M' = (K',\Sigma',\delta',q_o',H')$ are sfd-machines. Without the loss of generality, we may assume that $K \cap K' = \emptyset$.

Construct a sfd-machine M_1 from M and M' as follows:

Let $M_1 = (K_1, \Sigma_1, \delta_1, q_o, H')$, where:

$K_1 = (K-H) \cup K'$,

$\Sigma_1 = \Sigma \cup \Sigma'$ and δ_1 is defined so that:

1.) for every $q \in (K-H)$, $a \in \Sigma$:
if $\delta(q,a) = p$, then $\delta_1(q,a) = \bar{p}$,

where $$\bar{p} = \begin{cases} p & \text{if } p \notin H \\ q_o' & \text{if } p \in H \end{cases}$$

2.) for every $q' \in (K'-H')$, $a \in \Sigma$:
if $\delta'(q',a) = p'$, then $\delta_1(q',a) = p'$

3.) All other transitions are undefined. It is easily seen that
$w = w_1 w_2 \in L.L'$, where $w_1 \in L$, $w_2 \in L'$

$$\Leftrightarrow \begin{cases} q_o w_1 \overset{*}{\underset{M}{\vdash}} p \in H, \text{ and} \\ q_o' w_2 \overset{*}{\underset{M}{\vdash}} p' \in H' \end{cases}$$

$$\Leftrightarrow \{q_o w_1 w_2 \overset{*}{\underset{M_1}{\vdash}} q_o' w_2 \overset{*}{\underset{M_2}{\vdash}} p' \in H' : w = w_1 w_2 \in L(M_1)\}$$

Hence, the result.

Theorem (2.4)

The family $\mathcal{L}_{sd3}$ is closed under intersection.

Proof

Assume that $L = L(M)$ and $L' = L(M')$, where $M = (K, \Sigma, \delta, q_o, H)$, $M' = (K', \Sigma, \delta', q_o', H')$ are sfd-machines. Construct a sfd-machine M_2 from M and M' as follows:

Let $M_2 = (K_2, \Sigma, \delta_2, [q_o, q_o'], H_2)$, where

$K_2 = \{[q,q'] / q \in K \text{ and } q' \in K'\}$

$H_2 = \{[p,p'] / p \in H \text{ and } p' \in H'\} \subseteq K_2$,

and δ_2 is such that:

1.) for every $a \in \Sigma$, $q \in K$, $q' \in K'$:
$\delta_2([q,q'],a) = [p,p']$ iff $\begin{cases} \delta(q,a) = p, & \text{and} \\ \delta(q',a) = p' \end{cases}$

2. all other transitions are undefined.

First, we can easily see by induction on the length of w, where $w \in \Sigma^*$

$$[q,q'] w \vdash_{M_2}^{*} [p,p'] \iff \begin{cases} qw \vdash_{M}^{*} p, & \text{and} \\ q'w \vdash_{M}^{*} p' \end{cases}$$

Thus,

$$w \in L \cap L' \iff \begin{cases} q_o w \vdash_{M}^{*} p, & \text{for } p \in H \quad \text{and} \\ q'_o w \vdash_{M}^{*} p', & \text{for } p' \in H' \end{cases}$$

$$\iff \{[q_o,q'_o] w \vdash_{M_2}^{*} [p,p'], \text{ for } [p,p'] \in H_2\}$$

$$\iff w \in L(M_2)$$

Hence, the result

Theorem (2.5)

The family $\mathcal{L}_{sd3}$ is not closed under union.

Proof

First, we consider the following language:

$L = \{a^n b / n \geq 1\} \cup \{a^n bb / n \geq 1\}$.

By the prefix-free property of sfd-language, we can easily see that L is not a sfd-language.

We now prove that:

$L_1 = \{a^n b / n \geq 1\}$ and

$L_2 = \{a^n bb / n \geq 1\}$ are sfd-languages

Provide two sfd-machines M_1, M_2 that accept these languages

Let

$M_1 = (K_1, \{a,b\}, \delta_1, q_o, H_1)$, where:

$K_1 = \{q_o, q_1, q_h\}$,

$H_1 = \{q_h\}$, and δ_1 is defined so that:

$\delta_1(q_o, a) = q_1$

$\delta_1(q_1, a) = q_1$

$\delta_1(q_1,b) = q_h$

and $M_2 = (K_2,\{a,b\},\delta_2,q_o,H_2)$, where

$K_2 = \{q_o,q_1,q_2,q_h\}$,

$H_2 = \{q_h\}$, and δ_2 is such that:

$\delta_2(q_o,a) = q_1$

$\delta_2(q_1,a) = q_1$

$\delta_2(q_1,b) = q_2$

$\delta_2(q_2,b) = q_h$

Thus, $L_1 = L(M_1)$, $L_2 = L(M_2)$ are sfd-languages such that $L_1 \cup L_2$ is not a sfd-language.
Hence, the result.

3. SIMPLE MACHINE

Definition (3.1). (Definition (2.1.) in [1])

a.) A simple machine is a 4-tuple $M = (\Sigma,\Gamma,\delta,Z_o)$, where
Σ is a finite input alphabet,
Γ is a finite pushdown alphabet,
$Z_o \in \Gamma$ is the initial pushdown symbol,
and δ is a mapping from $(\Sigma \cup \{\lambda\}) \times \Gamma$ to Γ^* such that:
for every $Z \in \Gamma$ either $\delta(a,Z)$ contains exactly one element for each $a \in \Sigma$ and $\delta(\lambda,Z)$ is undefined, or $\delta(\lambda,Z)$ contains exactly one element and $\delta(a,Z)$ is undefined for all $a \in \Sigma$.

b.) A configuration of M is a pair (w,α), where: $w \in \Sigma^*$ is the portion of the input tape remaining to be read. $\alpha \in \Gamma^*$ is the current contents of the pushdown store, where the top of store is the rightmost symbol of α.

c.) We define the operator $\vdash_{\overline{M}}$ on configurations of M as follows:

for each $a \in \Sigma$, $w \in \Sigma^*$, $z \in \Gamma$, α, $\beta \in \Gamma^*$:

i.) $(aw,\alpha Z) \vdash_M (w,\alpha\beta)$ iff $\delta(a,Z) = \beta$ or

ii.) $(aw,\alpha Z) \vdash_M (aw,\alpha\beta)$ iff $\delta(\lambda,Z) = \beta$

Let $\vdash_M^*$ denote the transitive, closure of $\vdash_M$.

d.) We shall be concerned with acceptance of an input string by empty pushdown store. Accordingly, we define the language accepted by a simple machine M to be:

$L(M) = \{w \in \Sigma^* / (w,Z_o) \vdash_M^* (\lambda\ \lambda)\}$

A language L is said to be simple-context-free (sc-language) if $L = L(M)$ for some simple machine M. The family of all sc-languages is denoted by $\mathcal{L}_{sc}$.

Theorem (3.2)

a.) For every sfd-machine M, there is a simple machine M' such that $L(M') = L(M)$.

b.) There is a simple machine M_1 such that $L(M_1)$ is not a sfd-language.

Proof

a.) Let $M = (K,\Sigma,\delta,q_o,H)$ be a sfd-machine. We construct a simple machine M' from M, as follows:

Let $M' = (\Sigma,K,\delta',q_o)$, where:

δ' is defined so that:

for each $a \in \Sigma$, $q \in (K-H)$:

1.) $\delta'(a,q) = p$ iff $\delta(q,a) = p$,

2.) $\delta'(\lambda,p) = \lambda$ for all $p \in H$,

3.) all other transitions are undefined.

Evidently, $L(M') = L(M)$.

b. To show part (b), we reconsider the non-regular language (also non sfd-language) first seen in [5].

$L = \{a^n b^n / n \geq 1\}$

We now provide a simple machine M_1 that accepts this language:

$$M_1 = (\{a,b\}, \{Z_o,A,B\}, \delta_1, Z_o),$$

where δ_1 is as follows:

1.) $\delta_1(a,Z_o) = B$

2.) $\delta_1(a,B) = AB$

3.) $\delta_1(b,A) = \delta_1(b,B) = \lambda$

Definition (3.3). (Definition (3.1) in [1])

a.) An extended simple machine (abbreviated as es-machine) is a 4-tuple $M = (\Sigma,\Gamma,\delta,Z_o)$, where
Σ is a finite set of input symbols,
Γ is a finite set of pushdown symbols,
$Z_o \in \Gamma$ is the initial pushdown symbol,
and δ is a mapping from $\Sigma \times \Gamma$ to $\Gamma^* \times \{0,1\}$ and satisfies: for each $a \in \Sigma$, $Z \in \Gamma$ if $\delta(a,Z)$ is defined, then $\delta(a,Z)$ contains exactly one element.

Notation Given an es-machine, let $\vdash_M^*$ be the relation on $\Sigma^* \times \Gamma^*$ (The set of all configurations) defined as follows
For $Z \in \Gamma, \alpha,\beta \in \Gamma^*$, $a \in \Sigma$, $w \in \Sigma^*$:

i) $(aw,\alpha Z) \vdash_M (w,\alpha\beta)$ iff $\delta(a,Z) = (\beta,1)$, or

ii) $(aw,\alpha Z) \vdash_M (aw,\alpha\beta)$ iff $\delta(a,Z) = (\beta,0)$.

Define $\vdash_M^*$ to be the transitive, closure of $\vdash_M$.

b. An input string is accepted by es-machine M when the entire tape has been processed and the pushdown store is empty. That is,

$$L(M) = \{w \in \Sigma^* / (w,Z_o) \vdash_M^* (\lambda,\lambda)\}$$

We say that L is an es-language if there exists an es-machine M such that $L = L(M)$.
The family of all es-languages is denoted by $\mathcal{L}_{es}$.

Theorem (3.4). (Theorem (3.2) in [1])

a.) For every simple machine M, there is an es-machine M' such that $L(M') = L(M)$.

b.) There is an es-machine M_1 such that $L(M_1)$ is not a simple context-free language.

Proof

a.) Let $M = (\Sigma, \Gamma, \delta, Z_o)$ be a simple machine. Construct an es-machine M' from M as follows:

Let $M' = (\Sigma, \Gamma, \delta', Z_o)$,

where δ' is defined so that:

for each $a \in \Sigma$, $Z \in \Gamma$:

1.) if $\delta(a,Z)$ is defined, then $\delta'(a,Z) = (\delta(a,Z),1)$

2.) if $\delta(\lambda,Z)$ is defined, then $\delta'(a,Z) = (\delta(\lambda,Z),0)$ for all $a \in \Sigma$.

It is easily seen that $L(M') = L(M)$.

b.) To show part (b), we reconsider the non-simple context-free language first seen in [6]:

$$L = \{a^i b^i \$ / i \geq 0\} \cup \{a^i d^i \$ \quad i \geq 0\}$$

We now provide an es-machine M_1 that accepts this language:

$$M_1 = (\{a,b,d,\$\}, \{Z_o, A, B, D\}, \delta_1, Z_o)$$

where δ_1 is as follows:

$\delta_1(a,Z_o) = (AZ_o,1)$

$\delta_1(\$,Z_o) = (\lambda,1)$

$\delta_1(b,Z_o) = \delta_1(d,Z_o) = (\lambda,0)$

$\delta_1(b,A) = (B,1)$

$\delta_1(d,A) = (D,1)$

$\delta_1(b,B) = \delta_1(d,D) = (\lambda,0)$

$\delta_1(\$,B) = \delta_1(\$,D) = (\lambda,1)$

Theorem (3.5).

Let $M = (\Sigma, \Gamma, \delta, Z_o)$ be an es-machine satisfying the following condition:
"if $\delta(a,Z) = (\beta, 0)$, then $|\beta| \geq 1$, where $a \in \Sigma$, $Z \in \Gamma$ " then $L(M)$ is a sc-language. (where $|\beta|$ denotes the length of β).

