XIII

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

XIII.

Budapest, 1979.

Editorial board:

Bálint DÖMÖLKI (chairman)	Theoretical Laboratory, Institute for Co-ordination of Computer Techniques
Gábor DÁVID	Computer and Automation Institute, Hungarian Academy of Sciences
Ernő FARKAS (editor)	Computer and Automation Institute, Hungarian Academy of Sciences
Tamás GERGELY	Research Institute for Applied Computer Sciences
Tamás LEGENDI (editor)	Research Group on Mathematical Logic and Theory of Automata, Hungarian Academy of Sciences
Árpád MAKAI	Research Group on Mathematical Logic and Theory of Automata, Hungarian Academy of Sciences
György RÉVÉSZ	Computer and Automation Institute, Hungarian Academy of Sciences
Imre RUZSA	Loránd Eötvös University, Budapest
György SZÉPE	Research Institute of Linguistics, Hungarian Academy of Sciences
Dénes VARGA	National Planning Office

Distributor for: Albania, Bulgaria, China, Cuba, Czechoslovakia, German Democratic Republic, Korean People's Republic, Mongolia, Poland, Roumania, U.S.S.R., Socialist Republic of Viet-Nam, Yugoslavia

KULTÚRA

Hungarian Trading Co. for Books and Newpapers
1389 Budapest,
P.O.B. 149, Hungary

For all other countries:

JOHN BENJAMINS B.V.
Periodical Trade
Amsteldijk 44
Amsterdam, Holland

Responsible Publisher:

Prof. Dr. T. 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 articles 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.

Articles 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.

Articles 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 correspondence 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 (210x297 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 length 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 developments 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.) Distinguish between "oh," "zero"; "el," "one"; "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.

Készült a
SZÁMOK Reprográfiai Üzemében 9353

Contents

A. FORMAL LANGUAGES

Computational Linguistics and Computer Languages Vol. XIII. 1979.

COMPARISON OF SOME METHODS FOR THE DEFINITION OF STATIC SEMANTICS

E. Farkas
Computer and Automation Institute Hungarian Academy of Sciences
Budapest, Hungary

Probably every programmer has an intuitive image what the syntax and semantics of a programming language mean. However, it is very hard to find a commonly accepted exact definition for these concepts.

From the pragmatic point of view the syntax is a set of rules which a compiler can check, and semantics is the term for the features which we can detect only in runtime. This is a very common opinion.

On the other hand, in several theoretical works syntax is a set of context-free rules and all the others are called semantics. In this case semantics has two parts: features which we can check in compile time (so-called static semantics) and other features, called dynamic semantics.

Here we do not want to discuss about the terminology, about whether the first, or the second or a third nomenclature is rightful. We say that the context-free grammar will be called syntax, some rules regarding the proper use of the words playing part in programs will be called static semantics, while the expression dynamic semantics is used in the sense how input-output mapping created by a given program can be established.

We suppose that a program is composed rather of words than of individual characters. So in our case the terminal symbols in the context-free grammar will be the words.

In comparison with the natural languages we find that most of the programming languages have a very strange feature, namely a significant set of the legal words have no predefined meaning or the meaning is only partly defined. The necessary attributes of these words are fixed only in the program. For example, an identifier in 'ALGOL 68' can denote any type of constant or variable or can denote a label or an operation, etc. In 'BASIC' a letter followed by an open parenthesis can denote either a one-dimensional array or a two--dimensional array. Similarly most of the programming languages has a built-in mechanism to create syntactical/semantical attributes for the words of programs.

There are some very simple languages for calculators, industrial equipments etc, which have not such a creative power, so every word has a predefined meaning. In this case the language has no static semantics, and the context-free syntax can check whether a program is formally correct or not. (Although the proper work of the program is not guaranteed.)

The creation of attributes has usually a limited scope. It is not valid any longer than the program, furthermore it is often valid only in a smaller part of the program. When we speak about creation, we do not mean declarative inctructions only; a lot of so-called executable instructions have such a side effect. For example, in FORTRAN/II the 'DO' cycle instruction creates a new attribute for the cycle control variable, namely it must not be modified in the cycle.

In the existing compilers checking is solved by the aid of tables where the current attributes of the created words are registered, check goes in step by step from left to right in the program.

In spite of the fact that the method is commonly accepted, it is not fixed in any standard what checks must be fulfilled during compilation of the given language.

From the theoretical point of view, several abstract formalisms were developed. The works were done in three different directions:

1. replacement of context-free grammar by another, more powerful grammar,
2. enclosure of attributes and checks into the context-free grammar,
3. introduction and stepwise modification of a declarational state.

In the first direction several unsuccessful attempts have been done before the van Wijngaarden grammar was issued. The van Wijngaarden grammar was used in the definition of the 'ALGOL 68' language. Sevaral static semantical features were included into the revised report, but not all. It seems to be a nice didactic tool to understand the language but it is not clear how we can use it in compiler writing.

In the second branch the first important work was D.E. Knuth's paper. In this paper he introduces the idea that every syntactic unit has ascendent and descendent attributes. The affix grammar is a more developed form of this idea which is a theoretical basis for the CDL language. The functional grammar is a very similar solution. A similar method was also proposed by M. Griffiths.

The third solution is based on H.F. Ledgard's work. This was applied by M.H. Williams and by the author. In this case we have a table (or something like that) containing the current names with the current attributes and we have functions connected with the syntactic units for modification of the state of the table.

In the method elaborated by the author only the terminal symbols of the context-free grammar have state transition functions. The attributes of the higher-level syntactic units are included into the context-free grammar as the van Wijngaarden grammars. This method enables clear separation of syntax and static semantics.

THE VAN WIJNGAARDEN GRAMMAR

The van Wijngaarden grammar is a context-free grammar having infinite production rules. This infinite set of context-free rules are given in a constructive way.

The symbols of the grammar are denoted by long alphabetical strings, these are the so-called protonotions. The terminal symbols end with the word "symbol". The individual symbols are separated by commas.

For the construction of context-free production rules, schemes (so-called hyper-rules) are given. In the hyper-rules at both sides such symbols are given which are composed of small-letter words (protonotions) and capital-letter words (metanotions). The metanotions are parameters.

For each metanotion a context-free grammar generating a possibly infinite set of protonotions must be given.

A production rule is generated from a hyper-rule in such a way that all of the metanotions must be replaced by a corresponding protonotion so that the same metanotion must be replaced by the same protonotion. This is the uniform replacement rule.

From the initial symbol we can derive a string so that the non-terminal symbols will be substituted subsequently until the string contains only terminal symbols.

Using the metanotions we can generate production rules which can fulfil any algorithm in a style similar to a Markovian algorithm. Those symbols in the hyper-rules have a special role from which we can derive the empty string. Such symbols are called predicates. They can express a relation between the parameters denoted by metanotions. If the relation is true, we can derive the empty string. If it is false, the derivation stops and a non-terminal symbol remains in the derived string. So the derivation is not valid.

Example:

In the following example we can see how it can be checked that each variable is declared only once in the program.
(Note: the semicolon separates the alternatives.)

Metaproductions:

```
     'ALPHA': :A; B; . . . . X; Y; Z.
     'LETTER': : letter 'ALPHA'
     'NAME': :'LETTER'; 'LETTER' 'NAME'.
     'DEF': : 'NAME' has 'MODE'.
     'TABLE': : 'DEF'; 'TABLE' 'DEF'
     'DEFSETY': : 'TABLE'; 'EMPTY'.
     'EMPTY': : .
```

Hyper rules:

```
Program:  Begin symbol,
          Declare of 'TABLE',
          'TABLE' restrictions,
          'TABLE' statement train,
          end symbol.
```

(The " 'TABLE' restrictions" symbol is a predicate which checks the unique declaration.)

```
     'DEFSETY' 'NAME' has 'MODE' restrictions:
          where 'NAME' is not in 'DEFSETY', 'DEFSETY' restrictions;
          where 'DEFSETY' is 'EMPTY'.

     Where 'NAME1' is not in 'NAME2' has 'MODE' 'DEFSETY':
          where 'NAME1' differs from 'NAME2', where 'NAME1' is not in 'DEFSETY';
          where 'NAME1' differs from 'NAME2', where 'DEFSETY' is 'EMPTY'.

     Where 'EMPTY' is 'EMPTY': 'EMPTY'.

     Where 'NAME1' letter 'ALPHA1'differs from 'NAME2' letter 'ALPHA2':
          where 'NAME1'differs from 'NAME2';
          'ALPHA1' is not 'ALPHA2'.

     Where 'NAME' letter 'ALPHA1' differs from letter 'ALPHA2': 'EMPTY'

     Where letter 'ALPHA1' differs from 'NAME' letter 'ALPHA2': 'EMPTY'

     A is not B: 'EMPTY'.
     A is not C: 'EMPTY'.
     A is not D: 'EMPTY'.
          .
          .
          .
     etc.
```

As can be seen from the example, this type of definition is easily legible and comprehensive. On the other hand, we can see that the definition is rather redundant, not mathematically but in practice, since very simple functions are implemented in a tricky way, using sophisticated string manipulations. Such a string manipulation is solvable in a computer but it is surely an ineffective solution.

The van Wijngaarden grammar is a synchronous aspect of the language. It does not take into consideration that a program is written and translated advancing in time. Theoretically we can begin to analyse a program from any point. In practice, 'TABLE' is constructed during declaration, while in the other places of the program, 'TABLE' is used. This is not included into this model.

THE AFFIX GRAMMAR AND THE CDL

Another solution was proposed by D.E. Knuth. He proposed a context-free grammar in which every grammatical unit, i.e. every grammatical symbol has a set of attributes. An attribute is called "ascendent" if it is derived from the attributes of lower-level units and called "descendent" if it is originated from a higher level grammatical unit.

This concept and that one of the van Wijngaarden grammar was combined in the affix grammar.

In the affix grammar three types of objects have to be considered. The non-terminal symbols denote grammatical units, the terminal symbols are the words of the programs and the checks are predicates over the attributes. All the three types of objects have a definite number of attributes; the terminal symbols have no attributes.

There is a set of context-free like substitution rules given. A non-terminal symbol will be replaced by a string composed of terminal, non-terminal symbols and checks. On both sides of the rule attributes are connected with the objects. An attribute is denoted by a symbol or it can have constant value, too.

If we have a non-terminal symbol with attributes having a certain value, we can replace it by the string, standing on the right-hand side in a rule (having the non-terminal on the left-hand side). The attributes of the objects must be given so that attributes which were denoted by the same symbol must have the same value. (Cf. unique replacement rule.) Then, in the new string we must substitute for the non-terminal symbols over and over again and the checks must be evaluated. Every check means a recursive predicate over the attributes. If the check is true, it will be substituted for by the empty string. If it is false, it will be substituted for by a non-terminal symbol which has no further derivation. The derivation is finished when a string of terminal symbols is produced.

Example:

This example is equal to the previous one, however, another part of the derivation is given in detail

```
Program: Begin,
         Declaration+'TABLE' ,
         Restrictions+'TABLE' ,
         Statement train+'TABLE' ,
         End.
```

(Here the "restriction" is a predicate over the domain of the 'TABLE's. It checks that every name is unique in the 'TABLE'.)

```
Declaration+'TABLE': Declare+'MODE'+'TABLE';
                     Declare+'MODE'+'SUBTABLE1',
                     Declaration+'SUBTABLE2' ,
                     Union+'TABLE'+'SUBTABLE1'+'SUBTABLE2'.
```

("Union" checks that the 'TABLE' is the union of the two 'SUBTABLE's.)

```
Declare+'MODE'+'TABLE': Declarer+'MODE',
                        Idlist+'TABLE'+'MODE'.

Idlist+'TABLE'+'MODE': Identifier+'NAME',
                       Include+'TABLE'+'NAME'+'MODE',
                       Semicolon;
                       Identifier+'NAME',
                       Include+'TABLE'+'NAME'+'MODE',
                       Comma,
                       Idlist+'TABLE'+'MODE'.
```

("Include" checks that 'NAME' having 'MODE' is included in 'TABLE'.)

The 'CDL' language is a slightly modified form of the affix grammar. A 'CDL' program is a syntactical/semantical definition which can be translated into a code fulfilling the parsing of the program. In the form of the grammar there were modifications which turn the description of languages, and the execution of the parsing shorter.

The translation of a 'CDL' program is organised so that the grammatical symbols are translated into recursive procedures, while the terminal symbols and checks are translated into macros. The macros must be given by the user.

The body of a procedure is a sequence of macro and procedure calls. The calls are given in accordance with the sequence of the objects in the substitution rule. The parsing of the text goes on from top to bottom and from left to right. So the left-recursive rules are excluded.

The attributes are the parameters of the procedures and the macros. The descendent parameters are the input parameters and the ascendent ones are the output parameters.

In the affix grammar the values of attributes are strings, generated by context-free grammars. As we see, in the previous example on the van Wijngaarden grammar, we can express the necessary attributes in such a way. Nevertheless, we feel that tables and lists and others would be more natural. In the 'CDL' the attributes are integers and integer arrays. These physical data structures are used for the representation of the necessary logical data structures mentioned above.

If we compare the van Wijngaarden grammar with the affix grammar we can see that the conceptions and the solution are very similar, but the mechanism in the affix grammar is more explicit, so the implementation is much easier.

Though the use of the 'CDL' is wide-spread in Hungary, in our opinion the application of the method is far from being efficient. Inefficiency comes from two facts. One fact is the inefficiency of the recursive procedure calls. This can be minimized by sophisticated programming depending highly on the computer. Another problem is deeper and concerns the essence of the method. Most of the attributes are born on the level of the terminal symbols, but attributes are possessed usually by more higher-level syntactic units. So a great number of parameter passing is necessary to use the attributes.

STATE TRANSITION METHODS

The van Wijngaarden grammar is a synchronous model as was already mentioned above. Such model as a generative model of the language fits very well. Generative model means that first we decide what words and attributes we require and then we generate the program with the necessary words and attributes. Such a model is very suitable for users who want to create a program.

The affix grammar can be regarded as a synchronous and generative model. But this model may be completed by the consideration of the direction of the ascendent and descendent attributes. The model where the direction of the attributes is considered, determine how to build up the parsing tree of programs. In some sense this model is more exact. We feel, however that in a generative model such a consideration is unnecessary. Similarly, in practice the model causes problems, since the parsing must be in accordance with the direction of attributes. This means a restriction either for the grammar or for the parsing algorithm or for both.

The implementor's problem is absolutely different from the user's problem. The implementor must read and check existing programs. (The translation or interpretation is a further problem, which is not discussed here.) For such a purpose a diachrone model is more adequate. The diachronous model means that the program is considered in its development in time.. In the program new words are created first (defined, declared, etc.), later on these words get used. Sometimes they get new attributes which are valid in a limited scope. It is characteristical for most of the programming languages that words and attributes must first be created before they get applied.

'CDL', as an implementor-oriented realization of the affix grammar utilizes the diachronous nature of the languages. Sometimes this means a limitation for applicability. For example, we cannot use it for languages, where declaration can appear everywhere in the program, but it is valid retroactively.

Another approach was given by H.F. Ledgard, applied for the definition of static semantics of PL/I. This method was developed by M.H. Williams and by the author in different directions.

In this model the attributes are not included in the context-free grammar. They are enclosed into tables or alike. The state of the tables changes step by step, advancing in the program. Modification of the state is solved so that each syntactic unit, i.e. each substitution rule in the context-free grammar is connected to a state transition function. When the recognition of the syntactic unit is completed, we must perform the associated state transition function. When the recognition of the syntactic unit is completed, we must perform the associated state transition. When parsing is completed, associated state transition must be performed and it must be checked, whether the table in a legal final state.

In this model the diachronous nature of the language is thoroughly utilized and parsing goes on from left to right.

Example:

In this example we have two variables in the state, 'MODE' and 'NAME' and a table having the name 'TABLE'.

```
Program:   Begin, Declaration, Statement Train, End.
Declaration: Declare;
             Declare, Declaration.
Declare: Declarer /'MODE':= MODE/, idlist.
Idlist: Identifier /'NAME' := Identifier,
        'TABLE' := 'TABLE' + ('NAME', 'MODE')/,Semicolon;
      Identifier / 'NAME' := Identifier,
        'TABLE' := 'TABLE' + ('NAME' + 'MODE') /, Comma, IDLIST.
.
.
.
etc.
```

Finally it must be checked, whether every declaration was unique. This means the evaluation of the expression 'UNIQUE'('TABLE'). Here the 'UNIQUE' function is usually defined by a program, similarly to the functions in the substitution rules.

As can be seen, not every context-free rule must have a state transition function and the whole description seems to be more compact than the earlier ones. Nevertheless, it is true that the structure of the program means a natural structuring for the data table containing the words with the attributes. This is very clear in the case of the block-structure languages. In our method this structuring must be realized in the state table and in the table handling function.

Our method differs from the latter one in the point that the state transition functions are connected only to terminal symbols. The question arises, how we can implement the transitions which are in connection with higher-level syntactic units (e.g. blocks, etc.). The solution is very simple: such a higher-level unit has a first and a last terminal symbol and the necessary actions are connected to these ones. Nevertheless the presentation of our method differs significantly from other methods by the fact that the state transition function is not included into the 'CF' metalanguage but it is given in a separate table in the following form: in the first column the terminal symbol is given, in the second one the type of the lexical unit, and in the third one the state transition action. In the example the terminal symbols are indicated by underscored lettering. The prefix "D" indicates that it is a defining occurrence for the identifier.

Example:

```
Program:     Begin, Declaration, Statement Train, End.
             -----
Declaration: Declare;
             Declare, Declaration.
Declare: Declarer, Idlist.
         --------
Idlist: D. Identifier, Semicolon;
        -------------
        D. Identifier, Comma, Idlist.
        -------------
.
.
.
etc.
```

TERMINAL SYMBOL	LEXICAL UNIT	ACTION
DECLARER	MODE	'MODE' = MODE
D.IDENTIFIER	NAME	'TABLE' = 'TABLE' + (NAME,'MODE')
BEGIN	"BEGIN"	–
END	"END"	–
COMMA	,	–
SEMICOLON	;	–
.		
.		
.		

Our method has the advantage that lexical rules, syntax and static semantics are highly separated. This means a benefit in implementation, namely static semantics has no influence on the parsing method.

In H.F. Ledgard's work the declaration state was described by abstract mathematical objects and relations (like set, element, part of, etc.). In M.H. William's paper lists, stacks and variables are used for the description of the state, and for the transitions a special programming language was introduced. In the present paper general list structures (lists of lists) and 'LISP 1.5' - like list handling functions were used.

For certain programming languages one pass from left to right is insufficient, in these case more passes are required. A condition for a single pass and an algorithm how many passes from left to right are necessary, is given in G.V. Bochmann 's paper.

Summary

Most of the programming languages have the feature that a set of words has no predetermined meaning. The meaning of these words is rather determined by the actual program. The rules regarding the consistent use of these words is called static semantics in this paper.

As it has been shown, each of the formal and practical methods are based on the idea that the necessary attributes of these words are collected, registered and checked. There is a great difference between the methods, how correspondence between the words and attributes are represented and handled. Anyway, it is supposed in each method that for their handling general recursive functions are necessary. This has the consequence that each formal method is capable to accept or generate enumerable recursive sets.

Similarly we can see that in every method three types of attributes are used. It is not stated anywhere explicity, however that the three types of attributes are: the literal, the integer and the pointer-type. The literal-type attribute denotes the presence or the absence

of an attribute, the integer-type attribute has an integer value. The pointer-type attribute is the name of another word having attributes (in practice it is usually implemented by the help of pointers).

We can classify the definitional methods into two groups. In the first group the methods are more suitable to create programs, we can call them user-oriented methods. These are the generative synchronous models. The other class of methods is more suitable for checking existing programs, which can be denoted as implementor-oriented methods. Such are the state transitions methods.

R e f e r e n c e s

[1] J.C. Cleveland, R.C. Uzgalis: "Grammar for programming languages" Elsevier North-Holland. 1977.

[2] A. Van Wijngaarden et al.: "Report on the algorithmic language ALGOL 68" Offprint from Numerische Mathematik 14 79-218 (1969) Springer-Verlag Berlin.

[3] "Revised report on the algorithmic language Algol 68" ACTA Informatica 5 p1-236, (1974)

[4] Koster: "Affix grammars" In the book: Algol 68 implementation. North-Holland. 1971.

[5] Z. Laborczi: "Programnyelv teljesen formális leirása, forditóprogramiró program". Számológép 1972/3.

[6] A. Bedő et. al.: "Forditó leiró nyelv felhasználói kézikönyve" (CDL Manual) Internal Document, in Hungarian.

[7] D.E. Knuth: "Semantics of context-free languages" Mathematical system theory p. 127-145. 1968 (2). Correction math. sys. th. 1975. (5) / 1.

[8] H.F. ledgard: "Production system or can we do better than BNF? " Comm. of ACM 1974/2. p. 94-102.

[9] M.H. Williams: "Static semantics features of 'ALGOL 60', and 'BASIC'. The Computer Journal Vol 21. No. 3. p. 234-242.

[10] M. Griffiths: "Relationship between definition and implementation of a language." In the book: Advanced courses on software engineering. Lecture notes in economics and mathematical systems Springer-Verlag 1973.

[11] G. Riedwald: "Grammars of syntactical functions" Manuscript "Die Grammatik syntaktischer Funktionen-eine andere Form der van Wijgaarden-Grammatik." EIK 11 (1075) 7/8. 489-487.

[13] E. Farkas: "A compiler oriented syntax definition" Computational Linguistics and Computer Languages Vol.X. 1975.

[14] G.V. Bochmann:. "Semantic evaluation from left to right" Com. of ACM 1976/2. p.55-62.

Computational Linguistics and Computer Languages Vol. XIII. 1979.

A CORRESPONDENCE BETWEEN W–GRAMMARS AND FORMAL SYSTEMS OF LOGIC AND ITS APPLICATION TO FORMAL LANGUAGE DESCRIPTION

W. Hesse

SOFTLAB GmbH, München *

FRG

Abstract

A one–to–one correspondence is pointed out between formal systems of logic, the well–known tool for the formulation of logical calculi, and PW–grammars, i.e. W–grammars, whose hypernotions are all predicates. Using that correspondence, every formal system can easily be embedded into a W–grammar. Conversely, every PW–subgrammar of a W–grammar can be used as a logical calculus for the derivation of theorems. The various applications of that correspondence make W–grammars a well–suited tool for complete formal language descriptions.

Introduction and summary

In the revised ALGOL 68–report (A68RR), the possibilities for the application of W–grammars (''van Wijngaarden grammars'', ''two–level grammars'') have been considerably enlarged by the introduction of ''predicates''. Predicates are hypernotions (i.e. two–level sentential forms), which either produce the empty string ε or for which no production exists (the so–called ''blind alleys''). In this paper *PW–grammars* (*P*redicative *W*–grammars) are defined, whose hypernotions are all predicates. Instead of the language $L_g(G)$ generated by a PW–grammar (merely consisting of ε), the set $L_a(G)$ of all those hypernotions is considered, from which ε can be produced.

Formal systems (''canonical systems'', ''Post systems'') are a well–known tool for the formulation of logical calculi. We define *formal G–systems* which differ from the usual formal systems only in a stronger formalization of the well–formedness rules (section 1).

In section 2, it is shown, that PW–grammars are equivalent to formal G–systems by pointing out a simple way to translate them into each other.

PW–grammars can be embedded into general (non–predicative) W–grammars in a straightforward manner. This is shown in section 3.

With respect to formal description of programming languages, two consequences are immediate, which are dealt with in section 4:

1) Every part of a language definition formulated as a logical calculus can now easily be incorporated into a W–grammar. Examples are calculi for the formulation of context

* The work reported in this paper was carried out at the Instribut für Informatik der Technischen Universität München and was first published as technical report TUM-INFO-7727 of that institute.

conditions as well as for the semantics or parts of it.

2) The rules of PW–grammars may be used as axioms and derivation rules to prove theorems. For example, from a PW–grammar semantics description, program properties can directly be deduced this way.

These consequences open a practicable way to use W–grammars for complete language descriptions as proposed in /HES 76/. A model for such descriptions is given in section 5. The W–grammar method is proposed particularly for *defining* language descriptions, i.e. those documents which form the central, binding definition of a particular programming language for users and implementors likewise.

1. W–grammars, PW–grammars and formal G–systems

A *W–grammar* G is a 6–tuple (M, S, T, z, RM, RH), where

M is a finite set of *metanotions* (or meta–nonterminals),

S is a finite set of *s–notions* (in the ALGOL 68–terminology: "sequences of small syntactic marks not ending with–symbol"),

T is a finite set of *terminals* (M, S and T are pairwise disjoint sets),

$MT \subset S$ is a set of *metaterminals*,

$H \subset (M \cup S)^+$ is a set of *hypernotions*, 1)

$z \in S^+$ is the *start symbol*,

$RM \subset M \times (M \cup MT)^*$ is a set of *metarules*,

$RH \subset H \times (H \cup T)^*$ is a set of *hyperrules*. 2)

Remarks:

1) Differing from /A68R/, the hypernotion set does not contain the terminal set.

2) For brevity, we omit extra separation symbols within meta– and hyperrules (as e.g. introduced in /KOS 74/) and suppose, that all members of the right hand sides can always uniquely be separated (as is done in /A68R/).

Metarules $(x_0, x_1\ x_2\ \dots\ x_n)$ are denoted as $x_0 :: x_1\ x_2\ \dots\ x_n.$,
hyperrules $(x_0, x_1\ x_2\ \dots\ x_n)$ as $x_0 : x_1, x_2, \dots, x_n$.

Alternative meta–/hyperrules may be combined using the ";"– symbol, terminals are underlined.

The 5–tuple (M, S, T, RM, RH) – without start symbol – is called a *W–production system*. For $x \in M$, $L(x) = \{ m \in MT^* \mid x \xrightarrow[RM]{*} m \}$ (where $\xrightarrow{*}$ denotes the usual context–free production relation) is called the *metalanguage* of x. Every $y \in M \cup MT$ with $x \xrightarrow[RM]{*} y$ $(y \in L(x))$ is called a *(terminal) metaproduction* of x.

A set RP of *production rules* (which is in general infinite) is obtained from the hyperrules RH by *consistent substitution* of all metanotions by terminal metaproductions.

For a hypernotion $h \in H$, the set $L_g(h) = \{ t \in T^* / h \xrightarrow[RP]{*} t \}$ is called the language generated from h. Every $y \in H$ with $h \xrightarrow[RP]{*} y$ $(y \in L_g(h))$ is a *(terminal) production* of h. $L_g(G) = L_g(z)$ is called the *language generated by G*. (For more detailed definitions cf. /HES 76/.)

Example 1:

Consider the W–grammar $G = (M, S, \{ \underline{a}, \underline{b} \}, s, RM, RH)$ with $M = \{ N, EMPTY \}$, $S = \{ s, x, y, i \}$, $RM = \{ m_1, m_2 \}$, $RH = \{ h_1, \dots, h_5 \}$ and

(m_1) N :: EMPTY ; N i .	(m_2) EMPTY :: .
(h_1) s : N x , N y , N x .	
(h_2) x : EMPTY .	(h_3) N i x : N x , $\underline{a}$.
(h_4) y : EMPTY .	(h_5) N i y : N y , $\underline{b}$.

The language generated by G is $L_g(G) = \{ a^n\, b^n\, a^n / n = 0, 1, 2, \dots \}$

A *PW–grammar* $W = (M, S, RM, RH)$ is defined as a W–production system $(M, S, \emptyset, RM, RH)$ (the terminal set T is empty). Every hypernotion h of a PW–grammar is called a *predicate*. Instead of $L_g(h)$, which is empty or $\{ \epsilon \}$ for every predicate h, we consider the language $L_a(W) = \{ h \in S^+ / h \xrightarrow[RP]{*} \varepsilon \}$ and call it the *language accepted by W*.

Example 2:

$W' = (M, S, RM, \{ h_1, h_2, h_4 \})$ with M, S, RM, h_1, h_2 and h_4 defined as in example 1, is a PW–grammar with accepted language $L_a(W') = \{ s, x, y \}$.

A *formal G–system* GS is a 4–tuple (B, V, WFG, DR), where

B is a finite set of *basic symbols*,

V is a finite set of *variables*,

WFG = (V, B, w, WR) is a context–free grammar with nonterminal set V, terminal set B, start symbol $w \in V$ and production rules $WR \subset V \times (V \cup B)^*$. Every $x \in L(v)$, $v \in V$ $(x \in L(w))$ is called a *term (formula) schema*, or – if it does not contain any variable – a *well–formed term (formula)* of GS.

DR is a set of *derivation rule schemata*, i.e. (n + 1)–tuples of formula schemata $(n \geqslant 0)$. Instead of $(g_1, \dots, g_n, g) \in DR$ we write $g_1, \dots, g_n \vdash_{DR} g$ (or even omit "DR"). If $n = 0$, we write $\vdash_{DR} g$ and call the rule an *axiom schema*.

Remarks:

1) In contrast to usual definitions of formal systems, the wellformedness property is expressed by a grammar WFG rather than by an inductive definition in natural language.

2) Elements of V may serve as proper WFG–noterminals or as "metavariables" of the formal system. Rules of the form m: v. (where $m, v \in V$) may be interpreted as domain

rules assigning a "domain" v to a "metavariable" m.

3) An extension to two–level WFG–grammars is also possible (cf. /HES 76/).

Derivation rules are obtained from derivation rule schemata by (consistent) substitution of variables v by terms of $L_g(v)$. A formula f is *derivable* in GS (denoted by $\vdash_{GS} f$ or $\vdash f$), if there is a derivation rule $f_1, \dots, f_n \vdash f$ $(n \geq 0)$ and $f_1, \dots, f_n$ are derivable (inductive definition).

The set of all formulae derivable in GS is denoted by Th(GS).

2. The correspondence theorem

A one–to–one correspondence between formal G–systems and PW–grammars is established by

Theorem 1: *For every formal G–system GS there exists a PW–grammar* $W = q(GS)$ *and for every PW–grammar W there exists a formal G–system* $GS = q^{-1}(W)$*, such that* $x \in L_a(W)$ *iff* $q(x) \in Th(GS)$.

The correspondence is given by the following table, which juxtaposes GS–notions and their W–correspondences:

formal system GS	PW–grammar W
basic symbols B	s–notions S
variables V	metanotions M
substitution of variables	consistent substitution of metanotions
WFG–rules WR	metarules RM
derivation rules DR	hyperrules RH
(in particular:	
derivation rule $f_1, \dots, f_n \vdash f$	hyperrule $q(f) : q(f_1), \dots, q(f_n)$.
axiom $\vdash f$	ε -rule $q(f) : \varepsilon$.)

Thus, production in W corresponds to reduction of a conclusion to its premises in GS. Theorem 1 is proved by induction on the derivation of formulae or on the ε–productiveness of hypernotions, respectively.

Example 3:

Consider the formal G–system GS = (B, V, WFG, DR) for true boolean expressions in prefix form. (By the way, this system corresponds to the first example given by Smullyan for his "semantic tableaux"–method in /SMU 68/.)

$B = \{tt, ff, \neg, \wedge, \vee\}$, $V = \{prop, p, q\}$, WFG = (V, B, prop, WR), $WR = \{w_1, w_2, w_3\}$ with

(w_1) prop : tt ; ff ; $\neg$, prop ; $\wedge$, prop , prop ; $\vee$, prop , prop .

(w_2) p : prop . (w_3) q : prop .

(w_2 and w_3 assign to the "metavariables" p and q their "domain" prop.)

$DR = \{ d_1, \dots, d_7 \}$ with

$(d_1)\ \vdash tt$ $\qquad (d_2)\ \vdash \neg ff$

$(d_3)\ p \vdash \neg\neg p$

$(d_4)\ p, q \vdash \wedge pq$ $\qquad (d_5)\ \neg p \vdash \neg\wedge pq$

$\qquad\qquad \neg q \vdash \neg\wedge pq$

$(d_6)\ p \vdash \vee pq$ $\qquad (d_7)\ \neg p, \neg q \vdash \neg\vee pq$

$\qquad p \vdash \vee pq$

The corresponding PW–grammar is $W = (M, S, RM, RH)$ with
$M = \{ PROP, P, Q \}$, $S = \{$ true, false, not, and, or $\}$, $RM = \{ m_1, \dots, m_3 \}$ and
(m_1) PROP :: true ; false ; not PROP ; and PROP PROP ; or PROP PROP .

(m_2) P :: PROP . $\qquad (m_3)$ Q :: PROP .

$RH = \{ h_1, \dots, h_7 \}$ with

(h_1) true : ε . $\qquad (h_2)$ not false : ε .

(h_3) not not P : P.

(h_4) and P Q : P , Q . $\qquad (h_5)$ not and P Q : not P ; not Q .

(h_6) or P Q : P ; Q . $\qquad (h_7)$ not or P Q : not P , not Q .

The following W–production tree corresponds to the proof of the GS–theorem $\neg \vee \wedge\, tt\ ff\ ff$:

$$\begin{tikzcd}
& \text{not or and true false false} \arrow[dl, dash] \arrow[dr, dash, "(h_7)"] & \\
\text{not and true false} \arrow[d, dash, "(h_5)"] & & \text{not false} \arrow[d, dash, "(h_2)"] \\
\text{not false} \arrow[d, dash, "(h_2)"] & & {} \\
\varepsilon & &
\end{tikzcd}$$

3. Embedding PW–grammars into general (non–predicative) W–grammars

A PW–grammar G can be viewed as a W–grammar with trivial terminal layer. For many applications, it will not be satisfactory to describe a language L as accepted language $L_a(G)$ consisting of s–notions. One rather wants a W–grammar G′, for which $L_g(G') = L_a(G)$ holds, i.e. which additionally converts the s–notions of $L_a(G)$ into terminal strings. G′ can always be constructed from G using

Theorem 2: *For every PW–grammar G, there is a W–grammar G′ with* $L_g(G') = L_a(G)$.

Proof sketch: G′ is obtained from G by "duplicating" the upper level of G into a lower level of G′. For this reason, a new terminal set T is added to G, which is a 1–1–image of the s–notion set S of G. Every terminal t is called a *terminal representation* of its corresponding s–notion s, vice versa s is called a *nonterminal representation* of t.

A set of new hyperrules performs the conversion of s–notions into their terminal representations. (A detailed proof can be found in /HES 76/.)

Example 4:

The introduction of terminal representations into the PW–grammar of example 3 leads to the following W–grammar
$W' = (M, S', T, z, RM, RH')$ with $S' = S \cup \{z, \text{prop of}\}$,
$T = \{\underline{tt}, \underline{ff}, \underline{\neg}, \underline{\wedge}, \underline{\vee}\}$ and $RH' = RH \cup \{h_8, \ldots, h_{13}\}$, where

(h_8) z : prop of P , P .

(h_9) prop of true : $\underline{tt}$.

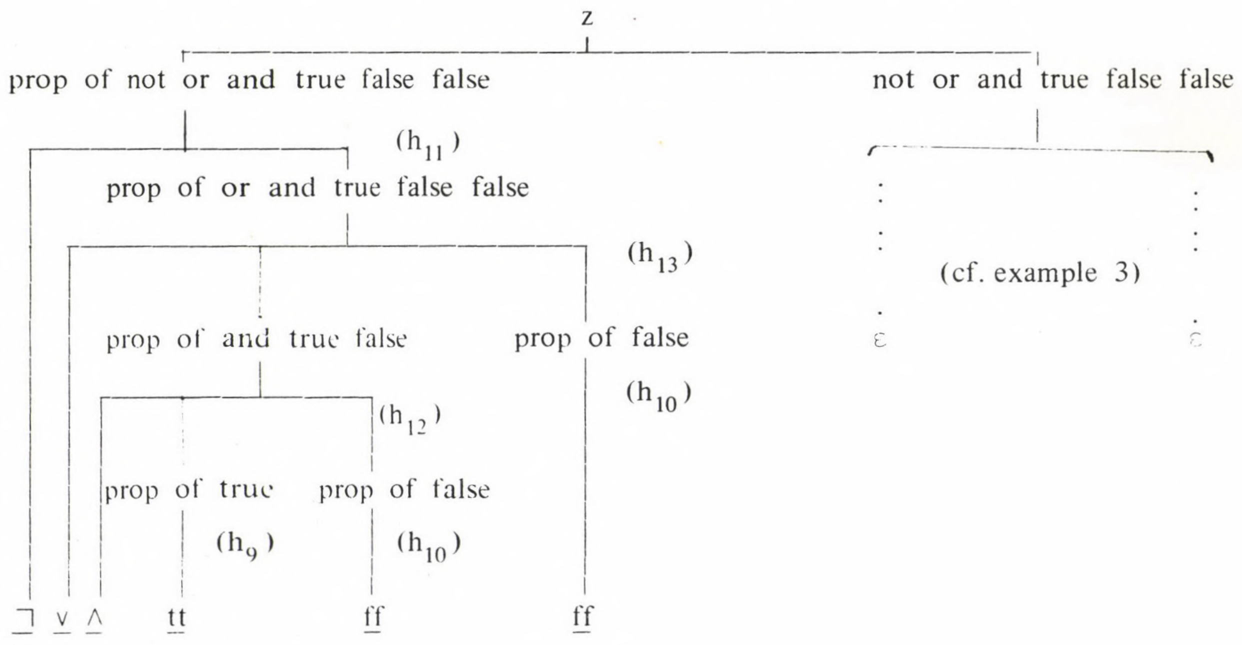
(h_{10}) prop of false : $\underline{ff}$.