Proof

Since $|\beta| \geq 1$, we can write $\beta = \beta_1 Z$, where $Z \in \Gamma$ $\beta_1 \in \Gamma^*$.

Construct a simple machine M' from M, as follows:
Let $M' = (\Sigma, \Gamma, \delta', Z_o)$, where δ' is defined so that, for each $a \in \Sigma$, $Z \in \Gamma$:

1.) if $\delta(a,Z) = (\beta, 1)$, then $\delta'(a,Z) = \beta$

2.) There exist $Z_1, Z_2, \ldots, Z_n$: $(1 \leq n \leq |\Gamma|)$ such that $\delta(a,Z_i) = (\beta_i Z_{i+1}, 0)$ for $i=1,2,\ldots,(n-1)$, where $Z = Z_1$ $Z_i \neq Z_j$ for $i \neq j$. Here we have three cases to consider:

2.a) if $\delta(a,Z_n) = (\beta_n, 1)$, then $\delta'(a,Z) = (\beta_1\beta_2\ldots\beta_n)$;

2.b) if $\delta(a,Z_n) = (\beta_n Z_i, 0)$, where $i \in \{1,2,\ldots(n-1)\}$, then $\delta'(a,Z)$ is undefined;

2.c) if $\delta(a,Z_n)$ is undefined, then $\delta'(a,Z)$ is undefined too.

3.) All other transitions are undefined. It is easily seen that $L(M') = L(M)$. Hence the result.

Notice, that a simple machine and es-machine is defined so that once pushdown store is empty, no further input can be processed. Thus, if L is a sc-language, or an es-language, then L must be prefix-free, and by their prefix-free property, we can easily prove the following:

Lemma (3.6).

If L is a sc-language (or es-language), then $\bar{L}$ is not a sc-language (not an es-language). Hence each of the families $\mathcal{L}_{sc}, \mathcal{L}_{es}$ is not closed under complementation.

By theorems (3.4), (3.2) and the proof of theorem (2.5) we can easily see the following:

Lemma (3.7).

Each of the families $\mathcal{L}_{sc}, \mathcal{L}_{es}$ is not closed under union.

Theorem (3.8).

Each of the families $\mathcal{L}_{sc}, \mathcal{L}_{es}$ is not closed under intersection.

Proof

First, we prove that the language $L = \{a^n b^n c^n \$ / n \geq 1\}$ is not context-free. Assume the contrary, and let $M = (K, \Sigma, \Gamma, \delta, Z_o, q_o, H)$ be a pushdown automaton such that $L = L(M)$. (The definition of a pushdown automaton in [4]). We now construct a pushdown automaton M' from M as follows: Let $M' = (K, \Sigma', \Gamma, \delta', Z_o, q_o, H)$, where:

$\Sigma' = \Sigma - \{\$\}$

and δ' is defined so that: for each $q \in K$, $Z \in \Gamma$, $d \in \Sigma'$:

1.) $(p,\alpha) \in \delta'(q,d,Z)$ iff $(p,\alpha) \in \delta(q,d,Z)$, where $p \in K$, $\alpha \in \Gamma^*$;

2.) If $(p,\alpha) \in \delta(q,\$,Z)$, then $(p,\alpha) \in \delta'(q,\lambda,Z)$, where $p \in K$, $\alpha \in \Gamma^*$;

3.) All other transitions are undefined.

It is easily seen that:

$$L(M') = \{a^n b^n c^n \ / \ n \geq 1\} \ .$$

But it is well-known that $\{a^n b^n c^n / n \geq 1\}$ is non-context free (see e.g. in [4]), and thus $L=\{a^n b^n c^n \$ / \ n \geq 1\}$ cannot be context-free either. Now let

$$L_1 = \{a^n b^n c^i \$ / n \geq 1, \ i \geq 1\} \quad , \qquad \text{and}$$

$$L_2 = \{a^i b^n c^n \$ / n \geq 1, \ i \geq 1\}$$

Evidently, $L = L_1 \cap L_2$. We prove that L_1, L_2 are sc--languages. Provide two simple machines M_1, M_2 that accept these languages:

Let $M_1 = (\{a,b,c,\$\}, \{Z_o,A,C\}, \delta_1, Z_o)$, where δ_1 is: such that

$\delta_1(a,Z_o) = CA$

$\delta_1(a,A) = AA$

$\delta_1(b,A) = \lambda$

$\delta_1(c,C) = C$

$\delta_1(\$,C) = \lambda$

and

$M_2 = (\{a,b,c,\$\},\{Z_o,A,B,C\},\delta_2,Z_o)$, where δ_2 is: such that

$\delta_2(a,Z_o) = A$

$\delta_2(a,A) = A$

$\delta_2(b,A) = CB$

$\delta_2(b,B) = BB$

$\delta_2(c,B) = \lambda$

$\delta_2(\$,C) = \lambda$

Thus, L_1, L_2 are sc-languages (and are also es-languages) such that $L_1 \cap L_2$ is not a sc-language (since $L_1 \cap L_2$ is non-context free, hence $L_1 \cap L_2$ is also not an es-language).

Theorem (3.9).

Each of the families $\mathcal{L}_{sc}, \mathcal{L}_{es}$ is closed under concatenation.

Proof

First consider $\mathcal{L}_{sc}$:

Assume, that $L = L(M)$, $L' = L(M')$, where $M = (\Sigma,\Gamma,\delta,Z_o)$, $M' = (\Sigma',\Gamma',\delta',Z_o')$ are simple machines.

Without loss of generality we may assume again that:

i.) $\Gamma \cap \Gamma' = \emptyset$;

ii.) $Z_o \notin \delta(a,Z)$ for $a \in \Sigma \cup\{\lambda\}$, $Z \in \Gamma$ such that $\delta(a,Z)$ is defined. (Assume the contrary, then we can introduce a new initial pushdown symbol $\overline{Z_o}$ and take the new machine $\overline{M} = (\Sigma,\Gamma \cup\{Z_o\},\overline{\delta},\overline{Z_o})$, where $\overline{\delta}(a,\overline{Z_o}) = \delta(a,Z_o)$ and $\overline{\delta}(a,Z) = \delta(a,Z)$ for all $Z \in \Gamma$, $a \in \Sigma \cup\{\lambda\}$).

Construct a simple machine M_1 from M and M′ as follows:

Let $M_1 = (\Sigma_1,\Gamma_1,\delta_1,Z_o)$, where:

$\Sigma_1 = \Sigma \cup \Sigma'$

$\Gamma_1 = \Gamma \cup \Gamma'$

and δ_1 is defined so that: for each $a \in \Sigma_1 \cup\{\lambda\}$ $Z \in \Gamma-\{Z_o\}$, $Z' \in \Gamma'$:

1. if $\delta(a,Z_o) = \alpha$, then $\delta_1(a,Z_o) = Z_o'\alpha$;
2. if $\delta(a,Z) = \alpha$, then $\delta_1(a,Z) = \alpha$;
3. if $\delta'(a,Z') = \alpha'$, then $\delta_1(a,Z') = \alpha'$;
4. All other transitions are undefined.

We now prove

$$L(M_1) = L\,L'$$

Let $w = w_1w_2$, where $w_1 \in \Sigma^*$, $w_2 \in \Sigma'^*$;

$$\begin{cases} w = w_1 w_2 \in L\,L' \\ \text{where } w_1 \in L,\ w_2 \in L' \end{cases} \Leftrightarrow \begin{cases} (w_1, Z_o) \mid_{\overline{M}}^{*} (\lambda,\lambda) \text{ , and} \\ (w_2, Z_o') \mid_{\overline{M}'}^{*} (\lambda,\lambda) \end{cases}$$

$$\Leftrightarrow \{(w_1 w_2, Z_o) \mid_{\overline{M}_1}^{*} (w_2, Z_o') \mid_{\overline{M}_1}^{*} (\lambda,\lambda)\}$$

$$\Leftrightarrow \quad w \in L(M_1)$$

Hence the result.

Now we consider $\mathcal{L}_{es}$

Let $L = L(M)$ and $L' = L(M')$, where

$M = (\Sigma, \Gamma, \delta, Z_o)$, $M' = (\Sigma', \Gamma', \delta', Z_o')$ are es-machines.

Similarly, we may assume again that

i.) $\Gamma \cap \Gamma = \emptyset$ and

ii.) if $\delta(a,Z) = (\alpha, i)$, where $a \in \Sigma$, $Z \in \Gamma$, $i \in \{0,1\}$ then $Z_o \notin \alpha$

Construct an es-machine M_2 from M and M', as follows:

$M_2 = (\Sigma_2, \Gamma_2, \delta_2, Z_o)$, where

$\Sigma_2 = \Sigma \cup \Sigma'$

$\Gamma_2 = \Gamma \cup \Gamma'$

and δ_2 is such that: for each $a \in \Sigma$, $Z \in \Gamma - \{Z_o\}$, $Z' \in \Gamma'$:

1.) if $\delta(a,Z_o) = (\alpha, i)$, where $\alpha \in \Gamma^*$, $i \in \{0,1\}$, then $\delta_2(a,Z_o) = (Z_o'\alpha, i)$;

2.) if $\delta(a,Z) = (\alpha, i)$, where $\alpha \in \Gamma^*$, $i \in \{0,1\}$, then $\delta_2(a,Z) = (\alpha, i)$;

3.) if $\delta'(a,Z') = (\alpha', i)$, where $\alpha' \in \Gamma'^*$, $i \in \{0,1\}$, then $\delta_2(a,Z') = (\alpha', i)$;

4. All other transition are undefined.

It is easily seen that $L(M_2) = L\,L'$.

4. SIMPLE DETERMINISTIC PUSHDOWN MACHINE

Let us consider the standard definition of a deterministic pushdown automaton (abbreviated as dp-machine) [2].
That is:

Let $M = (K,\Sigma,\Gamma,\delta,q_o,Z_o,H)$, where:

Σ is a finite set of input symbols,

K is a finite set of states,

Γ is a finite set of pushdown symbols,

q_o is an initial state (an element of K),

Z_o is an initial pushdown symbol (an element of Γ),

H is a subset of K (the set of final states), and

δ is a mapping from $K \times (\Sigma \cup \{\lambda\}) \times \Gamma$ to $K \times \Gamma^*$, and satisfy:

for each $q \in K$, $Z \in \Gamma$: either $\delta(q,a,Z)$ contains exactly one element for $a \in \Sigma$ and $\delta(a,\lambda,Z)$ is undefined; or $\delta(q,\lambda,Z)$ contains exactly one element and $\delta(a,q,Z)$ is undefined for all $a \in \Sigma$.

Notation

Given a dp-machine M, let $\vdash_M^*$ or $\vdash^*$ when M is understood, be the relation on $K \times \Sigma^* \times \Gamma^*$ defined as follows:
for $q,p \in K$, $a \in \Sigma \cup \{\lambda\}$, $Z \in \Gamma$, $\alpha, \beta \in \Gamma^*$:
let $(q,aw,\alpha Z) \vdash (p,w,\alpha\beta)$ iff $\delta(q,a,Z) = (p,\beta)$, and define $\vdash_M^*$ to be transitive, closure of $\vdash_M$.

Finally, we define the language accepted by M as:

$$L(M) = \{w \in \Sigma^* / (q_o,w,Z_o) \vdash_M^* (p,\lambda,\alpha), \quad \text{where } p \in H, \ \alpha \in \Gamma^*\}$$

Definition (4.1).

a.) A dp-machine $M = (K,\Sigma,\Gamma,\delta,q_o,Z_o,H)$ is called a simple (abbreviated as sdp - machine) if and only if the δ is a mapping from $(K-H) \times (\Sigma \cup \{\lambda\}) \times \Gamma$ to $K \times \Gamma^*$

b.) An input string is accepted by the sdp-machine M when the entire tape has been processed and the momentary state is an accepting state, and the pushdown store is empty. That is,

$L(M) = \{w \in \Sigma^* / (q_o,w,Z_o) \vdash_M^* (p,\lambda,\lambda)$ for some $p \in H\}$

A language L is said to be <u>simple deterministic context-free</u> (abbreviated as sdc-language) if $L = L(M)$, for some sdp-machine M. Finally, the family of all sdc-languages is denoted by $\mathcal{L}_{sd2}$. It is easily seen that: if $L \in \mathcal{L}_{sd2}$, then L must be prefix-free.

<u>Theorem (4.2)</u>

a.) For every es-machine M there is a sdp-machine M' such that $L(M') = L(M)$.

b.) There is a sdp-machine M_1 such that $L(M_1)$ is not an es-language.

<u>Proof</u>

a.) Let $M = (\Sigma,\Gamma,\delta,Z_o)$ be an es-machine. Without loss of generality we may assume again that:

if $\delta(a,Z) = (\alpha,i)$, for $a \in \Sigma$, $Z \in \Gamma$, $i \in \{0,1\}$,

then $Z_o \notin \alpha$

Construct a sdp-machine M' from M, as follows:

let $M' = (K,\Sigma,\Gamma',\delta',q_o,Z_o,H)$, where

$K = \{q_o,q_h\} \cup \{q_a / a \in \Sigma\}$, where

$q_o,q_h \notin \{q_a / a \in \Sigma\}$

$H = \{q_h\}$,

$\Gamma' = \Gamma \cup \{\$\}$ where $\$ \notin \Gamma$

and δ' is such that:

for each $a \in \Sigma$, $Z \in \Gamma - \{Z_o\}$:

1.) if $\delta(a,Z_o) = (\alpha,1)$, then $\delta'(q_o,a,Z_o) = (q_o, \$\alpha)$;

2.) if $\delta(a,Z_o) = (\alpha,0)$, then $\delta'(q_o,a,Z_o) = (q_a, \$\alpha)$;

3.) if $\delta(a,Z) = (\alpha,1)$, then $\delta'(q_o,a,Z) = (q_o,\alpha)$;

4.) if $\delta(a,Z) = (\alpha,0)$, then $\delta'(q_o,a,Z) = (q_a,\alpha)$;

5.) $\delta'(q_a,\lambda,Z) = \delta'(q_o,a,Z)$ for all $a \in \Sigma$, $Z \in \Gamma-\{Z_o\}$ such that $\delta'(q_o,a,Z)$ is defined;

6.) $\delta'(q_o,\lambda,\$) = (q_h,\lambda)$;

7.) all other transitions are undefined.

It is easily seen that: for $a \in \Sigma$, $w \in \Gamma^*$, $Z \in \Gamma-\{Z_o\}$:

$(aw,\alpha Z) \vdash_M^* (w,\beta) \Leftrightarrow (q_o,aw,\$\alpha Z) \vdash_{M'}^* (q_o,w,\$\beta)$.

Hence, we can easily prove by induction on the length of w:

$(w,Z_o) \vdash_M^* (\lambda,\alpha) \Leftrightarrow (q_o,w,Z_o) \vdash_{M'}^* (q_o,\lambda,\$\alpha)$

Thus, $w \in L(M) \Leftrightarrow (w,Z_o) \vdash_M^* (\lambda,\lambda)$

$\Leftrightarrow (q_o,w,Z_o) \vdash_{M'}^* (q_o,\lambda,\$) \vdash_{M'} (q_h,\lambda,\lambda)$

$\Leftrightarrow w \in L(M')$

Hence, the results

b. To show part (b), we reconsider the non es-language, first seen in [3]

$$L = \{a^i b a^i b / i \geq 1\} \cup \{a^i c a^i c / i \geq 1\}$$

We now provide a sdp machine M_1 that accept this language:

Let $M_1 = (K,\{a,b,c\},\Gamma,\delta_1,q_o,Z_o,H)$, where

$K = \{q_o,q_b,q_c,q_h\}$,

$H = \{q_h\}$,

$\Gamma = \{Z_o,A\}$, and

δ is defined so that

1.) $\delta\delta_1(q_o,a,Z_o) = (q_o,AAA)$

2.) $\delta\delta_1(q_o,a,A) = (q_o,AA)$

3a.) $\delta\delta_1(q_o,b,A) = (q_b,\lambda)$

3b.) $\delta\delta_1(q_o,c,A) = (q_c,\lambda)$

4a.) $\delta\delta_1(q_b,a,A) = (q_b,\lambda)$

4b.) $\delta\delta_1(q_c,a,A) = (q_c,\lambda)$

5a.) $\delta\delta_1(q_b,b,A) = (q_h,\lambda)$

5b.) $\delta\delta_1(q_c,c,A) = (q_h,\lambda)$

By the prefix-free property of a sdc-language, we can easily see following:

Lemma (4.3).

If L is a sdc-language, the $\bar{L}$ is not a sdc-language. Hence, the family $\mathcal{L}_{sd2}$ is not closed under complementation.

By theorem (4.2) and the proof of theorem (3.8), we can easily prove following

Lemma (4.4).

The family $\mathcal{L}_{sd2}$ is not closed under intersection.

By theorems (3.2),(3.4),(4.2) and the proof of theorem (2.5), we can easily see following.

Lemma (4.5).

The family $\mathcal{L}_{sd2}$ is not closed under union.

Lemma (4.6).

The family $\mathcal{L}_{sd2}$ is closed under concatenation.

Proof

Assume that L,L′ are accepted by two sdp-machines:

$M = (K,\Sigma,\Gamma,\delta,q_o,Z_o,H)$

$M' = (K',\Sigma',\Gamma',\delta',q_o',Z_o',H')$.

Whithout loss of generality, we may assume again that:

i.) $K \cap K' = \Gamma \cap \Gamma' = \emptyset$ and

ii.) For each $q \in K$, $q \in \Sigma \cup\{\lambda\}$, $Z \in \Gamma$: if

$\delta(q,a,Z) = (p,\alpha)$, then $Z_o \notin \alpha$.

We now construct a sdp-machine M_1 from M and M′, as follows;

$M_1 = (K_1,\Sigma_1,\Gamma_1,\delta_1,q_o,Z_o,H')$, where

$K_1 = K \cup K'$,

$\Sigma_1 = \Sigma \cup \Sigma'$,

$\Gamma_1 = \Gamma \cup \Gamma'$, and δ_1 is such that:

for each $a \in \Sigma_1 \cup \{\lambda\}$, $q \in K-H$, $q' \in K'-H'$, $Z \in \Gamma-\{Z_o\}$, $Z' \in \Gamma'$:

1.) if $\delta(q_o,a,Z_o) = (p,\alpha)$ then $\delta_1(q_o,a,Z_o) = (p,Z'_o\alpha)$

2.) if $\delta(q,a,Z) = (p,\alpha)$, then $\delta_1(q,a,Z) = (p,\alpha)$

3.) if $\delta'(q',a,Z') = (p',\alpha')$, then $\delta_1(q',a,Z') = (p',\alpha')$:

4.) For all $p \in H$: $\delta_1(p,\lambda,Z'_o) = (q'_o,Z'_o)$;

5.) All other transitions are undefined.

Prove: $L(M_1) = L\,L'$,

let $w = w_1w_2$, where $w_1 \in \Sigma^*$, $w_2 \in \Sigma^*$.

$w = w_1w_2 \in LL'$, where $w_1 \in L$, $w_2 \in L'$

$$\Leftrightarrow \begin{cases} (q_o,w_1,Z_o) \vdash^*_M (p,\lambda,\lambda), \text{ for } p \in H, \text{ and} \\ (q'_o,w_2,Z'_o) \vdash^*_{M'} (p',\lambda,\lambda), \text{ for } p' \in H' \end{cases}$$

$$\Leftrightarrow \{(q_o,w_1w_2,Z_o) \vdash^*_{M_1} (p,w_2,Z'_o) \vdash_{M_1} (q'_o,w_2,Z'_o) \vdash^*_{M_1} (p',\lambda,\lambda), \text{for } p' \in H'\}$$

$$\Leftrightarrow w \in L(M_1)$$

Hence the result.

Theorem (4.7).

The intersection of a sdc-language L and a sfd-language L' is a sdc-language.

Proof

Let $L = L(M)$, $L' = L(M')$, where:

$M = (K,\Sigma,\Gamma,\delta,q_o,Z_o,H)$ is a sdp-machine,

$M' = (K',\Sigma,\delta',q'_o,H')$ is a sfd-machine.

Construct a sdp-machine M_2 from M and M', as follows:

Let $M_2=(K_2,\Sigma,\Gamma,\delta_2,[q_o,q'_o],Z_o,H_2)$, where

$K_2 = \{[q,q']/\ q \in K \text{ and } q' \in K'\}$,

$H_2 = \{[p,p']/\ p \in H \text{ and } p' \in H'\}$,

and δ_1 is defined so that:

for each $a \in \Sigma$, $q \in K-H$, $q' \in K'-H'$, $Z \in \Gamma$

1.) $\delta_2([q,q'],a,Z) = ([p,p'],\alpha)$ iff $\begin{cases}\delta(q,a,Z) = (p,\alpha) \\ \delta'(q',a) = p'\end{cases}$ and

2.) $\delta_2([q,q'],\lambda,Z) = ([p,q'],\alpha)$ iff $\delta(q,\lambda,Z) = (p,\alpha)$

3.) All other transitions are undefined.

It is easily seen that for each $a \in \Sigma$, $q \in K$, $q' \in K'$:

$$([q,q'],a,\alpha) \vdash_{M_2}^{*} ([p,p'],\lambda,\beta) \iff \begin{cases}(q, a ,\alpha) \vdash_{M}^{*} (p,\lambda,\beta), \\ \text{and } q'a \vdash_{M'}^{*} p'\end{cases}$$

Hence, we can easily prove by induction on the length of w:

$$([q_o,q_o'],w,Z_o) \vdash_{M_2}^{*} ([p,p'],\lambda,\alpha) \iff \begin{cases}(q_o,w,Z_o) \vdash_{M}^{*} (p,\lambda,\alpha), \\ \text{and } q_o'w \vdash_{M'}^{*} p'\end{cases}$$

$$\text{Thus, } w \in L \cap L' \iff \begin{cases}(q_o,w,Z_o) \vdash_{M}^{*} (p,\lambda,\lambda), & \text{for } p \in H, \\ \text{and } q_o'w \vdash_{M'}^{*} p' , & \text{for } p' \in H'\end{cases}$$

$$\iff \{([q_o,q_o'],w,Z_o) \vdash_{M_2}^{*} ([p,p'],\lambda,\lambda), \text{ for } [p,p'] \in H_2\}$$

$$\iff w \in L(M_2) .$$

Hence the result.

Theorem (4.8).

Every sdc-language L can be expressed in the form $L = h(L_1 \cap L_2)$, where h is a homomorphism, and L_1 is a sfd-language, L_2 is a sc-language.

Proof

Assume that L is accepted by spd-machine:

$M = (K,\Sigma,\Gamma,\delta,q_o,Z_o,H)$.

First, construct an auxiliary spd-machine M' from M as follows:

Let $M' = (K,\Sigma',\Gamma,\delta',q_o,Z_o,H)$, where

$\Sigma' = \Sigma \cup \Sigma_1$ for $\Sigma_1 = \{e_{[q,Z]} / q \in (K-H), Z \in \Gamma$ such that $\delta(q,\lambda,Z)$ is defined$\}$, and

δ' is such that:

for each $a \in \Sigma$, $q \in (K-H)$, $Z \in \Gamma$

1.) $\delta'(q,a,Z) = (p,\alpha)$ iff $\delta(q,a,Z) = (p,\alpha)$

2.) if $(q,\lambda,Z) = (p,\alpha)$, then $\delta'(q,e_{[q,Z]},Z) = (p,\alpha)$;

3.) All other transitions are undefined.