(h_{11}) prop of not P : $\underline{\neg}$, prop of P .

(h_{12}) prop of and P Q : $\underline{\wedge}$, prop of P , prop of Q .

(h_{13}) prop of or P Q : $\underline{\vee}$, prop of P , prop of Q .

The production tree of example 3 may now be extended in the following way:


z
prop of not or and true false false
not or and true false false
(h_{11})
prop of or and true false false
(h_{13})
(cf. example 3)
prop of and true false
prop of false
ε
ε
(h_{10})
(h_{12})
prop of true
prop of false
(h_9)
(h_{10})
$\underline{\neg}$ $\underline{\vee}$ $\underline{\wedge}$ $\underline{tt}$ $\underline{ff}$ $\underline{ff}$

Co–existing nonterminal and terminal representations can already be found in /A68R/. For example,

"structured–with–integral–field–letter–i–and–reference–to–boolean–field–
–letter–r–letter–b"

is a nonterminal representation of a declarer, terminally represented as

struct (**int** I , **ref bool** RB) .

In a complete W–grammar language description (as advocated below), nonterminal representations will form the bridge between the "concrete" (string–producing) and the "abstract" (purely predicative) part of the description.

Combining theorem 1 and 2, we get

Theorem 3: *For every formal G–system GS, there is a W–grammar W with* $L_g(W) = L(GS)$.

4. Applications

4.1 Formal systems incorporated into W–grammar language descriptions

Using one direction of the correspondence theorem, every formal system can be incorporated into a W–grammar. In language descriptions, formal systems are used for the following two purposes:

a) for the formulation of context conditions, i.e. the context–sensitive part of the syntax (often called "static semantics")

b) for the formulation of the ("dynamic") semantics.

Examples for a) are formal systems, which describe the uniqueness of identifier declarations, the identification of applied occurrences of identifiers with their declarations, or the equivalence of modes. For the corresponding W–grammar predicates cf. section 7 of /A68RR/.

Chomsky 0–grammars (and Chomsky 1–grammars) can also be viewed as formal systems describing context dependences. The application of theorem 3 to them results in a new proof (detailed in /HES 76/) of the well–known fact that W–grammars are as powerful as Chomsky 0–grammars. This proof differs from the known ones (e.g. in /BAK 72/, /DEU 75/) in that the W–grammar productions simulate the Chomsky 0–grammar rules in a bottom–up manner rather than in a top–down manner.

Of even more importance are formal systems for the definition of the semantics or parts of it (b). The semantical aspects of the formal system / W–grammar–method are detailed in /HES7/. In the following only some examples are given:

1) The computation of recursive functions is described by the system of μ–recursive functions or by Kleene's system of general recursive functions. These systems have been formulated as W–grammars in /HES 76/. As a short example consider the PW–hyperrules for the recursively defined "plus"– and "times"–operations:

Example 5:

X plus o yields X : EMPTY .
X plus succ Y yields succ Z : X plus Y yields Z .
X times o yields o : EMPTY.
X times succ Y yields Z : X times Y yields W , W plus X yields Z .

If we assume, that every standard operation of a programming language is a recursive function, then we can describe all possible standard operations within the W–grammar system and do not need any additional descriptive means for this task (as many descriptions do).

2) A formal system for the λ–calculus may be used for the description of declarations of (user–defined) objects and, particularly, of (recursive) procedures. Various parameter mechanisms as leftmost–innermost substitution, leftmost–outermost substitution, Kleene's substitution rule and the ALGOL 68–parameter rule have been described by W–grammar predicates in /HES 76/.

3) There are formal systems for full semantics description, as Milner's system for denotational semantics based on Scott's logic (cf. /W–M 72/, /SCO 72/) and Hoare's system using the propositional method (cf. /HOA 69/). The PW–grammar formulation of Hoare's system is demonstrated in

Example 6:

PW–grammar W_H corresponding to Hoare's system H for propositional semantics

Metarules:

(m_1) STM :: ASS ; COMP ; ITER .
(m_2) ASS :: ID $\overline{:=}$ EXP .
(m_3) COMP :: $\bar{(}$ STM $\bar{;}$ STM $\bar{)}$.
(m_4) ITER :: $\overline{\textbf{while}}$ PROP $\overline{\textbf{do}}$ STM .

(These metarules describe the syntax of Hoare's example language, which comprises three sorts of statements (STM) : assignment (ASS), composition (COMP) and iteration (ITER). Nonterminal representations of delimiters are marked by overlining.

The metarules for proposition (PROP), identifier (ID) and expression (EXP), which are irrelevant for the system, have been omitted.)

(m_5) P :: Q :: R :: B :: PROP .
(m_6) S :: S1 :: S2 :: STM .
(m_7) X :: ID .
(m_8) E :: EXP .

(P :: Q :: . . . :: PROP is short for P :: PROP . and Q :: PROP . etc. These metarules serve for the introduction of metavariables, which are used in the hyperrules.)

Hyperrules:

(h_1 , rule of assignment)
Q | X $\overline{:=}$ E | P : subst X by E in P yielding Q.

(This rule corresponds to the H–axiom p^e_x ¦ x := e ¦ p, where x is a variable, e is an expression, p is a proposition and p^e_x denotes substitution of e for x in p.

The subst–predicate is defined such that

$$\text{subst } X \text{ by } E \text{ in } P \text{ yielding } Q \xrightarrow{*} \varepsilon, \text{ iff } Q = P^{E}_{X} .$$

For a detailed definition cf. /HES 76/.)

(h_2, rule of consequence)

$$P\,[S]\,R : P\,[S]\,Q\,,\,Q \supset R :$$
$$Q\,[S]\,R\,,\,P \supset Q\,.$$

(h_3, rule of composition)

$$P\,[\,\bar{(}\,S1\;\bar{;}\;S2\,\bar{)}\,]\,R : P\,[S1]\,Q\,,\,Q\,[S2]\,R\,.$$

(h_4, rule of iteration)

$$P\,[\overline{\textbf{while}}\;B\;\overline{\textbf{do}}\;S]\;P \wedge \neg B : P \wedge B\,[S]\,P\,.$$

(Further hyperrules are assumed to provide basic properties of logic and arithmetic.)

4) The full semantics of simple ALGOL–like example languages has been described in /C–U 73/ and in /WEG 74/, more complex languages are treated in /HES 76/ and /HES 77/. The latter contain among others the concepts of block structure, (recursive) procedures with ALGOL 68–parameter mechanism, identity declarations, variables with assignments, serial and collateral elaboration, conditional and repetitive clauses, composed objects (records and arrays) with selections, generators, in/output and nondeterminism in the notation of Dijkstra's guarded commands (cf. /DIJ 75/).

4.2 PW–grammars as logical calculi

Using the other direction of the correspondence theorem, PW–grammars may be used as formal systems for the derivation of theorems. In turn this applies to both PW–grammars which describe context conditions and PW–grammars for the semantics. In a formal system for the ALGOL 68 – mode equivalence, for example, the equivalence of the two modes

m = **struct** (**int** I , **ref m** R) and

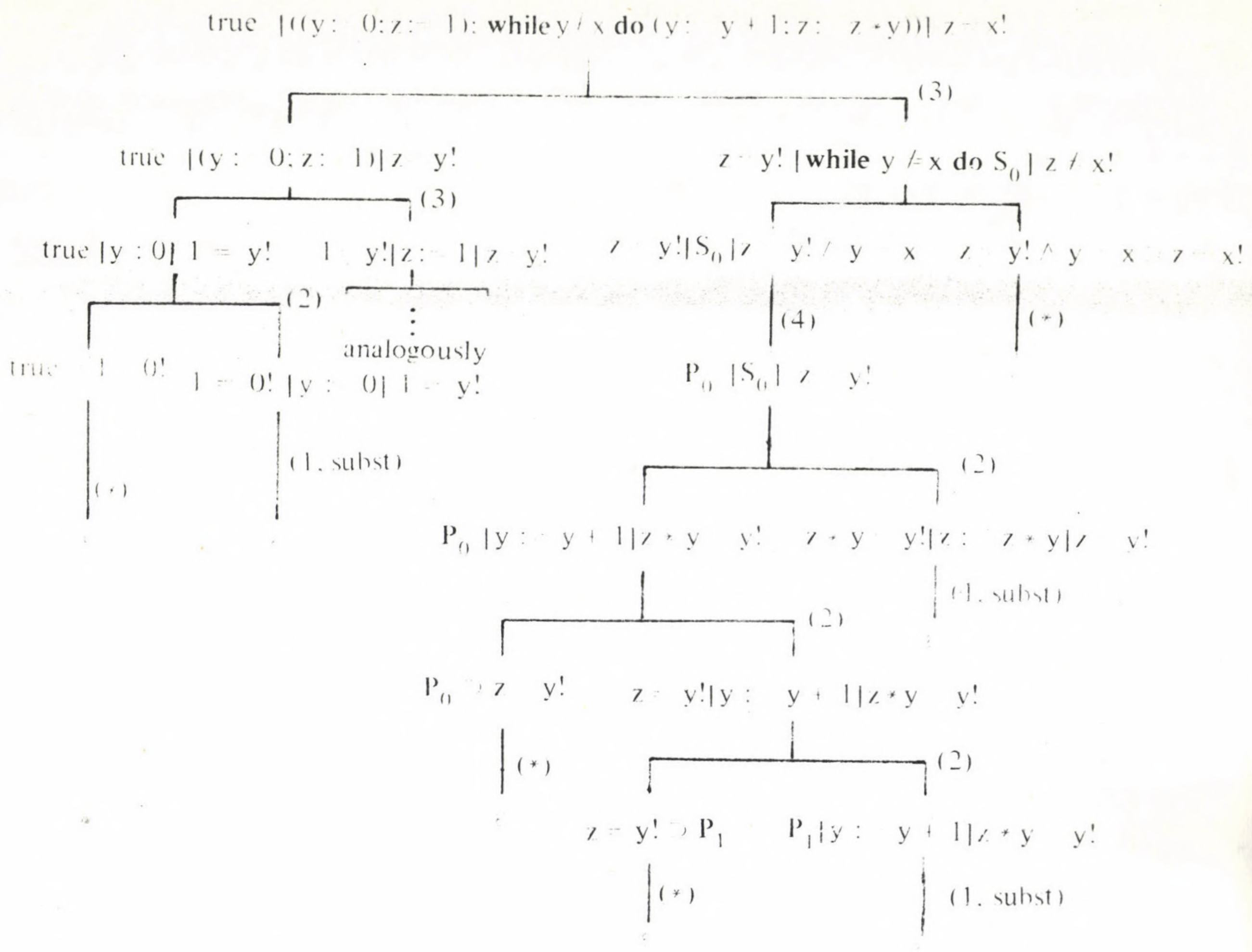
n = **struct** (**int** I , **ref struct** (**int** I , **ref n** R) R)

is a theorem which can be proved by the very hypperules of the language definition.

Partial correctness, termination or semantical equivalence are examples of theorems, which can be stated and proved (or disproved) by PW–grammars for the semantics.

Example 7:

From the PW–grammar W_H given in example 6, the (partial) correctness of a program computing the factorial function is proved by the following derivation tree. Let $x, y, z \ldots$ be terminal metaproductions of ID, analogously $x, y, \ldots,\; 0, 1, \ldots x + y,\; x * y,\; x!, \ldots$ for EXP: true, $x = y$, $x \neq y, \ldots$ for PROP. Let furthermore S_0 abbreviate the composed statement $(y := y + 1 ; z := z * y)$; P_0 the proposition $z = y! \wedge y \neq x$ and P_1 the proposition $z * (y + 1) = (y + 1)!$.


true |((y := 0; z := 1); while y ≠ x do (y := y + 1; z := z * y))| z = x!
(3)
true |(y := 0; z := 1)| z = y!
z = y! |while y ≠ x do S₀| z = x!
(3)
true |y := 0| 1 = y!
1 = y! |z := 1| z = y!
z = y! |S₀| z = y! ∧ y ≠ x
z = y! ∧ y = x ⊃ z = x!
(2)
analogously
(4)
(*)
true ⊃ 1 = 0!
1 = 0! |y := 0| 1 = y!
P₀ |S₀| z = y!
(*)
(1, subst)
(2)
P₀ |y := y + 1| z * y = y!
z * y = y! |z := z * y| z = y!
(1, subst)
(2)
P₀ ⊃ z = y!
z = y! |y := y + 1| z * y = y!
(*)
(2)
z = y! ⊃ P₁
P₁ |y := y + 1| z * y = y!
(*)
(1, subst)

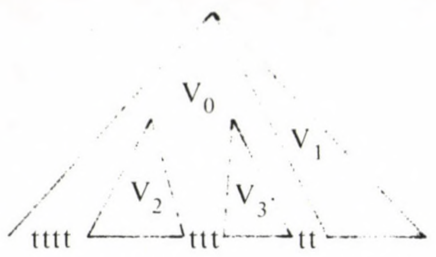
Natural numbers in parentheses indicate hyperrule numbers, (*) refers to logical or arithmetical rules, (subst) to substitution rules.

5. A model for complete formal language descriptions

In section 4 it has been shown, how W-grammars can be used for the formalization of the semantics as well as for syntax description including all context conditions. Combination of several W-grammars leads to the following model:

A *complete formal language description* is a W-grammar V consisting of subgrammars $V_0, V_1, \ldots, V_n$ $(n \geq 1)$ where V_0 is the *syntax subgrammar* (and the only string producing one), V_1 is the *semantics PW-subgrammar*, and $V_2, \ldots, V_n$, if necessary, are PW-subgrammars which describe context conditions. All syntactical and semantical domains are described by the metarules, while hyperrules serve a) for the production of terminal representations, and b) for the definition of predicates.

The typical form of a derivation tree obtained from such a language description (with $n = 3$) is shown in the following figure:



(tt . . . t : terminal strings)

Example 8:

Extend the PW–grammar W_H of example 6 by terminal representation rules for statements:

(h_5) repr of $X \overline{:=} E$: repr of X , $\underline{:=}$, repr of E .

(h_6) repr of $\bar{(} S1 \bar{;} S2 \bar{)}$: $\underline{(}$, repr of S1 , $\underline{;}$, repr of S2 , $\underline{)}$.

(h_7) repr of $\overline{\textbf{while}}$ P $\overline{\textbf{do}}$ S : $\underline{\textbf{while}}$, repr of P , $\underline{\textbf{do}}$, repr of S.

Together with the start rule

(h_0) z : repr of S , P [S] Q.

and the metarules the rules (h_5) – (h_8) form a W–grammar W_{H0}. The system $V = (W_{H0}, W_H)$ is a complete formal description of Hoare's language. Now, $z \xrightarrow[V]{} s$, iff there are a nonterminal representation $\bar{s}$ of s, and some pre– and postconditions p and q, such that $p \, \{ \bar{s} \} \, q$.

Some advantages of such a language description are:

– complete formalization (natural language is no longer needed as an auxiliary description tool, unless for comments),

– uniformity (one unique description tool for syntax, context conditions and semantics),

– universality, i.e. applicability to various semantical models,

– generality, compactness and modularity,

– usage of PW–subgrammars as logical calculi,

– existence of implementation systems for two–level grammars.

In particular, these benefits make W–grammars a well–suited tool for *defining* language descriptions, i.e. documents in the style and with the aim of reports such as /A60R/, /A68R/ or /A68RR/. The complete formalization of such reports is indispensable for rigorous and unambigous language definitions and would facilitate the derivation of particular descriptions for special purpose such as as program proving or implementation.

Acknowledgement

I thank my colleagues F. Geiselbrechtinger and B. Krieg–Brückner for many discussions.

References

[1] /A60R/ Naur, P. (Ed.) et al.: Revised report on the algorithmic language ALGOL 60, Num. Math. 4, 420–453 (1963).

[2] /A68R/ van Wijngaarden, A. (Ed.), Mailloux, B. J., Peck, J. E. L., Koster, C. H. A.: Report on the algorithmic language ALGOL 68, Num. Math. 14, 79–218 (1969).

[3] /A68RR/ van Wijngaarden, A. (Ed.) et al.: Revised report on the algorithmic language ALGOL 68, Acta Informatica 5 (1–3), 1–236 (1975).

[4] /BAK 72/ Baker, J. L.: Grammars with structured vocabulary: a model for the ALGOL 68 definition, Inf. and Controls 20, 351–395 (1972).

[5] /C–U 73/ Cleaveland, J. C., Uzgalis, R. C.: What every programmar should know about grammar, Lecture notes UCLA (1973).

[6] /DEU 75/ Deussen, P.: A decidability criterion for van Wijngaarden grammars, Acta Informatica 5 (4), 353–375 (1975).

[7] /DIJ 75/ Dijkstra, E. W.: Guarded commands, nondeterminacy and formal derivation of programs, Comm. ACM 18, 453–457 (1975).

[8] /HES 76/ Hesse, W.: Vollständige formale Beschreibung von Programmiersprachen mit zweischichtigen Grammatiken, Dissertation, TUM–INFO–7623, Techn. Univ. München (1976).

[9] /HES 77/ Hesse, W.: Formal semantics of programming languages described by predicative W–grammars, TUM–INFO–7728, Techn. Univ. München (1977).

[10] /HOA 69/ Hoare, C. A. R.: An axiomatic basis for computer programming, Comm. ACM 12, 576–580 (1969).

[11] /KOS 74/ Koster, C. H. A.: Two level grammars in: Compiler construction, an advanced course, Lecture Notes in Computer Sciene, 146–156, Springer 1974.

[12] /SCO 72/ Scott, D., Strachey, C.: Toward a mathematical semantics for computer languages, Oxford Mono. PRG–6, Oxford Univ. (1972).

[13] /SMU 68/ Smullyan, R. M.: First–order logic, Springer 1968.

[14] /W–M 72/ Weyrauch, R., Milner, R.: Program semantics and correctness in a mechanized logic in: Proc. First USA–Japan Computer Conference, 384–390 (1972).

[15] /WEG 74/ Wegner, L. M.: Two level grammars as a language definition system for syntax and semantics, M. S. thesis, Univ. Karlsruhe (1974).

Computational Linguistics and Computer Languages Vol. XIII. 1979.

PATHS AND TRACES

E.Knuth and Gy.Győry

Computer and Automation Institute Hungarian Academy of Sciences
Budapest, Hungary

Abstract

Cyclic concurrent systems can be described and analysed using "path expressions" introduced by Habermann, Lauer and Campbell. There is another formal device namely the concept of "traces" (i.e. partial orders induced by an independence relation) introduced by Mazurkiewicz which enables us to examine systems' behaviour. This paper is a first step to find the interconnections of the approaches mentioned and presents characterizations for some of the systems consisting of cyclic processes.

1. Introduction

When considering binary relations in the sequel we always relate *different* elements, therefore we are not interested in the fact whether a certain relation is reflexive or not. Notice this difference.

Let S be a set and $R \subset S \times S$ be a binary relation on it. We postulate *symmetry* by

(1) $$(s, z) \in R \Rightarrow (z, s) \in R, \qquad s, z \in S$$

and *antisymmetry* by

(2) $$(s,z) \in R \Rightarrow (z, s) \notin R, \qquad s, z \in S$$

supposing $s \neq z$, for there will be no case in consideration with $s = z$. In this sense we postulate a *partial order* as a binary relation being just antisymmetric and transitive. Remember the following facts: If P is a partial order, there always exists a total order containing P and

(3) $$P = \bigcap_{T \in \tau_P} T$$

where τ_P is the set of total orders containing P.

We shall call a relation R to be a *precedence relation* iff its transitive closure R^* is a partial order. Precedence relations are, therefore, antisymmetric, though, this fact still does not characterize them.

Let R be a precedence relation. Pairs not comparable in R, i.e. the complement of the relation $R \cap R^{-1}$ will be denoted by $I(R)$ and called *independent under* R. Relation $I(R)$ is obviously symmetric.

1.1 Mazurkiewicz's trace concept

Let V be a finite set called the *alphabet*. Let $I \subset V \times V$ be an arbitrary symmetric binary relation called the *independence* or *concurrence*. Denote $\sim_I$ the least equivalence relation in V^* satisfying

(4) $\quad (v, v) \in I \Rightarrow w'vvw'' \sim_I w'vvw'' \sim$

Since every $v, v \in V, w', w'' \in V^*$ *Traces* are defined as equivalence classes with respect to $\sim_I$.

The trace containing a given word $w \in V^*$ will be denoted by $[w]_I$ or symply $[w]$. When speaking of traces we abstract from the order of consecutive independent symbols. A trace could be considered as a partial order among symbol occurrences constituting it.

Usual operations among words, (concatenation, union, iteration) can also be defined for traces in the natural way allowing us to speak about *trace languages*, *regular trace languages*, etc. For details see Mazurkiewicz [1].

1.2 Simple traces

A *simple trace* is a trace containing at most one occurrence of each symbol of V. In the sequel simple traces will be considered only.

Introduce the following notations. Denote I_v the restriction of I to a given smaller domain $U \subset V$. Let T be a simple trace. Denote $|T|$ the set of words constituting T. Denote $op(T)$ the set of symbols occurring in it. Denote $a \rightarrow b$ the following facts:

i) $(a, b) \notin I$,
ii) a precedes b in one of the words (therefore in every word) constituting T.

Obviously, $\rightarrow$ is a partial order with

(5) $\quad I_{op(T)} = I(\rightarrow)$,

furthermore, from (3)

(6) $\quad \rightarrow = \bigcap_{w \in \{T\}} w$,

where each word is considered as a total ordering relation between the symbols occurring in it. The partial order $\rightarrow$ could be meant to be the trace T itself. A simple trace could also be considered equivalent to a *causal net* in the sense of Petri [2] with flow relation $\rightarrow$, and concurrence $I(\rightarrow)$.

1.3 Concurrence in safe nets

Mazurkiewicz [1] has axiomatically introduced the concept of *concurrent scheme* deriving it from the concepts of net theory [3]. In this paper, however, for the intuitive suggestiveness, we use a less formal way instead. We shall speak about safe nets simply, on which we shall mean 1-safe Petri-nets.

Let N be a safe net and M_0 be its initial marking. Transitions t_1, t_2 will be called *concurrent* iff

i) there is a reachable marking enabling both,
ii) the sets of their input (and therefore their output) places are disjoint (respectively).

The relation concurrence introduced will then be denoted by $C(N)$.

Mazurkiewicz [1] has reached the following results:

a) The firing order of concurrent transitions is immaterial and can be chosen arbitrarily.
b) Firing sequences leading from M_0 to any other given marking M_1 constitute a *regular trace language with respect to* $C(N)$.

1.4 Representation

The question of constructing nets from given trace languages is answered by Janicki [4] who has found a criterion whether a regular trace language is representable or not. Our present paper deals with the characterization of some cases. We also need, however, a proper representation concept and shall use the following.

Let R be regular language and I be a concurrence over V. Let N be a safe net and M_0, M_1 be two of its configurations (markings). We say the triplet (N, M_0, M_1) *represents* the language $[R]_I$ iff

i) There is a one–to–one mapping between transitions of N and operations occurring in R;
ii) $C(N) \subset I$, i.e. concurrent transitions correspond to independent operations;
iii) $[R]_{C(N)} = [R]_I$, i.e. concurrence of the net generates the same language as the one originally given;
iv) Every firing sequence starting from M_0 can be extended to an $M_0 \to M_1$ sequence i.e. a sequence ending in M_1.
v) Every $M_0 \to M_1$ sequence constitutes a word belonging to the language $|[R]_I|$;
vi) Every word belonging to $|[R]_I|$ is an $M_0 \to M_1$ sequence.

When considering languages of from R^* we always suppose $M_0 = M_1$.

2. Adequate marked graphs

Adequacy is used in the sense introduced in Lauer, Shields, Best [5]. A net N is *adequate* iff it is live–5 and safe. (N is live–5 iff any transition can fire after any previous firing sequence, see details in Lautenbach [6]).

Let t be simple trace. It follows from Mazurkiewicz's results that the language t^* is representable by an adequate marked graph (N, M_0, M_0) iff $M_0 \rightarrow M_0$ sequences containing unique occurrences of any symbol of t represent the simple trace t. These sequences will be called *elementary* $M_0 \rightarrow M_0$ words.

2.1 Sets of "bipole" cycles

Which we might call "*webs*" is the simplest special case of adequate marked graphs and will be defined as follows. A web is a collection of two–phase cycles having arbitrarily transitions in common. See e.g. fig. 1.

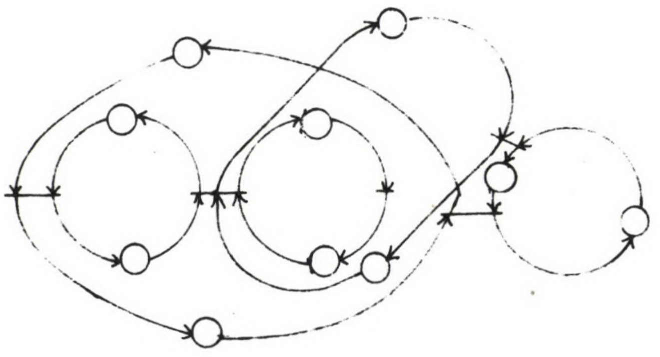


fig. 1.

It follows from Shields' adequacy theorem [7] that one can always choose an adequate initial marking for a given unmarked web structure. Obviously, every marked web is equivalent to a set of path expressions (defined e.g. in [5]) of the form

(7) $\quad | \underline{\text{path}}\ a; b;\ \underline{\text{end}} |, \quad a_i \neq b_i, \quad a_i, b_i \in V.$

Now let W be an adequate web. Introduce the following relation. Define $a \succ b$, $a, b \in V$ iff there is cycle in W containing them in the way shown in fig. 2.

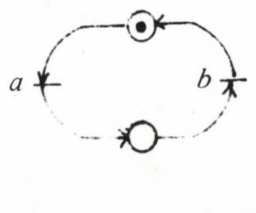


fig. 2.

Relation $\succ\!\!-$ is a precedence relation, that is, the transitive closure $\succ\!\!-^*$ is a partial order since in the contrary case there should exist a pair v, w such that $v \succ\!\!-^* w$ and $w \succ\!\!-^* v$, implying a circle through v and w containing no marker at all. This is impossible because of the well known liveness theorem for marked graphs.

Obviously, $a \succ\!\!-^* b$ implies that a must precede b in every elementary $M_0 \rightarrow M_0$ word. Moreover, $\succ\!\!-^*$ is the only condition of firability, i.e. transition b is firable iff it is not fired yet but every other transition a_i preceding b has already fired. In other words, elementary $M_0 \rightarrow M_0$ sequences represent a simple trace, namely $[s]_{C(W)}$ where the word s as a total order is an arbitrary extension of the partial order $\succ\!\!-^*$. Therefore, adequate webs always represent languages of the form t^*, where t is a simple trace.

Conversely, if t is a simple trace i.e. $t = [s]_I$ under given I then we can always construct a web representing t in the following straightforward way. Build a cycle according to fig. 2. whenever a letter a precedes an other letter b in the word s and a and b are not independent. The resulting whole net is an adequate web representing t^*.

We show a simple example. Let $V = \{a, b, c, d\}$, $I = \{(a, c), (b, d), (a, d)\}$ and $s = abcd$. Now $t = [s]_I$ consists of the single word $abcd$ for it contains no independent consecutive letters. The language t^*, however, contains words having independent consecutive letters and can be illustrated by an infinite precedence graph shown in fig. 3.

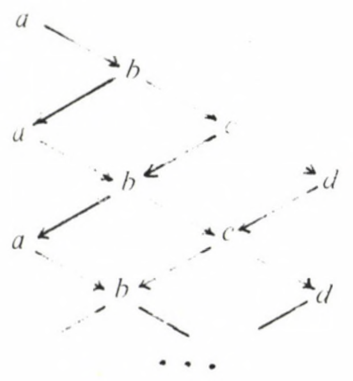


fig. 3.

The corresponding web expresses the same structure in the closed net shown in fig. 4.

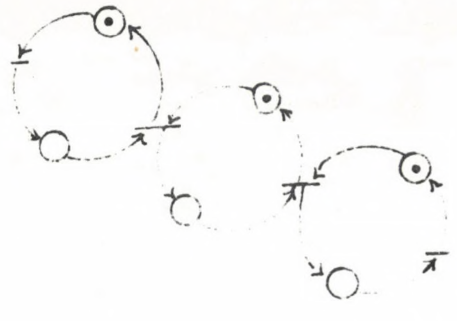


fig. 4.

Statements discussed so far can now be comprised in the following:

Theorem

i) For every web one can always give an adequate initial marking;

ii) Every web represents an iteration of a partial order that is a language of form t^* where t is a simple trace;

iii) Iteration of a simple trace can always be represented by an adequate web;

iv) Any adequate set of path expressions of the form $\underline{\text{path}}\ a_i; b_i\ \underline{\text{end}}$; $a_i \neq b_i$, $i = 1, 2, \ldots, n$. represents an iteration of a simple trace.

2.2 Relation between two kinds of independence

Having a set of path expressions $\{\underline{\text{path}}\ a_i; b_i\ \underline{\text{end}}\}$ or, equivalently, considering the precedence relation $\succ$ only, it is not obvious how the concurrence (independence) determined by the corresponding net (web) could be given. Though, it can easily be shown that

(8) $\quad t = [s]_{C(W)} = [s]_{I(\succ)}$

(where $I(\ )$ is defined in 1.), but $C(W)$ is usually different from $I(\succ)$. We can state only

(9) $\quad C(W) \subset I(\succ)$.

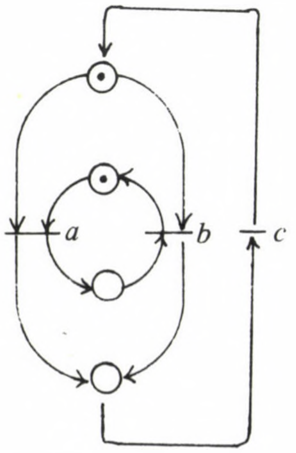
We shall show now, how the concurrence $C(W)$ can be derived from the characterizing precedence relation $\succ$.

Define a precedence relation $\succ$ to be complete iff

(10) $\quad a \succ^* b, b \succ^* c, a \succ c \Rightarrow a \succ b, b \succ c$

iff a and b occur in subexpressions separated by a comma. This relation expresses *exclusion* i.e. both operations can never occur together in an execution of p. (Operations c and d are exclusive in fig. 6.)

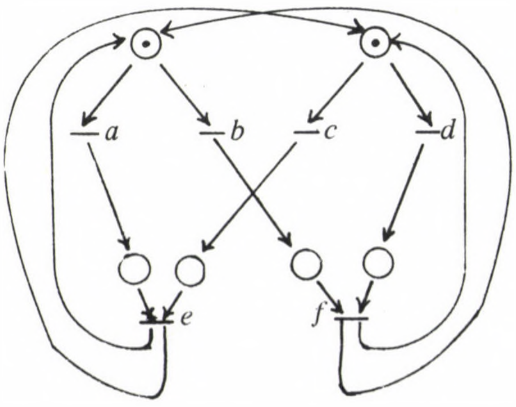
For a moment we might conjecture that a GE–net is adequate when there exist two disjoint relations $\succ$ and $\leftrightarrow$ over $op(E)$ projectible to all the relations $\succ_p$ and $\leftrightarrow_p$. Unfortunately, this is not the case. Fig. 7. shows a net which is adequate though there exists no disjoint projectible relation $\succ$ and $\leftrightarrow$ for $a \succ_{p_1} b$ but $a \leftrightarrow_{p_2} b$.



p_1 = path $a; b$ end
p_2 = path $(a, b); c$ end

fig. 7.

Moreover, it is possible that a net is not adequate in spite of the disjoint projectibility, see fig. 8.



p_1 = path $(a; e), (b; f)$ end
p_2 = path $(c; e), (d; f)$ end

Fig. 8.

Here transition α and d can fire and then the net is in a dead marking. We think, however, phenomena seen in figures 7. and 8. are the keys and a liveness theorem can be based on the properties of relations $\succ_p$ and $\leftrightarrow_p$.

The second problem we would consider very important is presenting trace languages representable by adequate GE–paths. This problem seems not very easy either. We have, however, a simple result transforming GE–paths into a normal form which is the extension of theorem 2. 3. iii, and which may lead to the desired language representation later.

Consider a net N derived from a set of GE–paths using transformation rules explained in 2. 2. of reference [5]. We introduce a new net N_{norm} in the following way:

i) Let $op(N_{\text{norm}}) = op(N)$;

ii) For every place of N form a subnet in N_{norm} according to fig. 9.

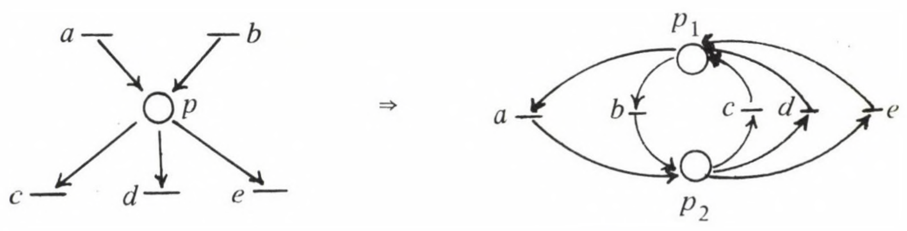


fig. 9.

iii) Mark p_1 if p is unmarked otherwise mark p_2.

As an example fig. 10. shows the normalized net corresponding to the net of fig. 8.

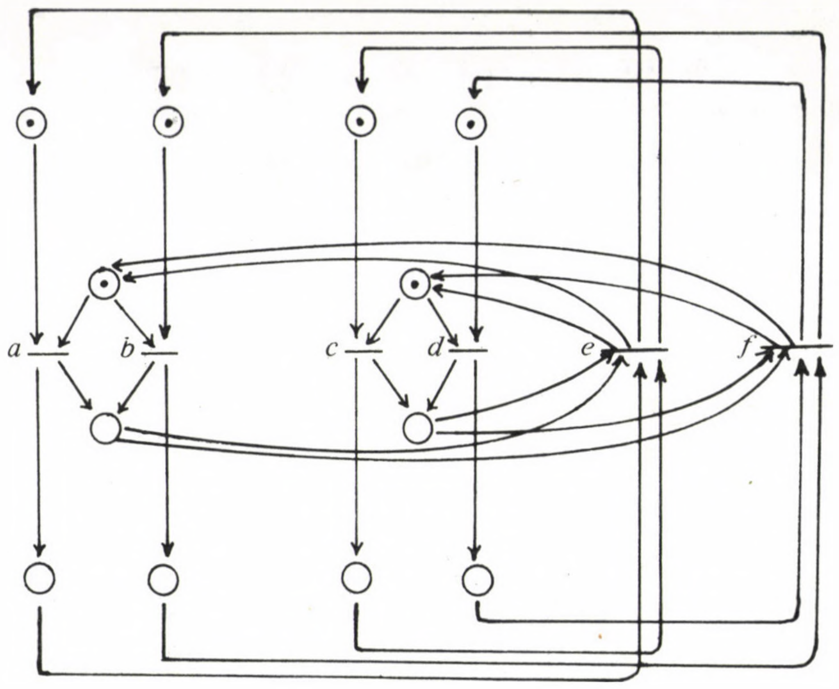


fig. 10.

Obviously, N_{norm} is always expressible by a set of paths of form

$$\underline{\text{path}}\ (a_{i_1}, \ldots, a_{i_{k_i}}); (b_{i_1}, \ldots, b_{i_{n_i}})\ \underline{\text{end}}$$

Our last theorem states that any set of GE–paths can be transformed into the above form, that is:

Theorem

Languages represented by N and N_{norm} are the same.

References

[1] Mazurkiewicz, A.: Concurrent program schemes and their interpretations, Aarhus Workshop on Verification of Parallel Processes, June 1977.

[2] Petri, C. A.: Nichtsequentielle Prozesse, GMD–ISF, Internal Report 76–6, Bonn 1976.

[3] Petri, C. A.: Concepts of net theory, MFCS 73 Proceedings, High Tatras, 137–146, 1973.

[4] Janicki, R.: Synthesis of concurrent schemes, MFCS 78 Proceedings, Lecture Notes in Computer Science 64. 298–308, 1978.

[5] Lauer, P. E., Shields, M. W., Best, E.: On the design and certification of asynchronous systems processes, Final report, ASM/45, University of Newcastle upon Tyne, 1978.

[6] Lautenbach, K.: Liveness in Petri nets, GMD–ISF, Internal Report, Bonn 1975.

[7] Shields, M. W.: Class of adequate path programs, ASM/42, University of Newcastle upon Tyne, 1977.

B. LINGUISTICS

Computational Linguistics and Computer Languages Vol. XIII. 1979.

HOW TO DO THINGS WITH MODEL THEORETIC SEMANTICS

T. Gergely and A. Szabolcsi

Research Institute for Applied Computer Science Budapest, Hungary

Institute of Linguistics of the Hungarian Academy of Sciences Budapest, Hungary

1. The aims of the paper

In recent decades our way of putting questions about natural language has reached a stage at which the application of nonnumerical i.e. qualitative mathematics is not merely possible but also appears to be of heuristic value. The two milestones seem to have been the introduction of the theory of formal languages into the study of natural language syntax, a merit of N. Chomsky (see e.g. [1]) and the introduction of model theory into the study of natural language semantics, most influentially by the works of R. Montague [2]. Here we will only be concerned with this latter.