It is easily seen that:

i.) For all $q \in K$, $Z \in \Gamma$: $\delta'(q,\lambda,Z)$ is undefined

ii.) $L = h_1(L(M'))$ for h_1 is a homomorphism of Σ' into Σ such that:

$$h_1(a) = \begin{cases} a & \text{if } a \in \Sigma \\ \lambda & \text{if } a \in \Sigma_1 \end{cases}$$

We now construct two machines M_1, M_2 from M', as follows:

Let $M_1 = (K_2, \Sigma_2, \delta_1, q_o, H)$, and

$M_2 = (\Sigma_2, \Gamma_2, \delta_2, Z_o)$, where:

$\Sigma_2 = \Sigma' \cup \Sigma''$, for $\Sigma'' = \{\varkappa_{[q,a]} / q \in (K-H), Z \in \Gamma\}$

and $\Sigma' \cap \Sigma'' = \emptyset$

$K_2 = K \cup \{[q,Z] / q \in (K-H), Z \in \Gamma\}$

$\Gamma_2 = \Gamma \cup \{[q,Z] / q \in (K-H), Z \in \Gamma\}$

and δ_1, δ_2 are such that:

for each $a \in \Sigma'$, $q \in (K-H)$, $Z \in \Gamma$:

1.) $\delta_1(q, \varkappa_{[q,Z]}) = [q,Z]$

$\delta_2(\varkappa_{[q,Z]}, Z) = [q,Z]$

2.) if $\delta'(q,a,Z) = (p,\alpha)$, then $\begin{cases} \delta_1([q,Z],a) = p, & \text{and} \\ \delta_2(a,[q,Z]) = \alpha \end{cases}$

3./ All other transitions are undefined.

It is easily seen that,

M_1 is a sfd-machine,

M_2 is a simple machine

Let h_2 be the homomorphism of Σ_2 into Σ' defined by

$$h_2(a) = \begin{cases} a & \text{if } a \in \Sigma' \\ \lambda & \text{if } a \in \Sigma'' \end{cases}$$

We now prove: $L(M') = h_2(L_1 \cap L_2)$, where $L_1 = L(M_1)$ and $L_2 = L(M_2)$

$$\Leftrightarrow \begin{cases} \text{there exist } u_1,\ldots,u_n \in \Sigma'' \text{ such that} \\ q_o u_1 a_1 \ldots u_n a_n \vdash_{M_1}^{*} p, \text{ for } p \in H, \text{ and} \\ (u_1 a_1 \ldots u_n a_n, Z_o) \vdash_{M_2}^{*} (\lambda,\lambda) \end{cases}$$

$$\Leftrightarrow \{\text{There exist } u_i,\ldots,u_n \in \Sigma'' \text{ such that } u_1 a_1 \ldots u_n a_n \in \in L_1 \cap L_2\}$$

Thus, $L(M') = h_2 (L_2 \cap L_2)$.

We now can easily see that $L = h(L_1 \cap L_2)$, for h is a homomorphism of Σ_2 into Σ, such that:

$$h(a) = \begin{cases} a & \text{if } a \in \Sigma \\ \lambda & \text{if } a \in \Sigma_1 \cup \Sigma'' . \end{cases}$$

Hence, the result.

By theorem (4.8) and theorem (3.2), we can easily see the following:

Corollary (4.9)

Every scd-language L can be expressed in the form $L = h(L_1 \cap L_2)$, where h is a homomorphism, and L_1, L_2 are two sc-languages.

5. TWO-MEMORY SIMPLE MACHINE

Definition (5.1).

a.) A two-memory simple machine (abbreviated as ts-machine) is a 6-tuple:
$M = (\Sigma, \Gamma, \Gamma', \delta, Z_o, Z'_o)$, where
Σ is a finite set of inputs,
Γ and Γ' are two finite sets of pushdown symbols,
$Z_o \in \Gamma$ and $Z'_o \in \Gamma'$ are two initial symbols of two pushdow stores, and
δ is a mapping from $\Gamma \times (\Sigma \cup \{\lambda\}) \times \Gamma'$ to $\Gamma^* \times \Gamma'^*$, and satisfy: for each $Z \in \Gamma$, $Z \in \Gamma'$ either $\delta(Z,a,Z')$ contains exactly one element for $a \in \Sigma$, and $\delta(Z,\lambda,Z')$ is undefined; or $\delta(Z,\lambda,Z')$ contains exactly one element and $\delta(Z,a,Z')$ is undefined for all $a \in \Sigma$.

b.) A configuration of M is a triple (α,w,α'), where:
$w \in \Sigma^*$ is the portion of the input tape remaining to be read.
$\alpha \in \Gamma^*$ (similarly, $\alpha' \in \Gamma'^*$) is the current contents of the pushdown store $[\Gamma]$ (and $[\Gamma']$), where the top of the store $[\Gamma]$ (and $[\Gamma']$) is the rightmost symbol of α (and α')

c.) We define the operator $\vdash_M$ on configurations of M as follows: for each $a \in \Sigma$, $w \in \Sigma^*$, $Z \in \Gamma$, $Z' \in \Gamma'$, α, $\beta \in \Gamma^*$, $\alpha', \beta' \in \Gamma'^*$:
i) $(\alpha Z, aw, \alpha' Z') \vdash_M (\alpha\beta, w, \alpha'\beta')$ iff $\delta(Z,a,Z') = (\beta,\beta')$,
or
ii) $(\alpha Z, aw, \alpha' Z') \vdash_M (\alpha\beta, aw, \alpha'\beta')$ iff $\delta(Z,\lambda,Z') = (\beta,\beta')$
Let $\vdash_M^*$ denote the transitive, closure of $\vdash_M$.

d.) We shall be concerned with acceptance of an input tape by empty pushdown stores. Accordingly, we define

the language accepted by a ts-machine M to be:

$$L(M) = \{W \in \Sigma^* / (Z_o, w, Z_o') \vdash_M^* (\lambda, \lambda, \lambda)\}$$

A language L is said to be a <u>two-memory simple context--free language</u> (<u>abbreviated as ts-language</u>) if L = L(M), for some ts-machine M. The family of all ts-languages is denoted $\mathcal{L}_{ts}$. It is easily seen that: if L is a ts--language, then L must be prefix-free.

<u>Theorem (5.2)</u>

a.) For every sdp-machine M there is a ts-machine M' such that L(M') = L(M).

b.) There is a ts-machine M_1 such that $L(M_1)$ is not a sdc--language.

<u>Proof</u>

a.) Let $M = (K, \Sigma, \Gamma, \delta, q_o, Z_o, H)$ be a sdp-machine. Without loss of generality, we may assume again that: for $q \in K$, $a \in \Sigma \cup \{\lambda\}$, $Z \in \Gamma$, if $\delta(q,a,Z) = (p,\alpha)$, then $q_o \neq p$ and $Z_o \notin \alpha$

Construct a ts-machine M' from M as follows:

Let $M' = (\Sigma, K, \Gamma', \delta', q_o, Z_o)$, where

$\Gamma' = \Gamma \cup \{\$\}$, for $\$ \notin \Gamma$, and

δ' is defined so that:

for each $a \in \Sigma \cup \{\lambda\}$, $q \in (K-H-\{q_o\})$, $Z \in (\Gamma - \{Z_o\})$:

1.) $\delta'(q_o,a,Z_o) = (p,\$\alpha)$ iff $(q_o,a,Z_o) = (p,\alpha)$;

2.) $\delta'(q,a,Z) = (p,\alpha)$ iff $(q,a,Z) = (p,\alpha)$;

3.) $\delta'(p,\lambda,\$) = (\lambda,\lambda)$ for all $p \in H$;

4.) All other transitions are undefined.

We can easily see that:

$W \in L(M) \Leftrightarrow (q_o,w,Z_o) \vdash_M^* (p,\lambda,\lambda)$, for $p \in H$

$\Leftrightarrow (q_o,w,Z_o) \vdash_{M'}^* (p,\lambda,\$) \vdash_{M'} (\lambda,\lambda,\lambda)$

$\Leftrightarrow$ $w \in L(M')$.

Hence, the result.

b.) To show part (b), we prove the following lemma:

Lemma (5.3).

There is a ts-machine M_1 such that $L(M_1)$is not context--free

Proof of lemma (5.3)

We consider the non context-free language, first seen in [4]:

$$L = \{a^n b^n c^n / n \geq 1\} .$$

We now provide a ts-machine M_1 that accepts this language:

Let $M_1 = (\Sigma, \Gamma, \Gamma', \delta_1, Z_o, Z_o')$, where

$\Sigma = \{a,b,c\}$,

$\Gamma = \{Z_o, A, \$\}$

$\Gamma' = \{Z_o', B, \$\}$ and

δ_1 = is such that:

1.) $\delta_1(Z_o, a, Z_o') = (\$AA, \$)$

2.) $\delta_1(A, a, \$) = (AAA, \$)$

3.) $\delta_1(A, b, \$) = (\lambda, \$B)$

4.) $\delta_1(A, b, B) = (\lambda, BB)$

5.) $\delta_1(A, c, B) = (\lambda, \lambda)$

6.) $\delta_1(\$, \lambda, \$) = (\lambda, \lambda)$

Thus, there is a ts-machine M_1 such that $L(M_1)$ is not context--free, and evidently is not a sdc-language, Hence, the result.

Given a ts-machine M, let $\delta(Z, a, Z') = (\alpha, \beta)$, where $a \in \Sigma \cup \{\lambda\}$, $Z \in \Gamma$, $Z' \in \Gamma'$, $\alpha \in \Gamma^*$, $\beta \in \Gamma^*$, we write:

$$Val_f\delta(Z,a,Z') = \alpha$$

$$Val_s\delta(Z,a,Z') = \beta$$

Theorem (5.4).

Let $M = (\Sigma,\Gamma,\Gamma',\delta,Z_o,Z'_o)$ be a ts-machine satisfying the following condition:

"$|Val_f\delta(Z,a,Z')| = |Val_s\delta(Z,a,Z')|$, for $a \in \Sigma \cup\{\lambda\}$, $Z \in \Gamma$ $Z' \in \Gamma'$ such that $\delta(Z,a,Z')$ is defined", (5.4)

then L(M) is a sc-language.

Proof

Let $M = (\Sigma,\Gamma,\Gamma',\delta,Z_oZ'_o)$ be a ts-machine satisfying the condition (5.4), where

$$\Gamma = \{Z_o,Z_1,\ldots,Z_n\},$$

$$\Gamma' = \{Z'_o,Z'_1,\ldots,Z'_m\} .$$

Construct a simple machine M' from M as follows:

Let $M' = (\Sigma,\bar{\Gamma},\delta',Z_{(o,o)})$, where:

$$\bar{\Gamma} = \{Z_{(i,j)}/i \in \{0,1,\ldots,n\} \text{ and } j \in \{0,1,\ldots,m\}\} ,$$

and δ' is such that: for each $a \in \Sigma \cup\{\lambda\}$:

1.) if $\delta(Z_i,a,Z'_j) = (Z_{il}\ldots Z_{ik},Z'_{jl}\ldots Z'_{jk})$, where $k \geq 1$ then $\delta'(a,Z_{(i,j)}) = Z_{(il,jl)}\ldots Z_{(ik,jk)}$;

2.) $\delta(Z_i,a,Z'_j) = (\lambda,\lambda)$, then $\delta'(a,Z_{(i,j)}) = \lambda$

3.) All other conditions are undefined.

It is easily seen that $L(M') = L(M)$

Hence the result.