In spite of the growing interest in *model theoretic semantics (MTS)* the penetration of those ideas into linguistic thinking does not proceed very smoothly. There have been several serious objections to MTS, tantamount to questioning its relevance for natural language. It is often difficult to tell whether those objections concern MTS as such or only particular uses of it; nevertheless, let us list some of the more general-looking ones:

(i) MTS overemphasizes the descriptive aspect of language, taking no notice of the communicative one,

(ii) meaning (in particular, word meaning) is a lot subtler than MTS believes,

(iii) MTS makes meanings relative to an arbitrary model and thus loses contact with the actual reality people talk about,

(iv) MTS has no psychological reality,

(v) MTS is primarily concerned with truth, which is irrelevant for natural language,

(vi) MTS is but an exercise in translation (of texts of the object language into some metalanguage) and so on.

It seems that these claims can take the form of objections because MTS is conceived of as a mere *device*, instead of being a *method*, and the possibility for this is provided by its introduction in the form "My model of language is such and such" – that is, in a purely mathematical form, without telling from which respects it is intended to be a model of language and from which it is not. Therefore, when wishing to do away with those objections, we first have to make clear what questions MTS puts and can possibly answer about language. This will be the task of Section 3. Such a specification can hardly be fully satisfactory on its own, however: it also needs to be shown how other questions, which are outside ot the scope of canonized MTS in view of Section 3 and which one would still like to put can be handled within the same methodological paradigm. The rest of the paper will be concerned with with some of these.

In other words, in this paper we do not aim at creating brandnew notions. Our aim is *to place MTS in a broader setting*, that is, to provide a coherent framework in which a number of current notions may receive their proper places. This is also a precondition to being able to decide how to improve the models we have available at present.

2. Methodology

Our task can only be accomplished if we make our backround assumptions as explicit as possible. This section is devoted to such preliminaries.

2.1 Language is an objectively existing abstract system, which is to be distinguished from its realizations and is to be studied as something self–contained. Being an abstract system, however, language can only be approached through its realizations. The results of investigation will thus to a great extent depend on what we regard as its relevant realizations.

Apart from the study of mere texts, the question of realization usually arises when one wishes to complete the notorious sentence "Language is a means of . . .". In general people tend to agree that models of language must somehow account for the fact the language can be and is used in cognition and communication. This is probably so because we have a functional view of language and a certain kind of language user in mind, which can in most general terms be called an *intelligent system* (see e.g. [3]). The least that this implies is that whatever one states about language must be compatible with whatever one happens to know about intelligent systems. We will actually use a stronger assumption, namely, that it is heuristically useful to look at language as functioning in some intelligent system (IS). Therefore a fundamental characteristics of the method to be followed in this paper is *activity–orientation*.

The second assumption is that seeking a unique answer to the ". . . a means of . . ." question is not fruitful. There are several language–using activities in which different, theoretically important aspects of language can be most readily studied. We shall first of all correlate such aspects with particular functions of language. Those functions appear in various activities, sometimes quite mixed up, sometimes rather clearly. Therefore we need to select such activities in which given functions of language feature most independently and perspicuously, in other words, *a simplest elementary activity necessary for realizing some theoretically important function*. We then form a model of that activity and study the model in order to see what it implies for language. The last step is to abstract from peculiarities of the activity and concentrate on language.

This assumption also implies that language is to be handled with a chain, or hierarchy, of models, rather than one single model.

2.2 Before turning to concrete problems, it is in order to dwell on the way of modelling those activities a bit longer. Having selected an elementary activity associated with a certain function of language, we consider a system realizing that function and an object at which the system's activity is directed, or which directly influences that activity. We refer to this object as the environment of the system in carrying out that activity. In this way, the function under consideration is made *relative* to the relation between the system and its environment. Moreover, we describe this relative situation from the position of an *ideal external observer,* thus introducing a *further level of relativization.* Let us spell out the observer's function more in detail.

(i) When studying something one always expresses one's basic assumptions about it by placing oneself into the position of an *ideal* observer. For instance, one says "Let's assume a system which is engaded in cognition". This does not mean that in empirical cases one would be able to unambiguously decide whether it is or it is not. However, this not being the point in the investigation, one may well assume that one possesses the sufficient knowledge to be able to tell. Furthermore, we assume along these lines that the observer's knowledge about the system, the environment, and their relation is sufficient for this models to be *adequate.* Thus the nature of idealization we employ can be in each case expressed in terms of the ideal observer's knowledge about the sample situation.

(ii) We also assume that the observer possesses a *(meta–) language* suitable for forming models of the sample situation. In order to keep the properties of the object (language) and the metalanguage strictly apart, we must assume that the observer is *external* to the sample situation.

(iii) Points (i) and (ii) also imply that models are not "absolute" – they are *relative* to the observer, both to this intentions and his limitations. It seems therefore methodologically useful to keep his position explicit throughout the discussion. We return to specific advantages of this at the end of Section 3.

3. Language in abstracto

3.1 The first question to put is: *what is language in the most abstract sense and what are its basic components.* (That is, we have an explication of syntax and semantics in mind.) Furthermore, we believe that this question is identical to the one model theory puts and therefore the nature of its answer is dependent on the conditions under which this question can be approached.

Can we take *communication* as the activity most representative of this problem? It is true that by studying communication linguists have gained revealing insights into the nature of language use but the aims are different in that case. As to our problem, we have to say that communication is neither simple nor transparent enough. It certainly has to do with semantics insofar as people "convey meanings" when trying to make themselves understood but in the case of communication the main thing is not merely what a text means but also how that meaning can be made available to the partner, also relative to the particular goals the communicative act is directed at. In other words, communication in the normal sense involves a need of *"adaptation" to the partner,* which is certainly irrelevant to our problem. The need of adaptation could be avoided if we assumed the two systems to be perfectly identical (with respect to both their "assumptions about the universe of the discourse" and their "means of expression"); this would make the situation more transparent but not yet simple. The reason is that it would be doubling a single system, rather than taking two systems, and thus we would be left with the question of what it means for a system to have "assumptions" and "means of expression". All this means that we ought to reduce communication to a trivial case and then in fact abstract from everything that makes it communication proper.

Therefore we suggest to abandon communication because of the interaction of at least two systems and propose to study the texts themselves before studying their transmission. This indicates that we must first investigate language as possessed by a single system and used for cognition. We take that the *cognitive activity* is the one in which texts are primarily produced.[1] Moreover, we will approach cognition from an angle which epistemological, rather than psychological.

3.2 In sum, we will consider *an elementary cognitive activity as going on between a system and its environment, as modelled by an ideal external observer.* By an elementary cognitive activity we mean that the system, possessing some language L, describes the objects in the environment. At the moment we abstract from the cognitive process itself, that is, from the possible experiments the system has to carry out, and from the goals of the system, from the precise knowledge obtained and its representation; furthermore, we also abstract from the internal organization of the system (whether it be a human or a machine and anything else). Our mere concern is the outcome of this activity, that is, descriptive texts and their relation to the environment.

Our dramatis personae will thus be a system S, the system's environment E, and an ideal external observer O. (We will pronominalize the system by "she" and the observer by "he", which has no significance aside from making the text more readable.)

To model this situation is a task of O; he forms models of S, of E, and of the S – E relation. In accordance with what we said above, as far as S is concerned, O only models her texts. A further important feature of O's modelling activity is that he models E independently of S. Note why it is so important: cognition is only possible if one distinguishes oneself from the object of one's reflections. In the present case we need not care about how S does so; nevertheless, this requirement is satisfied at O's modelling level.

1 This is not meant to exclude texts that are usually only used in conversation (e.g. questions, commands, performatives) from the scope of our discussion. It is the range of points of view of modelling rather than the range of texts that is restricted (see e.g. [4], [5]).

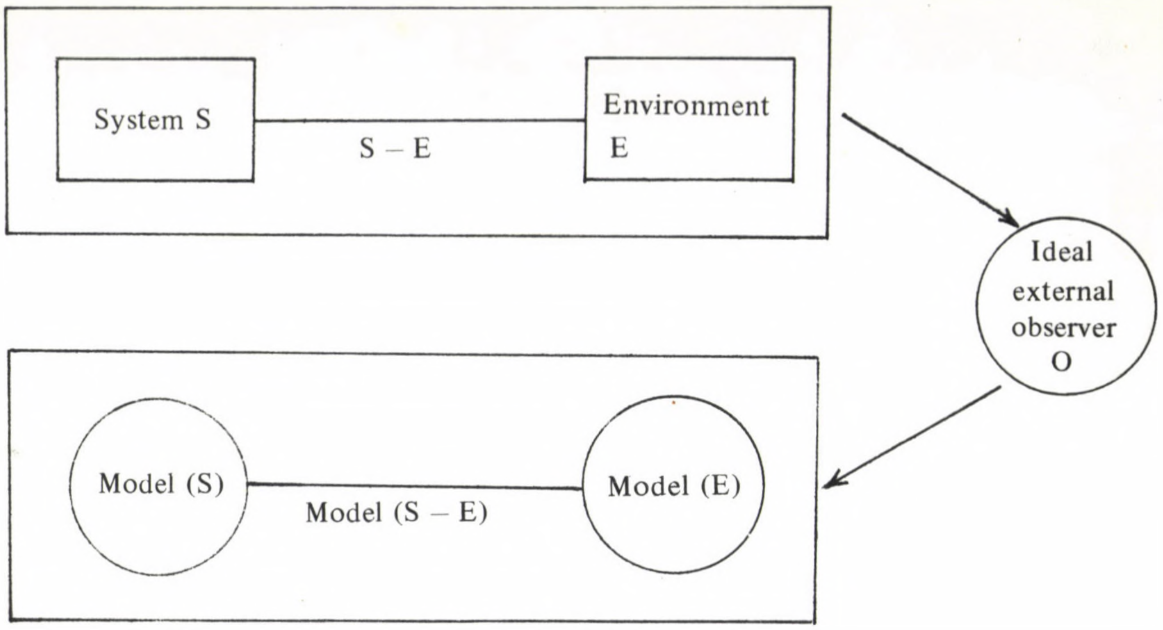


Figure 1. Language in abstracto

In order that O should be able to form the intended models, he must possess the following kinds of knowledge about the sample situation (and O being an ideal observer, we assume he really does):

(1) O knows the level at which S may perceive and describe her environment; in other words, he knows S's sensitivity.[2]

(2) O knows those fundamental aspects of E that S may describe.

(1)–(2) together ensure that O models E adequately with respect to S. Notice however that in spite of this adequacy, we cannot say that O established "S's model of E". At present we are not interested in how the system represents her environment (those questions will be tackled in Section 4) and this is not a matter of chance: for speaking about S's representation we first have to know what E itself is like (which, in view of the assumptions in 2.2 means that we have to speak about O's model of E). Without anticipating the

2 The sensitivity of S is actually manifest in the syntax of the language. For instance, let L_1 be the language of category theory, which handles objects and morphisms, without specifying what an object exactly is. Such an object may be a topological space or an algebra or a set etc. If now L_2 is the language of topology, or algebra, or set theory, then the sensitivity corresponding to L_2 is "greater" than that corresponding to L_1 since in L_2 one also takes the internal structures of L_1 objects into account.

discussion of representation, however, we can already state a point according to which Model (E) cannot coincide with "S's model of E":

(3) O knows that S is finite whereas E is both infinite and infinitely complex.

If we take infinity just in a spatial sense, we can say that, as a consequence of (3), S may never know in which part of E she is in. By infinite complexity we mean that E can be described at infinitely many different levels (say, at a "molecular" level, at a "meteorological" level, at a "touristic spectacles" level etc.), and having her fixed level of sensitivity, S may only grasp it at a finite number of levels. Therefore whenever S believes to be talking about some particular phenomenon, she is actually talking about all those possible ones that are identical from her respect but differ from each other in infinitely many other respects. With respect to the S – E relation this means that

(4) O knows that S's actual environment is accidental. The knowledge S may obtain at each stage of her cognition is compatible with infinitely many possible environments (differing in both "extension" and "depth"). The S – E relation is therefore uncertain: the texts of S always correspond to infinitely many environments, rather than a unique one.

On the basis of (1) – (4) O forms the following models:
The model of S will just be a system producing texts (more precisely, the material bodies of texts, whatever they should be). In case O happens to be a mathematician, Model (S) will be a formal grammar capable of generating the texts of the language.

The model of E is a metalinguistic description of the environment, adequate with S's sensitivity. For purely theoretical purposes, O only has to take into account that S has some fixed though arbitrary sensitivity, determining the possible character of the objects and phenomena of E S may describe. (When modelling some concrete language, S's sensitivity is also fixed though no longer arbitrary.) In case O happens to be a mathematician, Model (E) will be a mathematical object. Because of the uncertainty of the S – E relation, Model (E) is a class of models of infinitely many possible environments.

The model of the S – E relation is some correspondance between elements of texts and things in the world–models. In case O happens to be a mathematician, Model (S – E) can be a set of relations or functions.

We have reached the point where we may define language as it appears at this level of abstraction. *By an abstract language L_A we mean a triple ⟨Model (S), Model (E), Model (S – E)⟩*. Furthermore, we call Model (S) the syntax of L_A, and Model (E) and Model (S – E) together the semantics of L_A. All these models are formed by an ideal external observer and are described in his own language.

3.3 Let us now spell out some of the consequences of this way of modelling language, which we also claim to be the very level of idealization that MTS employs.

(a) MTS takes the descriptive function of language as its point of departure but not out of shere stubbornness. It does so because this is how the question as to the basic components of language can be answered most simply.

(b) MTS does not and need not have psychological reality because it has nothing to do with the representation of L in the system.

(c) The fact that MTS assigns a class of environment–models to a language (or, in other words, refers to an arbitrary model of it) is not a mere consequence of the mathematical apparatus MTS uses: it reflects the epistemological properties of the modelled situation, that is, the necessary uncertainty of the S – E relation.

(d) As a consequence of (b) and (c), word meanings proper are not objects of MTS; MTS may only take cognizance of their identity or non–identity. For more details, see Section 4 on representation.

(e) As to the objection that MTS is but a translation and thus leads to infinite regression. This argument might be assimilated to the following curious rephrasal of Gödel's theorem, which may bring out what is false in it: "The notion of consistency is useless, since you cannot prove that calculus$_1$ is consistent, within calculus$_1$. You have to use a metalanguage with calculus$_2$, whose consistency can only be proved within calculus$_3$. . . so you never can tell whether calculus$_1$ is consistent in an absolute sense". In other words, the requirement that you should be able to prove that your metalinguistic claim is consistent is equivalent to requiring that you should model your object and your own modelling activity simoultaneously. This absurdity is excluded by making the role of the observer explicit, i.e. by making the definition of L_A relative to an external observer.

(f) If already speaking about calculi, we may note that model theoretic semantics is not just an alternative to "calculus–semantics", since the question whether a calculus is sound and/or complete cannot be answered without telling what its intended class of models is.

(g) Note that we have not made any specific claim as to what kind of a mathematical apparatus is to be used, e.g. whether Model (E) is to contain classical relational structures or Kripke–models or intensional models or whatever else. That kind of choice depends on both the nature of the language O's models need to be adequate with and on O's own inventory of modelling tools. Here we may also note that truth only features in MTS as a metalinguistic device; to say that φ is true in a model m is but a mathematically comfortable way of expressing that a certain text (a sentence φ) corresponds to the situation modelled in m (cf. the model of the S – E relation).

In accordance with the assumption in 2.1, L_A is but one level in the hierarchy of idealizations one has to use when approaching language in its totality. Now we turn to the question of how a language can be represented in the system itself; its explication will hopefully also make the significance of L_A–idealizations clearer.

4. Language represented in an abstract system

4.1 We called language as defined as ⟨syntax, semantics⟩ an abstract language since when studying the cognitive situation we abstract from all properties of the system, except for one – we assumed S to have an arbitrary fixed sensitivity. (Notice though that even this was only made use of in modelling E, and not in modelling S herself.) L_A is not a system's language in the sense that a system might have it or use it; it is not even a language that systems might partially possess. L_A is an abstract construct, making the basic components of all such languages explicit. From a methodological point of view, L_A serves as a basis for investigating languages, used by systems, from the angle of their cognitive function.

Assuming that we shall once wish to construct some intelligent system IS we also have to explain what it means for a system to possess a language. For this, it is no longer enough to know what the basic components of language are – we also need to know how they are represented in a system. As a first step, we shall consider an abstract intelligent system and study the abilities necessary for representing and using some language in rather general terms. The concept of language to be formed on this basis is already more concrete than of L_A; we define a general IS–language, one corresponding to the representation of abstract language in an abstract system. For short, we call it a *representation language* L_R.

As we continue to study the cognitive function, the sample situation remains the same as we envisaged in 3.2. Nevertheless, this level being less abstract, the observer will be assumed to have some further knowledge about the situation. (Keeping in mind the synthesis problem we might alternatively think of this new situation as one in which someone, having the position of an external observer, introduces some language into an intelligent system: in that case instead of telling what O knows about S, we could tell what O grants to S.)

Our study being centered around semantics, let us assume S to have some text–generating device already (syntax). In order that S may use a language, however, she also needs something analogous to the semantics of L_A. The reason why we may only speak of an *analogue* of semantics here is that semantics, by definition, contains an infinite class of models of E, formed and described by the observer. S might only possess such a thing if she could treat herself and her own environment from the position of an external observer; an assumption which would only complicate the picture but would by no means eliminate the problems we have to cope with if we do not make it.

Let us consider O's knowledge about the situation. In 3.2 O already recognized that S describes E at a fixed level of sensitivity. Now, coming closer to S herself, O postulates that S has some kind of a *"perceptor"*, which determines the nature of her sensitivity (and through which she can receive influences from E – a precondition to internalizing it). Through this perceptor S gets *pictures* about her environment (the names "perceptor" and "picture" are intended to be most neutral as we continue to abstract from the specific organization of S; "pictures" can be thought of as S–specific changes in her internal structure, resulting from environmental influences and not vanishing with the moment but remaining stable in S). Pictures are assumed to be objectivistic in the sense that they are adequate representations of E–events, from some fixed point of view of adequacy. Nevertheless, pictures cannot be said to be "S's models of E" either. By a model we prefer to mean a result of some "software abstraction" (i.e. goal–oriented and deliberate). It is true that S's pictures are "abstract" – e.g. if she has a perceptor through which she can only perceive heat, then her pictures will be a "heat–abstraction" of her environment but this is not a model because this is a result of "hardware abstraction" (i.e. S cannot help abstracting from everything but heat). As a consequence of these, S cannot help identifying her pictures with real E–phenomena either (i.e. she does not treat her pictures as her own states).

In sum, the set of pictures S gets about her environment constitutes her internal representation of that environment.

Let us now see O's knowledge about the *environment.* At the level of L_A the observer did not need to care much about what the actual environment of S was like since he knew it to be accidental. At the representation level, however, this question also becomes crucial as S's actual environment determines her possible experiences and thus the nature of her pictures; we assume that O knows what S's actual environment is like. (Parallelly with saying that S has some arbitrary fixed sensitivity we can say that at this level O has to take into account some arbitrary fixed member of the class of models of E.) We call S's actual environment E – ACT. It is obvious that the objective relationship between pictures and real phenonema is impossible to determine at the level of the system (since her fixed sensitivity and her fixed E – ACT make her irrevocably subjective), it is only possible to determine at the observer's level.

The set of (lasting) pictures in S constitute *a system–dependent representation of some E – ACT.*

The next thing for O to observe is that there exists a *connection* between S's pictures and texts. (Some of the texts must be directly related to pictures whereas others only need to be reducible to them by operations.) This connection is granted to S from the outside so to say, by ostentic (deictic) definition. This assumption serves to emphasize the hardware character of the language represented (i.e. that S has no additional language to talk about this connection). As for the case when S has to apply this language to further E – ACTs, see 4.3.

Having specified O's knowledge about the sample situation, we can tell how he models it.

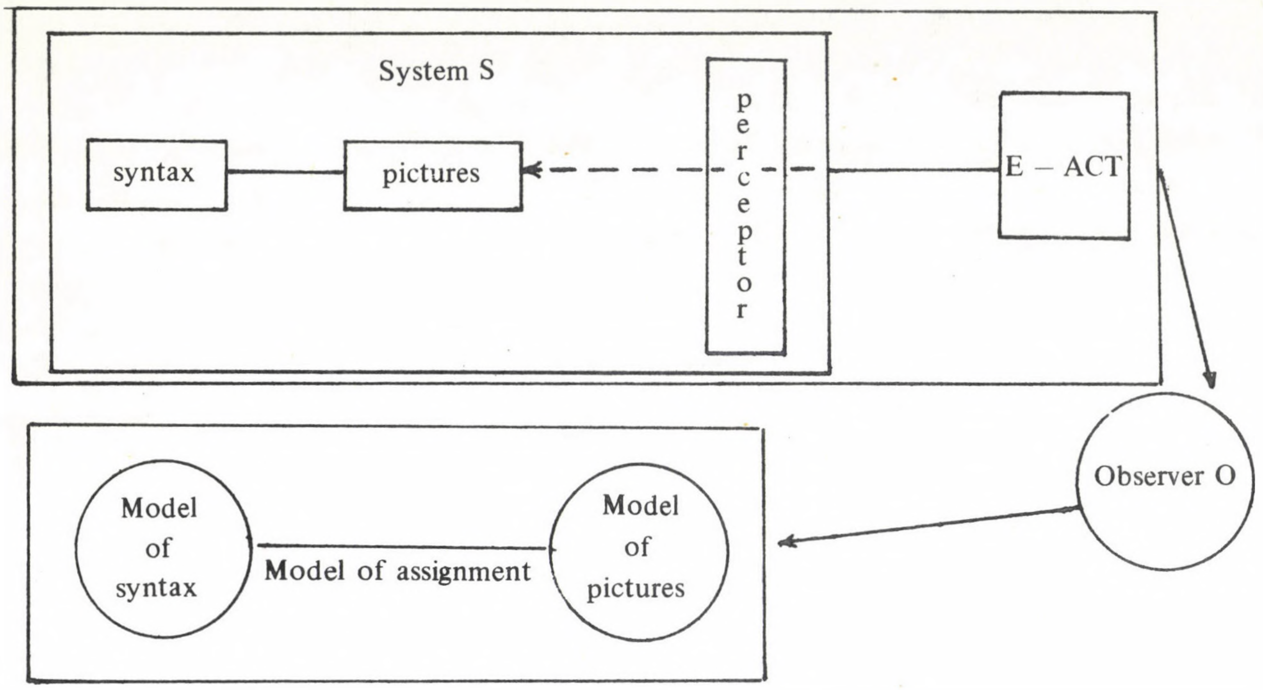


Fig. 2. Representation language

In the observer's metalanguage names of S's syntactic units and descriptions of S's pictures from pairs. These pairs count as definitions, definiendum being a syntactic unit and definiens being the description of a picture; their pairing models the connection between the respective items in S. O calls his own description of a picture the *meaning* of the syntactic unit it is assigned to. Notice the sharp difference between the statuses of pictures and meanings: in S there are only pictures – meanings exclusively belong to O's metalevel.

Fixing the terms we shall use in connection with L_R : *a representation language is a pair ⟨syntax, interpretation⟩, where interpretation consists of a set of meanings plus a meaning–assignment.*

4.2 Let us now add a few comments on L_R .

(a) As opposed to L_A , the semantics component of which contains a class of models of E, L_R contains no model of any environment whatsoever. Its interpretation component contains meanings, that is, a description of some system–dependent representation of some E – ACT.

(b) Semantics is objective (i.e. relative merely to the observer) whereas interpretation is subjective (i.e. relative to both observer and system). A corollary of this is that to an L_A there correspond infinitely many L_R 's, in accordance with the possible subjective ways of representing some part of the environment.

(c) L_R is the level where *word–meanings* can be treated in the usual linguistic fashion, that is, by analysing them in terms of oppositions, features etc. In other words, "word–semantics" as opposed to "sentence–semantics" (as these terms are used in linguistic jargon) is a representation problem. The claim that meanings belong to the observer's metalevel does not contradict this: remember that in usual word–semantics one does not investigate what is actually in "people's heads", either. Furthermore, the fact that we treated the language represented on a purely hardware level (i.e. we did not assume the system to be able to talk about her pictures) makes no big difference in this respect. On the one hand, in the course of such an investigation one actually always ignores the metalinguistic abilities of the language–user. On the other hand, whatever metalinguistic abilities one may attribute to a system, those may not have the whole of her representation language in their scope: any system must have a purely hardware level language, analyzable for some external observer only. (For a treatment of word–meanings similar to ours, see Pavilionis [6].)

(d) The general treatment of the $L_A - L_R$ relation would also require us to model the relationship between pictures and E – ACT and, further, to model the relationship between E – ACT and E. These tasks are outside of the scope of the present paper.

4.3 Let us see what happens if the sample situation remains the same as in 4.1 with the exception that S is assumed to face more than one fixed though arbitrary E – ACT. More exactly, we assume that in connection with one and the same syntactic unit S may form various pictures on the basis of several E – ACTs and she may also link them together. As far as this linking is concerned, there are basically two possibilities. The first possibility, which is actually very close to the situation sketched in 4.1 is that these pictures function as elements belonging to different L_R's, with the possible variation that S may also be granted further quasi–metalinguistic devices for identifying the syntactic units related to those pictures. The L_R–model of this situation will in turn also contain a model of this quasimetalinguistic connection and a model of the relations between pictures assigned to the same syntactic unit. A more interesting second possibility is that S does not merely link those pictures together in the manner described above but she also forms some kind of a secondary picture out of them (where by 'forms' we may either mean some goal–governed software abstraction or just some hardware abstraction in the case of which those secondary pictures are actually formed by the teacher–observer and are built into the system). Such secondary pictures are conceived of as concepts that comprise the features common to all (primary pictures of) real phenomena that are associated with the same syntactic unit. The sharp difference between this possibility and the former one consists in the fact that in case S is faced with an n + 1th E – ACT, in the first case she has to wait as long as O also teaches her to handle this E – ACT as well, whereas in the second case she may apply the syntactic units she already has to the pictures she gets about this new E – ACT, using the respective concept as a mediating device. In other words, although her concepts are formed on the basis of some designated E – ACTs, these concepts are further applicable to brand–new ones as well, which grants a great amount of independence to S (a manipulated kind of independence, though). We may also note that concepts (secondary pictures) are already quite similar to what we would like natural linguistic meaning to be models of.

Both possibilities agree in that they have important consequences for the L_A –level. So–called classical (that is, purely extensional) models of E seem to be intuitively adequate for the more primitive situation which we described in 4.1, whereas the more complex cases of representation are matched by intensional models of E at the L_A – level. Intensional models (or, non–classical models in general) differ from classical ones in that they regard different possible worlds or things occuring in different possible worlds as alternatives to one another, in other words, as possible realizations of one and the same thing. In this sense intensions are L_A – level counterparts to L_R – level conceptual meanings: meanings specify *why* the system applies the same syntactic unit to different things she encounters, whereas intensions specify *how* she would have to use them in all possible situations. The fact that S's concepts are formed on the basis of a subclass of all possible E –ACTs can be reflected at the L_A – level by the use of meaning postulates as employed by Montague: meaning postulates being non–logical axioms, they can be thought of as restricting the class of all possible worlds to the subclass which conforms to S's initial E – ACTs.

In sum, intensions and (conceptual) meanings belong to two radically different levels of idealization but still closely correspond to each other, which justifies the use of intensions as reflections of meanings at the L_A – level. The radical difference between them, however, makes it clear why intensional semantics may not be expected to account for traditional questions arising in connection with word–meanings: the meanings that are analysable in terms of oppositions etc. are par excellence L_R – level units (i.e. models of S's pictures).

5. Goals and cognition

Very briefly, we may sketch a third level of abstraction in this paradigm, in which we also take the existence of nonlinguistic components of S into account. We may say that the functioning of those non–linguistic components can be at an abstract level characterized with a set of goals (e.g. that S wants to survive, at least). In order for S to achieve those goals she must have a perceptor adequate with her goals (e.g. if she has to be afraid of microbes, she must be able to perceive them so that she can avoid them). Therefore the kind of L_R she has is a function of her goals. Obviously, this implies that goals are granted to S at a hardware level, too. In the case of an intelligent system proper (e.g. if S is an adaptive system) we assume that in addition to hardware–goals she can set a number of further goals in the course of her functioning, which also necessitates that she should be able to alter her sensitivity and L_R . This problem, however, belongs to the scope of the theory of intelligent systems rather than to the scope of linguistics, and since at present we focus on questions strictly connected with language we do not elaborate at this point here.

Notice that in Sections 3, 4 and 5 we did not talk about three different systems: the system remained the same but was viewed at different levels of abstraction.

Even if one does not wish to go into linguistic and mathematical details it is apparent that at least two large sets of problems are missing from the above treatment: (i) the application of the results of Sections 3 and 4 to more complex cases, approximating the complexity of natural language, and (ii) the treatment of communication in the same methodological paradigm. We are convinced that (i) and (ii) do belong here: nevertheless, they must be objects for futher research.

References

[1] Chomsky, N.: Syntactic Structures, Mouton, 1957.

[2] Thomason, R. (ed.): Formal Philosophy – Selected Papers of Richard Montague, Yale University Press, 1974.

[3] Bratko, A.A. and T. Gergely: Definition of intelligent system, Soviet Automatic Control N°3, 1977.

[4] T. Gergely and L. Úry: A theory of interactive programming within the frame of first order classical logic (to appear).

[5] A. Szabolcsi: Model theoretic semantics of performatives, in Kiefer, F. (ed.) Hungarian Contributions to General Linguistics, John Benjamins, Amsterdam, 1979.

[6] Pavilionis, R.: On "the global" conception of meaning, Kalbotyra XXVI 3, pp. 27–35, Vilnius, 1975.

Computational Linguistics and Computer Languages Vol. XIII. 1979.

TRANSFORMATIONS OF GENERATIVE GRAMMAR: THE RISE OF TRACE THEORY

I. Kenesei

Attila József University

Szeged, Hungary

Abstract

Empirical arguments againts the 'standard theory' of generative grammar led to its revision in the form of the 'extended standard theory' (EST). Owing to its impracticability, EST has now been replaced by the 'revised extended standard theory'. Trace theory, as this latest offspring of generative grammar has come to be called, revived the idea of semantic interpretation in one block – with near-surface structures as input. This paper examines the sources of the revisions as well as their effect on the syntactic components of the various models, and discusses the claims concerning crucial issues such as the power of grammars, the autonomy of syntax and restrictions on rules *versus* universality. Since it draws heavily on the available literature, the paper makes no pretence to originality in a number of problems touched upon.

1. The first cracks

1.1 When the uniformity of the principles of linguistic description of Chomsky's *Aspects of the Theory of Syntax* was replaced by the 'mixed' grammar of the extended standard theory (EST), it was the result of the effects of several interrelated findings. In the first place, some of the basic ideas of the earlier version proved to be untenable, as anticipated already in *Aspects*: "this claim [that the semantic interpretation depends only on deep structure] seems to me somewhat too strong [. . .]. It seems clear that the order of 'quantifiers' in surface structures sometimes plays a role in semantic interpretation" (Chomsky 1965:224). Secondly, the idea of the semantic interpretation of deep structures in one block ensued from Katz and Postal's thesis which stated that transformations do not change meaning. At the time it was thought to be an interesting assumption but was subsequently found to be too strong in the light of empirical evidence. Thirdly, there was a growing concern in certain formal properties of both phrase structure and transformational rules. Not long after the *Aspects* model had won popularity it was shown, most of all by Peters and Ritchie (1969, 1971, 1973), that the unrestricted nature of the apparatus made the grammar equivalent to a Turing–machine. "That is to say, any language that can be defined by a Turing–machine or an unrestricted rewriting system can be defined by a transformational grammar and vice versa. This result is somewhat disconcerting. It shows that claiming that transformational theory provides a theory of possible natural languages is making no stronger claim than that natural languages are systems of some sort." (Bach 1971:4) Note that there were already a number of conditions of various types on rules of grammar in *Aspects* without, however, any effect upon restricting the power of the grammar. Furthermore, it is important to see that these conditions notwithstanding no formal differences were found between natural languages and other systems generally believed to be much more powerful. For those who accept the doctrine of 'linguistics within psychology' this was tantamount to having uncovered nothing specific in the structure of the mind, whereas for those who disbelieve it, the theory must inevitably have appeared as formally uninteresting or even vacuous. Serious work therefore began to overcome these difficulties. Finally, the dispute between the proponents of 'generative semantics' and those of the 'standard theory' on the autonomy of syntax had a catalytic effect on Chomsky's and his collaborators' attempts at refurbishing the old model.

1.2 Before beginning to outline what the 'extended standard theory' proclaimed to have achieved, let us take a closer look at the issues listed above.

Obviously, the first two points are interrelated in a fashion that throws doubt upon the significance of one or the other. For the question whether transformations preserve or change meaning is the same question as whether deep structures alone determine meaning (i.e. only deep structures are interpreted by the semantic component) or surface structures as well contribute to semantic interpretation. What seems reasonable to ask is only whether in a particular grammar two distinct phonetic strings of sentences derived from identical deep structures are synonymous or not, as a piece of empirical evidence for (or more often againts) that particular grammar or some subpart thereof. Our contention is that it is no property of any transformation whether or not it preserves meaning unless we also accept the assumption that transformations operate on strings of semantic units. We can of course rephrase the original question by asking whether the meanings of sentences with identical deep structures will differ as a result of the application of transformations and only of those. Sure enough, in the 1965 model they are not allowed to. Surface structures are paired with semantic interpretations, therefore it is per definitionem impossible to obtain two surface structures that are derived from the same deep structure and are not synonymous. This is no doubt a formal answer and internal in the sense that it is related to a particular theory. But it must be clearly understood that the terms meaning (or rather semantic interpretation), deep structure, surface structure, transformations and so on are defined within a particular grammar, i.e. theory of language. What we do when we check two sentences for synonymy or the like is testing the empirical validity of the grammar.

It may of course very well happen that a grammar of English or any other language contains transformations which are required for well–motivated reasons but whose operations on certain deep structures will ultimately result in non–synonymous superficial sentences as established by independent and reliable tests. But then the grammar is simply empirically (or, in a more technical term, descriptively) inadequate. If, for example, in a grammar of the *Aspects* type, the movement of quantifiers by some optional rule of Passive in presumably identical deep structures ultimately yields two distinct and (at least on one reading) non–synonymous sentences such as (1) and (2):

(1) Many men read few books.

(2) Few books are read by many men.

then we ought to conclude that either there can be no Passive transformation and/or there are distinct deep structures underlying (1–2), but not that transformations (or some of them, some of the time) change meaning.[1]

This problem was irrelevant in the first period of transformational grammar since grammar was regarded as "autonomous and independent of meaning", and it became irrelevant again when Chomsky had completely abandoned the view of making semantic interpretation a function of deep structures. But, as has been shown, it was probably uninteresting if not void even when it could have been of any relevance.

1.3 The issue discussed in the previous section concerned a 'substantive' property. The power of grammars is, however, a purely formal topic, since in this case the relationship of grammars to systems in general is at stake. Undisputably, the statement that language (*la langue*) is a system was very important, and many of de Saussure's findings still hold valid in their own context. However, to content ourselves with what is a generality today (even though the answer to the question what language is a system *of* may vary from time to time and school to school) would be a grave error. Why language is a unique system (if it is) and not just one of an infinite array of systems, is a problem that can be solved by specifying the restrictions, formal or substantive, which operate in grammars. Examples for formal restrictions are the thesis of the recoverability of deletion (Chomsky 1965) or the A–over–A principle (Chomsky 1964), while some of the substantive restrictions are the various constraints on the movement of constituents (Ross 1967) or on surface structures (Perlmutter 1971). It may turn out that formal restrictions are universal, whereas substantive ones are language–particular. But even if that were the case, it would not preclude the possibility of finding general enough or even universal principles underlying substantive constraints. Although Peters and Ritchie's research and Chomsky's own efforts were directed towards assessing and constraining, respectively, the power of the syntactic component, it should be kept in mind that it is the power of the grammar as a whole which is investigated. And if restrictions on the rules of syntax are imposed at the expense of increasing the power of some other (notably the semantic) component, nothing will of course change in the overall picture.

2. Topics in the dispute on generative grammar

2.11 The extended standard theory (EST) was called a 'mixed' grammar to indicate that it differs from the standard theory (ST) in that EST carries out semantic interpretation at two stages while in ST semantic readings are assigned to deep structures in one block. As was mentioned above, the change was called for by a large amount of data which were used to demonstrate that interpretation based on deep structure alone was insufficient to determine the semantic properties of sentences.

Another important and, with some modification, still valid innovation over the *Aspects* model was the introduction of the 'X–Bar Convention' which delimited the possible types of base rules. Let us first recall what the rewriting rules of ST were like:

> "A rewriting rule is a rule of the form
>
> $(\ )\ A \rightarrow Z / X — Y$
>
> where X and Y are (possibly null) strings of symbols, A is a single category symbol, and Z is a nonnull string of symbols. This rule is interpreted as asserting that the category A is realized as the string Z when it is in the environment consisting of X to the left and Y to the right. Application of the rewriting rule ( ) to a string $\ldots XAY \ldots$ converts this to a string $\ldots XZY \ldots$"
> (Chomsky 1965:66).

Apart from the purely theoretical interest in limiting the types of base rules, syntactic similarities between nominal and verbal constructions such as (3) and (4) were also instrumental in introducing the new device.

(3) (3) a. the enemy's destruction of the city
b. the city's destruction by the enemy

(4) (4) a. The enemy destroyed the city.
b. The city was destroyed by the enemy.

Still more important was the fact that there were a number of incongruities found between verbal expressions (tensed sentences and gerundive constructions) and nominal ones, cf.:

(5) (5) a. John is likely to win the prize.
b. John's being likely to win the prize.
c. * John's likelihood to win the prize.

The difficulty of accounting for the semantic dissimilarity of (6) and (7–8) also contributed to the shaping of the new principle since (6) can be paraphrased by neither of the pair (7–8), although the subject NP of (6) used to be thought to derive from something like that of (7) or (8).

(6) (6) John's intelligence is his most remarkable quality.
(7) (7) The fact that John is intelligent is his most remarkable quality.
(8) (8) The extent to which John is intelligent is his most remarkable quality.

Since the problems just presented can be solved in two ways, two positions crystallized by the end of the sixties. The transformationalist, which held that there was a transformational connection between tensed sentences, gerundive constructions and nominal expressions, i.e. they are all derived from the same underlying structure with specifications on the nonapplication of the rules; and the lexicalist, which maintained that there was no nominal-ization transformation since structures like (3a) can be derived from a deep structure which is distinct from that of (4a) but incorporates the relevant relationships by means of subcategorization excluding certain contexts for nominals. This is achieved by imposing on the rules of the base the X–Bar Convention (Chomsky 1970), which in effect requires that every category of the type X be rewritten as a category of the type X and a phrase associated with X and labelled as *Specifier of* X. The X immediately dominating another category X is marked $\bar{X}$, the category dominating $\bar{X}$ is then $\bar{\bar{X}}$. To put it formally

(9) $\bar{\bar{X}} \rightarrow$ [Spec $\bar{X}$] $\bar{X}$

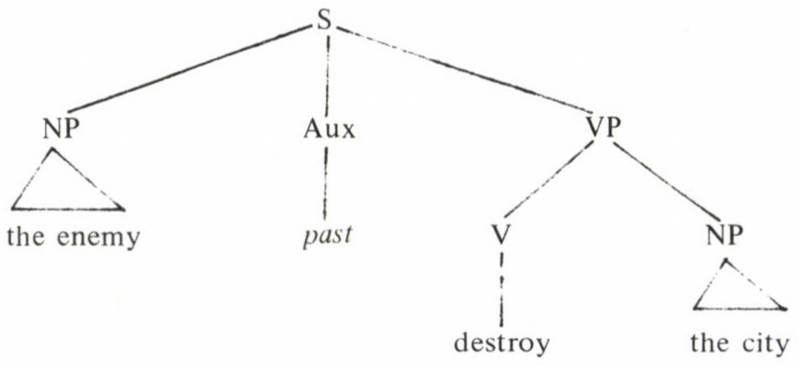
where X can be N, V, or *Adj*. Then $\bar{\bar{V}}$ can stand for S, $\bar{\bar{N}}$ for *NP*, and $\bar{\bar{A}}$ for *AdjP*. The next rule is (10):

(10) $\bar{X} \rightarrow X \ldots$

where in place of the three dots complements of V, N, or *Adj* can occur – themselves possibly of the type $\bar{\bar{X}}$.[2]

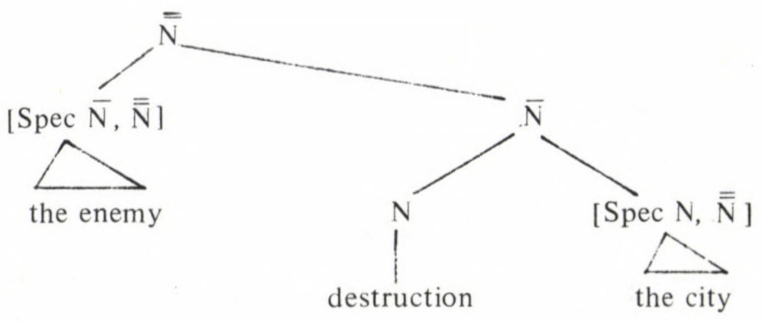
Thus, according to the transformationalist position, the tree in (11) ultimately underlies the constructions in (3–4) as well as the related gerundive nominal:

(11)

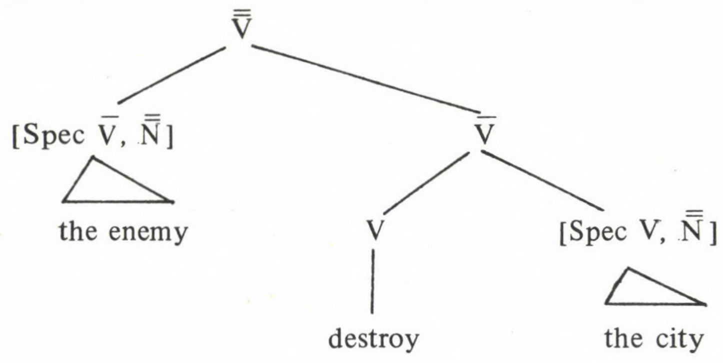


whereas within the lexicalist framework there are two distinct structures (12a) and (12b), and gerundive nominals are derived from (12b) in effect:

(12a)



(12b)



Of course, several details are omitted from (12a–b) but they do not affect the validity of the argument. A base component which incorporates the X–Bar Convention (or theory, as it has come to be called) will not only capture a significant general property of the language whose theory it is a subpart of, but will also simplify the transformational component to a great extent by eliminating rules of nominalizations, the conditions on which are extremely difficult to specify. Furthermore, it is now necessary to drop the view that the domain of the transformational cycle is the sentence since the examples in (3) and (4) show that NPs too may undergo transformations like Passive. In other words, there are now two cyclic categories in the grammar: S and NP, or equivalently, $\bar{\bar{V}}$ and $\bar{\bar{N}}$.

2.12 By the time of the formulation of EST the influence of John Robert Ross's seminal dissertation was widespread. It set out to constrain movement transformations by limiting the domain of individual rules rather than stating general restrictions on the transformational component in toto. Its basic purport was to show which constituents cannot be moved from which configurations (cf. Emonds 1970).

Here we present the Coordinate Structure Constraint to give an example. We can regard question and relative clause formation as a movement transformation on an NP in roughly the fashion of the examples below:

(13) a. [Q Henry plays wh–something] → What does Henry play

b. [the lute [Henry plays the lute]] → the lute which Henry plays

However, if the NP to be questioned or relativized is part of a larger phrase of the type as below in (14a–b) no grammatical structure results:

(14) a. [Q Henry plays wh–something and sings madrigals] →
→ * What does Henry play and sings madrigals?

b. [the lute [Henry plays the lute and sings madrigals] →
→ * the lute which Henry plays and sings madrigals

Therefore the following constraint is in order:
"In a coordinate structure, no conjunct may be moved, nor may any element contained in a conjunct be moved out of that conjunct." (Ross 1967: § 4.2)

This and Ross's other constraints could be easily built into the *Aspects* version – they were in fact its extensions and specifications. But at the same time they were the first stepping stones to constructing a general framework of restrictions on transformations.

2.2 These two lines of research were, however, overshadowed for some while by the emergence of a new conception of the relationship between syntax and semantics in generative grammar. The quotation from Chomsky (1965) in 1.1 indicated his discontent with the idea that semantic interpretation should be based exclusively on deep structure. Simultaneously with the elaboration of ST (eg. Rosenbaum 1967) a loosely knit group began to be formed of linguists who contended that a large body of data was incompatible with any grammar of the *Aspects* type (Lakoff 1967, 1970; McCawley 1968, 1970a, among others) and leaned towards a grammar whose initial phrase markers would be more like semantic structures. Then attacks from different quarters on ST followed; speech–act theorists notably John Searle (1972) doubted whether the description of language presupposed the independence of grammar as proposed by Chomsky. Thus it was a narrow and a broad sense of the autonomy of formal description that proponents of ST had to defend.

2.21 Searle is convinced that language has a primary function, communication in the broadest sense:
"The common–sense picture of human language runs something like this. The purpose of language is communication in much the same sense that the purpose of the heart is to pump blood. In both cases it is possible to study the structure independently of function but pointless and perverse to do so, since structure and function so obviously interact. We communicate primarily with other people, but also with ourselves as when we talk or think in words to ourselves." (Searle 1972:19)

Although we might have mild reservations to attributing such a paramount significance to what Searle himself calls "quite ordinary [. . .] common–sense assumptions", mostly because common–sense is not always too reliable a guidance for research (as was shown by the remoteness from common–sense assumptions of, for example, quantum–mechanics), it is certainly true that nothing is elevated to the rank of theory solely by being far removed from common–sense. Yet it is not impossible to conceive of the problem of "the central function of language" as somewhat overemphasized. In another connection, which we will discuss in a moment, Max Black wrote:
"There is a fairly obvious trap here, into which too many acute minds have fallen. A question about the primary use (or: purpose, function) of an instrument as simple as a hammer is readily answered in a single formula. But even something as simple by comparison with language as, say, paper, is designed to serve a multitude of purposes: it has the primary functions of being used for writing, for wrapping parcels, for lining compartments and

packages, and so on. Paper, unlike a hammer, has multiple uses.

"Given the obvious complexity and versatility of language, and the enormous variety of purposes to which words seem to be put, it might be expected that language would, from the every outset, be recognized as having a multiplicity of uses. In fact, however, there is an ancient tradition of regarding language as an instrument with a single primary use – more like a hammer than like paper." (Black 1968:118)

Black's remarks seem to apply with remarkable force to Searle's considerations, although they were originally meant to criticize a position which, among others, Chomsky has been supporting. "Language, it is argued, is 'essentially' a system for expression of thought. I basically agree with this view." (Chomsky 1975a:57) And when he blames Searle for taking "communication in [a] broader sense [which is] an unfortunate move [. . .] since the notion 'communication' is now deprived of its essential and interesting character" (Chomsky 1975a:57), Chomsky himself becomes equally vulnerable to similar criticism of his own broader notion of 'thought'. Furthermore he is also suspect of substituting 'thought' for 'whatever can be meant'.

"But then a thought, in this indefensibly extended sense, can very well be a feeling, an intention, and much else. It is misleading and unhelpful to use 'thought' to cover all those things. On this broad interpretation, language serves many purposes: the way is open for a *multiple* theory of linguistic uses." (Black 1968:117)

It is important to realize that the problem of the central function of language is uninteresting largely because it is inevitably accompanied by an overstretched use of words on the one hand, and often by the assumption that there is an immediate derivative connection between function and form on the other:

"It is quite reasonable to suppose that the needs of communication influenced the structure. For example, transformational rules facilitate economy and so have survival value: we don't have to say, 'I like it that she cooks in a certain way', we can say, simply, 'I like her cooking.'" (Searle 1972:19)

Searle is right in part of course inasmuch as the general statement is concerned. But hundreds of examples lend support to the view that economy, and the parallel notion of redundancy, fall fatally short of accounting for changes in language. No argument from communication alone will explain the phases of development of a language like English. Further, to attribute to certain transformations properties such as this: "transformations [. . .] facilitate communication" (Searle 1972:19) would unexceptionally lead to a cross–classification of languages according to 'degrees of ease of communication,' which is no doubt an undesirable result. Chomsky can of course, escape this second consequence simply by using 'thought' for 'whatever can be said' in his definition of the primary function of language.^{3}

Searle likes to think of speech act theory as at best including a formal theory of language, i.e. syntax: "The obvious next step in the development of the study of language is to graft the study of syntax onto the study of speech acts." (Searle 1972:23) He also concedes that the study of the meaning of sentences and that of the uses of expressions in speech situations "are complementary, not competing." However, he also claims that there is an "essential connection between meaning and speech act" since "an essential part of the meaning of any sentence is its potential for being used to perform a speech act." (Searle 1972:23) This theory of meaning was criticized partly by Chomsky for the reintroduction of 'literal meaning' as an unexplained notion, and partly by Deirdre Wilson for two reasons:

"The view that knowing the meanings of words involves knowing what speech–acts they are characteristically used to perform [. . .] seems largely false. In the first place, it seems entirely irrelevant to specifying the meanings of such ordinary words as *table* or *ventilate*. [. . .] More generally, there is a theoretical objection to the use theory of meaning which parallels the objection often raised to truth–conditional theories, that they rest on a prior and unexplicated notion of necessary truth–relations. In the case of the use–theory, the objection is that it rests on an unexplicated notion of rules for appropriate use. When one enquires into the definition of *appropriateness* which is relevant for semantics, one is forced, I think, to one of two conclusions. Either there is no distinction between knowing when a given sentence could be appropriately used and knowing when it would in fact be true: in this case the use theory is not distinct from a truth–conditional theory. Or the notion of appropriateness includes, but goes beyond, the notion of truth–conditions. In this case the problem is to define the non–truth–conditional aspects of appropriateness. These seem to me clearly non–homogeneous, including reference to social conventions, discourse–conventions, psychological considerations and contextual factors of many different types. Moreover, they seem to me in most, if not all cases, to be clearly non–linguistic, and certainly not matters of speaker–hearer's competence." (Wilson 1975:14)

It would certainly go far beyond the goals of this paper to discuss fundamental questions of semantics in any detail, but it may perhaps be suggested that a possible way out could be conceived either through the integration of formal linguistic and communication theories along the lines of Lewis (1974) with a semantic theory in the fashion of Lewis (1972) or through an extended version of truth–conditional semantics in the wake of Kempson (1975) and Wilson (1975).

All I have tried to show in this section was the futility of inferring anything interesting regarding the nature of formal linguistic theory from the assumption that there is a central function of language. The overexaggeration of functional explanations for syntax may, in addition, lead to such monstrosities as are examplified by Sadock (1975), who was duly criticized even by the author of the theory he had intended to draw on (cf. Searle 1976).

2.22 Turning now to the narrow sense of autonomy, it is clear that this notion can be, and has been, censured within the comparatively less extensive framework of generative grammar. Indeed, that is the basic issue which separates generative semanticists and proponents of the lexicalist–interpretivist position. To put it briefly, the heart of the matter is whether the central component of the grammar is syntax or semantics. Almost all of the other problems (such as lexical decomposition, the existence of the level of deep structure, global rules, etc.) are derivative from it.

Generative Semantics (GS), in its more extreme version, proclaims to be virtually the theory of cognition:

"In the theory of generative semantics [. . .] the abstract objects generated are not sentences but quadruples of the form (S, LS, C, CM) where S is a sentence, LS is a logical structure associated with S by a derivation, C is a finite set of logical structures (characterizing the conceptual context of the utterance), and CM is a sequence of logical structures, representing the conveyed meanings of the sentence in the infinite class of possible situations in which the logical structures of C are true.

"But even this is inadequate. One must take into account much more than conceptual contexts (that is, assumptions of speaker and hearer). Rules of grammar also require that one take into account the stylistic type of discourse one is in." (Lakoff 1974:163)[4]

A more sober view is voiced in the following summary by Pieter Seuren:

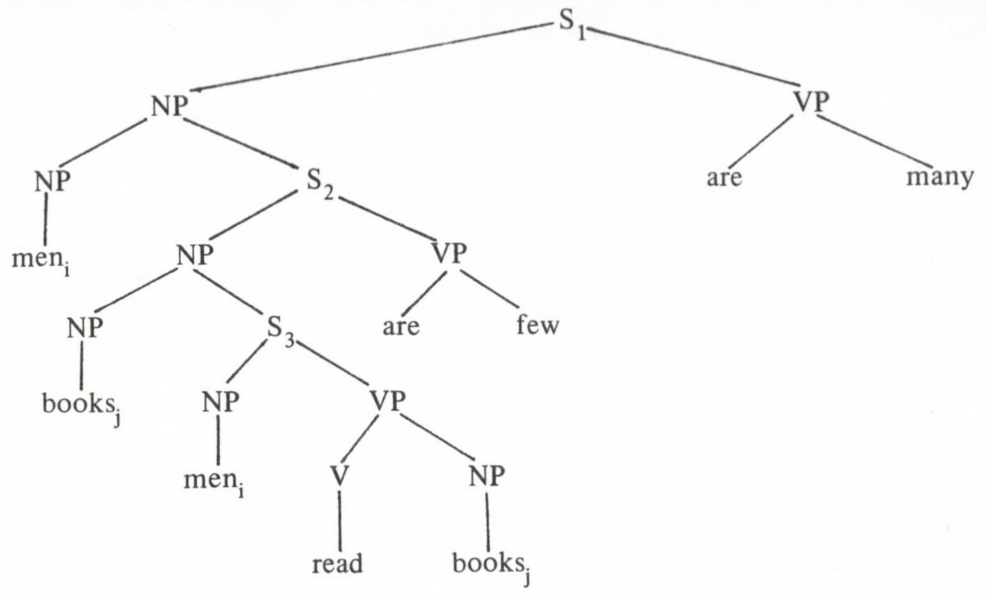
"Semantic Syntax [. . .] maintains that [the] ultimate underlying structure is the SR [=semantic representation], and the transformational rules map SS's [=surface structures]. In this theory the formation rules have a very different status [from that of the corresponding rules in ST or EST]: they define the wellformedness of SR's. Given the great deficiency of our knowledge of SR's as well as of cognitive structures, it would be impractical to formulate such rules at present." (Seuren 1974a:110)

In order to show the difference between ST and GS I have taken over the following standard example from Lakoff (1971). The derivation of the two sentences (1) and (2), according to GS, goes back to two distinct initial structures (14a) and (14b):[5]

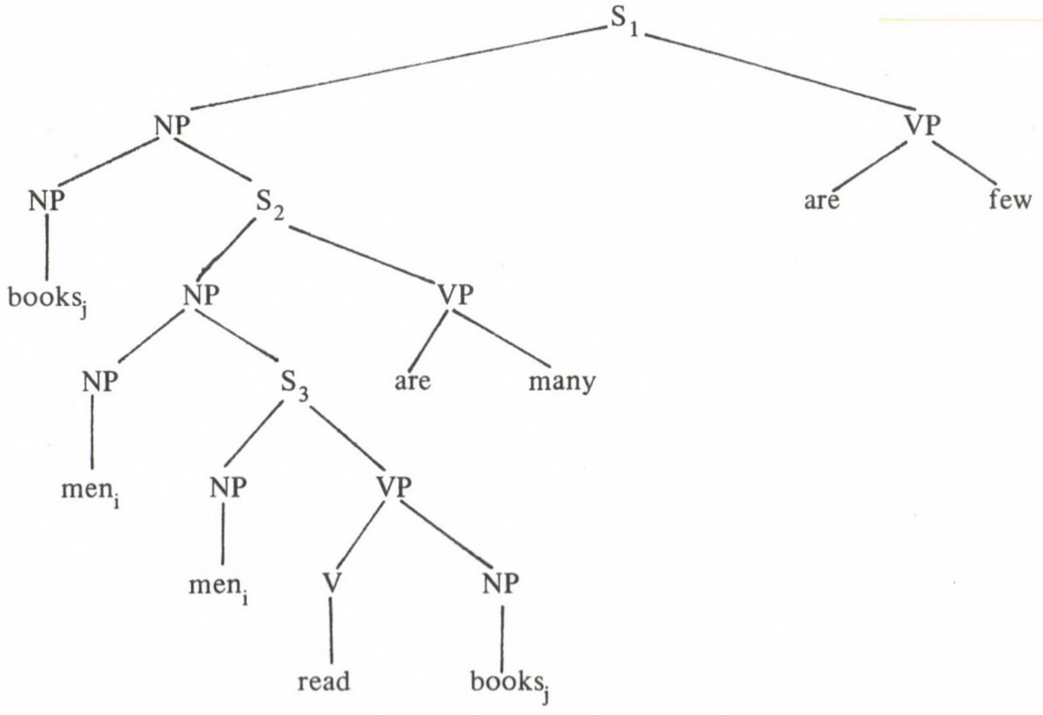
(1) Many men read few books.

(2) Few books are read by many men.

(14) a.


S₁
NP
VP
NP
S₂
are
many
men_i
NP
VP
NP
S₃
are
few
books_j
NP
VP
men_i
V
NP
read
books_j

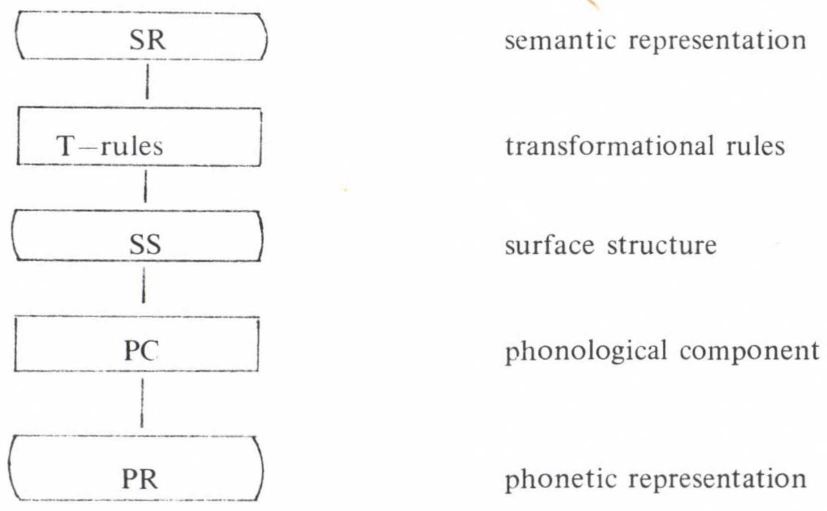
(14) b.


S₁
NP
VP
NP
S₂
are
few
books_j
NP
VP
NP
S₃
are
many
men_i
NP
VP
men_i
V
NP
read
books_j

The rule of 'Quantifier–lowering' will apply first to the S_2 cycle yielding *men read few books* in (14a), then to the S_1 cycle with the result of (1). The structure (14b), however, undergoes Passive in the S_3 cycle (*books are read by men*), then cyclic Quantifier–lowering, first on S_2 (*books are read by many men*), then on S_1 to give (2).

But Passive is an optional rule: it may or may not apply to either of the structures (14a) and (14b), and if it does apply to (14a), the resulting sequence will be indistinguishable from (2), but it has the meaning of (1). Conversely, if Passive does not apply to (14b), we will have the surface sentence (1) with the meaning of (2) as defined by GS. In order to overcome this difficulty it was necessary to introduce in the transformational component a 'global rule' which preserves the order of quantifiers between the different stages of a derivation.

Seuren gives the following schematic representation for the GS (or, in his parlance, Semantic Syntax) model:



It is a corollary of the GS hypothesis that there is no distinct level of deep structure, since the input for T–rules is SR's (in other words, the mapping relationship is between SR's and SS's) on the one hand, and that lexical insertion is not carried out in one block (as in ST) but is part of the functions of transformations, which can substitute single lexical entries (eg. *kill*) for complex subtrees (eg. *cause to die*) within one derivation. Global rules will attend to wellformedness of (ultimately) surface structures by stating a relationship between distinct (not necessarily consecutive) stages of a single derivation, while a transderivational rule is "a relationship between some stages in a derivation and something outside of the derivation – e.g., some stage of another derivation, or some logical inference from the semantic structure of the derivation in question" (McCawley 1974:266), and filters out ambiguous structures, for example.

Generative Semantics thus makes considerably stronger assumptions about the organization of grammar on the one hand, which may well be viable but are difficult to refute (cf. Seuren's remark on our knowledge of semantic representations) and, on the other hand, about the tasks of linguistics, which could lead to the dissolution of linguistics as a discipline in the related fields of logic, psychology, sociology, and so on (cf. Lakoff 1974). It is not the linguist's fear of losing his job that has made various people raise serious doubts about such an approach, but rather the conviction that the elusive nature of meaning in itself does not perhaps provide sufficient reason for an unprecedented and uncalled for extension of the boundaries of linguistics as GS demands.

To make a final point, there is a tangible undercurrent in GS when it refuses to accept syntactic structures as underlying surface sentences by indicating that it is more 'natural' or closer to common–sense to suppose that sentences in natural language are produced by first deciding on their content, which will in turn determine their form. Even if some of the more conscientious writers of GS would certainly reject such an accusation, it is an implicit possibility in GS and has undoubtedly contributed to its widespread acceptance and popularity (cf. eg. Bouveresse's remarks on Chomsky and GS in Parret 1974). Another, related, misunderstanding has been the identification of generative–transformational grammar with some kind of model of production requiring an immediate, one–to–one correspondence between linguistic theory and the psychological processes which underlie speech production (cf. eg. Chafe 1970, Bartsch and Vennemann 1972).

2.23 As far as the other position, the autonomy of syntax, is concerned, predictably enough the best arguments in favour of it have been advanced by Chomsky himself. The most important preliminary argument has helped to make it clear that the problem of 'semantically' or 'syntactically based grammars' was of no interest, since grammars do not 'first' generate deep structures and 'then' map them by means of transformations ultimately into surface structures, but generate *n*–tuples of abstract objects such as deep structure (in ST at least), surface structure, semantic representation, and phonetic representation (Chomsky 1971, 1972, 1974). In this sense it is irrelevant to speak of a central component in grammar.

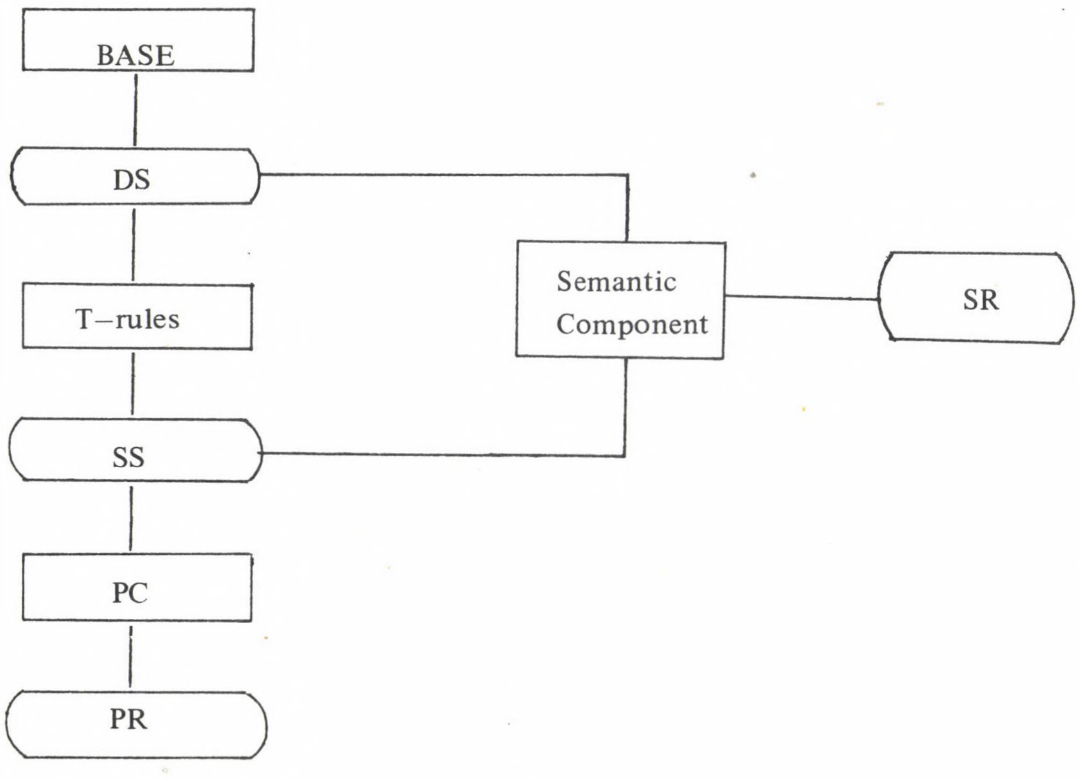
The problem of autonomy is of course void for all those who postulate some kind of semantic or logical structures as input to transformations. It would again go beyond the purpose of this discussion to remove the misunderstandings that surround the issue, but it should be noted that the concept of 'autonomy of syntax' has undergone considerable change since its inception, the publication of *Syntactic Structures*, The earlier formulations of generative grammar relied upon a strong version of autonomy:

"The absolute autonomy thesis implies that the formal conditions on 'possible grammars' and a formal property of 'optimality' are so narrow and restrictive that a formal grammar can in principle be selected (and its structures generated) on the basis of a preliminary analysis of data in terms of formal primitives excluding the core notions of semantics, and that the systematic connections between formal grammar and semantics are determined on the basis of this independently selected system and the analysis of data in terms of the full range of semantic primitives." (Chomsky 1975b:21)

It does not, however, follow from this thesis that semantic considerations do not take part in the choice of a theory of linguistic form. It is nevertheless possible to put forward a weaker, and consequently less interesting, thesis of the autonomy of syntax, according to which "the theory of linguistic form may still be a theory with significant internal structure, but it will be constructed with 'semantic parameters'. The actual choice of formal grammar will be determined by fixing these parameters." (Chomsky 1975b:22) The choice between these and other possible versions of autonomy is an empirical problem though of a rather abstract nature. But the refutation of a relatively stronger version of the thesis does not of course entail the abandonment of a weaker formula. The 'parameterized autonomy' thesis has recently gained ground among the proponents of a lexicalist version of generative grammar with the parameters localized in the dictionary.

3. Halfway between *Aspects* and traces

3.1 The model of EST is not a radical departure from that of ST; it simply incorporates the finding that deep structure alone is not sufficient to determine the meanings of sentences. The schematic representation of the model is thus not unlike that of the grammar of *Aspects*– with the only exception of a mapping operation between surface structure and semantic representation:

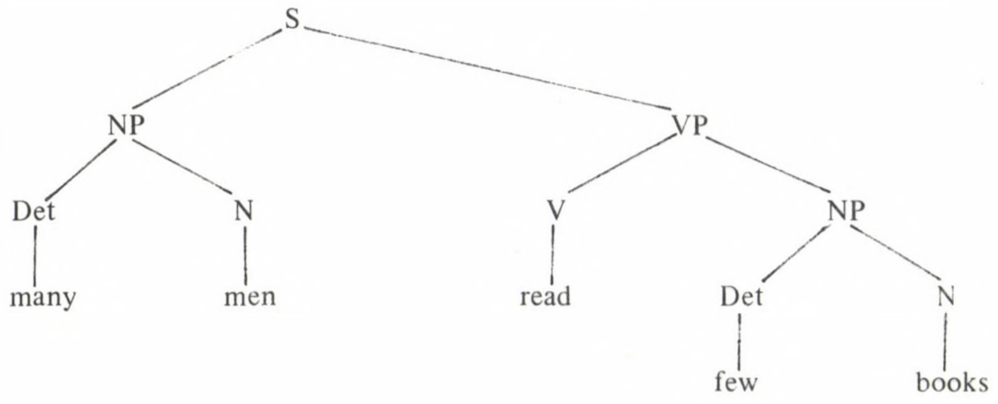


Deep structures determine 'thematic relations' (Agent, Goal, Instrument, etc.), whereas surface structures provide data deciding the scope of quantifiers, the anaphoric relationship between pronoun and antecedent, topic–comment relations, and so on. For example, the sentences (1) and (2) are derived from the same underlying structure (15):

(1) Many men read few books.

(2) Few books are read by many men.

(15)



from which, if Passive does not apply, (1) is derived, and and, if it does, (2). However, the surface linear order of *many* and *few* in (2) differs from that in (1), therefore *many* is within the scope of *few* in (2), whereas the reverse holds for (1). The 'thematic relations' of the sentence have of course remained unchanged.

Similarly, 'precede' and 'command" relations will determine coreference between pronoun and antecedent. We will see below that while these relations are formulated in GS as (global) constraints on (pronominalization) transformations, they are construed in EST as rules of interpretation, since EST has rid itself of the idea of transformational derivation of pronouns. The following four sentences in (16):

(16) a. If John is ill, he has to stay in bed.

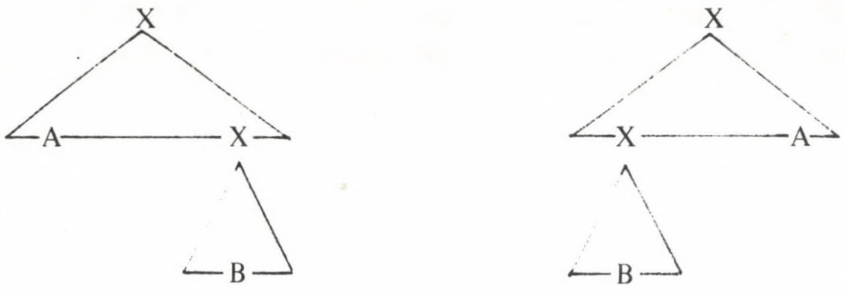
b. If he is ill, John has to stay in bed.

c. John has to stay in bed, if he is ill.

d. He has to stay in bed if John is ill.

show that the only case *John* and *he* cannot be coreferent, is (16d). Now we say, following Langacker (1969), that a node *A* commands another node *B* if neither *A* nor *B* dominates the other, and the first cyclic node (i.e. S or NP) that most immediately dominates *A* also dominates *B*. The precedence relationship is self–explanatory. Then in (17):

(17) a. b.



where X is a cyclic node, A commands B, and precedes B in (17a) but is preceded by it in (17b). Returning to (16) it must now be obvious that a pronominal NP cannot both precede and command the NP it is correferent with, or in other words, if a pronominal NP precedes and commands another NP there can be no correference relationship between them.

3.2 It was precisely pronominal anaphora that was used to demonstrate as a corroboration of the lexicalist position that certain phenomena previously regarded as transformationally derivable were better described by means of lexical insertion paired with semantic interpretation.

Recall that in ST pronouns were accounted for by positing two full–blown lexical NPs in deep structure marked for identity by, say indexing, and subsequently by replacing one or the other with a pronoun as in (18a–b):

(18) a. John_i knew John_i was stupid.

b. John_i knew he_i was stupid.

However, evidence was discovered that pronominalization (as the process was called) could lead to undesirable consequences. The famous Bach–Peters sentences (Bach1970) were data of this kind:[6]

(19) $[_{NP_i}$ the pilot who shot at $\text{it}_j]$ hit $[_{NP_j}$ the mig that chased $\text{him}_i]$

(20) $[_{NP_i}$ the man who shows he_i deserves $\text{it}_j]$ will get $[_{NP_j}$ the prize he_i desires$]$

It goes without saying that the transformational derivation of the pronouns in (19–20) would involve multiply, if not infinitely, embedded sentences in somewhat like the following way:

(21) $[_{NP_i}$ the pilot who shot at $[_{NP_j}$ the mig that chased $[_{NP_i}$ the pilot who . . .]]]
hit $[_{NP_j}$ the mig that chased $[_{NP_i}$ the pilot who shot at $[_{NP_j}$ the mig that . . .]]]

Another type of counterargument was based on examples like (22a–b):

(22) a. Every Italian$_i$ thinks he$_i$ is handsome.
b. Every Italian$_i$ thinks every Italian$_i$ is handsome.

According to the transformationalist position, (22b) was supposed to underlie (22a), although they have no common readings and furthermore they differ syntactically, cf.

(23) a. Every Italian$_i$ thinks he$_i$ alone is handsome.
b. *Every Italian thinks every Italian alone is handsome.

If, however, we assume that surface pronouns derive from 'deep' pronouns inserted from the lexicon into deep structure, which is a procedure independently needed for non–intrasententially anaphoric pronouns (cf. the extrasentential anaphoric reading of the pronoun in (18) or (22a), i.e. when *he* refers to an NP outside the context of the sentence it is in, as in the discourse: *A*: *Peter$_i$ has jumped off the wall. B*: *John$_j$ knew he$_i$ was stupid*), all that is now necessary is a relatively simple interpreting device which will tell whether the pronoun, or more generally the pro–form, can or cannot be the anaphor of whichever NP or other constituent.

Another, slightly more complicated argument against pronominalization was put forward by Joan Bresnan (1970a). It is based on the undisputed fact that the transformational cycle works 'from bottom up'. If in (24a), where identical indexes mark coreferent constituents, *some* is the unstressed /sm/, nothing will prevent *there* Insertion from operating on the embedded S cycle, yielding (24b):

(24) a. [Some students$_i$ believe [some students$_i$ are running the show]]
b. [Some students$_i$ believe [there are some students$_i$ running the show]]

Pronominalization can operate only on the next, the matrix S cycle, but now it either has to change (24b) to (25a) by replacing *some students$_i$* with *they$_i$*, or can leave (24b) alone. Either way the result is ungrammatical:

(25) a. *Some students$_i$ believe there are they$_i$ running the show.
b. *Some students$_i$ believe there are some students$_i$ running the show.

The case of (25a) is straightforward; the asterisk must be assigned to (25b) since the coreference required by the indexes does not go through.

3.3 EST had two advantages over its predecessor, indeed over any competing theory: the X–Bar Convention and the deep pronoun hypothesis; the fact that one is a metatheoretical, the other an empirical problem anticipated the directions of further research. The restructuring of the model which now allowed for the interpretation of surface structure in addition to that of deep structure was a necessary adjustment rather than an achievement of theoretical value.

4. The rise of trace theory

Since there was not much controversy about EST it was rather peacefully superseded by the latest offspring of the standard model, generally known as 'trace theory'. This time there were no signs of growing discontent with the extant model in the literature, there was no presentation of data to demonstrate that the model was inadequate. Yet the change was probably not unmotivated.