By the prefix-free property of a ts-language, we can easily see the following:

Lemma (5.5)

If L is a ts-language, then $\bar{L}$ is not a ts-language. Hence, the family $\mathcal{L}_{ts}$ is not closed under complementation. By theorems (3.2),(3.4),(4.2), (5.2) and the proof of theorem (2.5), we can easily prove the following:

Lemma (5.6)

The family $\mathcal{L}_{ts}$ is not closed under union.

Lemma (5.7)

The family $\mathcal{L}_{ts}$ is closed under concatenation.

Proof

Assume that $L = L(M)$ and $L' = L(M')$, where:

$M = (\Sigma, \Gamma, \Gamma', \delta, Z_o, Z'_o)$, and

$M' = (\Sigma', \bar{\Gamma}, \bar{\Gamma}', \delta', \bar{Z}_o, \bar{Z}'_o)$ are ts-machines.

Without loss of generality, we may assume again that:

i.) $\Gamma \cap \bar{\Gamma} = \Gamma \cap \bar{\Gamma}' = \emptyset$

ii.) $Z_o \notin Val_f \delta(Z,a,Z')$, $Z'_o \notin Val_s \delta(Z,a,Z')$, for $a \in \Sigma \cup \{\lambda\}$, $Z \in \Gamma$, $Z' \in \Gamma'$ such that $\delta(Z,a,Z')$ is defined.

Construct a ts-machine M_1 from M and M', as follows:

Let $M_1 = (\Sigma_1, \Gamma_1, \Gamma'_1, \delta_1, Z_o, Z'_o)$, where

$\Sigma_1 = \Sigma \cup \Sigma$

$\Gamma_1 = \Gamma \cup \bar{\Gamma}$

$\Gamma'_1 = \Gamma' \cup \bar{\Gamma}'$; and

δ_1 is such that, for each $a \in \Sigma_1 \cup \{\lambda\}$, $Z \in \Gamma - \{Z_o\}$, $Z' \in \Gamma' - \{Z'_o\}$, $\bar{Z} \in \bar{\Gamma}$, $\bar{Z}' \in \bar{\Gamma}'$

1.) if $\delta(Z_o,a,Z'_o) = (\alpha,\beta)$, then $\delta_1(Z_o,a,Z'_o) = (\bar{Z}_o\alpha, \bar{Z}'_o\beta)$

2.) if $\delta(Z,a,Z') = (\alpha,\beta)$, then $\delta_1(Z,a,Z') = (\alpha,\beta)$

3.) if $\delta'(\bar{Z},a,\bar{Z}') = (\bar{\alpha},\bar{\beta})$, then $\delta_1(\bar{Z},a,\bar{Z}') = (\bar{\alpha},\bar{\beta})$;

4.) All other transitions are undefined.

It is easily seen that, let $w = w_1w_2$, where $w_1 \in \Sigma^*$, $w_2 \in \Sigma^*$,

$$\{w = w_1w_2 \in LL', \text{ where } w_1 \in L,\ w_2 \in L'\} \Leftrightarrow \begin{cases} (Z_o,w_1,Z_o') \mid \overset{*}{\overline{M}}(\lambda,\lambda,\lambda), \text{ and} \\ (\bar{Z}_o,w_2,\bar{Z}_o') \mid \overset{*}{\overline{M}}(\lambda,\lambda,\lambda) \end{cases}$$

$$\Leftrightarrow \{(Z_o,w_1\ w_2,Z_o') \mid \overset{*}{\overline{M}_1}(\bar{Z}_o,w_2,\bar{Z}_o') \mid \overset{*}{\overline{M}_1}(\lambda,\lambda,\lambda)\}$$

$$\Leftrightarrow \quad w = w_1w_2 \in L(M_1).$$

Hence the result.

Theorem (5.8)

The intersection of a ts-language L and a sfd-language L' is a ts-language.

Proof

Let $L = L(M)$, and $L' = L(M')$, where

$M = (\Sigma,\Gamma,\Gamma',\delta,Z_o,Z_o')$ is a ts-machine,

$M' = (K,\Sigma,\delta',q_o,H)$ is a sfd-machine.

Without loss of generality, we may assume again, that:

i.) $K \cap \Gamma = \emptyset$ and

ii.) $Z_o \notin \mathrm{Val}_f\delta(Z,a,Z')$, $Z_o' \notin \mathrm{Val}_s\delta(Z,a,Z')$, for $a \in \Sigma \cup \{\lambda\}$, $Z \in \Gamma$, $Z' \in \Gamma'$ such that $\delta(Z,a,Z')$ is defined.

Construct a ts-machine M_2 from M and M', as follows:

Let $M_2 = (\Sigma,\Gamma_2,\Gamma_2',\delta_2,Z_o,Z_o')$, where

$\Gamma_2 = \Gamma \cup K$

$\Gamma_2' = \Gamma' \cup \{[q,Z'] / q \in K \text{ and } Z' \in \Gamma'\}$ and

δ_2 is such that:

for each $a \in \Sigma$, $Z \in \Gamma - \{Z_o\}$, $Z' \in \Gamma' - \{Z_o\}$, $\alpha \in \Gamma^*$ $\beta \in \Gamma'^*$ $q \in K-H$:

1.) if $\delta(Z_o,\lambda,Z_o') = (\alpha,\beta)$, then $\delta_2(Z_o,\lambda,Z_o') = (\alpha q_o,\beta)$

2.) if $\begin{cases}\delta(Z_o,a,Z_o') = (\alpha,\beta), \\ \text{and } \delta'(q_o,a) = p\end{cases}$ then $\delta_2(Z_o,a,Z_o') = (\alpha\bar{p},\beta)$,

where $\bar{p} = \begin{cases} p & \text{if } p \notin H \\ \lambda & \text{if } p \in H \end{cases}$

3.) $\delta_2(q,\lambda,Z') = (\lambda,[q,Z'])$, for all $q \in (K-H)$, $Z' \in \Gamma'$;

4.) if $\delta(Z,\lambda,Z') = (\alpha,\beta)$, then $\delta_2(Z,\lambda,[q,Z']) = (\alpha q,\beta)$, for all $q \in (K-H)$

5.) if $\begin{cases}\delta(Z,a,Z') = (\alpha,\beta), \\ \text{and } \delta'(q,a) = p\end{cases}$, then $\delta_2(Z,a,[q,Z']) = (\alpha\bar{p},\beta)$

where $\bar{p} = \begin{cases} p & \text{if } p \notin H \\ \lambda & \text{if } p \in H \end{cases}$

6.) All other transitions are undefined.

It is easily seen that, for each $a \in \Sigma$, $Z \in \Gamma - \{Z_o\}$, $Z' \in \Gamma' - \{Z_o'\}$, $\alpha, \alpha_1 \in \Gamma^*$, $\beta, \beta_1 \in \Gamma'^*$, $q \in (K-H)$:

$$\begin{cases}(\alpha Z,a,\beta Z') \vdash_M^* (\alpha_1,\lambda,\beta_1), \\ \text{and } qa \vdash_{M'}^* p\end{cases} \Leftrightarrow \{(\alpha Z,a,\beta[q,Z']) \vdash_{M_2}^* (\alpha_1\bar{p},\lambda,\beta_1)\} \quad (5.8.1)$$

where $\bar{p} = \begin{cases} p & \text{if } p \notin H \\ \lambda & \text{if } p \in H \end{cases}$

Hence, we prove by induction on the length of w:

$$\begin{cases}(Z_o,w,Z_o') \vdash_M^* (\alpha,\lambda,\beta), \\ \text{and } q_o w \vdash_{M'}^* p\end{cases} \Leftrightarrow (Z_o,w,Z_o') \vdash_{M_2}^* (\alpha\bar{p},\lambda,\beta) \quad (5.8.2)$$

where $\bar{p} = \begin{cases} p & \text{if } p \notin H \\ \lambda & \text{if } p \in H \end{cases}$

Indeed,

* A case $w = a \in \Sigma$: evidently, by statement (5.8.1), and the forms (1), (2), (3), (4), (5).

* Assume that statement (5.8.2) is valid for all $w \in \Sigma^*$, such that $|w| < n$.

We now consider the word $w = a_1 \ldots a_{n-1} a_n$, and can write: $w = w_1 a_n$, where $w_1 = a_1 \ldots a_{n-1}$.

Let $\begin{cases} (Z_o, w_1 a_n, Z_o') \mid\!\overset{*}{\overline{M}}\; (\alpha_1 Z, a_n, \beta_1 Z') \mid\!\overset{*}{\overline{M}}\; (\alpha, \lambda, \beta) \\ \text{and } q_o w_1 a_n \mid\!\overset{*}{\overline{M}}\; q a_n \mid\!\overset{*}{\overline{M}}\; p, \quad \text{where } q \in K-H \end{cases}$

Since $|w_1| < n$, thence statement (5.8.2) is true, and we have:

$$\begin{cases} (Z_o, w_1, Z_o') \mid\!\overset{*}{\overline{M}}\; (\alpha_1 Z, \lambda, \beta_1 Z'), \\ \text{and } q_o w_1 \mid\!\overset{*}{\overline{M}}\; q, \text{ where } q \in K-H \end{cases} \iff (Z_o, w_1, Z_o') \mid\!\overset{*}{\overline{M_2}} (\alpha_1 Zq, \lambda, \beta_1 Z').$$

On the other hand, by the form (3) and statement (5.8.1), we can see that,

$$\begin{cases} (\alpha_1 Z, a_n, \beta_1 Z') \mid\!\overset{*}{\overline{M}} (\alpha, \lambda, \beta) \\ \text{and } \quad q a_n \mid\!\overset{*}{\overline{M}}\; p \end{cases} \iff$$

$$\iff \quad (\alpha_1 Zq, a_n, \beta_1 Z') \mid\!\overset{*}{\overline{M_2}}\; \{(\alpha_1 Z, a_n \beta_1 [q, Z']) \mid\!\overset{*}{\overline{M_2}} (\alpha \bar{p}, \lambda, \beta)\} .$$

Thus, statement (5.8.2) holds.

Finally, let $w \in \Sigma^*$

$$w \in L \cap L' \iff \begin{cases} (Z_o, w, Z_o') \mid\!\overset{*}{\overline{M}} (\lambda, \lambda, \lambda), & \text{and} \\ q_o w \mid\!\overset{*}{\overline{M}}\; p, \text{ for } p \in H \end{cases}$$

$$\iff \quad (Z_o, w, Z_o') \mid\!\overset{*}{\overline{M_2}}\; (\bar{p}, \lambda, \lambda), \quad \text{and } \bar{p} = \lambda$$

$$\iff \quad w \in L(M_2)$$

Hence, the result.

Theorem (5.9)

Every ts-language L can be expressed in the form $L = h(L_2 \cap L_2)$, where h is a homomorphism and L_1, L_2 are sc--languages.

Proof

Assume that L is accepted by a ts-machine

$M = (\Sigma, \Gamma, \Gamma', \delta, Z_o, Z_o')$.

Contruct two simple machines M_1, M_2 from M, as follows:

$M_1 = (\Sigma', \Gamma_1, \delta_1, Z_o)$,

$M_2 = (\Sigma', \Gamma_2, \delta_2, Z_o')$, where

$\Sigma' = \Sigma \cup \Sigma_1$ for $\Sigma_1 = \{\chi_{[Z,Z']} / Z \in \Gamma, Z' \in \Gamma'\}$,

$\Gamma_1 = \Gamma \cup \{[Z,Z'] / Z \in \Gamma, Z' \in \Gamma'\}$

$\Gamma_2 = \Gamma' \cup \{[Z,Z'] / Z \in \Gamma, Z' \in \Gamma'\}$, and δ_1, δ_2 are such that: for each $a \in \Sigma \cup \{\lambda\}$, $Z \in \Gamma$, $Z' \in \Gamma'$:

1.) $\delta_1(\chi_{[Z,Z']}, Z) = [Z,Z']$
$\delta_2(\chi_{[Z,Z']}, Z') = [Z,Z']$ for all $Z \in \Gamma$, $Z' \in \Gamma'$

2.) if $\delta(Z,a,Z') = (\alpha,\beta)$, then $\begin{cases} \delta_1(a,[Z,Z']) = \alpha \\ \delta_2(a,[Z,Z']) = \beta \end{cases}$

3.) All other transitions are undefined.