4.1 First of all EST made no clear statements about how exactly 'double' semantic interpretation should be carried out, that is, how one kind of semantic data (gained from surface structure) is to be integrated into another kind (those determined by deep structure). Let us call this the 'matching deficiency'. But even if this matching deficiency could be overcome there would still remain the, rather elusive, disadvantage of double interpretation being relatively less simple than interpretation in one block. We can call this the 'aesthetic deficiency'. A third kind of difficulty arose in connection with a curious phenomenon in English, Verb + *to* contraction. Let us examine this problem in some detail.

Lakoff (1970) mentions the pair of sentences (26–27):

(26) Teddy is the man I want to succeed.

(27) Teddy is the man I wanna succeed.

using them as evidence for an argument supporting global rules by making the following comment (in the quotation the numbers of examples have been changed):

"Here (26) is ambiguous, and can be understood as either of the following:

(28) I want Teddy to succeed.

(29) I want to succeed Teddy.

But (27) can only be understood in the sense of (29), since *want to* cannot contract to *wanna* if there is an intervening NP between *want* and *to* at an earliner point in the derivation, as there is in (28)." (Lakoff 1970:632)

Clearly, no syntax of the type of EST, which refuses to employ global rules, could cope with a problem like this, since once a constituent is deleted the resulting construction will be indistinguishable from a similar construction which did not contain the constituent in question at any stage of its derivation.

We have seen that semantic interpretation must take surface structure into account, deep structure is simply insufficient in a number of respects to determine the meanings of sentences, so the road back to ST is blocked. But there is another way out: the interpretation of surface structure could surmount the problems of the matching and the aesthetic deficiency if only the vital information from deep structure could be retained. The 'thematic relations', the sole essential factor in deep structure for semantic interpretation, are changed by movement and deletion transformations only. The solution is then self–evident. Roughly speaking, the position from which a constituent was moved must be marked for that constituent (by means of, for example, co–indexing), and instead of deletion transformations some kind of dummy nodes should be introduced. Note that this innovation has sufficient syntactic motivation: the difference between (26) and (28) can now be accounted for. Taking (26) as synonymous with (28) the relevant aspects of the surface structures immediately underlying (26) and (27) will be somewhat like (30) and (31), respectively:

(30) Teddy is the man [I want *t* to succeed]

(31) Teddy is the man [I want PRO to succeed *t*]

where *t* is the trace left by the movement transformation and PRO is the dummy to be interpreted for coreference (with *I*).[7] Now the trace between *want* and *to* in (30) will not only help semantic rules to find what is the subject of *succeed*, but will also prevent contraction. PRO, however, is defined as allowing contraction. Of course, neither trace nor PRO has phonetic outcome, that is, both are phonetically null.[8]

4.2 Traces can be regarded as a special type of anaphora, indeed they are interpreted as such by the semantic component. Transformations can, of course, move constituents both 'forward' and 'backward' and, like in the case of ordinary anaphora, traces must be properly 'bound', that is, in case of NPs, for example, no trace can precede its 'antecedent'. To illustrate this we will show the derivation of a passive construction according to trace theory (in its early period). The deep structure is thé same as in ST or EST (cf. (3), (4), (12)):

(32) $[_{S}[_{NP}$ the barbarians$]$ $[_{VP}$ destroy $[_{NP}$ the city$]$ $[_{PrepP}$ by NP$]]]$.

But there is nothing like Passive transformation in this new version. Instead, NP Movement will place the subject into the empty node:

(33) $[_{S}$ $[_{NP}$ $t]$ $[_{VP}$ was destroyed $[$the city$]$ $[_{PP}$ by $[_{NP}$ the barbarians$]]]]$

leaving the trace t_{NP} in its original position. If no more transformation applies to the structure (33), it will be ungrammatical. That the ungrammaticality is not the result of its lacking a subject is demonstrated by the corresponding nominal, which, according to the X–Bar Convention, is derived along similar lines :[9]

(34) $[_{NP}$ $[_{NP}$ the barbarians$]$ $[_{\bar{N}'}$ destruction $[_{NP}$ the city$]$ $[_{PP}$ by NP$]]]$

in which NP Postposing gives (35):

(35) $[_{NP}\ [_{NP}\ t]\ [_{\bar{N}}$ destruction $[_{NP}$ the city] $[_{PP}$ by $[_{NP}$ the barbarians]]]]

Now if *t* remains unsaturated in (35), or in (33), some rule of anaphora will filter out the structure as ungrammatical, since the trace precedes the corresponding NP. No doubt, the strings (36–37) are illformed:

(36) *was destroyed the city by the barbarians

(37) *destruction of the city by the barbarians.

The trace can be erased as a result of another movement transformation. So if an NP, like *the city*, is preposed in (33), it will take the position of the trace:

(38) $[_{S}\ [_{NP}$ the city] $[_{VP}$ was destroyed] $[_{NP}\ t]\ [_{PP}$ by the barbarians]]]

Then the trace in the tree (38) will be that of the NP *the city*, and will show that it is the deep object of the verb. Besides, the derived subject NP *the city* now properly binds its trace, so the structure will pass the wellformedness conditions set by the rules of anaphora.

Similar NP Preposing in the nominal construction will give (39):

(39) $[_{NP}\ [_{NP}$ the city's] $[_{\bar{N}}$ destruction $[_{NP}\ t]\ [_{PP}$ by the barbarians]]]

When, however, no NP Preposing applies to (35), it can still be rendered grammatical by the insertion of the definite article, which is allowed to occur in the *Spec* $\bar{N}$ node having replaced the trace:

(40) $[_{NP}\ [_{Spec}$ the] $[_{\bar{N}}$ destruction] $[_{NP}$ the city] $[_{PP}$ by the barbarians]]]

with the final outcome (41):

(41) the destruction of the city by the barbarians.[10]

4.31 Before beginning to discuss trace theory in a little more detail I will review two proposals which have gained rapid and wide–scale acceptance in the current period of GTG.

Bresnan's (1970b) suggestion concerns the introduction of a node Complementizer into base phrase markers, and is thus a revision of the base rules. The idea is simple and easy to prove. Embedded sentences (but not nominal constructions) may display one of three devices, called complementizers since Rosenbaum (1967), which integrate them into matrix or higher sentences: *for–to*, possessive–*ing*, and *that*, as shown in (42), (43), and (44), respectively:

(42) For John to leave would be insane.

(43) His singing annoys everyone in the room.

(44) Peter knew that John would leave.

Bresnan argues that introducing complementizers through transformations is illegitimate, since it would require that complementizers be determined by matrix verbs. Therefore, the relevant transformations would have to work 'downward', inserting material into a lower sentence, i.e. one that has already been passed by the cycle, and this is a violation of the well–known Insertion Prohibition which was first formulated by Chomsky (1965:146) and never disproved since. Instead of a transformational analysis, Bresnan suggests that a rule of the form of (45) should introduce complementizers:

(45) $\overline{\mathrm{S}} \rightarrow$ COMP S

As to the content of the COMP node, there is no general agreement: some claim that poss–*ing* constructions are derived from sentences through transformations but are ultimately dominated by an NP node (see 2.11). Bresnan's original formulation, which was intended to cover embedded sentences only, has since been extended to become one of the initial rules of the base and matrix sentences are also supposed to have complementizers such as [+WH] for questions, [–WH] for noninterrogatives. If the COMP node is in an embedded sentence, it will introduce indirect questions or *that*–clauses (and possibly relative clauses), depending on the plus/minus sign. So in observance of the consensus, COMP is analysed as follows:

(46) $\text{COMP} \rightarrow \begin{Bmatrix} \textit{for} \\ \pm\text{WH} \end{Bmatrix}$

4.32 The other proposal, Emonds' (1970, 1972) typology of transformations, was also hailed as a welcome innovation, although a number of its details are still being argued about. Emonds distinguished three kinds of transformations. The first division is between minor and major transformations: the former comprise small–scale 'readjustment' rules which involve non–phrase nodes, e.g., Affix Movement, while the latter move phrase nodes such as S, NP, AdjP, etc., in rules like Extraposition, Passive, Subject–Auxiliary Inversion. This second group is then divided into 'root' and 'structure–preserving' transformations. Root transformations operate only on matrix sentences and include rules like Subject–Aux Inversion (in case of questions, for example). Structure–preserving rules, however, apply all along the cyclic nodes under an unexceptional condition that states that "node X in a tree T can be moved, copied, or inserted into a new position in T [. . .] only if [. . .] the new position of X is a position in which a phrase structure rule, motivated independently of the transformation in question, can generate the category X". (Emonds 1972:22). The new position X is generated as an empty node; in Emonds' original formulation, it was a node filled by some 'recoverable' form like *it*, *there*, etc. In more recent frameworks, the usual transformations are followed by filters whose task is, among others, to mark as ungrammatical the trees which contain empty nodes.

An illustration for structure–preserving rules could be the derivation of the passive constructions in (32) to (41), or various instances of sentence Extraposition, which moves an embedded sentence to the rightmost position in the cycle in question, as in (47) and (48):

(47) a. They pronounced the man $[_S$ who was accused of murder$]$ guilty $[_S \Delta]$

b. They pronounced the man guilty who was accused of murder.

(48) a. We heard $[_S$ that he had been stranded for days$]$ from his own lips $[_S \Delta]$

b. We heard from his own lips that he had been stranded for days.

To see that this is a non–root transformation, it suffices to prefix a matrix clause skeleton, like *He asked whether* . . ., degrading the sentences of (47–48) to subordinate status.

Obviously, in the cases of both Passive and Extraposition there is sufficient independent motivation for the relevant empty nodes in underlying structure. Indeed it was probably this observation that led Emonds to an interesting conclusion, which was arrived along a separate path by Bresnan (1970a), viz. That *for* plus infinitival constructions and *that* clauses are dominated not by NP (as was claimed by Rosenbaum 1967), but by S. Furthermore, the classic case of Rosenbaum's Extraposition from Subject will now work the other way round, that is, the relevant S nodes are generated in a position adjacent to the verb, and a root transformation of Intraposition moves them to subject position. In Rosenbaum's solution, (49) derives from something like (50):

(49) It is important for John to please Mary.

(50) For John to please Mary is important.

However, Emonds demonstrates that neither construction in question can occur in the subject position of an embedded sentence:

(51) a. *Peter said that for John to please Mary was important.

b. Peter said that it was important for John to please Mary.

(52) a. *John wanted to know whether that he was too loud annoyed me.

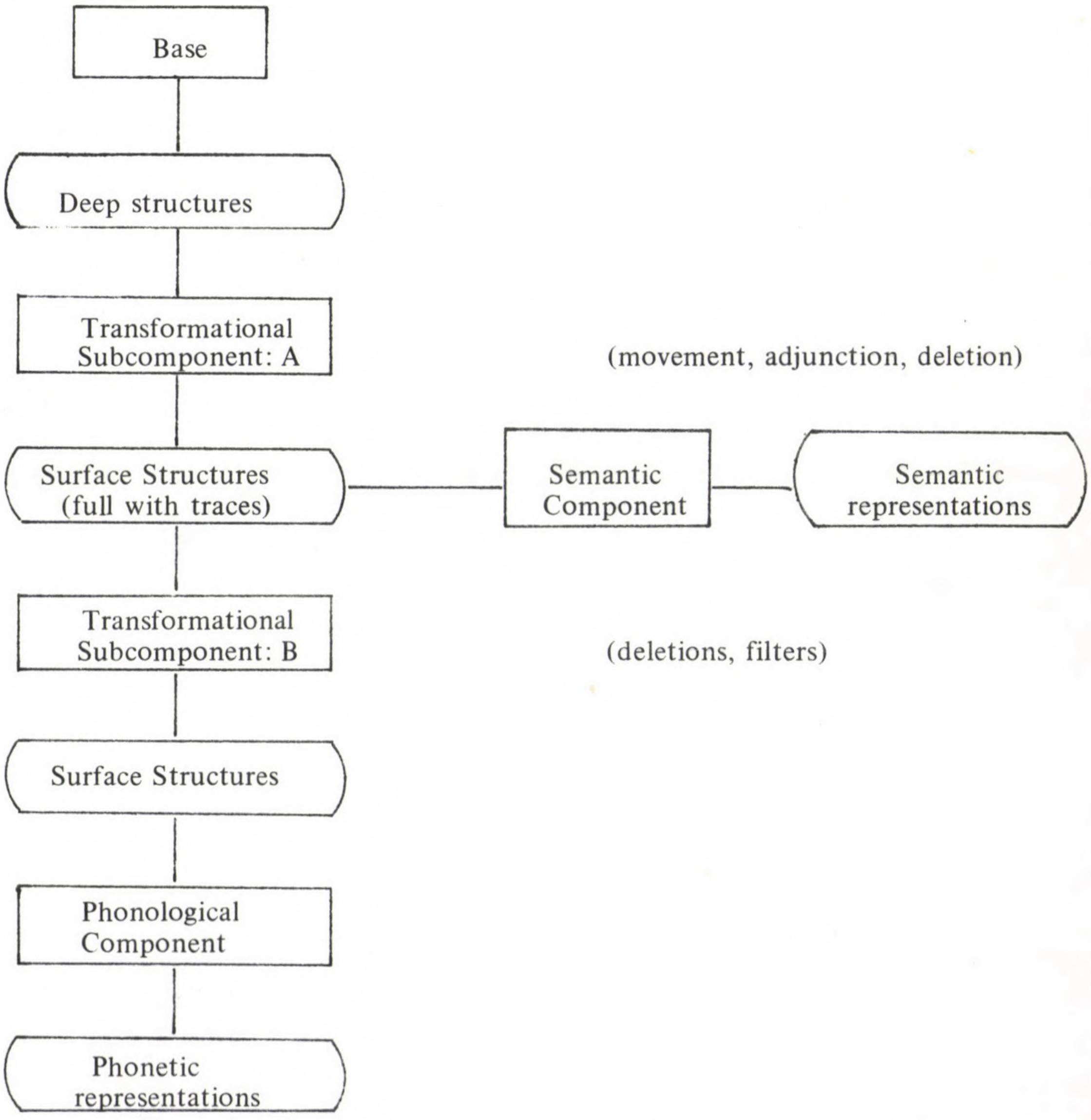
b. John wanted to know whether it annoyed me that he was too loud.

Therefore, we either have to make do with an ad hoc prohibition (in the style of Ross 1967), requiring that there be no Ss in sentence–interior position, i.e. Extraposition is obligatory in embedded sentences (and also in matrix sentences if as a result of a transformation they become sentence–internal, cf. **Is that he came late surprising*?), or we admit that there is no Extraposition from Subject, so it is (49) that underlies (50) rather than conversely.

4.4 The current composition of trace theory is the result of the considerations entailed by the innovations sketched above. It is easy to see, for example, that if surface structures are the input to the semantic component, no deletion transformation must apply before the structures become available for semantic interpretation; otherwise some information would again be irretrievably lost. In other words, transformations must now be ordered in two consecutive blocks: movement, adjunction, and substitution transformations will all be followed by deletion operations. But it is at the predeletion stage that 'surface structures' receive semantic interpretation (Chomsky and Lasnik 1977).

4.41 Thus the schematic outline of trace theory can be given as follows:

(53)



The other factor which has contributed to the shaping of the new model has been with us for quite some time: the universal nature of linguistic hypotheses. Chomsky and Lasnik (1977) assume that "there is a theory of core grammar with highly restricted options, limited expressive power and a few parameters. Systems that fall within core grammar constitute 'the unmarked case'; we may think of them as optimal in terms of evaluation metric. An actual language is determined by fixing the parameters of core grammar and then adding rules or rule conditions, using much richer resources, perhaps resources as rich as contemplated in earlier theories of [transformational grammar]". (430)
Rules of the base are restricted by the X–Bar Theory.

The generality required of what was called here Transformational Subcomponent: A (henceforth TSC:A) is achieved by the removal of ordering and obligatoriness as constraints on transformations. In consequence,
"the transformational rules of the core grammar are unordered and optional. Structural conditions are severely restricted. [. . .] The operations are restricted to movement, left– and right–adjunction, and substitution of a designated element. [. . .] Only a finite and quite small number of transformations are available in principle". (Chomsky and Lasnik 1977:431)

An important formal restriction on TSC:A is Emonds' structure preserving constraint.

4.42 In order to make up for the removal of ordering and obligatoriness in transformations a new subcomponent has been created which also has the function of the structural analyses of the old type transformations: filtering. Filters
"will have to bear the burden of accounting for constraints which in the earlier and far richer theory, were expressed in statements of ordering and obligatoriness, as well as contextual dependencies that cannot be formulated in the framework of core grammar". (Chomsky and Lasnik 1977:433)

Filters, as well as deletions, are language specific in contrast to the Base and TSC:A. They are placed in TSC:B and are ordered so that all deletions precede filters. A sophisticated enough system of filters, like the one Chomsky and Lasnik (1977) propose, should be capable of accounting for a very large number of ungrammatical structures. The general form of filters is given as follows:

(54) $^*[_\alpha\, \varphi_1, \ldots, \varphi_n]$, unless C, where

a. α is either a category or left unspecified

b. φ_i is either a category or a terminal symbol

c. C is some condition on $(\alpha, \varphi_1, \ldots, \varphi_n)$

"If α of (54a) is unspecified, the bracketed construction is arbitrary; otherwise the filter applies in the domain α. [. . .] "We might interpret (54) as follows. Given a construction (either unspecified or of the category α) that can be analyzed into the terminal strings $X_1, \ldots, X_n$, where X_i is a φ_i (in the sense of the theory of transformations), then assign * to the construction (or, equivalently to the sentence in which it appears) unless C holds of $(\alpha, \varphi_1, \ldots, \varphi_n)$." (Chomsky and Lasnik 1977:488–89)

4.43 Let us now see a somewhat simplified example to witness the function of filters. In the sentential complements to nouns like *desire* the complementizer is *for* (cf. (46)). That is, the relevant deep structure is (55):

(55) $[_{NP}$ the desire $[_{\bar{S}}$ $[_{COMP}$ for] $[_S$ John to leave]]]

If, however, the subject of the embedded sentence is the phonetically null and referentially 'empty' PRO, as in (56a), the complementizer is to be deleted to yield (56b):

(56) a. $[_{NP}$ the desire $[_{\bar{S}}$ $[_{COMP}$ for] $[_S$ PRO to leave]]]

b. the desire to leave

That is to say, there should be a rule deleting *for*, informally given as (57)

(57) delete *for*

But, according to the requirements of trace theory, the conditions on the deletion can only be given among the filters. Thus the filter (58) will in effect make (57) obligatory if *for* is followed by *to* without any intervening lexical NP (note that PRO is not lexical):

(58) *[for–to]

This, however, is not enough. If (57) is optional, it may very well apply to (55), which would result in the ungrammatical structure (59):

(59) *the desire John to leave

This will be prevented by another filter, (60):

(60) $*[_\alpha$ NP to VP] unless α is in the context

for — or
V —

Since the relevant NP–to–VP construction in (59) is preceded by the noun *desire*, (59) will be marked as ungrammatical by the filter (60). Obviously, structures like (55) will be allowed to go through unaffected by both filters (58) and (60). On the other hand, (60) forbids the deletion of *for* also in full adjectival phrases such as the one in (61):

(61) a. It is $[_{AP}$ illegal $[_{\bar{S}}$ for John to leave]]

b. *It is $[_{AP}$ illegal $[_{\bar{S}}$ John to leave]]

but it will make deletion obligatory if the subject is PRO in the embedded sentence, cf.

(62) a. It is $[_{AP}$ illegal $[_{\bar{S}}$ for PRO to leave]]

b. *It is illegal for to leave.

c. It is illegal to leave.

4.44 Although there are a number of incongruities in the formulations of the specific filters by Chomsky and Lasnik and some of the claims Chomsky (1977) makes in order to reduce the number and types of transformations are unsubstantiated, the threats that trace theory must counter go back to different considerations. One may speculate whether the abandonment of the idea of the semantic interpretation of deep structure cannot lead to an eventual identification of deep structure and (pre–deletion) surface stucture. More specifically, if Emonds' hypothesis is correct, cannot cyclic movement rules be abolished and quasi–surface structures be generated complete with trace (note that trace and empty nodes are syntactically identical)?

We may also ask the question whether the removal of all constraints of ordering and obligatoriness has not increased the power of core grammar. If it has, we may easily have come back to the Universal Base Hypothesis as criticized by Peters and Ritchie (1969).

To sum up, trace theory arose basically as a theory of syntax. Its semantic component is rather crude, and until this obstacle is overcome no overall view of the theory will be available. It can, however, be safely said that as a syntactic theory it has a high degree of internal consistency and practicability.

Notes

*This paper is a revised version of the introductory section of my dissertation *Trace theory and relative clauses* (Budapest, 1978).

1. Compare the following: "In the standard theory [. . .], as developed, e.g. in [*Aspects*], it is postulated that deep structure determines meaning. Thus nonsynonymous sentences cannot be assigned the same deep structure. In this respect, semantic considerations provide a partial criterion for the selection of grammars [. . .]."
(Introduction from 1973 to Chomsky 1955; p. 49)

2. These categories can be easily reformulated in terms of features. Thus, with bars neglected, S, VP, and V will be [+verb, –noun], NP and N [–verb, +noun], AP and Adj [+verb, +noun].

3. It should be kept in mind that the polemic is about the logical, not the historical priority of functions.

4. It is an immediate consequence of this view that sentences like (i):
(i) John called Mary a Republican and then *she* insulted *him*.

(where italics indicate heavy stress) are marked as well–formed or ill–formed according to whether or not the speaker believes that calling someone a Republican is an insult.

5. Any details of trees and labelled bracketing not pertinent to the discussion are omitted here and throughout.

6. Two of their sentences are given only; the bracketing and the indexes are somewhat altered.

7. At this point it is of no interest whether a 'lexical' NP or a relative pronoun was moved and then deleted.

8. For more detailed discussion see Lightfoot (1976, 1977). Note, however, that contraction phenomena do not point to a single possible explanation. That is to say, they do not provide unambiguous support for trace theory. For this see Emonds' comments on Lightfoot (1977) in Culicover (1977), as well as Postal and Pullum (1978).

9. There is no 'phrase–name' for $\overline{\mathrm{N}}$; a possible candidate could be NOM (from NOMINAL), but that has not been generally accepted.

   Some elements of the terminal string are missing, such as the preposition *of*. They can be fed into the tree automatically if no lexical information to the opposite effect is contained in it, since they constitute the 'unmarked' case among syntactic relations and are legitimately left unspecified, following the X–Bar Theory.

10. The above discussion of passive is modelled after Fiengo (1977), but it differs from his analysis in that it is somewhat simplified and reflects an earlier stage in the development of trace theory. However, even Fiengo's solutions became outdated on the publication of Chomsky and Lasnik (1977).

    Critics from various quarters have since pointed out that neither is an entirely satisfactory solution to passive constructions. Within the framework of trace thaory, one can equally well envisage a derivation of passive in which there is an empty subject and the agent phrase is already in position:
    "[The] rule of trace replacement in which the determiner replaces the trace and in which phrases with no determiner (*destruction of private property is illegal*) are explained as a result of replacing the trace by null determiner [is an] analysis completely unconvincing and ad hoc." (Bach 1977:142) The 'empty subject' assumption for passive constructions fares better also because it entails a more consistent treatment of traces.

References

[1] Bach, Emmon (1970), "Problominalization", *Linguistic Inquiry 1.*,. 121–122.
(1971), "Syntax since *Aspects*", In: Richard O'Brien (ed.), *Report of the Twenty–Second Annual Round Table Meeting on Linguistics and Language Studies*, Georgetown University Press, Washington, 1971, pp. 1–18.

[2] —— (1977), "Comments on Chomsky (1977)," In: Culicover et al. (1977), pp. 133–155.

[3] Bartsch, Renate, and Theo Vennemann (1972), *Semantic Structures*, Atheneum Verlag, Frankfurt/Main.

[4] Black, Max (1968), *The labyrinth of language*, Encyclopaedia Britannica, New York.

[5] Bresnan, Joan (1970a), "An argument againts pronominalization", *Linguistic Inquiry1.*, 122–123.

[6] —— (1970b), "On complementizers: toward a syntactic theory of complement types", *Foundations of Language 6.*

[7] Chafe, Wallace L. (1970), *Meaning and the structure of language,* University of Chicago Press, Chicago.

[8] Chomsky, Noam (1955), *The logical structure of linguistic theory,* Plenum Press, New York [1975].

[9] —— (1957), *Syntactic structures,* Mouton, The Hague.

[10] —— (1964), "Current issues in linguistic theory", In: Fodor and Katz (1964), pp. 50–118.

[11] —— (1965), *Aspects of the theory of syntax,* MIT Press, Cambridge.

[12] —— (1970), "Remarks on nominalization", In: Jacobs and Rosenbaum (1970), pp. 184–221.

[13] —— (1971), "Deep structure, surface structure and semantic interpretation", In: Steinberg and Jakobovits (1971), pp. 183–216.

[14] —— (1972), "Some empirical issues in the theory of transformational grammar", In: Peters (1972), pp. 63–130.

[15] —— (1974), "[Dialogue with] Noam Chomsky", In: Parret (1974), pp. 27–54.

[16] —— 1975a), *Reflections on language,* Collins, London. London.

[17] —— (1975b), *Questions on form and interpretation,* P. de Ridder Press, Lisse.

[18] —— (1976), "Conditions on rules of grammar", *Linguistic Analysis 2.*, 303–351.

[19] —— (1977), "On *wh*–movement, " In: Culicover et al. (1977), pp. 71–132.

[20] Chomsky, Noam, and Howard Lasnik (1977), "Filters and control", *Linguistic Inquiry 8.*, 425–504.

[21] Culicover, Peter W., Thomas Wasow, Adrian Akmajian (eds.) (1977), *Formal syntax,* Academic Press, New York.

[22] Emonds, Joseph (1970), *Root and structure preserving transformations,* unpublished Doctoral dissertation, M. I. T., Cambridge.

[23] —— (1972), "A reformulation of certain syntactic transformations," In: Peters (1972), pp. 21–62.

[24] Fiengo, Robert (1977), "On trace theory," *Linguistic Inquiry 8.*, 35–62.

[25] Fillmore, Charles J. (1968), "The case for case," In: Bach and Harms (1968), pp. 1–90.

[26] Fillmore, Charles J., and Terence D. Langendoen (eds.) (1971), *Studies in linguistic semantics,* Holt, New York.

[27] Fodor, Jerry A., and Jerrold J. Katz (eds.) (1964), *The structure of language: readings in the philosophy of language,* Prentice–Hall, Englewood Cliffs.

[28] Grinder, John T. (1976), *On deletion phenomena in English,* Mouton, The Hague.

[29] Halitsky, David (1975), "Left branch $\overline{\text{S}}$'s and $\overline{\text{NP}}$'s in English," *Linguistic Analysis 1.*, 279–296.

[30] Harman, Gilbert (1974), *On Noam Chomsky: critical essays,* Anchor Books, New York.

[31] Jacobs, Roderick A., and Peter S. Rosenbaum (eds.) (1970), *Readings in English transformational grammar,* Ginn, Waltham.

[32] Katz, Jerrold J., and Paul M. Postal (1964), *An integrated theory of linguistic description,* MIT Press, Cambridge.

[33] Kempson, Ruth M. (1975), *Presupposition and the delimitation of semantics,* Cambridge University Press, Cambridge.

[34] Lakoff, George (1967), *Deep and surface grammar,* unpublished mimeo.

[35] —— (1970), "Global rules," *Language 46.*, 627–638.

[36] —— (1971), "On generative semantics," In: Steinberg and Jakobovits (1971), pp. 232–296.

[37] —— (1974), "[Dialogue with] George Lakoff," In: Parret (1974), pp. 151–178.

[38] Langacker, Ronald W. (1969), "On pronominalization and the chain of command," In: Reibel and Schane (1969), pp. 160–186.

[39] Levine, Arvin (1976), "Why argue about rule ordering," *Linguistic Analysis 2.*, 115–124.

[40] Lewis, David (1972), "General semantics," In: Donald Davidson and Gilbert Harman (eds.), *Semantics of natural language,* D. Reidel, Dordrecht, 1972, pp. 169–218.

[41] Lewis, David (1974), "Languages and Grammar," In: Harman (1974), pp. 253–266.

[42] Lightfoot, David (1976), "Trace theory and twice moved $\overline{\text{NP}}$'s," *Linguistic Inquiry 7.*, 559–582.

[43] —— (1977), "On traces and conditions on rules," In: Culicover et. al. (1977), pp. 207–238.

[44] McCawley, James D. (1968), "The role of semantics in a grammar," In: Bach and Harms (1968), pp. 125–170.

[45] —— (1970a), "Where do noun phrases come from? " In: Jacobs and Rosenbaum (1970), pp. 166–183.

[46] —— (1970b), *Syntactic and logical arguments for semantic structures,* unpublished mimeo.

[47] —— (1974), "[Dialogue with] James McCawley," In: Parret (1974), pp. 249–278.

[48] Parret, Herman (1974), *Discussing language,* Mouton, The Hague.

[49] Partee, Barbara Hall (1971), "On the requirement that transformations preserve meaning," In: Fillmore and Langendoen (1971), pp. 1–22.

[50] Perlmutter, David M. (1971), *Deep and surface structure constraints in syntax,* Holt, New York.

[51] Peters, Stanley (ed.) (1972), *Goals of linguistic theory,* Prentice–Hall, Englewood Cliffs.

[52] Peters, S., and Robert W. Ritchie (1969), "A note on the universal base hypothesis," *Journal of Linguistics 5.*, 150–152.

[53] —— (1971), "On restricting the base component of transformational grammars, *Information and Control 18.*, 483–501.

[54] —— (1973), "On the generative power of transformational grammars," *Information Sciences 6.*, 49–83.

[55] Postal, Paul M., and Geoffrey Pullum (1978), "Traces and the description of English complementizer contraction." *Linguistic Inquiry 9.*, 1–29.

[56] Reibel, David A., and Sanford A. Schane (eds.) (1969), *Modern studies in English,* Prentice–Hall, Englewood Cliffs.

[57] Rosenbaum, Peter S. (1967), *The grammar of English predicate complement constructions,* MIT Press, Cambridge.

[58] Ross, John Robert (1967), *Constraints on variables in syntax,* unpublished Doctoral dissertation, MIT, Cambridge.

[59] —— "The cyclic nature of English pronominalization," In: Reibel and Schane (1969), pp. 187–200.

[60] Sadock, Jerrold M. (1975), *Toward a linguistic theory of speech acts,* Academic Press, New York.

[61] Searle, John R. (1969), *Speech acts: an essay in the philosophy of language,* Cambridge University Press, Cambridge.

[62] —— (1972) "Chomsky's revolution in linguistics," *The New York Review of Books,* June 29, 1972, pp. 16–24. (Also in Harman 1974)

[63] —— (1976), "Review of Sadock (1975)," *Language 52.*, 966–971.

[64] Seuren, Pieter A. M. (1974a), "Autonomous versus semantic syntax," In: Seuren (1974b), pp. 96–122.

[65] —— (ed.) (1974b), *Semantic syntax,* Oxford University Press, London.

[66] Steinberg, Danny D., and Jakobovits Leon A. (eds.) (1971), *Semantics: an interdisciplinary reader in philosophy, linguistics and psychology,* Cambridge University Press, Cambridge.

[67] Wasow, Thomas (1973), "More migs and pilots," *Foundations of Language 9.*

[68] Wilson, Deirdre (1975), *Presuppositions and non–truth–conditional semantics,* Academy Press, New York.

Computational Linguistics and Computer Languages Vol. XIII. 1979.

MACHINES IN THE SERVICE OF THE HUNGARIAN SUBSTANTIVE AS A MACHINE

F. Papp
University L. Kossuth, Debrecen
Hungary

1. INTRODUCTION

A morphological automaton receives the necessary grammatical information and the desired correct surface form is produced: Eng. TABL + SING – → table, TABLE + PLUR – → tables; Lat. TABULA + SING + ACC – → tabulam, TABULA + PLUR + GEN –→ tabularum, etc. Two prerequisites are indispensable for a morphological automaton to work correctly:

(i) the structure of the automaton must be described, and

(ii) depending on, as well as serving, the description given in (i), every single word in the lexicon has to be supplied with information of a certain type. Task (i) will be dealt with in greater detail below (1.2.). In general terms, the structure of (ii) is such that it indicates

a) the change the stem of a word must undergo as well as
b) the condition on which this change must take place.

Thus, one finds the following with the word lady:

a) *y* = *ie*
b) if +PLUR, to ensure the correct form ladies in the case of an orthographic output. The dependence of (ii) on (i) means that, depending on the way the structure of the automaton is described (in fact, on the way the automaton itself is constructed), more or less information of this or that type is necessary to be fixed in the lexicon. Thus, it is conceivable that one finds the following with the word lady:

a) *y* = *i*
b) if +PLUR, the automaton being devised in a way that it adds the ending -es to the stem already modified (therefore the form ladies is created at about the same place as the form dishes). It is also conceivable that the word lady as a lexical unit is not accompanied by any information whatever (more exactly, zero modification can be found, which means that it is a question of a "regular and standard stem wholly to be processed by the automaton"), but then such a rule has to be built into the automaton as first transforms all final *-y*'s into *-ie*'s, then adding the plural ending *s*. Forms like man – – men need separate treatment. A block has to be built in so as to stop the formation of the plural of some words or to give a unique solution (thus, the word information in the lexicon must be accompanied by a warning like "sing. t." or some such instruction that, in the case of the input information PLUR, one should get as an output form, say, this: kinds/pieces of information), etc. It can be seen, therefore, that even in the case of English, where the stem could be accompanied merely by the two possible values of the category NUM as input data (SING, PLUR), one

has to deal with a rather complex automaton. If this automaton is intended to be realized in a computer program, for this purpose one needs a good many instructions even in an advenced programming language and a wellprepared lexicon, etc. Therefore, the task is not trivial even in English (Latin, French, etc.).

It is even less trivialin the case of the Hungarian substantive. From the given viewpoint, Hungarian nouns have the following characteristics:

(i) while English, French, etc. substantive stems can be followed only by the two-value category (NUM) mentioned above, just as Latin, Russian, etc. substantive stems, after which two categories may stand (NUM and CAS, the second with several values: NOM, ACC, GEN, etc.), Hungaian substantive stems can be followed by

a) the category NUM more than once, according to the detailed rules given below,
b) the category PERS with three values, the category POSS with two values, and the category CAS with seventeen-twenty-odd values, according to various scholars.

(ii) These pieces of information follow each other in a certain definite order.
(iii) While in English (French, Russian, etc.) lexicon phenomena like man – – men, information – – ∅ (pieces of information) occur in a random fashion, in a Hungarian lexicon, besides contingencies similar to these, there is a cardinal piece of information concerning vowel harmony for synthesis, which systematically pervades the whole lexicon (every single word has either palatal or velar vowel harmony, or rather several dozen lexemes are subject to standard variation between the two vowel harmonies). This information can be unearthed from the sound shape of individual words by means of a separate small autmaton. And if one has such an automaton supplying units in the lexicon with the proper information on vowel harmony, it is worth expanding it in a way that it should be able to provide other pieces of information necessary for synthesis on the basis of the processing of the lexicon. The theoretical point of interest of such equipment lies in the fact that through it, information necessary for the synthesis of an arbitrary new lexeme can also be produced correctly with great probability, the analysis not being bound up with a given lexicon, large as it may be. At the same time, this has presumably resulted in a modelling of a Hungarian native speaker's activity when individual newly-met lexemes are provided with appropriate information from the lexicon, and their respective forms are compared in accordance with this:
SPUTNIK + SING + ACC – → szputnyikot (velar harmony, the ending of the accusative being -ot), BEATLES + PLUR + NOM – → bitleszek (palatal harmony, the ending of the plur. nom. being -ek), HOTEL + SING + LOC – → hotelban/ hotelben 'in (a) hotel' (unstable vowel harmony, the ending of the sing. locative – – inessive being -ban or ben), etc. Obviously, any lexicon must be open and must ensure manageability in the course of synthesis for new lexical units entering it. It is evident, at the same time, that this is not so trivial a task for a language like Hungarian as for English, where the problem lies in attaching (*e*)s for the expression of the plural, or for Russian in the case of consonantal stems: various declined forms of word 'Beatles' are easily produced, as in битлза

sing. gen./acc., битлзу sing. dat., битлзы plur. nom., etc.

The automaton that is able to generate all paradigmatic forms of Hungarian nouns, and an essential part of which was realized in a computer program several years ago will be described in brief below (cf. [3], [6]), to be followed by an outline presentation of what kind of information was necessary for this, and how it was obtained through a machine processing of the lexicon [1] (see [4] for details).

2. THE AUTOMATON GENERATING THE PARADIGMATIC FORMS OF THE HUNGARIAN SUBSTANTIVE

It follows from the foregoing, that while, with respect to full paradigms, an English noun has altogether two paradigmatic forms, a Latin substantive possesses ten, a Russian noun twelve – disregarding the Latin and Russian vocative as well as a few deficient Russian cases –, a Hungarian substantive with a full paradigm has a great many paradigmatic forms, according to some scholars as many as 714 (see [2, 50]), which is somewhere around the lower limit. Theoretically, however, – on account of a feature of the Hungarian substantive to be noted below – the number of forms individual Hungarian nouns have could be infinite (i.e. it could consist of an infinitely long series of letters). This situation is presented in Fig.1., to which the following comments are added.

The automaton works in the direction of the arrows: a stem with the information that determines its paradigmatic forms enters the automaton (further on: MI 'morphological information'), and the full form comes out in orthographic shape (of course, it could come out in phonetic transcription as well). The peculiarities and meaning of certain parts of MI are the following:

NUM – – is the category of number. Unlike Old Greek and Old Church Slavonic, etc. where the dual also existed, it can take on only two values: SING (singular) and PLUR (plural). However, as was noted briefly above, it may occur more than once within the full MI (in fact, it may occur an infinite number of times after the category POSS, which see below). It always refers to the singular or plural form of a preceding element (immediately to the left). Therefore, STEM + SG = the singular of the stem, PERS + PL = the plural of the person, etc.

PERS – – is the category of person. It may take on one of three values: 1st, 2nd, and 3rd person, an example being: <u>könyvem</u> 'my book', <u>könyved</u> 'your book', <u>könyve</u> 'his/her book'. Combined with NUM: KÖNYV + SG + ∅PERS → <u>könyv</u> 'book', KÖNYV + PL + ∅ PERS → <u>könyvek</u> 'books', KÖNYV + SG + 1PERS + PL + NOM → → <u>könyvünk</u> 'our book', KÖNYV + PL + 1PERS + SING + NOM → <u>könyveim</u> 'my books', KÖNYV + PL + 1PERS + PL + NOM → <u>könyveink</u> 'our books'.

POSS – – is the category of possession. It may take on either of two values. 'possession' or 'nonpossession'. For example, KÖNYV + SG + ∅PERS + POSS + SG + + NOM ⟶ <u>könyvé</u> 'that of the book'. (A folyóirat papírja szép, de a könyvé nem az. 'The paper of the journal is fine, but that of the book isn't.') The suffix <u>-é</u> of the possession may appear after more complex antecedents as well: <u>könyvemé</u> 'that of my book', <u>könyveké</u> 'that of the books', <u>könyvünké</u> 'that of our book', <u>könyveinké</u> 'that of our books', etc. Naturally, the possession itself may also be put into the plural: <u>könyvéi</u> 'those of the book', <u>könyveméi</u> 'those of my book', etc. When this is the case, the possessive suffix may be added repeatedly in the literary language: <u>könyvéié</u> 'that of those of the book', which may again be followed by a plural ending, etc. theoretically ad infinitum. This is indicated at this place in Fig.1. by the loop (cycle). In reality, more than a twofold repetition hardly ever occurs <u>(-éiéi-)</u>, therefore a counter ought to be built into the cycle: if it has occurred twice, the cycle is ended. In some Hungarian dialects the possessive suffix <u>-é</u> cannot be followed by the PL, therefore this cycle does not occur at all in these dialects.

CAS – – is the category of case. It may take on a value out of at least 17, for instance NOM, ACC, DAT, etc. Somewhat exotic case values can be demonstrated by the following examples: instrumental – – <u>könyvvel</u> 'with (a) book', superessive – – <u>könyvön</u> 'on (a) book', subessive – – <u>könyvre</u> 'on(to) (a) book' (with direction indicated), delative – – <u>könyvről</u> 'off (a) book', inessive – – <u>könyvben</u> 'in (a) book', etc. As Fig. 1 shows, arbitrarily complicated forms may be generated. For example: KÖNYV + SG + 2PERS + PL + INE ⟶ <u>könyvetekéiben</u> 'in those of your (plur.) book'.

The algorithm realized by the computer program presented in Fig.1., consisted of the following blocks: <u>1.</u> An input lexicon: SG, PL; 1PERS, 2PERS, 3PERS; POSS; NOM, ACC, DAT, etc. <u>2.</u> a block controlling the syntax of input information: some value of NUM may figure anywhere, except after itself and some value of CAS. The other categories may follow the stem only in this order: PERS – – POSS – – CAS. If there was an error in stating the task (in the sequential order of individual categories), the machine gave a feed-back signal. <u>3.</u> An output lexicon, which contained real (surface) Hungarian endings. <u>4.</u> A converter block contained the input lexicon on the one hand, and the output one on the other. In addition it also contained the requirements which had to be met for individual input data to be transformed into surface output data of one or another kind. Examples:

(i) Everywhere at the output ∅ corresponded to the input information of the SG.

(ii) (Among others) *k* corresponded as a concrete output to PL input information if 1PERS figured (cf. above: <u>könyvünk</u> 'our book'): the *k* shows that it is 'ours' and not 'mine' but an output *i* corresponded to it if the suffix of PL figured after POSS. The rules operate cyclically till all input information is converted into final output forms. In the course of their operation, the rules also generated intermediary symbols, which were not part of the input lexicon, but which were not Hungarian endings either, so that they gradually had to be transformed into real endings.

3. INFORMATION GAINED FROM THE LEXICON AND HOW IT IS GENERATED

The automaton described under item 2. worked without any information from the lexicon whatever. In its converter block (4), there were also such general rules among the prerequisites that extracted the necessary information for a correct selection of endings from the shape of the stem. In this way, however,

a) exceptions were to have wrong forms (as if in English forms like $^{+}$mans, $^{+}$tooths, etc. had been generated);

b) in some cases two forms were generated and one had to choose from the two;

c) forms missing from the full paradigm of a given lexeme were also received. It is obvious that information from the lexicon is needed for generating correct forms and only the correct forms. It is advisable to detach the generation of information from the lexicon from the synthesis itself.

3.1. NOUNS WITH DEFECTIVE PARADIGMS

Standard European languages surrounding Hungarian recognize, in essence, only two types of defective paradigm: "SG T" – – singularia tantum, and "PL T" – – pluralia tantum. A more complete picture is demonstrated by Table 1.

CAS / NUM	TOT CAS	NOM T	OBL T
TOT NUM	words with a full paradigm	APPELL	? ?
SG T	SG T	APPELL	se, sibi
PL T	PL T	APPELL	ADV

Table 1.

This table shows that two kinds of defectiveness can be discovered in respect of cases as well:

a) if (in both numbers or in either of the two numbers) only the nominative exists, it is a question of appellatives;

b) if only the oblique cases, or some of them, exist (as in the case of the Latin reflexive pronoun: se, sibi), it is a question of nouns with a defective paradigm or, in fact, adverbs. It is clear why this two-dimensional table was enough: the full paradigm could be set up only according to the two obligatory categories of case and number.

Hungarian presents a considerably more complicated case: paradigm, in addition to the above two categories, can also be formed according to the obligatory category of PERS. That is why Hungarian forms with full/defective paradigms can only be represented in three dimension, as can be seen in Fig. 2. (The POSS is not an obligatory category; that is why some authors do not even treat it among paradigmatic forms. If this category were also obligatory – – i.e. if a separate group of nouns were characterized by the fact that they are unable to have a possession, and, consequently, would not be able to have a form in *é* – –, the situation could be represented only in four dimension.) As in Table 1., Figure 2. also reveals the introduction of the simplification that it was not examined which oblique cases were missing from the list of cases. Only such distinctions are introduced that the NOM is missing, or that some/all of oblique cases are missing.

It can be seen that non-defective elements (having a full paradigm) can be found in the upper left corner of Fig.2., with the classical SG T and PL T below them.

Fig. 3. sets out to demonstrate what ways are missing from the synthetizing automaton outlined above in the process of generating Hungarian nouns with defective paradigms. (It should be noted that the processing of about 35 000 nouns to be found in [1] showed that they included approximately 3000 SG T and a further 700 defective elements.) Here are some examples to illustrate Hungarian nouns having rather peculiar defective paradigms, in accordance with the numbering given in Fig., 2.:

19 TOT NUM, PERS T, TOT CAS. That is to say all the cases of the two numbers exist, but the word can only appear in a form having a personal ending. Such are fia 'his/her son', neje 'his wife' and some words having similar semantic characteristics.

22 SG T, PERS T, TOT CAS: holtom, holtad, holta 'my death, your death, his/her death', főztöm, főztöd, főztje, 'my cooking, your cooking, his/her cooking'. It should be noted that 'death in general' also exists, but it is a different lexeme. The same can be stated about other defective nouns. (Occasionally, however, there occur nouns that do not exist independent of some person. Such are főztöm, főztöd, etc. It is possible to have in mind food, meals, lunch etc. in general, but the concept itself of 'the result of somebody's cooking' can only have forms with these possessive endings.)

25 PL T. PERS T, TOT CAS: eleim, eleid, elei . . . 'my ancestors, your ancestors, his ancestors . . . '. It is important to emphasize that there is a lexeme having the same meaning but with a full paradigm: ős 'ancestor'. The existence of elei proves precisely that it is a question of real defectiveness of paradigm here rather than, say, that the concept designated by the given lexeme as such prefers to figure in the company of several others like possessive definiteness, etc. One is not interested in reality or notions referring to parts of reality here: one has to establish the defectiveness of a lexeme as a linguistic unit. That is why in the given sense one would never come across lexemes like *elő, *elők, etc. in the course of analysis, for it is the appropriate forms of the lexeme ős that always figure in such cases. In the course of synthesis, however, this lexeme may, in theory, be accidentally selected for this meaning. Information from the lexicon indicating defectiveness serves precisely for the purpose of stopping the generation of non-existent forms of this kind, putting another stem with the same meaning but having a full paradigm into the input of the automaton.

Finally, there is a small group of nouns with a defective paradigm, which is hard to place. The reason for this can be understood on the basis of Fig. 3.: here a path rather than a point of the graph describing the automaton is missing (marked by a white arrow in the figure). That is to say, these lexemes either figure

a) in the plural and may have forms with or without personal endings or
b) in the singular having only forms with personal endings.

Such are:

a) léptek 'steps', léptei 'his/her steps',
b) lépte 'his/her step' ('step in general': lépés, with a full paradigm);
a) párthivek 'adherents of a party', páthivei 'his/her/its adherents',
b) párthive 'his/her/its adherent'. It is interresting that, perhaps, an international term also behaves like this: the lexeme homonima 'homonym' occurs either in the plural (homonimák 'homonyms') or with a personal ending (homonimája 'its homonym') '(one word is) the homonym of another'. However, this is a typical case when it is not the linguistic phenomenon (lexeme) that is 'defective', but reality itself: in the nature of things a 'homonym' implies more things than one, or a word can be said to be the homonym of another, therefore, for it to express the meaning 'of' it is put in the form of 3 PERS. That is why the Hungarian word 'homonym', for example, may figure in bare singular too, namely in the language use of linguists, who may talk of a homonym in the abstract.

Synthesis proper can start only after this point has been controlled, i.e. if it has been established that the lexeme in question has a full paradigm according to the information supplied by the lexicon (or at least its form appearing at the input exists).

3.2. TYPES OF STEM

If our synthetizing automaton contained a rule stating that "a stem is the maximum (longest) identical (unchanging) segment of word forms realizing lexemes", then, in written form, the stem of the English word lady would be lad-, as the latter *y* is to change. Similarly, the stem of the verb write would be only *wr-* (since there is at least one such paradigmatic form in which the subsequent *-i-* is replaced by something else: wrote), etc. A grammar containing such a rule would also give correct results, i.e. it would meet the requirements of the principle of explanatory adequacy. Only considerations extending beyond the scope of grammar, i.e. considerations concerning economy, the simplicity and elegance of the description, make one vote rather for a grammar containig the rule: "a stem is the maximum similar segment of word forms realizing lexemes" and the 'similar' would cover automatic changes like *y* - - *ie* (lady - - ladies) easil operating on stems.

By relying partly on Hungarian grammatical tradition, Hungarian substantive stems could be divided into 12 groups. In addition, there were a few hundred su bstantive lexemes either unbalanced between two pure types of stem or which behaved rather individually. In 81% of the 35 000 nouns mentioned earlier, no modification what ever had to be effected on the stems, while in 13% of them an easily definable modification had to be carried out (long *á, é,* etc. appear before some endings in their short forms: almát 'apple acc.' – – alma 'apple - - nom.'). The remaining 6% of substantives was divided among representatives of rare stem types as well as unbalanced substantives. The automaton operating without information from the lexicon presented under item 2, worked with a precision of 94%: it was able to carry out 'zero modification' as well as 'long-short stem final vowel modification'. New words like szputnyik or beatles mentioned under item 1, all belong to the unchanged stem type (various rare changes in stems are naturally historical relics, and new lexemes are highly unlikely to get into any of these groups). Therefore, the memory of native speakers of Hungarian is burdened only by these relics (about two thousand in number), and this number will hardly increase, in fact the opposite can be anticipated.

3.3. A VOWEL BETWEEN THE STEM AND CERTAIN ENDINGS

Certain endings with initial consonants cannot directly be linked to the stem, so a vowel (a stem-final short vowel at an earlier stage of the history of Hungarian) is wedged between them, for instance, kalap-o-m 'my hat', ház-a-m 'my house', könyv-e-m 'my book' . . . ; the coresponding plural forms are kalap-o-k, ház-a-k, könyv-e-k; the corresponding accusative forms being kalap-o-t, ház-a-t, könyv-e-t, etc. Special information concerning this vowel is also fixed by every single lexeme in the lexicon.

3.4. CERTAIN FORMS WITH PERSONAL ENDINGS

It is a peculiar problem for synthesis if the stem is followed by one of the following two input information series:

(1) SG 3PERS J (J = SG or PL)

(2) PL IPERS J (I = 1 or 2 or 3, J as in (1)

(after this *é* element of POSS may or may not figure, and any CAS ending may follow). The problem is caused by the fact that in such cases a phoneme |j| either appears between the stem and the appropriate string of endings or does not – in a distribution that seems to be random:

a) it either appears before the same group of stem-final vowels or consonants or does not: türelm-e 'his/her patience' without a |j|, but film-j-e 'his/her film with a |j|;

b) occasionally it does not present itself after a single stem-final consonant, on the other hand it does turn up following an overloaded stem-final consonant cluster, further extending it: ablak-a 'his/her window' (cf. the final single (*k*)), but barack-j-a 'his/her apricot' (cf. the cluster *-ckj* obtained as a result!);

c) there are many cases of fluctuation between forms with a |j| and forms without one.

In the course of a manysided machine processing of the material, it was established that the phoneme |j| appeared if – precisely because of rare and complicated, etc. stem-final clusters – this stem-final position was necessary to be indicated separately. Fluctuations could be anticipated at places where the cluster was neither too typical not too rare. In the case of an identical stem-final consonant (cluster) (cf. a) above) |j| presents itselt in new loan-words (film), but does not appear in old words, where a rare (changing) stem type marks stem-final position anyway (türelem 'patience' nom. - - the stem is modified: türelm-). That is to say, native speakers of Hungarian also strive to operate a minimum-size lexicon, and |j| promotes a non-lexicon-bound automatic analysis in marking stem-final position in these forms.

3.5. VOWEL HARMONY

It is a well-known fact that the principle of vowel harmony in Hungarian has changed: virtually it is the vowel of the last syllable that decides whether the stem is velar or palatal (cf. [6]). In the course of the machine ordering of the material of [1], exceptions as well as substantives showing fluctuation as to vowel harmony were sifted out, then a cyclic rule for the bulk of the nouns (and obviously for most new words to come) was successfully established, which, on the basis of the vowels constituting a lexeme, automatically determined its pattern of vowel harmony. The automaton presented under item 2, solved this task too with a precision of 90%. The velar or palatal quality of all endings containing a vowel depends on VH. Individual PERS, CAS, etc. endings have allomorphs depending on this principle: Inessive - - ban/ben; Allative - - hoz/hez/höz, 1 PERS SG om/em/öm, etc.

4. FINAL REMARKS

Hitherto machines and Hungarian substantives have been brought into contact in two respects

a) the numerous (perhaps infinitely great) number of Hungarian substantive forms could be represented in a way that an automaton generated them;

b) the information from the lexicon necessary for this was advisable to be obtained through a machine processing of the data in the lexicon. One cannot, however, be silent about a further potential connection between the Hungarian substantive and automata (perhaps logical languages?) either. The well-organized substantive forms of Hungarian or other agglunative languages built on an agglutinative basis may serve as a model for constructing certain elements of a logical language.

REFERENCES

[1] A magyar nyelv értelmező szótára (An Explanatory Dictionary of the Hungarian Language) Vols. I-VII. Budapest, 1959-1962.

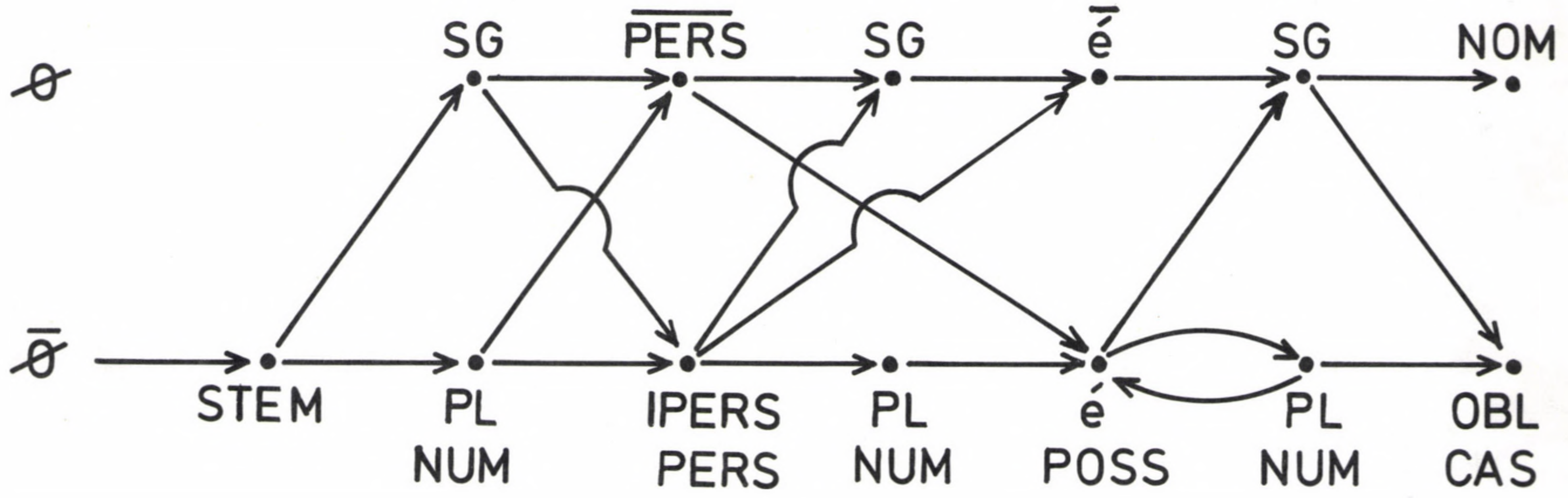
[2] Antal László, A magyar esetrendszer (The Hungarian Case System) Budapest, 1961.

[3] Papp Ferenc, A magyar főnévragozás három modellje (Three Models of Hungarian Declension) In: Magyar Nyelv 62.2. 194-206 (1966)

[4] Papp Ferenc, A magyar főnév paradigmatikus rendszere (leirás és automatikus szintézis) (The Paradigmatic System of the Hungarian Substantive. Description and Automatic Syntesis), Budapest, 1975.

[5] Stein Mária, Synthese des ungarischen Hauptwortes mit einer elektronischen Rechenmaschine. In: Computational Linguistics 5. 169-176 (1966)

[6] Szépe György, Vegyes magánhangzójú szavainak illeszkedésének kérdéséhez. (On Some Questions of Vowel Harmony in Hungarian Words Containing Mixed Front and Back Wovels) In: Nyelvtudományi Értekezések 17. 105-129 (1958).



SG
$\overline{\text{PERS}}$
SG
$\bar{\acute{e}}$
SG
NOM
$\bar{\emptyset}$
STEM
PL NUM
IPERS PERS
PL NUM
é POSS
PL NUM
OBL CAS

Fig. 1.

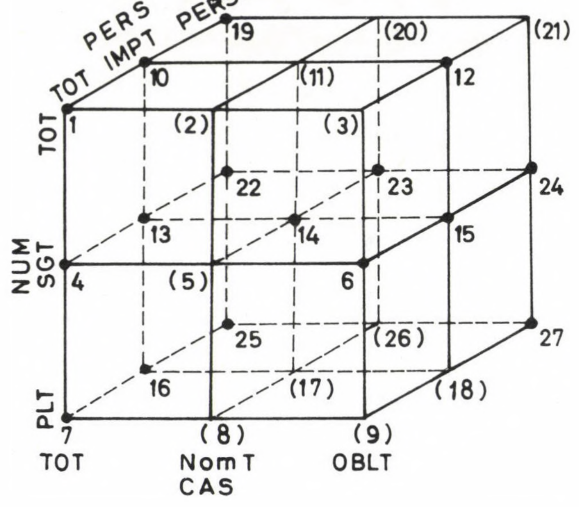

PERS
TOT
IMPT
PERST
1
(2)
(3)
4
(5)
6
7
(8)
(9)
10
(11)
12
13
14
15
16
(17)
(18)
19
(20)
(21)
22
23
24
25
(26)
27
TOT
NUM
SGT
PLT
TOT
NomT
CAS
OBLT

Fig.2.

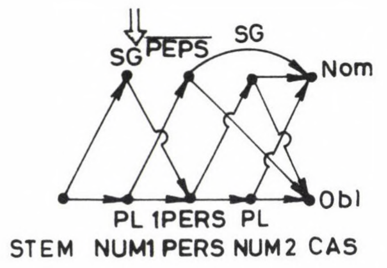

SG
PEPS
SG
Nom
Obl
PL
1PERS
PL
STEM
NUM1
PERS
NUM2
CAS

Fig. 3.

C. SOFTWARE METHODOLOGY

C. SOFTWARE METHODOLOGY

Computational Linguistics and Computer Languages Vol. XIII. 1979.

INDUCTIVE GENERALIZATION AND PROOFS OF FUNCTION PROPERTIES

P. Degano and F. Sirovich

Instituto di Scienze dell'Informazione.
Universita' degli Studi di Pisa.
Italy

Instituto di Elaborazione dell'Informazione.
Consiglio Nazionale delle Ricerche. Pisa
Italy

Abstract

The paper presents a formal system and an inductive method for proving function properties and investigates the relationships between inductive and deductive proofs. Induction is performed by stepwise generalizing specific given elements of the function domains which are known to satisfy the property itself. The method is based on symbolic computation, reflexivity lemmas and function behaviour estimate, and is proven sound when functions and predicates belong to a constructively defined class. Finally, an example is completely worked out.

Keywords

primitive recursive functions, inductive reasoning, generalization, symbolic computation, program properties, function behaviour estimate.

1. Introduction

Inductive reasoning has been a popular task in Artificial Intelligence since the very early beginnings, and it has been attempted in such domains as (just to mention a few) series completion, grammar inference, automatic programming, and theory formation [5, 6, 8, 9, 12, 15, 17, 18, 21].

In deductive sciences inductive reasoning plays a peculiar role in the process of knowledge development. In fact, the correctness of an induction can be confirmed by a rigorous proof, even though a semi-decision procedure only might be available, depending on the formal system complexity. Moreover, the rejection of an inductive hypothesis may bring forth explanations about the failure which help refining the hypothesis. Finally, the aptness of the inductive method itself can be precisely stated and confirmed by a formal proof.

It goes far beyond the scope of this paper to give a complete account of all the work on inductive reasoning in formal systems. We might mention just a few references. Meltzer

[13], Michalski [14], Plotkin [16] and Vere [19] tackle various forms of inductive reasoning tasks in the predicate calculus. Brown and Tärnlund [4] are concerned with finding a close form solution to difference equations. They discuss a taxonomy of inductive methods and propose a temporal method based on proofs, where the proof of the (non) correctness of a solution is used to formulate a new inductive hypothesis. In the context of program verification, Boyer and Moore [2], Brotz [3], and Aubin [1] need to generalize theorems to be then proven by application of a suitable induction principle. The problem domain we have chosen is akin to the latter.

The inductive generalization method we present here induces properties of (recursive) functions. The properties themselves are defined as (boolean valued) recursive functions. The method depends on the fact that a given property holds for given points in the domains of the involved functions *(ground property)*, and on the corresponding ground property computation. We can prove that the method is *sound*, i.e. it induces formulae which are actually theorems if the involved functions and properties belong to a constructively defined class.

The evidence the method starts from is that a very trivial (with no quantified variables) theorem holds, because indeed the actual computation is a proof. Yet more information could be exploited. The definitions of properties and functions are available and must be taken into account, because no relevant inductive reasoning can be made irrespectively of their nature. Secondly, induction can depend on the details about the flow of computation, i.e. which function, when and where, was applied during the computation.

We will first present the formal calculus environment for the inductive method and define the class of functions and properties for which the method can be proven sound. The inductive generalization method is presented, and the proof of its soundness is given in Section 7. In Section 8 related research work is discussed, and the Appendix provides an example completely worked out.

2. A formal calculus environment

We use a simple recursive function formalism, TEL (Term Equation Language) which was introduced [11] for proving theorems by symbolic computation (see for example [2]) and is similar to other independently developed formalisms [1, 7]. For the present application we add types to TEL so that the resulting language is so similar to Aubin's that the formal treatment and all results of his carry over Typed TEL. We now briefly overview TTEL, borrowing some nomenclature from Aubin's.

Every term, i.e. every variable and function application, has a type. Each type is introduced by a set of type equations which also define type constructors. All the types and constructors occuring in the equation (apart from the type and constructor being defined) must have been previously introduced. The language is quantifier free, because every variable occurring in an equation is implicitly universally quantified over its type. Examples are

{TYPE(TRUE()) = BOOL;
TYPE(FALSE()) = BOOL}

{TYPE(ZERO()) = NAT;
TYPE(S(NAT)) = NAT}

{TYPE(NIL()) = LIST;
TYPE(CONS(LIST, LIST)) = LIST}

{TYPE(LNIL()) = LLIST;
TYPE(LCONS(LIST, LLIST)) = LLIST}

The constructor TYPE is used only to denote type equations. In the sequel we will omit the argument list of 0-adic constructors. We say that a type is *reflexive* if it is defined in terms of itself (e.g. NAT, LIST and LLIST). Analogously, we say that constructors like S and CONS are reflexive and the argument position where the type they construct occurs is called the *reflection argument position*. Non reflexive constructors will also be called *terminators* of the type. Given a term $c(t_1, \dots, t_n)$ where c is a constructor, $t_1, \dots, t_n$ are terms, then t_i $(1 \leqslant i \leqslant n)$ is an *immediate predecessor* of $c(t_1, \dots, t_n)$ if t_i occurs in a reflection argument position.

Defined functions are introduced by stages. A function definition is a pair, whose first component is a type equation which defines the types of the arguments and the type of the result. For example TYPE(EQN(NAT, NAT)) = BOOL. The second component consists in a set of equations (*rewrite equations*), which allows a definition by cases [7, 10]. Rewrite equations obey the schema $f(a_1, \dots, a_n) = \langle\text{rewriting term}\rangle$ where f is the function being defined, and $a_1, \dots, a_n$ are terms (*formal arguments*) which may consist either in a variable, or in a constructor applied to variables only (*recursion argument*). The only variables which may occur in the ⟨rewriting term⟩ are the formal argument variables. If f occurs in the ⟨rewriting term⟩ (recursive equation), its recursion arguments must be immediate predecessors of the formal recursion arguments.

The definition by cases is restricted as follows. If an argument position is recursive in one equation, then it must be a recursive argument position for all the equations, and for each constructor of the required type there must be exactly one rewrite equation[1]. Thus, total functions only can be defined in TTEL. This is a concrete example.

{EQN(ZERO, ZERO) = TRUE; (ENb1)
EQN(ZERO, S(y)) = FALSE; (ENb2)
EQN(S(x), ZERO) = FALSE; (ENb3)
EQN(S(x), S(y)) = EQN(x, y)} (ENr)

The definition of computation of a TTEL term follows.

1 If the function simultaneously recurs on two or more arguments, the exactly one rewrite equation must be given for each tuple of constructors of the required types (an example follows).

i) Specialize a rewrite equation so that its left–hand side becomes identical to a (sub)term of the evaluating term;

ii) Substitute in the evaluating term the specialized equation right–hand side for the (sub)term;

iii) Repeat from Step (i) until no equation left–hand side can be made identical to a (sub)term of the evaluating term.

The interpreter adopts the call–by–need computation rule which is known to be optimal for recursion equations [20].

Let us now extend the TTEL term definition by introducing free (typed) variables. A *symbolic term* is a term which free variables occur in, otherwise the term is *ground.* A free variable can be instantiated to any term of its type, possibly introducing new free variables. We can extend the definition of computation to handle symbolic evaluating terms. Step (i) only needs to be extended so that the involved test for identity can cause evaluating (sub) term free variables to be instantiated. The computation of a symbolic term may be non–deterministic, due to the inherent non–determinism of free variable instantiation. Thus symbolic computations are (possibly infinite) trees. A concrete example is the following. The term[1] EQN $(S(\underline{x}), S(S(\underline{y})))$ is reduced to EQN$(\underline{x}, S(\underline{y}))$ by (EN2) and then either to FALSE if $\underline{x}$ is instantiated to ZERO by (ENb2), or to EQN$(\underline{x}_1)$ if $\underline{x}$ is instantiated to $S(\underline{x}_1)$ by (ENr). From this point on, all EQN equation can be applied.

We might provide the same inference rules available in Aubin's system, we instead omit here since we are interested on inductive proof methods. It is only important to notice that computations yielding TRUE are proofs because the interpreter itself implements the tactics Aubin calls simplification. Moreover, a theorem with universally quantified variables is proven by straight computation if free variables are substituted for quantified variables and the obtained symbolic term computation yields TRUE without instantiating the introduced free variables. Even if the metalinguistic constraints on function definition keep function definitions very simple, yet all primitive recursive functions over type NAT can be defined in TTEL. It is well known that only a semi–decision procedure can be given for the primitive recursive function theory. In order to be able to give a decision procedure, we add the following metalinguistic constraints. The non boolean functions must belong to the class be defined according to the following schemata.

U–schema

$$f(x_1, \ldots, x_n) = x_i \qquad 1 \leqslant i \leqslant n$$

N–schema

$$n(T, y_2, \ldots, y_n) = \varphi(y_2, \ldots, y_n) \qquad \text{(Nb)}$$

$$n(c(x_1, x_2, \ldots, x_m, y_1), y_2, \ldots, y_n) = c'(\delta_1(x_1, \ldots, x_m), \ldots, \delta_q(x_1, \ldots, x_m), n(y_1, y_2, \ldots, y_n)) \qquad \text{(Nr)}$$

1 Free variables will be underlined.

R–schema

$r(T, y_2, \dots, y_m) = \psi(y_2, \dots, y_n)$ (Rb)

$r(c(x_1, \dots, x_m, y_1), y_2, \dots, y_n) = n_R(r(y_1, \dots, y_n), c'(\gamma(x_1, \dots, x_m), T'))$ (Rr)

$n_R(T', y_2) = \phi_R(y_2)$

$n_R(c'(x_1, \dots, x_m, y_1), y_2) = c''(x_1, \dots, x_m, n_R(y_1, y_2))$

S–schema

$s(x_1, \dots, x_n) = f(h_1(x_1, \dots, x_n), \dots, h_m(x_1, \dots, x_n))$ [1]

where:

– ϕ, ψ, and ϕ_R either are constructors or must be defined according to N– or R–schema only (linear functions);

– δ_i, γ, f, h_i either are constructors or belong to the class;

– c, c', c'' are reflexive constructors, and T, T' are terminators.

Note that n_R definition obeys a restricted N–schema. The boolean functions must be defined according to the schema

$b(T, T') =$ BCONST1 $b(c(x, y), T') =$ BCONST2

$b(T, c'(z, w)) =$ BCONST3 $b(c(x, y), c'(z, w)) = b1(b2(x,z), b(z,y))$

$b1($BCONST4$, y) =$ BCONST5 $b1($BCONST6$, y) = y$

where BCONSTi are boolean constants (note that BCONST4 ≠ BCONST6), and $b2$ is a symmetric transitive boolean function belonging to the class. In the sequel boolean functions such as b and $b2$ will be called *predicates.* Note that functions defined on type BOOL (such as $b1$) are allowed only as auxiliary functions in predicate definitions. Then, the form of the theorems is $b(t_1, t_2)$, where b is a predicate, and t_1, t_2 are terms built on functions from reflexive types to reflexive types only.

3. Inductive generalization in TTEL

We can now describe a method to induce function properties from examples. Suppose that TTEL type and function definitions are given, and a ground property is computed by means of the TTEL interpreter. Thus, the resulting property value is available together with a computation trace consisting of a sequence of pairs ⟨rewritten subterm, applied rewrite equation⟩. We have chosen a very simple formal calculus in order to be able to describe in a compact way the flow of computation, and to provide a detailed explanation of why a given function property holds for specific actual arguments.

1 Definition by composition is actually allowed as a short–hand for S– and U–definition.

In other words a theorem and its proof are given, and the goal is to induce from this evidence a formula which subsumes the ground property, and hopefully is a theorem itself. This inductive generalization task is accomplished in three steps. First, the most general theorem is found whose proof consists in the given computation trace. This is a proper generalization which can actually be checked. Secondly, properties of predicates such as equality are exploited to further generalize the theorem. A proof of the new theorem is not given, but it could easily be obtained by simplification and induction. Thirdly, a final generalization is obtained on the grounds of few rules which we will prove that do yield valid results.

4. Generalization based on proof

The first clue to start from is the given computation trace. In fact, the very same computation trace may prove a theorem stronger than the given one. In first place, if a function application term is never evaluated during the computation, the term can safely be substituted by a universally quantified variable and the given computation will still be a proof of the obtained theorem. A typical example is provided by the following ground property

```
EQLENGTH(APP(CONS(PLUS(S(ZERO), ZERO), NIL),          (1)
                    CONS(S(ZERO), CONS(ZERO, NIL))),
              APP(CONS(S(ZERO), CONS(ZERO, NIL)),
                    CONS(PLUS(S(ZERO), ZERO), NIL)))
```

where the involved functions are defined as follows.

```
(TYPE(EQLENGTH(LIST, LIST)) = BOOL; ¦EQLENGTH(NIL, NIL) = TRUE;
 EQLENGTH(CONS(x, y), NIL) = FALSE;  EQLENGTH(NIL, CONS(z, w)) = FALSE;
 EQLENGTH(CONS(x, y), CONS(z, w)) = EQLENGTH(y, w)¦ )

(TYPE(APP(LIST, LIST)) = LIST;
¦APP(NIL, z) = z;  APP(CONS(x, y), z) = CONS(x, APP(y, z))¦)

(TYPE(PLUS(NAT, NAT)) = NAT;
¦PLUS(ZERO, y) = y; PLUS(S(x), y) = S(PLUS(x, y))¦)
```

Since the test for list length equality obviously ignores the nature of the list elements, the terms PLUS(S(ZERO), ZERO) are never evaluated, and can be substituted by universally quantified variables. The following theorem is obtained

```
EQLENGTH(APP(CONS(x, NIL),
                    CONS(S(ZERO), CONS(ZERO, NIL))),
              APP(CONS(S(ZERO), CONS(ZERO, NIL)),
                    CONS(y, NIL))).
```

The same generalization can be done on those data terms (i.e. terms built on constructors only) whose structure is irrelevant to the computation. In the example, one would like to induce

EQLENGTH(APP(CONS(x, NIL)
        CONS(z_1, CONS(z_2, NIL)))
    APP(CONS(w_1, CONS(w_2, NIL)),
        CONS(y, NIL))).

The relevant part of the occurring data terms can be singled out by resorting to the function rewrite equations, whose formal argument terms describe exactly the most general data term the rewrite equation can be applied to. Since the symbolic computation can instantiate free variables in order to apply a rewrite equation, the required generalization is obtained by firstly substituting free variables for the maximal data terms in the theorem, and then by evaluating the resulting symbolic term, forcing the computation to exactly follow the given computation trace. The original free variables will turn out to be instantiated only as far as needed to obtain the given computation. Let us work out the above example to clarify the matter.

Suppose the maximal data terms are replaced as follows

EQLENGTH(APP($\underline{v}_1$, $\underline{v}_2$),   (2)
    APP($\underline{v}_3$, $\underline{v}_4$))

Since the computation trace reports that the recursive APP equation was applied to the APP terms, the variables $\underline{v}_1$ and $\underline{v}_3$ need to be instantiated to CONS($\underline{v}_{11}$, $\underline{v}_{12}$) and CONS($\underline{v}_{31}$, $\underline{v}_{32}$) respectively. Analogously, $\underline{v}_{12}$ is subsequently instantiated to NIL and $\underline{v}_{32}$ to CONS($\underline{v}_{33}$, NIL). All instantiations are finally collected, and the still remaining free variables are substituted by universally quantified variables, thus obtaining the following theorem

EQLENGTH(APP(CONS(v_{11}, NIL),
        CONS(v_{21}, CONS(v_{23}, NIL))),
    APP(CONS(v_{31}, CONS(v_{33}, NIL)),
        CONS(v_{41}, NIL))).