Let h be the homomorphism of Σ' into Σ, defined by:

$$h(a) = \begin{cases} a & \text{if } a \in \Sigma \\ \lambda & \text{if } a \in \Sigma_1 \end{cases}$$

We now prove: $L(M) = h(L_1 \cap L_2)$, where

$$L_1 = L(M_1) \quad \text{and} \quad L_2 = L(M_2).$$

First, we can easily see the following:
for $a \in \Sigma$, $Z \in \Gamma$, $Z' \in \Gamma'$, $\alpha, \alpha_1 \in \Gamma^*$, $\beta, \beta_1 \in \Gamma'^*$:

Let $w = a_1 \ldots a_n \in \Sigma'^*$, we prove by induction on the length of w:

$$(q_o, a_1 \ldots a_n, Z_o) \vdash_{\overline{M}}^{*} (p, \lambda, \alpha) \Leftrightarrow \begin{cases} \text{there exist } u_1 \ldots u_n \in \Sigma'', \text{such that} \\ q_o u_1 a_1 \ldots u_n a_n \vdash_{\overline{M_1}}^{*} p, \quad \text{and} \\ (u_1 a_1 \ldots u_n a_n, Z_o) \vdash_{\overline{M_2}}^{*} (\lambda\ \alpha) \end{cases} \quad (4.8)$$

Indeed,

A case $w = a \in \Sigma'$; is easily seen that:

$$(q_o, a, Z_o) \vdash_{\overline{M}'} (p, \lambda, \alpha) \quad \text{iff} \quad \begin{cases} \text{there exist } \varkappa_{[q_o Z_o]} \in \Sigma'', \text{ and} \\ q_o \varkappa_{[q_o Z_o]} a \vdash_{\overline{M_1}} [q_o, Z_o] a \vdash_{\overline{M_1}} p \quad \text{and} \\ (\varkappa_{[q_o, Z_o]} a, Z_o) \vdash_{\overline{M_2}} (a, [q_o, Z_o]) \vdash_{\overline{M_2}} (\lambda\, \alpha) \end{cases}$$

Assume, that the statement (4.8) is valid for all $w \in \Sigma'^*$, such that $|w| < n$. We now consider a word $w = w_1 a_n$, where $w_1 = a_1, \ldots, a_{n-1}$.

Let $(q_o, w_1 a_n, Z_o) \vdash_{\overline{M}'}^{*} (q, a_n, \alpha_1 Z) \vdash_{\overline{M}'}^{*} (p, \lambda, \alpha)$, where

$q \in (K-H)$ and $\delta'(q, a_n, Z) = (p, \beta)$, and $\alpha = \alpha_1 \beta$

Since $|w_1| < n$, we have

$$(q_o, w_1, Z_o) \vdash_{\overline{M}'}^{*} (q, \lambda, \alpha_1 Z) \quad \text{iff} \quad \begin{cases} \text{there exist } u_1, \ldots, u_{n-1} \in \Sigma'' \text{such that} \\ q_o u_1 a_1 \ldots u_{n-1} a_{n-1} \vdash_{\overline{M_1}}^{*} q \text{ , and} \\ (u_1 a_1 \ldots u_{n-1} a_{n-1}, Z_o) \vdash_{\overline{M_2}}^{*} (\lambda, \alpha_1 Z) \end{cases}$$

On the other hand, we can easily see that:

for $q \in (K-H)$, $z \in \Gamma$, there exist $u_n = \varkappa_{[q,Z]}$ such that

$$\begin{cases} q \varkappa_{[q,Z]} a_n \vdash_{\overline{M_1}} [q,Z]\, a_n \vdash_{\overline{M_1}} p \quad \text{and} \\ (\varkappa_{[q,Z]} a_n, Z) \vdash_{\overline{M_2}} (a_n, [q,Z]) \vdash_{\overline{M_2}} \beta \end{cases}$$

Thus, statement (4.8) holds.

Finally, let $w = a_1 \ldots a_n \in \Sigma'^*$:

$$w \in L(M') \Leftrightarrow \{ (q_o, w, Z_o) \vdash_{\overline{M}'}^{*} (p, \lambda, \lambda), \quad \text{for } p \in H \}$$

$$(\alpha Z,a,\beta Z') \vdash_{\overline{M}}^{*} (\alpha_1,\lambda,\beta_1) \Leftrightarrow \begin{cases} \text{there exist } u \in \Sigma_1^*, \text{ such that} \\ (ua,\alpha Z) \vdash_{\overline{M}_1}^{*} (\lambda,\alpha_1), \text{ and} \\ (ua,\beta Z') \vdash_{\overline{M}_2}^{*} (\lambda,\beta_1) \end{cases} \tag{5.9.1}$$

Thence, we prove by induction on the length of w,

let $w = a_1 \ldots a_n$:

$$(Z_o,a_1 \ldots a_n,Z_o') \vdash_{\overline{M}}^{*} (\alpha,\lambda,\beta) \Leftrightarrow \begin{cases} \text{there exist } u_1,\ldots,u_n \in \Sigma_1^*, \\ \text{such that } (u_1a_1 \ldots u_na_n,Z_o) \vdash_{\overline{M}_1}^{*} (\lambda,\alpha), \\ \text{and } (u_1a_1 \ldots u_na_n,Z_o') \vdash_{\overline{M}_2}^{*} (\lambda,\beta) \end{cases} \tag{5.9.2}$$

Indeed,

* A case $w = a \in \Sigma$ evidently

* Assume that statement (5.9.2) is valid for all $w \in \Sigma^*$, such that $|w| < n$. We now consider the word $w = w_1a_n$, where $w_1 = a_1 \ldots a_{n-1}$.

Let $(Z_o w_1 a_n, Z_o') \vdash_{\overline{M}}^{*} (\alpha_1 Z, a_n, \beta_1 Z') \vdash_{\overline{M}}^{*} (\alpha,\lambda,\beta)$.

Since $|w_1| < n$, thence statement (5.9.2) is true, and we have:

$$(Z_o,w_1,Z_o') \vdash_{\overline{M}}^{*} (\alpha_1 Z,\lambda,\beta_1 Z') \Leftrightarrow \begin{cases} \text{there exist } u_1,\ldots,u_{n-1} \in \Sigma_1^*, \\ \text{such that } (u_1a_1 \ldots u_{n-1},Z_o) \vdash_{\overline{M}}^{*} (\lambda,\alpha_1 Z), \\ \text{and } (u_1a_2 \ldots u_{n-1}a_{n-2},Z_o') \vdash_{\overline{M}_2}^{*} (\lambda,\beta_1 Z') \end{cases}$$

On the other hand, by statement (5.9.1), we can see:

$$(\alpha_1 Z,a_n,\beta_1 Z') \vdash_{\overline{M}}^{*} (\alpha,\lambda,\beta) \Leftrightarrow \begin{cases} \text{there exist } u_n \in \Sigma_1^*, \text{ such that} \\ (u_n,a_n,\alpha_1 Z) \vdash_{\overline{M}_1}^{*} (\lambda,\alpha), \text{ and} \\ (u_na_n,\beta_1 Z') \vdash_{\overline{M}_2}^{*} (\lambda,\beta) \end{cases}$$

Thus, statement (5.9.2) holds.

Finally, let $w = a_1 \ldots a_n \in \Sigma^*$:

$$w = a_1 \ldots a_n \in L(M) \Leftrightarrow (Z_o,a_1 \ldots a_n,Z_o') \vdash_{\overline{M}}^{*} (\lambda,\lambda,\lambda) \Leftrightarrow$$

$$\Leftrightarrow \begin{cases} \text{there exist } u_1,\ldots,u_n \in \Sigma^*, \text{ such that} \\ (u_1a_1 \ldots u_na_n,Z_o) \vdash_{\overline{M}_1}^{*} (\lambda,\lambda), \text{and} \\ (u_1a_1 \ldots u_na_n,Z_o') \vdash_{\overline{M}_2}^{*} (\lambda,\lambda) \end{cases}$$

$$\Leftrightarrow \begin{cases} \text{There exist } u_1,\ldots,u_n \in \Sigma'^*, \\ \text{such that } u_1a_1\ldots u_na_n \in L_1 \cap L_2 \end{cases}$$

Thus, $L = h(L_1 \cap L_2)$.

Corollary (5.10)

Every ts-language L can be expressed in the form $L = h(L_1 \cap L_2)$, where h is a homomorphism, and L_1, L_2 are sdc--languages.

REFERENCES

[1] Friedman, E.P.: Simple context-free languages and free monadic recursion schemes. Mathematical Systems Theory. 11 /1977/, 9-28 pp

[2] Ginsburg, S. and Greibach, S.: Deterministic context-free languages. Information and Control 9 /1966/, 62o-648 pp

[3] Friedman, E.P.: Equivalence problems for deterministic context-free languages and monadic recursion schemes. To appear in ICSS

[4] Révész, Gy.: Bevezetés a formális nyelvek elméletébe. Akadémiai Kiadó, Budapest /1979/

[5] Salomaa, A.: Formal languages. ACM Monograph Series, Academic Press, New York-London, /1973/

[6] Korenjac, A.I. and Hopcrooft, I.E.: Simple deterministic languages. IEEE Conference record of 7th Annual Symposium on switching and automata theory /1966/, 36-46 pp

Acknowledgement

I wish to express my thanks to Gy. Révész for his useful comments and for the criticism of the manuscript of this paper.

D. KNOWLEDGE REPRESENTATION

Computational Linguistics and Computer Languages Vol. XIV. 1980.

CONSTRUCTIVE APPROACH TO KNOWLEDGE REPRESENTATION

by

V.M. PONOMAREV and V.V. ALEXANDROV

Research Computer Centre of USSR Academy of Sciences

Leningrad, USSR

ABSTRACT

In the present paper we'd like to draw attention to some new aspects of conception of constructive approach to knowledge representation. The necessity of new approach is called by the particular role of computer system in the field of collecting, representing and application of knowledge. In fact the progressive role of computer is connected first of all with its capacity of storing and mass data manipulating. And here the problem arises: how to get quickly and reliably scientific information, how to organize synthesis of new formal knowledge based on former information and how to provide selfdevelopment of intelligent data base. The paper includes two parts. In the first part we discuss some specific questions concerning knowledge representation. The second part is devoted to a study of constructive principles of data representation and processing.

I. MAIN FEATURES OF RISE, DEVELOPMENT AND REPRESENTATION OF KNOWLEDGE

From the point of view of knowledge representation in computer it is important that knowledge is a means of efficient coding (description) of experimental facts. Thus this defini-

tion implies the following interconnected chain: Problem under study - Experimental facts - Formalysed knowledge, which shows that appearence of new knowledge is connected with constant demand to compare experimental facts with formalysed knowledge. Such comparison is double-aimed: from one side to check once more the correspondence of the process under study to formalysed knowledge that we have and from the other side to establish limits of application of given formalysed knowledge and to find those experimental facts which demand to extend the former mathematical model and hence stimulate knowledge development. Thus knowledge that we have and the current experimental facts are continous source of knowledge development. Knowledge development in its turn is always connected with search of law, theorem, algorithm. We underline that the main aim of such a search is a reduction of initial empirical facts. Figure 1 shows a model of informational development of knowledge. Here δ_{t_o} - is capacity of accumulated knowledge at the moment t_o; t_o, t_1,...t_j - are levels of special knowledge which symbolise a tendency of development of knowledge, marked by (-). From Figure 1 one can see that for development of special knowledge the search of information in neighbouring fields of knowledge stored at previous levels of development of general knowledge always takes place. And the problem of special knowledge application is connected first of all with the rate of getting information. It is possible to conclude that reliability of special knowledge depends on analysis of general capacity of stored knowledge, and the rate of its application depends on capacity of special knowledge.