Although this is actually the most general theorem whose proof consists in the given computation, the result is not satisfactory. Since there is no relationship between the EQLENGTH argument terms, the theorem cannot be further generalized, for example to induce the commutativity of APP with respect to EQLENGTH. A weaker generalization which would retain the links between EQLENGTH argument terms would instead admit further generalization. On the other hand, the forced computation will never be able to reconstruct from a term such as (2) the identities of $\underline{v}_1$ to $\underline{v}_4$, and of $\underline{v}_2$ to $\underline{v}_4$. In other words, if forced symbolic computation allows for some variables to remain free in the input data terms, it cannot at the same time bind some of them together. Consequently, the substitution of free variables for input data terms must be done carefully. In order to obtain interesting theorems, the variables occurring in one actual argument term of simultaneously recurring predicates, such as EQLENGTH, should occur in the other argument term as well. Thus, different free variables are substituted for identical data terms in one argument term, and the introduced variables are carried over the other argument term. For example, if the ground property is

EQLENGTH(APP(CONS(ZERO, NIL),
                CONS(ZERO, NIL)),
        APP(CONS(ZERO, NIL),
                CONS(ZERO, NIL)))

then free variables are introduced as follows

EQLENGTH(APP($\underline{v}_1$, $\underline{v}_2$),
        APP($\underline{v}_2$, $\underline{v}_1$))

and forced computation yields the theorem

EQLENGTH(APP(CONS(v_{11}, NIL),
                CONS(v_{21}, NIL)),
        APP(CONS(v_{21}, NIL),
                CONS(v_{11}, NIL))).

Actually, $\underline{v}_1$ and $\underline{v}_2$ in the second APP term could also be introduced the other way around, thus yielding

EQLENGTH(APP($\underline{v}_1$, $\underline{v}_2$),
        APP($\underline{v}_1$, $\underline{v}_2$))

which is a trivial theorem. All possible free variables introductions must be done, and the resulting hypotheses are tested and possibly rejected by a triviality checker or by subsumption.

The limitation of the proof based generalization method stems from its extravagant dependence on the involved functions. For example, consider the following ground property.

EQL(APP(CONS(ZERO, NIL),   (3)
        APP(CONS(ZERO, NIL), REV(NIL))),
    APP(APP(CONS(ZERO, NIL),
            CONS(ZERO, NIL)),
        REV(NIL)))

where EQL is defined as

(TYPE(EQL(LIST, LIST)) = BOOL; {EQL(NIL, NIL) = TRUE;
EQL(CONS(x, y), NIL) = FALSE; EQL(NIL, CONS(z, w)) = FALSE;
EQL(CONS(x, y), CONS(z, w)) = AND(EQN(x, z), EQL(y, w))})

(TYPE(AND(BOOL, BOOL)) = BOOL;
{AND(TRUE, TRUE) = TRUE; AND(TRUE, FALSE) = FALSE;
AND(FALSE, FALSE) = FALSE; AND(FALSE, TRUE) = FALSE})

(TYPE(REV(LIST)) = LIST;
¦REV(NIL) = NIL; REV(CONS(x, y)) = APP(REV(y), CONS(x, NIL))¦)

No free variable introduction helps, because the forced computation will anyway yield back the ground property. A careful analysis of the EQL definition points out that EQL is a much more demanding equivalence relation than EQLENGTH, because EQN accurately checks the list elements by means of the function EQN.

On the other side, equivalence relations are of such a paramount importance that it is reasonable to let the inductive generalization be sensitive to them. The next Section presents a generalization method which exploits the presence of equivalence predicates in the theorem, as a first start on generalizing the proof.

5. Generalization based on reflexivity

Let us describe the role of reflexivity by means of example (3). The forced computation starts with the following term

$$\begin{array}{l}
\mathrm{EQL(APP(}\underline{v}_1, \\
\qquad\qquad \mathrm{APP(}\underline{v}_2, \mathrm{REV(}\underline{v}_3))), \\
\quad \mathrm{APP(APP(}\underline{v}_1, \underline{v}_2), \\
\qquad\qquad \mathrm{REV(}\underline{v}_3)))
\end{array}$$

The computation trace forces the instantiation of $\underline{v}_1$ to let the outermost APP terms produce (through the recursive APP equation) two CONS terms. Thus the symbolic term is rewritten as follows

$$\begin{array}{l}
\mathrm{EQL(CONS(}\underline{v}_{11}, \mathrm{APP(}\underline{v}_{12}, \\
\qquad\qquad\qquad\qquad \mathrm{APP(}\underline{v}_2, \mathrm{REV(}\underline{v}_3)))), \\
\quad \mathrm{CONS(}\underline{v}_{11}, \mathrm{APP(APP(}\underline{v}_{12}, \underline{v}_2), \\
\qquad\qquad\qquad\qquad\quad \mathrm{REV(}\underline{v}_3)))).
\end{array}$$

Now the EQL recursive equation is applied yielding

$$\begin{array}{l}
\mathrm{AND(EQN(}\underline{v}_{11}, \underline{v}_{11}), \\
\quad \mathrm{EQL(APP(}\underline{v}_{12}, \\
\qquad\qquad \mathrm{APP(}\underline{v}_2, \mathrm{REV(}\underline{v}_3))), \\
\qquad \mathrm{APP(APP(}\underline{v}_{12}, \underline{v}_2), \\
\qquad\qquad \mathrm{REV(}\underline{v}_3)))).
\end{array}$$

This is the first place where reflexivity can help generalizing. Since EQN is an equivalence relation the reflexivity lemma EQN(x, x) can easily be proven by means of a suitable induction principle. The lemma can be used to evaluate EQN($\underline{v}_{11}$, $\underline{v}_{11}$) in place of the EQN rewrite equations, and that part of the computation trace describing the evaluation of EQN can be by-passed. The advantage of using the lemma, and of turning aside from (a part of) the computation trace, is that the variable $\underline{v}_{11}$ is left free.

More precisely, whenever a term such as $e(t_1, t_2)$ is found during the forced computation, and e is a reflexive predicate, the reflexivity lemma $e(x, x)$ is used in place of the corresponding (sub)computation trace, provided that t_1 and t_2 are identical, and that they are sub–terms of the given input symbolic term. In such a case, a new free variable is introduced for both t_1 and t_2.

In the given example, the forced computation proceeds with the evaluation of the EQL term. By instantiation of $\underline{v}_{12}$ to NIL, the evaluating term becomes the following [1]

$$\begin{aligned}&\mathrm{EQL}(\mathrm{APP}(\underline{v}_2, \mathrm{REV}(\underline{v}_3)),\\ &\quad \mathrm{APP}(\underline{v}_2, \mathrm{REV}(\underline{v}_3))).\end{aligned}$$

The reflexivity lemma is not applied because the second APP term is not part of the input symbolic term but it comes from the computation of the term $\mathrm{APP}(\underline{v}_1, \underline{v}_2)$. On the contrary, after a few steps the variable $\underline{v}_2$ ripples out and the evaluating term becomes

$$\begin{aligned}&\mathrm{EQL}(\mathrm{REV}(\underline{v}_3),\\ &\quad \mathrm{REV}(\underline{v}_3)).\end{aligned}$$

Both $\mathrm{REV}(\underline{v}_3)$ terms satisfy the conditions above, and can be generalized. Finally, collecting all instantiations, the induced theorem is the following

$$\begin{aligned}&\mathrm{EQL}(\mathrm{APP}(\mathrm{CONS}(v_{11}, \mathrm{NIL}),\\ &\qquad\quad \mathrm{APP}(\mathrm{CONS}(v_{21}, \mathrm{NIL}),\\ &\qquad\qquad v_4)),\\ &\quad \mathrm{APP}(\mathrm{APP}(\mathrm{CONS}(v_{11}, \mathrm{NIL}),\\ &\qquad\qquad \mathrm{CONS}(v_{21}, \mathrm{NIL})),\\ &\qquad v_4)).\end{aligned} \tag{4}$$

The given computation trace is no more a proof of the induced theorem. Yet, a proof can easily be obtained by a simplification inference rule, and by use of reflexivity lemmas. Hence, the induced formula is actually a theorem, even if a complete proof of it is not carried out.

Although the formula induced at this point is valid, it is a poor generalization, because the structure of the given data terms may still over–specialize the theorem. We would like for example generalize the CONS terms in (4), and obtain the following associativity theorem for APP

$$\begin{aligned}&\mathrm{EQL}(\mathrm{APP}(x, \mathrm{APP}(y, z)),\\ &\quad \mathrm{APP}(\mathrm{APP}(x, y), z))).\end{aligned}$$

[1] For brevity sake, we forget about the outermost AND application.

6. Generalization based on stretching the computation length

The data structures still present in the theorem mirror (part of) the structure of the given data terms and only those parts of the computation trace which are concerned with reflexive function evaluations have been generalized. A further generalization is now possible.

The first task is to select in the theorem data terms which are so general that they appear as "skeletons" for their type in the assumption that such skeletons could be further generalized. A *skeleton* of type T is a data term which satisfies the following conditions (p1):

– All constructors of type T occur in it;

– In each argument positions of type $T' \neq T$ a variable of type T' must occur.

Examples of skeletons are S(ZERO) and CONS(v_1, CONS(v_2, NIL)), while the following data terms do not classify as skeletons: TRUE, because it does not contain the constructor FALSE, and CONS(S(ZERO), CONS(v_1, NIL)), because S(ZERO) is not a variable of type NAT.

The focussing on skeletons embodies a peculiar notion of computation. In fact, the "structure" of data terms and the "structure" of computations are strongly interrelated, and in so far as skeletons are representatives of type structures, the given computation trace is a representative of all the possible computation traces. The hierarchical definition of types is believed to induce a corresponding nesting in the computation, hence to require hierarchical generalizations, as we will see later.

However, two different skeletons may be related to each other in such a way that their generalization to two different variables would not be valid. Discovering relevant relationships is a very difficult task, and we adopt just a naive notion of relationship, i.e. two skeletons are related if and only if they share a universally quantified variable.

After the above check (condition (p2)), good candidates for generalization are defined to be the maximal unrelated skeletons. But this alone is certainly a weak basis for generalizing the candidate. In fact, the given theorem may hold only if the actual data terms obey exactly the structure described by the corresponding skeletons. A classical example is the following

```
EQN(PLUS(S(ZERO), S(S(ZERO))),
      TRIPLE(S(ZERO))).
```

where TRIPLE is defined as

```
(TYPE(TRIPLE(NAT)) = NAT;
¦TRIPLE(x) = PLUS(TWICE(x), x)¦)

(TYPE(TWICE(NAT)) = NAT;
¦TWICE(x) = PLUS(x, x)¦).
```

Further generalization should then be based on a careful analysis of the given theorem and of the involved functions definitions. These can help understanding which are the effects of skeleton modifications, and a candidate skeleton can be generalized on the grounds that candidate skeleton modifications do not appear to bring about radical computation structure changes.

The starting point for such an analysis are predicates whose actual terms contain the candidate skeleton. Predicates implicitly define a measure on the argument values, and the difference between the argument measures determines which base equation is applied to terminate the computation. We say that a term depends on a skeleton if and only if it consists either in the skeleton itself, or in a function application term one argument of which depends on the skeleton. If the actual argument measures depend at different degree on the measure of the candidate skeleton, a skeleton modification may cause a different base equation to be used to terminate the (modified) computation. To be on the safe side, the candidate is discarded.

For each function, an arithmetical expression (called the *norm* of the function) is computed which, roughly speaking, expresses the measure of the function value in terms of the measure of its arguments [1].

Since we want to capture the effect of the modification of a specific skeleton, the partial derivatives of the auxiliary function norms w.r.t. the argument position under generalization are computed, and the generalization is accepted only if the derivatives can be simplified to the same expression. Let us consider the candidate S(ZERO) in the example above. The norm of PLUS is computed as $p_1 + p_2$ [2], and the norm of TRIPLE is $3 * p_1$. The partial derivative of $p_1 + p_2$ w.r.t. p_1 is 1, while the derivative of $3 * p_1$ w.r.t. p_1 is 3, thus the candidate S(ZERO) is not generalized.

The algorithm to associate a norm to a TTEL function is the following

i) The norm of a constructor is $1 + p_1$ if the constructor is reflexive;

0 otherwise;

ii) The norm of a U–function is Norm(f) = p_i;

iii) The norm of a S–function is
Norm(s) [Norm(h_1), . . . , Norm(h_m)/p_1, . . . , p_m];

iv) The norm of a N–function is
Norm(n) = p_1 + Norm(ϕ)[p_2, . . . , p_n/p_1, . . . , p_{n-1}];

1 Function composition in the actual argument terms of the predicates can be accounted by introducing two suitable auxiliary functions (defined by composition) such that the predicate argument terms can be rewritten as applications of the auxiliary functions to data terms only.

2 The variable simbol p_i occurring in norm expressions stands for the measure of the i-th argument term.

v) The norm of a R–function is

$$\mathrm{Norm}(r) = \mathrm{Norm}(\psi)\,[p_2, \dots, p_n \,/\, p_1, \dots, p_{n-1}] + p_1 * \mathrm{Norm}(\psi_R)\,[p_2, \dots, p_n \,/\, p_1, \dots, p_{n-1}].$$

Norm expressions are then reduced to a sum–of–products normal form.

Besides the conditions on the derivatives of the actual predicate argument terms, the predicate computation trace is tested for the following conditions. The first condition (s1) requires the forced computation to terminate using a base equation of the predicate. The second condition (s2) requires that, when a base equation is eventually applied, the whole skeleton does no more appear in the actual predicate argument terms. The third condition (s3) requires the candidate skeleton be substituted by a skeleton with a different measure, and the predicate term be re–evaluated yielding the same result.

In the next Section we will prove that, under the conditions p1, p2 and s1–s3, the generalizations performed are safe and the induced formula is actually a theorem.

7. Proof of the inductive method correctness

In order to let the proof be more readable, we will prove, without loss of generality, the correctness of the inductive method in the assumption that the arity of the involved functions is at most 2.

Let us introduce a different notation which uses the notion of sequence.
A *sequence* X is denoted as

$$\langle x_1, x_2, \dots, x_n \rangle$$

Let d and T be the reflexive and non reflexive constructors of a given type. The *d–generated* data structure

$$d(x_1, d(x_2, \dots, d(x_n, T) \dots)$$

will be also denoted as

$d[\langle x_1, x_2, \dots, x_n \rangle]$ and

$T = d[\langle\rangle]$

We define an operation from d–generated data structure to the sequence of its component elements

$$|\, d[\langle x_1, x_2, \dots, x_n \rangle] \,| = \langle x_1, x_2, \dots, x_n \rangle$$

Let be $X = \langle x_1, x_2, \dots, x_n \rangle$, $Y = \langle y_1, y_2, \dots, y_m \rangle$ then let:

– $X^i = x_i$ (*i* – th *component*);

– $x_i \in X \quad 1 \leqslant i \leqslant n$ (*component of* relation);

– $|X| = n$ (*length* of X);

– $\overline{X} = \langle x_n, x_{n-1}, \ldots, x_1 \rangle$ (*inverse* of X);

– $X ; Y = \langle x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m \rangle$ (*concatenation* of X, Y).

The component–wise application of a function γ is defined as

$$\gamma^\circ X = \langle \gamma x_1, \gamma x_2, \ldots, \gamma x_n \rangle$$

Note that we omit parenthesis around monadic terms whenever there is no ambiguity. Let ϕ be a linear map over a data structure, i.e.

$\phi\, d[X]$ is either

$d'[\rho^\circ X]$ or $d'[\rho^\circ \overline{X}]$,

then the linear application of ϕ to X is defined as

$\phi \square X = \rho^\circ X$ or

$\phi \square X = \rho^\circ \overline{X}$

The notation above allows us to compactly express the evaluation of an N– or R–function.

Lemma N. *The term*

$$N(d_1[X], d_2[Y]) \tag{LN.1}$$

where N *obeys the N–schema can be rewritten as*

$$d_3[\gamma^\circ X; \phi \square Y] \tag{LN.2}$$

Proof. Let $X = \langle x_1, x_2, \ldots, x_n \rangle$, by the application of the Nr recursive equation we obtain from (LN.1)

$$d_3(\gamma x_1, N(d_1[\langle x_2, \ldots, x_n \rangle], d_2[Y]))$$

By $(n-1)$ applications of the Nr equation

$$d_3[\gamma^\circ X; \nmid N(T_1, d_2[Y]) \nmid]$$

By Nb equation

$$d_3[\gamma^\circ X; \nmid \phi(d_2[Y]) \nmid]$$

Since ϕ is a linear map we obtain

$$d_3[\gamma^\circ X; \phi \square \nmid d_2[Y] \nmid]$$

and finally we get (LN.2).

Q.E.D.

Lemma R. *The term*

$$R(d_1[X], d_2[Y]) \tag{LR.1}$$

where R *obeys the R–schema can be rewritten as*

$$d_3[\psi \square Y; (\rho_R\, \delta)^\circ \overline{X}] \tag{LR.2}$$

Proof. Let $|X| = n$. By n applications of the Rr recursive equation (LR.1) can be rewritten as (superscripts keep track of the function applications)

$$N_R^1(\ldots(N_R^{n-1}(N_R^n(\psi d_2[Y], d_4(\delta x_n, T_4)), d_4(\delta x_{n-1}, T_4)), \ldots, d_4(\delta x_1, T_4))$$

Since ψ is a linear map we have

$$N_R^1(\ldots(N_R^{n-1}(N_R^n(d_3[\psi \Box Y], d_4(\delta x_n, T_4)), d_4(\delta x_{n-1}, T_4)), \ldots, d_4(\delta x_1, T_4))$$

Let $|Y| = k$. By $k * n$ application of Nr equation

$$d_3[\psi \Box Y; \mid N_R^1(\ldots(N_R^{n-1}(N_R^n(T_3, d_4(\delta x_n, T_4)), d_4(\delta x_{n-1}, T_4)), \ldots, d_4(\delta x_1, T_4)) \mid]$$

Applying Nb equation

$$d_3[\psi \Box Y; \mid N_R^1(\ldots(N_R^{n-1}(\phi_R(d_4(\delta x_n, T_4)), d_4(\delta x_{n-1})T_4)), \ldots, d_4(\delta x_1, T_4)) \mid]$$

Since ϕ_R is a linear map

$$d_3[\psi \Box Y; \mid N_R^1(\ldots(N_R^{n-1}(d_3(\rho_R\ \ x_n, T_3), d_4(\delta x_{n-1}, T_4)), \ldots, d_4(\delta x_1, T_4)) \mid]$$

By evaluating the $(n-1)$ N_R function applications

$$d_3[\psi \Box Y; \mid d_3(\rho_R \delta x_n, N_R^1(\ldots(N_R^{n-1}(T_3, d_4(\delta x_{n-1}, T_4)), \ldots, d_4(\delta x_1, T_4))) \mid]$$

By repetition of the three steps above $(n-1)$ times we have

$$d_3[\psi \Box Y; \mid d_3[\langle \rho_R \delta x_n, \rho_R \delta x_{n-1}, \ldots, \rho_R \delta x_1 \rangle] \mid]$$

According to the notation introduced above, we obtain (LR.2).

Q.E.D.

Lemma E. *The term*

$$E(d_1[X], d_2[Y])$$

where E *obeys an S–schema, i.e.*

$$E(x, y) = F(P(x, y), Q(x, y))$$

can be rewritten as

$$d[H_1; \phi_1 \Box X; H_2; \phi_2 \Box X; \ldots; \phi_n \Box X, H_{n+1}] \tag{E.1}$$

where

- $\phi_i \Box X$, $1 \leqslant i \leqslant n$, *denote linear mappings of* X;
- H_i, $1 \leqslant i \leqslant n$, *denote (possibly empty) sequences which do not contain linear mappings of* X.

Proof. Let us first assume that F, P, and Q are not defined according an S–schema. Then eight cases may occur depending on the definition schema the three involved functions obey. Let us consider only one of them, because the remaining ones are analogous. Suppose

$$P(d_1[X], d_2[Y]) = d_3[\gamma^\circ X; \phi\Box Y],$$

$$Q(d_1[X], d_2[Y]) = d_4[\psi\Box Y; (\rho_R\, \delta)^\circ \overline{X}],$$

and F be an N–function.
Then, by Lemma N on F we have

$$E(d_3[\gamma^\circ X; \phi\Box Y], d_4[\psi\Box Y; (\rho_R\, \delta)^\circ \overline{X}]) = d[\gamma_F{}^\circ(\gamma^\circ X; \phi_F\Box(\psi\Box Y; (\rho_R\, \delta)^\circ \overline{X})].$$

By $^\circ$–distributing γ_F we obtain

$$d[(\gamma_F\gamma)^\circ X; \gamma_F{}^\circ \phi\Box Y; \phi_F\, \Box(\psi\Box Y; (\rho_R\, \delta)^\circ \overline{X})]$$

By $\Box$–distributing ϕ_F, two cases may occur depending on whether ϕ_F inverts its argument sequence. If ϕ_F does not invert its argument, we obtain

$$d[(\gamma_F\gamma)^\circ X; \gamma_F{}^\circ \phi\Box Y; \rho_{\phi_F}{}^\circ \psi\Box Y; (\rho_{\phi_F}\rho_R\, \delta)^\circ \overline{X})] \qquad \text{(E.2)}$$

and (E.1) is proven.
Conversely, if ϕ_F inverts the argument, we have

$$d[(\gamma_F\gamma)^\circ X; \gamma_F{}^\circ \phi\Box Y; (\rho_{\phi_F}\, \rho_R\, \delta)^\circ X; \rho_{\phi_F}{}^\circ \psi\Box \overline{Y}] \qquad \text{(E.3)}$$

and again (E.1) is proven.

The reader can see that (E.1) is proven because γ_F can be $^\circ$–distributed and ϕ_F can be $\Box$–distributed, so that if a linear mapping of X occurs in the argument sequences of function E, a linear mapping of X occurs in the resulting sequence too. In other words, the number of linear mappings of X is preserved, and the same is true for the remaining seven cases, as we mentioned above.

The number of linear mappings of X occurring in the sequence (E.3) is exactly 2 because F, P and Q were, by hypothesis, defined by N– or R–schema only and the formal argument x occurred twice in the right–hand side of the definition of E. If P and Q are allowed to be defined by a U–schema the number of linear mappings of X occurring in the resulting sequence may decrease but (E.1) remains (possibly vacuously) true. If F, P and Q are defined by composition, we can simply substitute them by their definitions, until only functions defined by N–, R– or U–schema appear in the definition of E. Still, the resulting sequence will obey a schema like (E.1) because of the distribution property of component–wise and linear applications.

Q.E.D.

Let $ET(d_x[X], d_1[Y_1], \ldots, d_y[Y_y])$ denote any term built on the function symbols $F_1, \ldots, F_f$ and the data structures $d_x[X], d_1[Y_1], \ldots, d_y[Y_y]$. Then by Lemma E the term can be rewritten as

$$d[H_1 ; \phi_1 \square \underline{Y}; H_2; \ldots ; \phi_n \square X; H_{n+1}]$$

where

– H_i, $1 \leqslant i \leqslant n+1$, are (possibly empty) $F_1 -, \ldots, F_f$–mappings of sequences $Y_1, \ldots, Y_y$, but not of X;

– $\phi_i \square X$, $1 \leqslant i \leqslant n$, denote the $F_1 -, \ldots, F_f$–linear mappings of X.

Now assume that symbolic execution has proven

$$\begin{aligned}&B(ET1(d_x[X], d_{11}[Y_{11}], \ldots, d_{1y_1}[Y_{1y_1}]),\\ &\quad ET2(d_x[X], d_{21}[Y_{21}], \ldots, d_{2y_2}[Y_{2y_2}])) = \text{BCONST1}\end{aligned} \tag{T.1}$$

and the algorithm surmises

$$\begin{aligned}&B(ET1(x, d_{11}[Y_{11}], \ldots, d_{1y_1}[Y_{1y_1}]),\\ &\quad ET2(x, d_{21}[Y_{21}], \ldots, d_{2y_2}[Y_{2y_2}])) = \text{BCONST1}\end{aligned} \tag{T.2}$$

where x is a universally quantified variable of the type generated by d_x. Before attempting to prove that (T.2) holds we need a few definitions and a preparatory Lemma. In fact, by Lemma E (T.1) can be rewritten as

$$B(d_1[V], d_2[Z]) = \text{BCONST1}$$

where

$$V = H_1 ; \phi_1 \square X; \ldots ; \phi_n \square X; H_{n+1} \quad \text{and}$$

$$Z = K_1 ; \psi_1 \square X; \ldots ; \psi_n \square X; K_{n+1}.$$

Note that V and Z contain the same number of mapped X's because the partial derivatives of the norms of the predicate argument terms are equal.

Problems will arise in the proof depending on the "relative position" of V's and Z's subsequences. Let us make precise such notions. Consider the following sequences of indexes (non–negative integers)

$$\begin{aligned}L_V &= \langle 0, |H_1|, |H_1| + |\phi_1 \square X|, \ldots, \sum_{i=1}^{n} (|H_i| + |\phi_i X| + |H_{n+1}|)\rangle \\ &= \langle 0, |H_1|, |H_1| + |X|, \ldots, n \cdot |X| + | \sum_{i=1}^{n} |H_i| + |H_{n+1}|\rangle\end{aligned}$$

and

$$L_Z = \langle 0, |K_1|, |K_1| + |X|, \ldots, n \cdot |X| + \sum_{i=1}^{n} |K_i| + |K_{n+1}|\rangle$$

Let us define *cut index* any index c such that

i) $c = L^p_V = L^q_Z$;

ii) $(p-2) \div 2 = (q-2) \div 2$, i.e. the sequences $\langle V^1, \ldots, V^c \rangle$ and $\langle Z^1, \ldots, Z^c \rangle$ contain the same number of mapped X's.

If I cut indexes exist for two given sequences V and Z, the following $2 * I$ sub–sequences called *intervals* can be defined

$$V_i = \langle V^{c_i+1}, \ldots, V^{c_{i+1}} \rangle \; 1 \leqslant i \leqslant I-1$$

$$V_I = \langle V^{c_I+1}, \ldots, V^{\min(|V|, |Z|)} \rangle$$

$$Z_i = \langle Z^{c_i+1}, \ldots, Z^{c_{i+1}} \rangle \; 1 \leqslant i \leqslant I-1$$

$$Z_I = \langle Z^{c_I+1}, \ldots, Z^{\min(|V|, |Z|)} \rangle$$

Let us define *singular index* any index s such that

i) $s = L^p_V = L^q_Z$;

ii) $(p-2) \div 2 \neq (q-2) \div 2$, i.e. the sequences $\langle V^1, \ldots, V^s \rangle$ and $\langle Z^1, \ldots, Z^s \rangle$ contain a different number of mapped X's.

An interval V_i is *singular* iff at least one singular index s exists such that

$$c_i + 1 < s < c_i + |V_i|$$

i.e. a singular index falls in the interval.
Note that if V_i is singular Z_i is singular too.
If V_i and Z_i are singular intervals where k_i singular indexes occur, the following $2 * (k_i + 1)$ singular sub–intervals of V_i, Z_i can be defined

$$V_{i1} = \langle V_i^1, \ldots, V_i^{s_1 - c_i} \rangle$$

$$V_{ij} = \langle V_i^{s_{j-1} - c_i + 1}, \ldots, V_i^{s_j - c_i} \rangle \quad 2 \leqslant j \leqslant k_i$$

$$V_{i(k_i+1)} = \langle V_i^{s_{k_i} - c_i + 1}, \ldots, V_i^{|V_i|} \rangle$$

$$Z_{i1} = \langle Z_i^1, \ldots, Z_i^{s_1 - c_i} \rangle$$

$$Z_{ij} = \langle Z_i^{s_{j-1} - c_i + 1}, \ldots, Z_i^{s_j - c_i} \rangle \quad 2 \leqslant j \leqslant k_i$$

$$Z_{i(k+1)} = \langle Z_i^{s_{k_i} - c_i + 1}, \ldots, Z_i^{|Z_i|} \rangle$$

We can now prove the following.

Lemma S. *Let*

- $V = H_1; \phi_1 \square X; \dots ; H_n; \phi_n \square X; H_{n+1}$ *and*
  $Z = K_i, \psi_1 \square X; \dots ; K_n; \psi_n \square X; K_{n+1}$
  be two singular intervals;
- $|X| = n$;
- $s_1, \dots , s_k$ *be the singular indexes of* V *and* Z;
- V_i, Z_i $1 \leqslant i \leqslant k+1$ *be the singular sub–intervals of* V *and* Z;
- V'_i, Z'_i *be the sequences obtained by substituting* X' *for* X,
  $|X'| = m,\ m \neq n,\ m \neq 0$, *in* V_i *and* Z_i *respectively.*

Then

$$\sum_{i=1}^{j} |V'_i| \neq \sum_{i=1}^{j} |Z'_i| \quad 1 \leqslant j \leqslant k \tag{S.1}$$

i.e. V'_i and Z'_i are no more singular sub–intervals.

Proof. (S.1) can be rewritten as

$$\sum_{i=1}^{j} (v_i * m + h_i) \neq \sum_{i=1}^{j} (z_i * m + h_i) \tag{S.2}$$

where v_i and z_i denote the number of occurrences of (mapped) X' in V'_i and Z'_i respectively, and h_i and h'_i denote the total length of those subsequences of V'_i and Z'_i which do not involve X'. Note that by definition of singular index we have

$$\sum_{i=1}^{j} v_i \neq \sum_{i=1}^{j} z_i \quad 1 \leqslant j \leqslant k \tag{S.3}$$

and

$$v_i * n + h_i = z_i * n + h'_i \quad 1 \leqslant i \leqslant k+1 \tag{S.4}$$

Let $m = n + m'$ (S.2) can be rewritten as

$$\sum_{i=1}^{j} (v_i * n + h_i) + \sum_{i=1}^{j} v_j m' \neq \sum_{i=1}^{j} (z_i * n + h'_i) + \sum_{i=1}^{i} z_i m'$$

and by (S.4) as

$$m' * \sum_{i=1}^{j} v_i \neq m' * \sum_{i=1}^{j} z_i$$

which is proven by (S.3) and $m' \neq 0$. Q.E.D.

Note that if V' (Z') is obtained from $V(Z)$ by substituting X' for X (and $|X'| \neq |X|$) V' and Z' may still be singular intervals. Roughly speaking, Lemma S states that the new singular indexes, if any, define different singular sub–intervals.

We now have the tools to prove the main theorem.

Theorem. *Assume symbolic execution has proven*

$$B(ET1(d_x[X], d_{11}[Y_{11}], \dots, d_{1y_1}[Y_{1y_1}]),$$
$$ET2(d_x[X], d_{21}[Y_{21}], \dots, d_{2y_2}[Y_{2y_2}])) = \text{BCONST1} \tag{T.1}$$

and conditions (p1), (p2), (s1)–(s3) *hold. Assume the algorithm surmises*

$$B(ET1(x, d_{11}[Y_{11}], \dots, d_{1y_1}[Y_{1y_1}]),$$
$$ET2(x, d_{21}[Y_{21}], \dots, d_{2y_2}[Y_{2y_2}])) = \text{BCONST1} \tag{T.2}$$

where x is a universally quantified variable of the type generated by d_x, then (T.2) holds.

Proof. By Lemma E, (T.1) can be rewritten as

$$B(d_1[V], d_2[Z]) = \text{BCONST1} \tag{T.3}$$

where

$$V = H_1; \phi_1 \,\Box\, X; \dots; H_n; \phi_n \,\Box\, X; H_{n+1} \quad \text{and}$$

$$Z = K_1; \psi_1 \,\Box\, X; \dots; K_n; \psi_n \,\Box\, X; K_{n+1}.$$

Because of conditions (p1) and (p2), all the components of X are distinct universally quantified variables. Thus, in order to prove that (T.2) holds, it suffices to prove that (T.3) does not depend on the length of X. The proof is achieved by showing that

i) condition (s1), which holds for the computation of (T.3), holds for any length of X, and

ii) the same base equation for B is finally applied, for any length of X.

The proof of (ii) is very simple. In fact, if condition (s1) holds, the base equation for B which is finally applied is determined by the difference between $|V|$ and $|Z|$. The value of the difference is

$$d = |V| - |Z| = \sum_{i=1}^{n+1} |H_i| + n * |X| - \sum_{i=1}^{n+1} |K_i| - n * |X| = \sum_{i=1}^{n+1} |H_i| - \sum_{i=1}^{n+1} |K_i|$$

which obviously does not depend on $|X|$.

We now prove (i), i.e. that for any $|X|$ holds

$$B2(V^i, Z^i) = \text{BCONST2} \qquad 1 \leq i \leq \min(|V|, |Z|) \tag{T.4}$$

Let V_i, Z_i, $1 \leq i \leq I$, be the intervals of V and Z. By the definition of interval, if we substitute any X' for X in V, Z, V_i and Z_i, thus obtaining the sequences V', Z', V'_i, Z'_i, then V'_i and Z'_i are intervals of V' and Z'. This property of intervals allows to confine ourselves to prove (T.4) on intervals only, i.e. to prove that for any X holds

$$B2((V_i)^j, (Z_i)^j) = \text{BCONST2} \qquad 1 \leqslant j \leqslant |V_i| = |Z_i| \tag{T.5}$$

Note that such an interval–wise reduction of the proof is valid also for the intervals V_l and Z_l which are not bounded on the right by a cut index. However, by condition (s2) they still contain the same number of occurrences of mapped X's (even if one occurrence may be not complete).
For the sake of brevity, let us forget the subscript i in the sequel, and denote by V and Z any V_i and Z_i intervals. Two cases may arise, depending on whether V and Z are singular intervals.

Case 1. The intervals V and Z are not singular. By the definition of interval we know that neither cut indexes nor singular indexes may occur in the intervals. We will make use of this property only in the proof which follows (and thereby we will exploit the same argument also in Case 2).

Let us single out the sub–sequences, i.e.

$V = V_1 ; V_2 ; \ldots ; V_p$ and

$Z = Z_1 ; Z_2 ; \ldots ; Z_p$

where V_i (Z_i) is either a mapped X, or some H_j (K_j).
Let us consider the sequences of indexes

$$M_V = \langle 0, |V_1|, |V_1 ; V_2|, \ldots, |V| \rangle$$

$$M_Z = \langle 0, |Z_1|, |Z_1 ; Z_2|, \ldots, |Z| \rangle$$

We denote by M the sequence obtained by merging M_V and M_Z, keeping indexes in ascending order and eliminating the duplicate 0's and $|V|$ and $|Z|$. We can now define $2 * (|M| - 1)$ sub–intervals of V and Z, as follows

$$V_i^+ = \langle V^{m^{i-1}+1}, \ldots, V^{m^i} \rangle \qquad 2 \leqslant i \leqslant 2p - 1$$

$$Z_i^+ = \langle Z^{m^{i-1}+1}, \ldots, Z^{m^i} \rangle$$

The sequence of indexes M is defined in such a way that the following property holds for the induced sub–intervals. For any $1 \leqslant i \leqslant 2p - 2$, then either one, but not both of the following cases occur

i) $(V_i^+)^{|V_i^+|}, (V_{i+1}^+)^1 \in V_j$
and

$(Z_i^+)^{|Z_i^+|} \in Z_k$ and $(Z_{i+1}^+)^1 \in Z_{k+1}$

ii) $(V_i^+)^{|V_i^+|} \in V_j$ and $(V_{i+1}^+)^1 \in V_{j+1}$

and

$$(Z_i^+)^{|V_i^+|}, (Z_{i+1}^+)^1 \in Z_k \qquad \text{(T.6)}$$

Now let us consider the sub–intervals $V_k^+, V_{k+1}^+, Z_k^+, Z_{k+1}^+$, $1 \leqslant k \leqslant 2p-2$. By (T.5) we know that

$$B2((V_k^+)^i, (Z_k^+)^i) = \text{BCONST2} \qquad 1 \leqslant i \leqslant |V_k^+| \qquad \text{(T.7)}$$

$$B2((V_{k+1}^+)^i, (Z_{k+1}^+)^i) = \text{BCONST2} \qquad 1 \leqslant i \leqslant |V_{k+1}^+|$$

By conditions (p1) and (p2) we can extend (T.7) to

$$B2((V_k^+)^i, (Z_k^+)^j) = \text{BCONST2} \qquad 1 \leqslant i,j \leqslant |V_k^+| \qquad \text{(T.8)}$$

$$B2((V_{k+1}^+)^i, (Z_{k+1}^+)^j) = \text{BCONST2} \qquad 1 \leqslant i,j \leqslant |V_{k+1}^+|$$

Since property (T.6) holds let us suppose, without loss of generality, that $(V_k^+), (V_{k+1}^+)^1 \in V_g$ (Case (ii) is just symmetric) then by (p1) and (p2) we have

$$B2((V_k^+; V_{k+1}^+)^i, (Z_k^+)^j) = \text{BCONST2} \qquad \begin{array}{l} 1 \leqslant i \leqslant |V_k^+ ; V_{k+1}^+| \\ 1 \leqslant j \leqslant |V_k^+| \end{array}$$

$$B2((V_k^+, V_{k+1}^+)^i, (Z_{k+1}^+)^j) = \text{BCONST2} \qquad \begin{array}{l} 1 \leqslant i \leqslant |V_k^+ ; V_{k+1}^+| \\ 1 \leqslant j \leqslant |V_{k+1}^+| \end{array}$$

Then we can conclude

$$B2((V_k^+ ; V_{k+1}^+)^i, (Z_k^+; Z_{k+1}^+)^j) = \text{BCONST2}, \; 1 \leqslant i,j \leqslant |V_k^+; V_{k+1}^+| \qquad \text{(T.9)}$$

We have just shown that the sequences $(V_k^+; V_{k+1}^+)$ and $(Z_k^+; Z_{k+1}^+)$ can be considered as sub–intervals, because (T.9) corresponds to (T.8) and property (T.6) holds. By repetition of the same argument $(2p-1)$ times we can conclude that

$$B2((V)^i, (Z)^j) = \text{BCONST2} \qquad 1 \leqslant i,j \leqslant |V| \qquad \text{(T.10)}$$

By (p1) and (p2), (T.10) holds for any X, and since (T.5) is a specialization of (T.10), Case 1 is proven.