Well known is the fact that communicating in constant environment: sport, mathematical scientific seminars, etc. people use a concise tesaurus to accelerate informational interchange.

We shall briefly enumerate main stages, providing knowledge development:

1. Reduction as a method of accelerating of informational interchange and special knowledge application.

2. Increasing of capacity of general knowledge together with enlargement of special knowledge using concise tesaurus. Conciseness of tesaurus is a necessary condition of optimal coordination between the rate of special knowledge application and capacity of general knowledge.

3. Creation of "translaters". Relation between capacities of general and special knowledge is constantly increasing. This fact evokes the problem of "translator".

4. Understanding - is a coordination of special tesauri provided by learning.

Knowledge understanding represented on Figure 1 shows place and role of learning. Let us underline that tesauri coordination is possible only through previous special levels. That is the difficulty of creating of "translators". The following example will explain this situation: specialists in one narrow field of knowledge speaking different natural languages will quicker understand each other than specialists in zoology and physics for example, using the same natural language.

It is quite evident that it is the problem of understanding of special knowledge by specialists in neighbouring fields taking into consideration permanent development of these fields that causes the necessity of creating of problem-oriented dialogue systems and intelligent data bases with the help of computer.

II. CONSTRUCTIVE APPROACH TO DATA REPRESENTATION. REDUCTION PROBLEM.

There is only one problem on the initial stage of know-

ledge representation - approximation of experimental facts connected with demand of effective reduction in their memorising.

Let E be a set of objects under study represented by experimental facts list X. $\{|X|\}$ - is a notation for symbol power of initial alphabet of description E. Mathematically the problem of reduction consists in searching for algorithm φ of enumeration of objects from E using such an alphabet of description Y in order to minimize $\{|Y|\} \to \min$. By symbol power we understand reduced number of bit information necessary for reestablishment of initial alphabet description of the set E during computer memorizing. Objects under study forming the set E are constructive objects, because the experimental facts may be always represented by finite symbol configuration.

This definition of constructive objects corresponds to that of (1). It is known that every constructive object may be coded by a word of s table alphabet (1). The following reduction is based on possibility of reenumeration of words in finite alphabet using for example lexicographical ordering. Thus, in principle it is enough to examine enumeration $\varphi : E \subseteq X$ only on integers (1,3).

Interesting is the fact that from one point of view integers represent a natural model of ordering of objects of a set E, and from the other point of view they represent adequate addresses of memory elements in computer.

Final solution of problems like approximation, pattern recognition, cluster-analysis and identification of objects in informational system may be conventionally represented in the following formalism:

Let E be a given set of objects under study represented by a list X of its experimental facts, which we mark by points

in space R^n , we should find a general recursive function of enumeration of objects in E, $X \subseteq R^n$ on R^+.

Problem of pattern recognition is connected with the fact that this algorithm implies division of objects of a training set (Fig. 2a).

Problem of cluster-analysis is connected with the fact of condensing of objects relatively to a threshold element δ_n and distinguishing of a class of objects (Fig. 2b).

Problem of ordering of objects in E is connected with distinguishing of objects with certain property δ (Fig. 2c).

Approximation is characterised by searching of a real dimension of a problem under study.

Given: E, $X \subseteq R^n$, we should find φ under condition that the difference function

$$f\left(X_a, \varphi(X_a)\right) \to \min \qquad \text{and}$$

$$\varphi(X_a) = Y \subseteq R^m, \quad m \to \min, \quad a \subseteq E,$$

here m - is a real dimension of initial problem. In traditional (classic) representation φ - is an approximation basis, f - is an approximation quality criterium.

Problems of pattern recognition and of cluster-analysis are further enlargements of a problem of approximation because a quality criterium of approximation depends on a representation of the set E - union of homogenouos groups: $E = \cup A_i$, where A_i are known groups in pattern recognition, or it is necessary to find S_j a priori such that $E = \cup S_j$ (cluster-analysis).

Problem of associative search and identification:
Given $E = \{a, b, c \ldots\}$, $X \leq R^n$, enumeration function:

$$\varphi_n: R^n \to R^+ \quad \text{and} \quad X_k^* \subseteq R^m, \quad R^m \not\subseteq R^n, \quad X_k \not\subseteq E$$

Find $X_k \subseteq R^n$, with $f(X_\ell^*, X_k^*) < \varepsilon$;

$X_k^* \subseteq R^m$; $X_\ell^* \subseteq R^m$; $l = \overline{\ell, E}$;

$m < n$ $\varphi_m: R^m \to R^+$

$\min\limits_{j \le l} |J_j - J_k|$; $\varphi_n^{-1} : (J_j \subseteq R^+) \to X_j \subseteq R^n$

$$X_k = \{\underbrace{X_1, X_2 \ldots X_m}_{X_k^* \subseteq R^m}, \underbrace{X_{m+1} \ldots X_n}_{X_j \subseteq R^{n-m}}\}$$

Here k, ℓ, j are indices of enumeration of objects of the set E . n, m - are indices of enumeration of description features of X.

To solve problems of above mentioned class we use principles of one-to-one application of points of spaces of different dimensions, that we have already realised (3), with the help of general recursive function based on multidimensional analog of Peano curve.

The first method of establishment of one-to-one correspondence of points with different dimensions was proposed by G. Cantor, but this correspondence was not continuous. Lack of continuity of mapping makes it impossible to solve successfully practical problems with reduced dimension. Later on impossibility to construct one-to-one continuous application was proved. The found out that continuity of mapping should be provided only in one direction - from lower dimension to higher one. Space-filling curves (SFC), historically connected with the name of italian mathematician Peano, are just one-to-one continuous mappings of unit interval on n-dimensional unit hiper-cube. From the end of XIX century till recent time these mappings were of pure theoretical interest, but nowadays atten-

tion to them and to their practical usage in different fields has considerably grown. See for example (2,3,4,5).

Each SFC is a continuous non-differentiable curve which pass through all the points of n-dimensional unit hiper-cube R^n. It represents a limit of sequence of curves, called "approximations of SFC" (Fig.3). Approximations of SFC establish one-to-one correspondence between quantified hiper-cube R^n_m (m - is a number of approximation) and quantified unit interval R^1_m .

Quanta of m-th division are elements between which the correspondence by means of m-th approximation of SFC is established. Every such quantum represent n-dimensional hiper-cube with a side k^{-m}, where k - is a number of parts of division of a side of cube R^n, for construction of first approximation.

From geometrical point of view m-th approximation of SFC is a broken line connecting in certain order centres of quanta of m-th division. This broken line repeats several times (m-1)--th approximation of SFC according to a law of the first approximation, that's why SFC are called also recursive self-similar curves.

It is important that for application problems we need not know precise correspondence between coordinates of points in R^n and R^1 . As a rule we need only definite number of decimal places. Such a constraint of accurancy makes it possible to use corresponding approximation instead of SFC for reduction of dimension of initial problem. This fact simplifies considerably resolution of practical problems, because it allows to find correspondence in finite number of steps of algorithm (4,6). We shall use the following notation for such a mapping:

$$\varphi : R^1 \to R^n \qquad \text{and} \qquad \varphi^{-1}: R^n \to R^1$$

Having algorithm of constructing of any approximation of SFC we get an opportunity to ennumerate all the quanta of any division in quantified space R_m^n and, if necessary, to examine objects connected with the quanta (for example, to calculate value of a function given in R^n).

Thus the power of SFC method is always equal to that of exhaustive search of all possible solutions. Though it has several properties which allow to decrease considerably a number of variants for concrete problems without decreasing power of this method (6).

Its first property is called "convergence by division" and is a result of recursive definition of construction of approximations of SFC. The main point of this property is that it provides convergence of resolution for the searching of correspondence between coordinates of points in R^1 and R^n, when division number m is growing. In other words using two different approximations SFC for solution of concrete problem we shall get two solutions of proposed problem, both of them lying in the region of true solution and the one with greater number of approximation of SFC being more precise.

The second basic property of approximations of SFC is called "quasicontinuity", because a limit of approximations of SFC with $m \to \infty$ is a continuous curve. This property shows that "close" points in R^1 have images which are "close" in R^n during search of correspondence by means of any approximation of SFC, and if ΔX - is a distance between two points in R^1, then $\| \Delta \varphi(X) \|$ - distance between their images in R^n - may be estimated by a formula:

$$\| \Delta \varphi(X) \| \leq L |\Delta(X)|^{1/n}, \quad L = \text{const} < \infty$$

Most important practical result of this property of quasicontinuity is the fact that Lipshitz property is preserved in R^1 by functions given in R^n and mapped on R^1 by means of

SFC.

We should underline that no other mappings possess such properties (television raster, spiral raster, random walk) of quantified spaces.

Let us consider solutions of some application problems using SFC.

1. Search of global extremum of multi-variable function.

Given continuous real function $\varphi(y)$, $y \subseteq R^n$, having in R^n a restrained Lipshitz constant. For sequence of points X_o, X_1, ... we find values of functions $\varphi(\varphi(X_o)$, $\varphi(\varphi(X_1)$, ... and we may search for extremum in the region of R^1, because the function $\varphi(\varphi(-X)$ possesses also the constrained Lipshitz constant in R^1 because of the property of quasi-continuity. The later property allows to calculate the value of function using a constrained number of points but garantees convergence to global extremum under condition of using of global algorithm of one-dimension search. It in important that we did not demand that a function should be differenciable or convex.

Similarly,method of SFC may be used to solve equation or unequation systems.

2. Pattern recognition.

Given sets $A_1,...A_q$ of points in R^n. We know that points of one set belong to the same image. We shall map these points from R^n onto R^1. In this case we may construct such a covering S that images of sets $A_1,...,A_q$ on R^1 will not cross (7). Thus we can decide to which set $A_1,...,A_q$ an object Q belongs by mapping Q from R^n onto R^1, and analysing its position relatively to covering S. All this makes it possible to exclude from a procedure of decision making such informal component as a choise of metric in multi-dimensional space. The algorithm based on method of SFC is similar to that of "some close neigh-

bours" according to its quality characteristics but it functions much faster.

3. Cluster-analysis.

Given a set E of N points in R^n, each symbolysing one of N objects of classification with n measured features, coordinates of points being the values of these features. Then, using a mapping φ^{-1} we may map all the points from R^n onto R^1 and solve the problem in one-dimensional variant. In this case objects found in one quantum belong to the same class, and object found in neighbouring quanta are also united in the same class. It is worth mentioning that because of the property of quasi-continuity of mapping objects found in neighbouring quanta in R^{-1} are neighbouring in R^n, besides according to the property of convergence by division some one-dimensional mappings of the set E with different m precise consequently coordinates of objects in R^1. In comparison with other methods of cluster-analysis the method of SFC is distinguished by its visible demonstrativity and by its independence of number of classified objects (time of mapping is very short).

Both described procedures that of pattern recognition and that of cluster-analysis are particular cases of decision making by means of "precedent": inquiring object Q, k features of which are known, appropriates values (n-k unknown features) of the closest neighbour in R^1 according to above principles of associativity for constructive objects.

More detailed discussion of problems concerning different applications of SFC one can find in (6).

SUMMARY.

In the first part of this paper we discussed some general principles of forming, development and representation of knowledge, as a basic element of transference of experience from

one generation to another. And from this point of view only forms of knowledge representation change through time: from oral narration to written documents, from written documents to computer memory as a result of necessity of collecting and application of permanently growing knowledge.

The second part is devoted to one of possible algorithms of knowledge representation in computer based on constructive approach of creating general recursive function of one-to-one mapping of constructive objects represented by a list of experimental facts in elements of computer memory. We use multidimensional analog of Peano curve as a general recursive functions.

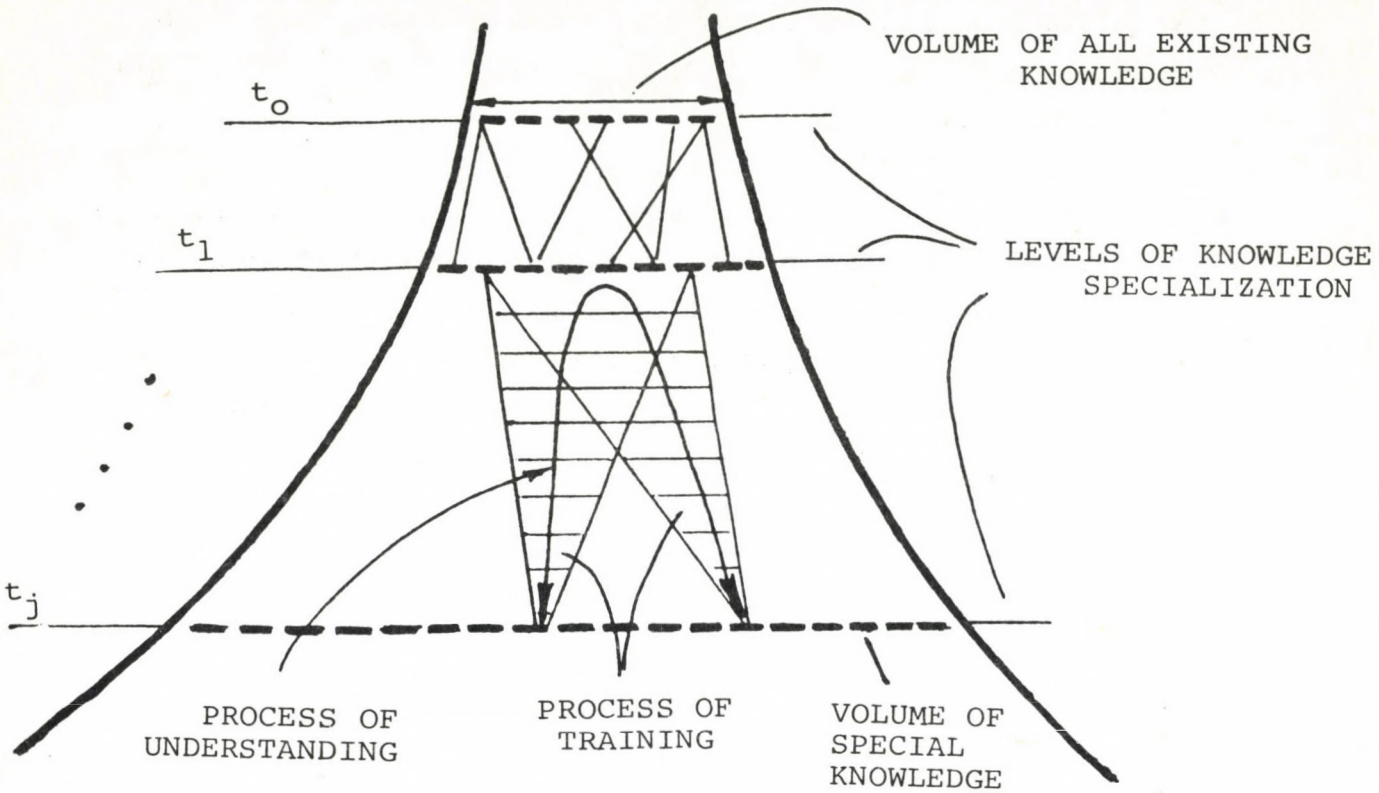
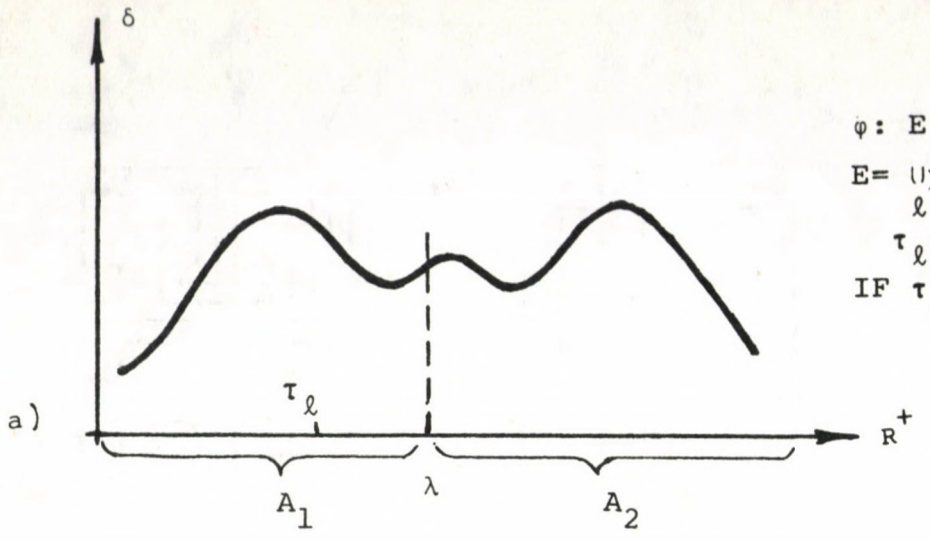


Fig.1. A model of informational knowledge development

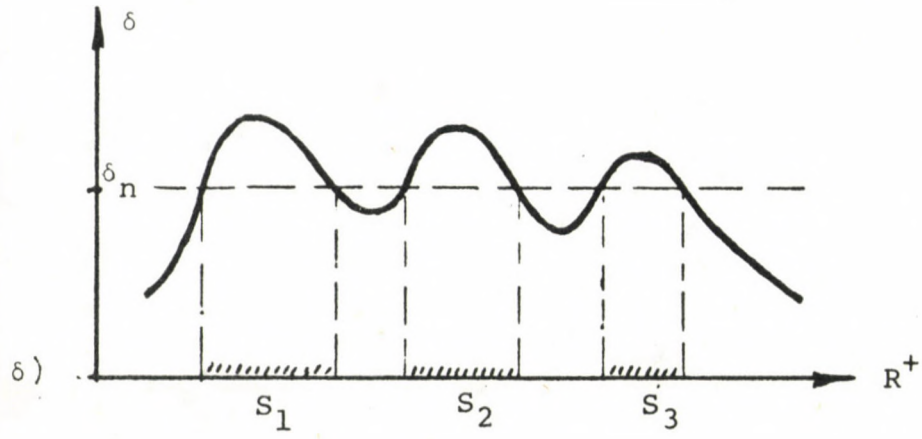


$\varphi: E \subseteq R^n \to R^1$

$E = \bigcup_\ell y_\ell = \cup A_1, A_2$

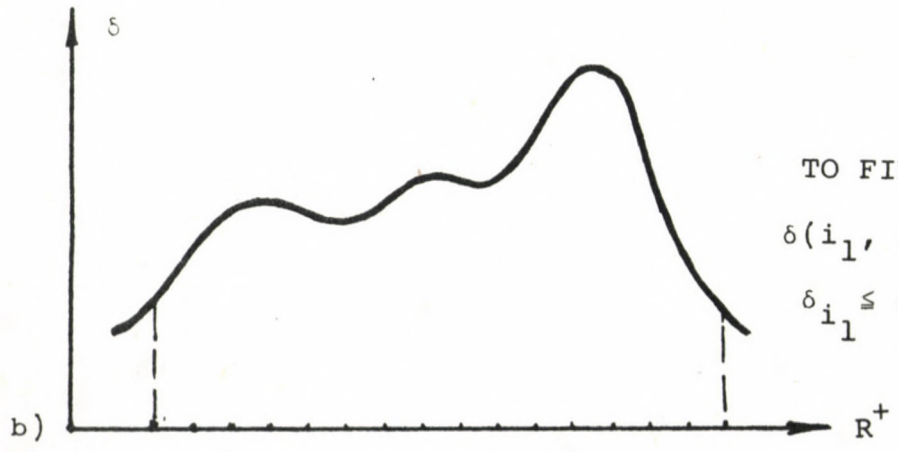
$\tau_\ell = \varphi(y_\ell)$

IF $\tau_\ell < \lambda$ THEN $y_\ell \subseteq A_1$



$S_1 = \{y_\ell \subseteq E,$

$\tau_1 < \varphi(y_\ell) < \tau_2\}$



TO FIND A PEPMUTATION

$\delta(i_1, i_2, \ldots, i_{|E|})$, IF

$\delta_{i_1} \leq \delta_{i_2} \leq \ldots \leq \delta_{i_{|E|}}$

$\rho: (E, Y^n) \to (E, R^+, \leq)$

Fig.2. a) pattern recognition;
δ) cluster-analysis;
b) ordering and identification of objects

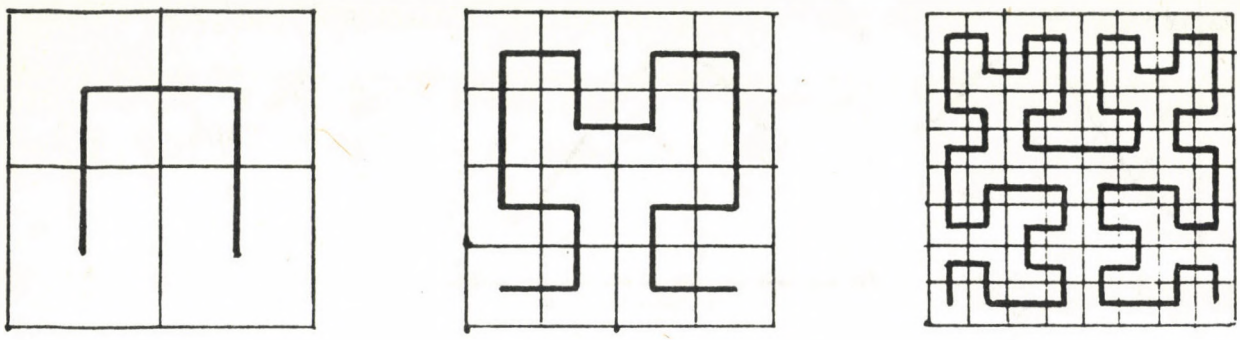


Fig.3. SFC approximation for n=2

REFERENCES

[1] Мартин-Лёф: Очерки по конструктивной математике. М. 1976.

[2] Butz A.R.: Convergence with Hilbert's Space Filling Curves. I. Comp. Sys. Sci. v.3. N2,1969.

[3] Александров В.В., Поляков А.О.: Структурные методы классификации. В сб. "Адаптация в системах со сложной структурой", М. 1977.

[4] Alexandrov V.V., Polyakov A.O., Latchinov V.M. Synthese et application d'analogue multidimensionnal de courbe Peano. Comminication aux jorneese "Reconnaissance des formes et Traitement des images", fevrier, 1978.

[5] Quinqueton I. Classification rapide utilisant un balayage de Peano. Communication aux jorneese "Reconnaissance des formes et Traitement des images", fevrier, 1978.

[6] Александров В.В., Горский Н.Д., Поляков А.О.: Рекурсивные алгоритмы обработки и представления данных. Рабочий отчет ЛНИВЦ АН СССР, 1978.

[7] Александров В.В., Горский Н.Д.: О возможности использования отображений дискретных пространств для решения многопараметрических задач. Тезисы III Всесоюзной конференции по исследованию операций. Горький, 1978.

CONTENTS

81.1380/3 – MSZH Nyomda, Budapest – F. v.: Nagy László