Case 2. The intervals V and Z are singular, i.e.

$V = V_1 ; V_2 ; \ldots ; V_k$ and

$Z = Z_1 ; Z_2 ; \ldots ; Z_k$

where V_i and Z_i are the singular sub–intervals induced by the $(k-1)$ singular indexes. Since neither cut indexes nor singular indexes may occur in the sub–intervals, by the same argument used for proving (T.8), we know

$$B2((V_i)^j, (Z_i)^h) = \text{BCONST2} \qquad 1 \leqslant j, h \leqslant |V_i| \tag{T.11}$$

In order to prove the theorem, we need to show that it is possible to "merge" the singular sub–intervals. We resort to condition (s3) and exploit the fact that the sub–intervals are bounded by singular indexes.

Let us denote by V', Z', V'_i, Z'_i the sequences obtained by substituting X' for $X (0 \neq |X'| \neq |X|)$ in V, Z, V_i, Z_i respectively. Then, by condition (s3) we have

$$B2((V')^i, (Z')^i) = \text{BCONST2} \qquad 1 \leqslant i \leqslant |V'|.$$

Let us now consider the sequences V_1, Z_1, V_2, Z_2. By Lemma S we have $|V'_1| \neq |Z'_1|$. Assume $r = |V'_1| > |Z'_1| = s$ (the case $|V'_1| < |Z'_1|$ is just analogous). Since V'_2 and Z'_2 are skeletons we have $|V'_2| \neq 0$ and $|Z'_2| \neq 0$. Thus, by symbolic evaluation we know

$$B2((V'_1)^i, (Z'_2)^{i-s}) = \text{BCONST2} \qquad s + 1 \leqslant i \leqslant r$$

and by conditions (p1) and (p2)

$$B2((V'_1)^i, Z'_2)^j) = \text{BCONST2} \qquad \begin{array}{l} 1 \leqslant i \leqslant r \\ 1 \leqslant j \leqslant |Z'_2| \end{array}$$

which by condition (p1) is not affected by X'. Therefore

$$B2((V_1)^i, (Z_2)^j) = \text{BCONST2} \qquad \begin{array}{l} i = 1, \ldots, |V_1| \\ j = 1, \ldots, |Z_1| \end{array} \tag{T.12}$$

By (T.11) and (T.12) we have

$$B2((V_1 ; V_2)^i, (Z_1 ; Z_2)^j) = \text{BCONST2} \qquad 1 \leqslant i, j \leqslant |V_1 ; V_2|$$

We have thus proven that the two pairs of singular sub–intervals can be merged and B2 relation extended to the merge. By repeating the same merging $(k - 1)$ times we extend B2 relation to the whole singular interval and the proof of Case 2 is complete.

Q.E.D.

The main theorem shows that the performed skeleton generalizations preserve the values of the predicates involved in the given theorem. Thus the induced formula is actually a theorem.

8. Comparison with related work

The inductive method we have presented has been influenced in many respects by previous research on induction. First of all, the generalization based on proof can be classified as a successive refinement method. We do not need to iterate the process of guessing and refining, because the formal calculus environment provides for a powerful technique (forced symbolic computation) to exactly adjust the initial guessing. Analogously, the idea of using the given computation trace as a proof to strengthen the theorem, relates to Brown's and Tarnlund' temporal method based on proofs [4]. By the way, let us note that the problem of estimating function behaviour is exactly the problem of solving (restricted) difference equations (over the naturals) tackled by Brown and Tarnlund.

Moreover, we have had the advantage of being able to draw on the ideas of Boyer and Moore [2] and of Aubin [1], which tackle the problem of generalizing the theorem to be proven by induction. Such a generalization is necessary when a backward search strategy is adopted. One basic problem is that of distinguishing different occurrences of a variable. Boyer and Moore generalize terms which the involved functions recur on, thus relating generalization to function definition. In their footsteps, Aubin points out the close relationship among generalization, proof by induction and symbolic computation. His method generalizes precisely those variables which an interpreter would first instantiate during symbolic computation. Thus, only the first and fourth occurrences of x are generalized in the theorem EQL(APP(x, APP(x, x)), APP(APP(x, x), x)). Along the same line, we bring the whole computation structure to bear on the problem, and we can capture rather complex relationship, as shown in the worked example in the Appendix. Because of the peculiar role played by the LREV function, Aubin's method would incorrectly generalize the first and third occurence of x in the theorem EQSTRUCT(LREV(LAPP(x, x)),LAPP(LREV(x), LREV(x))).

As final remark, we might mention that a heuristic inductive system based on the method presented here has been actually developed as a tool to assist the user to debug his TTEL functions, when they are used as formal program specifications. Running formal specifications is of little help to understand whether they do express the user intents. Instead, surmising function properties from testing computation, may effectively assist the user in matching intentions to specifications.

9. Concluding remarks

We have tackled the problem of proving function properties by inductive generalization from examples. The method presented here is proven sound for a constructively defined class of recursive functions and properties. In spite of the limitations, significant functions, properties and theorems fall into the class. It is certainly not possible to express significance by a figure, but it may be interesting to note that about 35% of the theorems listed in [1] and involving single reflexive data types are induced by the present method and belong to the above class. Classical examples are EQL(APP(x, APP(y, z)), APP(APP(x, y), z)), EQL(REV(REV(x)), x), and EQL(REV(APP(x, y)), APP(REV(y), REV(x))). The reflexivity theorem for EQN, EQL, EQLENGTH, and EQSTRUCT fall into the class and are induced applying the method.

At the present, the application of such inductive method to theorem proving has one advantage but suffers from a few limitations. The advantage is that no combinatorial explosion arises in the proof. Free variable introduction is indeed a non–deterministic process, but no nesting of non–deterministic choices is involved. On the contrary, the major limitation of the proposed method is that it is not proven complete. Thus, it can be used only as an auxiliary tool, which may result in a failure, but requires at least a bounded amount of resource. Moreover, the method can prove only function properties whose restrictions have been described above. This is a heavy limitation for a general purpose theorem prover.

In the framework of program verification where generally the goal is that of proving properties of functions, it is more likely that the inductive method can help, provided that the theorem to be proven (or a subgoal generated during the proof) is within its reach. We believe that function property induction can in the future play a role in program verification systems.

Appendix

Let us consider the following ground property

```
EQSTRUCT(LREV(LAPP(LCONS(CONS(PLUS(S(ZERO),ZERO),                          (EX 1)
                              CONS(S(ZERO), NIL)),
                         LNIL),
                    LCONS(CONS(PLUS(S(ZERO), ZERO),
                              CONS(S(ZERO), NIL)),
                         LNIL))),
           LAPP(LREV(LCONS(CONS(PLUS(S(ZERO), ZERO),
                               CONS(S(ZERO), NIL)),
                         LNIL)),
                LREV(LCONS(CONS(PLUS(S(ZERO), ZERO),
                               CONS(S(ZERO), NIL)),
                         LNIL)))).
```

where EQSTRUCT, LAPP, and LREV are defined as follows

```
(TYPE(EQSTRUCT(LLIST,LLIST))=BOOL; { EQSTRUCT(LNIL,LNIL)= TRUE;
 EQSTRUCT(LCONS(x, y), LNIL) = FALSE;  EQSTRUCT(LNIL, LCONS(z, w)) = FALSE;
 EQSTRUCT(LCONS(x, y), LCONS(z, w)) = AND(EQLENGTH(x, z), EQSTRUCT(y, w)) })

(TYPE(LAPP(LLIST, LLIST)) = LLIST;
{LAPP(LNIL, z) = z;  LAPP(LCONS(x, y), z) = LCONS(x, LAPP(y, z)) })

(TYPE(LREV(LLIST)) = LLIST;
{ LREV(LNIL) = LNIL;  LREV(LCONS(x, y)) = LAPP(LREV(y), LCONS(x, LNIL)) }
```

First of all, the term PLUS(S(ZERO), ZERO) is never evaluated, and a free variable nx is substituted for it, thus obtaining

```
EQSTRUCT(LREV(LAPP(LCONS(CONS(nx, CONS(S(ZERO), NIL)),                     (EX 2)
                         LNIL),
                    LCONS(CONS(nx, CONS(S(ZERO), NIL)),
                         LNIL))),
           LAPP(LREV(LCONS(CONS(nx, CONS(S(ZERO), NIL)),
                         LNIL)),
                LREV(LCONS(CONS(nx, CONS(S(ZERO), NIL)),
                         LNIL)))).
```

Secondly, free variables nllx and nlly are introduced to perform forced computation. Free variable introduction yields two symbolic terms, i.e.

$$\begin{array}{l}\text{EQSTRUCT}(\text{LREV}(\text{LAPP}(\underline{\text{nllx}}, \underline{\text{nlly}})),\\ \qquad\qquad \text{LAPP}(\text{LREV}(\underline{\text{nlly}}),\\ \qquad\qquad\qquad \text{LREV}(\underline{\text{nllx}})))\end{array} \qquad \text{(EX 3)}$$

and

$$\begin{array}{l}\text{EQSTRUCT}(\text{LREV}(\text{LAPP}(\underline{\text{nllx}}, \underline{\text{nlly}})),\\ \qquad\qquad \text{LAPP}(\text{LREV}(\underline{\text{nllx}}),\\ \qquad\qquad\qquad \text{LREV}(\underline{\text{nlly}}))).\end{array} \qquad \text{(EX 4)}$$

Symbolic computation is forced on both terms. Let us consider the forced computation of (EX 3), which yields (collecting all free variable instantiations)

$$\begin{array}{l}\text{EQSTRUCT}(\text{LREV}(\text{LAPP}(\text{LCONS}(\text{CONS}(\underline{\text{nx}}_1, \text{CONS}(\underline{\text{nx}}_2, \text{NIL})),\\ \qquad\qquad\qquad\qquad\qquad\qquad \text{LNIL}),\\ \qquad\qquad\qquad\qquad\quad \text{LCONS}(\text{CONS}(\underline{\text{ny}}_1, \text{CONS}(\underline{\text{ny}}_2, \text{NIL})),\\ \qquad\qquad\qquad\qquad\qquad\qquad \text{LNIL}))),\\ \qquad\qquad \text{LAPP}(\text{LREV}(\text{LCONS}(\text{CONS}(\underline{\text{ny}}_1, \text{CONS}(\underline{\text{ny}}_2, \text{NIL})),\\ \qquad\qquad\qquad\qquad\qquad\qquad \text{LNIL})),\\ \qquad\qquad\qquad \text{LREV}(\text{LCONS}(\text{CONS}(\underline{\text{nx}}_1, \text{CONS}(\underline{\text{nx}}_2, \text{NIL})),\\ \qquad\qquad\qquad\qquad\qquad\qquad \text{LNIL}))))\end{array} \qquad \text{(EX 5)}$$

The type LLIST terms are due to the applications of LAPP equations, while the type LIST terms come from the applications of EQLENGTH equations. Note that NAT variables appear because EQLENGTH checks for LIST length equality only. Analogously, from (EX 4) we obtain

$$\begin{array}{l}\text{EQSTRUCT}(\text{LREV}(\text{LAPP}(\text{LCONS}(\text{CONS}(\underline{\text{nx}}_1, \text{CONS}(\underline{\text{nx}}_2, \text{NIL})),\\ \qquad\qquad\qquad\qquad\qquad\qquad \text{LNIL}),\\ \qquad\qquad\qquad\qquad\quad \text{LCONS}(\text{CONS}(\underline{\text{ny}}_1, \text{CONS}(\underline{\text{ny}}_2, \text{NIL})),\\ \qquad\qquad\qquad\qquad\qquad\qquad \text{LNIL}))),\\ \qquad\qquad \text{LAPP}(\text{LREV}(\text{LCONS}(\text{CONS}(\underline{\text{nx}}_1, \text{CONS}(\underline{\text{nx}}_2, \text{NIL})),\\ \qquad\qquad\qquad\qquad\qquad\qquad \text{LNIL})),\\ \qquad\qquad\qquad \text{LREV}(\text{LCONS}(\text{CONS}(\underline{\text{ny}}_1, \text{CONS}(\underline{\text{ny}}_2, \text{NIL})),\\ \qquad\qquad\qquad\qquad\qquad\qquad \text{LNIL}))))\end{array} \qquad \text{(EX 6)}$$

Now, consider the generalization method based on equivalence applied to (EX 5). The predicate EQLENGTH used in the computation is reflexive, and the computation trace shows that the term

$$\begin{array}{l}\text{EQLENGTH}(\text{CONS}(\underline{\text{ny}}_1, \text{CONS}(\underline{\text{ny}}_2, \text{NIL})),\\ \qquad\qquad \text{CONS}(\underline{\text{ny}}_1, \text{CONS}(\underline{\text{ny}}_2, \text{NIL})))\end{array}$$

was evaluated, yielding TRUE (note the role of LREV). Since the EQLENGTH argument terms are identical and do occur in (EX 5), they are generalized to a new variable $\underline{\text{nly}}$. The same situation occurs for the data term CONS(nx_1, CONS(nx_2, NIL)) which is generalized to $\underline{\text{nlx}}$. The following theorem is thus induced

$$\begin{aligned}&\text{EQSTRUCT(LREV(LAPP(LCONS(}\underline{\text{nlx}}\text{, LNIL),}\\ &\qquad\qquad\text{LCONS(}\underline{\text{nly}}\text{, LNIL))),}\\ &\quad\text{LAPP(LREV(LCONS(}\underline{\text{nly}}\text{, LNIL)),}\\ &\qquad\text{LREV(LCONS(}\underline{\text{nlx}}\text{, LNIL))))).}\end{aligned} \qquad \text{(EX 7)}$$

Instead, when the computation trace of (EX 6) is analyzed, the following EQLENGTH computation is found

$$\begin{aligned}&\text{EQLENGTH(CONS(}\underline{\text{ny}}_1\text{, CONS(}\underline{\text{ny}}_2\text{, NIL)),}\\ &\quad\text{CONS(}\underline{\text{nx}}_1\text{, CONS(}\underline{\text{nx}}_2\text{, NIL)))}\end{aligned} \qquad \text{(EX 6.1)}$$

Since the argument terms are not identical, the method does not apply, and (EX 6) is (correctly) no more generalized.

Finally, let us consider the application to (EX 7) of the generalization method based on estimated function behaviour. The only skeletons are LCONS($\underline{\text{nlx}}$, LNIL) and LCONS($\underline{\text{nly}}$, LNIL), and EQSTRUCT is the only s.r. function which applies to terms containing them. Since the partial derivative of LREV(LAPP(p_1, p_2)) w.r.t. p_1 is 1, and the partial derivative of LAPP(LREV(p_1), LREV(p_2)) w.r.t. p_1 is again 1, the skeleton LCONS($\underline{\text{nlx}}$, LNIL) is generalized to $\underline{\text{nllx}}$. The other skeleton is analogously generalized to $\underline{\text{nlly}}$, and the method induces the following theorem

$$\begin{aligned}&\text{EQSTRUCT(LREV(LAPP(nllx, nlly)),}\\ &\quad\text{LAPP(LREV(nlly),}\\ &\qquad\text{LREV(nllx))).}\end{aligned} \qquad \text{(EX 8)}$$

Conversely, when (EX 6) is considered, the skeletons CONS($\underline{\text{nx}}_1$, CONS($\underline{\text{nx}}_2$, NIL)) and CONS($\underline{\text{ny}}_1$, CONS($\underline{\text{ny}}_2$, NIL)) are not generalized. In fact, the predicate EQLENGTH is directly applied to them (see (EX 6.1)), i.e. we have a term of the form EQLENGTH(p_1, p_2). The partial derivative of p_1 w.r.t. p_2 is 1, while the partial derivative of p_2 w.r.t. p_1 is 0, thus the skeleton CONS($\underline{\text{ny}}_1$, CONS($\underline{\text{ny}}_2$, NIL)) is not generalized. The same happens for CONS($\underline{\text{nx}}_1$, CONS($\underline{\text{nx}}_2$, NIL)). Again (EX 6) is no more generalized, and actually further generalization would not be correct.

Let us suppose to skip the generalization by equivalence, in order to describe the stepwise application of the generalization method based on estimated function behaviour without introducing a more complex example. The application of this method to (EX 5) allows to generalize the skeletons CONS($\underline{\text{nx}}_1$, CONS($\underline{\text{nx}}_2$, NIL)) and CONS($\underline{\text{ny}}_1$, CONS($\underline{\text{ny}}_2$, NIL)) thus obtaining (EX 7). Note that the generalization method based on equivalence cannot be removed, although in this case the same generalization is produced, since it can generalize data terms which are not skeletons, while the last generalization method cannot.

References

[1] Aubin, R.: Mechanizing structural induction. Ph.D.Th., Univ. of Edinburgh, Dept. of Artif. Intel. (1976).

[2] Boyer, R. S., and Moore, J. S.: Proving theorems about LISP functions. J. ACM 22 (1975), 129–144.

[3] Brotz, D. K.: Embedding heuristic problem solving methods in a mechanical theorem prover. STAN–CS–74-446, Computer Science Dept., Stanford Univ. 1974.

[4] Brown, F. M., and Tärnlund, S.: Inductive reasoning in mathematics. Proc. Internat. Joint Conf. on Artif. Intell., Cambridge (1977), 844–850.

[5] Buchanan, B. G.: Logic of scientific discovery. Stanford Artificial Memo. No. 47. Dept. of Comp. Science, Stanford Univ., 1966.

[6] Buchanan, B. G., Feigenbaum, E. A., and Lederberg, J.: A heuristic programming study of theory formation in science. Proc. Internat. Joint Conf. on Artif. Intell., London (1971), 40–48.

[7] Burstall, R. M.: Proving properties of programs by structural induction. Comp. J. 12, 1 (1969), 41–48.

[8] Evans, T. G.: A program for the solution of geometric–analogy intelligence test. Semantic Information Processing, M. Minsky Ed., MIT Press, 1968, pp. 271–656.

[9] Feldman, J., Gips, J., Horning, J. J., and Reder, S.: Grammatical complexity and inference. CS 125. Stanford Artificial Intelligence Project, Stanford, 1969.

[10] Hoare, C. A. R.: Recursive data structures. Int. J. Comp. and Inf. Sc. 4, 2 (1975), 105–162.

[11] Levi, G., and Sirovich, F.: Proving properties, symbolic evaluation and logical procedural semantics. Lecture Notes in Computer Science, Vol. 62, Mathematical Foundations of Computer Science, Springer, Berlin, 1975, pp. 569–574.

[12] Mc Carthy, J., and Hayes, P.: Some philosophical problems from the standpoint of artificial intelligence. In Machine Intelligence 4, B. Meltzer, and D. Michie Eds., Edinburgh Univ. Press, 1969. pp. 466–502.

[13] Meltzer, B.: Power amplification for theorem–provers. in Machine Intelligence 5, B. Meltzer, and D. Michie Eds., Edinburgh Univ. Press, 1969, pp. 165–179.

[14] Michalski, R. S.: A system of programs for computer–aided induction: a summary. Proc. Internat. Joint Conf. on Artif. Intell., Cambridge (1977), 619–620.

[15] Newell, A.: Heuristic programming: ill–structured problems. In Progress in Operational Research, III, Aronofsky J. S. Ed., Wiley and Sons, 1969, pp. 662–415.

[16] Plotkin, G. D.: A further note on inductive generalization. In Machine Intelligence 6, B. Meltzer, and D. Michie Eds., Edinburgh Univ. Press, 1971, pp. 101–124.

[17] Popplestone, R. J.: An experiment in automatic induction. In Machine Intelligence 5, B. Meltzer, and D. Michie Eds., Edinburgh Univ. Press, 1969, pp. 206–215.

[18] Simon, H. A., and Kotovsky, K.: Human acquisition of concepts for sequential patterns. Psychol. Rev., 70 (1966), 546–564.

[19] Vere, S. A.: Induction of concepts in the predicate calculus. Proc. Internat. Joint Conf. on Artif. Intell., Tbilisi (1975), 281–287.

[20] Vuillemin, J. E.: Proof techniques for recursive programs. Memo AIM–218/STAN–CS–76–696, Dept. of Comp. Science, Stanford Univ., 1976.

[21] Waldinger, R. J., and Lee, R. C. T.: "PROW: a step toward automatic program writing." Proc. Internat. Joint Conf. on Artif. Intell., Walker, D. E., and Norton, L. N. Eds., Washington (1969), 241–252.

Computational Linguistics and Computer Languages Vol. XIII. 1979.

RELATIONAL PROGRAMMING ILLUSTRATED BY A PROGRAM FOR THE GAME OF MASTERMIND

M. H. van Emden

Department of Computer Science University of Waterloo

1. Introduction

Many difficulties in programming are caused by the use of *imperative* languages (those which are based on commands) such as Fortran, Algol, or Pascal. These difficulties can be avoided by using a *definitional* language as Lisp or Prolog. Lisp is based on the lambda-calculus and is typically used for the definition of functions. Prolog is based on first-order predicate logic and is typically used for the definition of relations.

This paper aims to show an advantage of specifying a relation instead of a function: the same relation can specify several different functions, depending on which arguments are given. As a result, the Prolog interpreter can use the relational specification to compute several of the different functions implied in the relation. Several examples of this phenomenon are exhibited and discussed in this paper.

Prolog programs are an essential part of this paper. Between different implementations of Prolog there are minor variations in syntax and in the effect of system-defined predicates. The original Marseille implementation [GDM] is the common ancestor of the systems developed in Budapest [PPL], Edinburgh [CLP], and Waterloo [IOP]. The programs in this paper have been run on the latter system. A more advanced version is IC-Prolog [ICP], which is being developed in Imperial College, London, by K.L. Clark, R. A. Kowalski, and F. G. McCabe. Another recent development is by J. A. Robinson and E. Sibert in the University of Syracuse, where an implementation of logic programming is embedded in Lisp.

Prolog is based on Kowalski's proposal [PLPL]for using logic as a programming language, which, in turn, is based on J. A. Robinson's resolution principle [MOL]. Although not conceptually related, Prolog has similarities to several earlier systems, such as [ABSYS], [PLANNER], and [ABSET].

2. The Principle

The principle of relational programming is explained by means of the following binary relation between natural numbers:

$$R = \{ (1,1), (2,4), (3,9), (4,16) \}$$

Depending on which argument is given, the relation specifies a subset of the squaring function, or a subset of the square-root function.

In logic the relation can be specified by the conjunction of the following atomic formulas:

$R(1,1)$.
$R(2,4)$.
$R(3,9)$.
$R(4,16)$.

The atomic formulas are special cases of *clauses,* the components of a Prolog program. Not only in this example, but in general also, Prolog programs are regarded as specifying relations.

The Prolog interpreter can be instructed to find a y such that $R(3,y)$ is provable from the specification. It will respond with $y = 9$, thus computing a value of the squaring function. Or the Prolog interpreter may be instructed to find an x such that $R(x,16)$ is provable from the specification. It will respond with $x = 4$, thus computing a value of the square-root function.

Both of these computations are, of course, nothing but table look-ups. The remainder of this paper is devoted to less trivial applications, which can, however, be viewed as look-up in virtual tables implicit in specifications consisting of clauses which have a less restricted form than those of the present example. This point of view is elaborated in [CDI].

3. A simple example of relational programming

In logic programs, as in first-order logic, terms denote objects. The syntax requires that a term is either a constant (in Prolog an identifier or a decimal number), or a variable (in Prolog a constant preceded by an asterisk), or $f(t_1, \ldots, t_m)$ where f is an m-place functor and $t_1, \ldots, t_m$ are terms.

For example,

$.(c, .(b,nil))$

is a term, where "." is a 2-place functor and b,c,nil are constants. Prolog allows infix notation for 2-place functors so that we can also write

$c.(b.nil)$

As a further convenience, we are allowed to write

$t_1. \ldots .t_n$

and to specify whether it means

$t_1.(t_2.(\ldots .t_n) \ldots)$

or

$(\ldots (t_1.t_2). \ldots .t_{n-1}).t_n$

Throughout this paper we assume that the former option is in force.

In this section we discuss a specification of the relation, called append, between three lists which holds if the last list is the result of appending the first two. In this example the objects to be denoted by terms are lists. A nonempty list is a composite object: it consists of a head, which is the first element, and a tail, which is the list of the remaining elements. A non-empty list is denoted by a term of the form $t_1 . t_2$ where the term t_1 is the head and the term t_2 is the tail. An empty list has no components and is therefore denoted by a constant (in this paper, as is usual, by *nil*).

The *logic program* specifying the append relation is the conjunction of the tollowing two clauses:

append(*nil*, $^*y$, $^*y$).

(3.1)

append($^*u.\ ^*x$, $^*y$, $^*u.\ ^*z$) $\leftarrow$ append($^*x$, $^*y$, $^*z$).

In general we are concerned with clauses of the form

$$A \leftarrow B_1 \ \& \ldots \& \ B_n, \qquad n \geqslant 0$$

where $A, B_1, \ldots, B_n$ are atomic formulas containing the variables $x_1, \ldots, x_p$, where $p \geqslant 0$. The clause should be read as

for all $x_1, \ldots, x_p$, A if (B_1 and . . . and B_n)

In case $n = 0$ we drop the left arrow. In that case the clauses should be read as an unconditional assertion. It should now be clear that the clauses (3.1) are true of the append relation between lists.

A program, which consists of clauses, is activated by a *goal statement,* which has the form

$$\leftarrow A_1 \ \& \ldots \& \ A_k, \qquad k \geqslant 0$$

where $A_1, \ldots, A_k$ are atomic formulas, called *goals.* One of the goals in a non-empty goal statement is distinguished and is called the *selected goal.*

From a goal statement

(3.2) $\quad \leftarrow A_1 \ \& \ldots \& \ A_k$

with selected goal A_i there may be derived the goal statement

(3.3) $\quad \leftarrow (A_1 \ \& \ldots \& \ A_{i-1} \ \& \ B_1 \ \& \ldots \& \ B_n \ \& \ A_{i+1} \ \& \ldots \& \ A_k)t_0$

if the program contains a clause

$$A \leftarrow B_1 \ \& \ldots \& \ B_n, \qquad n \geqslant 0,$$

which matches the goal A_i. This is said to be the case if $At = A_i t$ for some substitution t of terms for variables. If such a t exists, then there also exists a 'most general' one, here called t_0, which is such that At can be obtained by substitution (possibly null) from At_0; t_0 is the *substitution produced* by the derivation of (3.3) from (3.2).

A *proof* is a sequence of goal statements, ending in an empty goal statement, and such that each successive goal statement is derived from the preceding one. Suppose now that a proof exists with $\leftarrow A_1 \ \& \ldots \& \ A_k$ as initial goal statement and with $t_0, \ldots, t_N$ as substitutions. What is now proved by the proof is that

$$\text{for all } x_1, \ldots, x_q, (A_1 \ \& \ldots \& \ A_k)t_0 \ldots t_N$$

follows from the conjucnction of the clauses in the program, where $x_1, \ldots, x_q$ $(q \geqslant 0)$ are the variables in $(A_1 \ \& \ldots \& \ A_k)t_0 \ldots t_N$.

Let us look at an example, using the logic program (3.1), where we require the result of appending the three lists *c.nil, a.nil,* and *b.nil.* This requirement is specified by the initial goal statement

← append(*a.nil,b.nil,* **x*) & append(*c.nil,* **x,* **y*)

If a proof is found, then the composition of all substitutions in the proof substitutes for **y* the required result.

For the Prolog interpreter the selected goal is always the leftmost. Hence the interpreter will attempt initially to match the leftmost goal in the above goal statement with the first clause of (3.1), which is not possible. The second clause does match, deriving the goal statement

← append(*nil,b.nil,* **x1*) & append(*c.nil,a.* **x1,* **y*)

Now the first clause matches the leftmost goal, deriving

← append(*c.nil,a.b.nil,* **y*)

The next goal statement is

← append(*nil,a.b.nil,* **y1*)

with a substitution replacing **y* by *c.* **y1*) The selected goal now matches the first clause, so that the next goal statement is empty. A proof has been found. The variable **y* in the initial goal statement is replaced by *c.a.b.nil.*

So much for the basic mechanism of Prolog. Here we are concerned with relational programming; that is, we want to make use of the fact that (3.1) specifies a relation between the three arguments of append, rather than a function from the first two to the third. Take for example the goal statement

← append(*a.nil*, * *y*,*a.b.c.nil*)

In finding a proof, Prolog will substitute *b.c.nil* for **y*, thus performing list subtraction. Below (3.4, 3.5, 3.6) we list several other examples of goal statements causing Prolog to compute functions other than the append function, all by means of the same relational specification (3.1).

(3.4) ← append(* *x*,*c.nil*,*a.b.cnil*)

substitution:

**x* := *a.b.nil*

(3.5) ← append(* *u*,*c*. * *v*,*a.b.c.nil*)

substitution:

**u* := *a.b.nil*
**v* := *nil*

(3.6) ← append(* *u*,*b.c.nil*, * *v*) & append(**v*, * *w*,*a.b.c.d.nil*)

substitution:

**u* := *a.nil*
**v* := *a.b.c.nil*
**w* := *d.nil*

The goal statement (3.4) causes another form of list subtraction. The goal statement (3.5) has the effect of checking whether *c* occurs in *a.b.c.nil*; this suggests the following definition of list membership:

```
append(nil, *y, *y).
append( *u. *x, *y, *u. *z) ← append( *x, *y, *z).
member( *c, *w) ← append( *u, *c. *v, *w).
```

The goal statement (3.6) has the effect of checking whether *b.c.nil* is a sublist of *a.b.c.d.nil*; this suggests the following definition of the sublist relation:

```
append(nil, *y, *y).
append( *u. *x, *y, *u. *z) ← append( *x, *y, *z).
sublist( *x, *z) ← append( *u, *x, *v) & append( *v, *w, *z).
```

This definition of sublist is not restricted to completely specified list as first argument. For example,

← sublist(**x*. **y*. **x.nil*, *m.a.d.a.m.nil*) will result in

$*x := a$
$*y := d$

In other words, sublist can be used to search for incompletely specified sublists: things that may well be called "patterns".

We have shown that a single specification can be used to compute a variety of functions, each of which would require a different program in a conventional language. We call relational programming the technique of using this phenomenon. Another advantage is that the more general relational specification may be easier to find than the particular function required.

Prolog is far from perfect as a vehicle for relational programming. Finding a proof depends on having in each goal statement the correct choice of selected goal or, given the selected goal, using the correct choice of clause in case more than one matches. Prolog often fails to find a proof because it always selects the leftmost goal and because it always tries to match the clauses in the order in which they occur in the program. IC-Prolog [ICP] will find proofs in cases where Prolog does not because it is more flexible in determining the selected goal.

4. The game of Mastermind

In the abstract game of Mastermind the following types of object exist:

CODE = PROBE = the set of ordered 4-tuples with elements in a set of colours
SCORE = the set of ordered pairs with elements in the set of numbers

0 through 4

f: PROBE $\times$ CODE $\rightarrow$ SCORE; we call f the 'scoring function'.

The elements of the ordered 4-tuples correspond to the 'code pegs' of the concrete game, and they may be black,blue, green, red, white, or yellow. In the abstract game we take the set of colours to be

{ BLACK, BLUE, GREEN, RED, WHITE, YELLOW }

The first (second) component of an element of SCORE corresponds to the number of 'black (white) key pegs' of the concrete game. We will find it convenient to represent in the abstract game these numbers in successor notation, because then the relation between predecessor and successor can be specified succinctly, without explicitly referring to the sum relation. The successor function is denoted by + so that $+(x)$ is the successor of x in functional notation. However, suffix notation is more concise and traditional; therefore we represent the set of numbers 0 through 4 as

$\{0, 0+, 0++, 0+++, 0++++\}$

The scoring function has the property that the higher the score, the greater the similarity between its arguments. This statement is only intended to help the intuition, as it is formally meaningless without a definition of order among scores or of similarity between codes and probes. The value $f(p,C)$ of the scoring function contains a black key peg for every position where p and C have the same colour. Such an occurrence is called a 'strong match'. For every one of the remaining positions, $f(p,C)$ contains a white key peg for every element of p with the same colour as an element of C. Such an occurrence is called a 'weak match'

The game is played as follows. There are two players, the Codemaker and the Codebreaker. The Codemaker selects a code C which is concealed from the Codebreaker. The Codebreaker can obtain information about C by selecting a probe p in response to which the Codemaker reveals the result $s = f(p,C)$ of the scoring function.

This is repeated until the Codebreaker has selected a probe equal to C. In other words, the Codebreaker constructs a sequence $p_1, \ldots, p_n$ of probes with $p_i \neq C$ for $i \neq n$ and $p_n = C$. The Codemaker constructs a sequence $s_1, \ldots, s_n$ such that $s_i = f(p_i, C)$. The selection of p_i by the Codebreaker may depend on $(p_1, s_1), \ldots, (p_{i-1}, s_{i-1})$. It is the Codebreaker's objective to make n as small as possible.

5. A logic program for the scoring function

Logic programs compute relations. Therefore, if one wants to compute a function, it has to be expressed as a relation. The logic program for the scoring function defines a relation *MM* such that

$$MM(p,c,s) \text{ iff } f(p,c) = s,$$

where f is the scoring function discussed before. The relation *MM* is defined by the clause

```
MM( *P, *C, *S1. *S2) ← BLACKS( *P, * C, * P1, *C1, *S1)
                        & WHITES( *P1, *C1, * S2).
```

The first component of the score is $s1$, the number of black key pegs. BLACKS is true if $s1$ is the number of strong matches between p and c and if $p1$ and $c1$ are the results of removing the strongly matching elements from p and c respectively. WHITES is true if $s2$ is the number of weak matches between $p1$ and $c1$. This clause reflects the informal description of f given in the previous section.

The following statements in logic, which are true of *MM* and of the auxiliary relations, have been run as a Prolog program for computing the scoring function. Note that we represent an ordered n-tuple consisting of the colours $c_1, \ldots, c_n$ by the term $c_1 . \ldots . c_n . nil$; that is, just as we represented lists in section 3.

```
OP(+,SUFFIX,150).
OP(=,RL,150).

MM(*P,*C,*S1.*S2) <- BLACKS(*P,*C,*P1,*C1,*S1) & WHITES(*P1,*C1,*S2).

BLACKS(NIL,NIL,NIL,NIL,0).
BLACKS(*U.*P,*U.*C,*P1,*C1,*S+) <- BLACKS(*P,*C,*P1,*C1,*S).
BLACKS(*U.*P,*V.*C,*U.*P1,*V.*C1,*S) <- NOT(*U=*V)
                                        & BLACKS(*P,*C,*P1,*C1,*S).

WHITES(NIL,*C,0).
WHITES(*U.*P,*C,*S+) <- DEL(*U,*C,*C1) & WHITES(*P,*C1,*S).
WHITES(*U.*P,*C,*S) <- NONMEM(*U,*C) & WHITES(*P,*C,*S).

DEL(*U,*U.*Y,*Y).
DEL(*U,*V.*Y,*V.*Y1) <- NOT(*U=*V) & DEL(*U,*Y,*Y1).

NONMEM(*U,NIL).
NONMEM(*U,*V1.*V) <- NOT(*U=*V1) & NONMEM(*U,*V).

*X=*X.
```

Fig. 1: Prolog program for the scoring function

6. A codebreaker obtained by relational programming

Suppose we want to obtain a program playing the part of the Codebreaker. Then we have to devise a playing strategy, that is, some function with the sequence $(p_1, s_1), \ldots, (p_{i-1}, s_{i-1})$ as argument and with a value which can be used as value for p_i, the next probe. We will use a strategy reported in [EFP] which is to take any p_i such that $f(p_j, p_i) = s_j$ for $j = 1, \ldots, i-1$, if $i \geqslant 2$. The first probe is arbitrary. In the first place, such a p_i always exists, because, for example, the unknown code has this property. In the second place, such a p_i can be expected to be, in some sense, close to the unknown code, the more so the larger i is.

This last observation can be made more precise if we consider the first component of the score, namely the number s' of black key pegs. Note that $4\text{-}s'$ is the so-called Hamming distance, which is a metric, between fourtuples of colours. The set of p_i such that for $j = 1, \ldots, i-1$, $f(p_j, p_i) = s_j$ is therefore contained in the set of codes having given distances to the points $p_1, \ldots, p_{i-1}$ in a metric space. This set contains the unknown code and it can be expected to be smaller for large i.

The strategy therefore requires the following equation to be solved for x: $f(p,x) = s$. We already wrote a program for solving for x: $f(p,c) = x$. Apparently, the relation *MM*, introduced for a logic program to compute the scoring function, also specifies the computation needed by a Codebreaker.

It should now be clear why we have chosen the game of Mastermind as a case study in relational programming: we aim to obtain a program for the more difficult Codebreaker's part by specifying in relational form the easily programmed scoring function and then to use this relation with the probe and score as given arguments to obtain a guess at the unknown code.

However, it would be a mistake to believe that the program in section 5 can be used as a code-breaking program. The reason is that we have inadvertently specified a relation different from the one intended. We need a relation which is a subset of the Cartesian product PROBE × CODE × SCORE. PROBE and CODE contain only fourtuples of colours. It is apparent that the relation specified in section 5 can have in its first two argument places tuples containing arbitrary elements, not necessarily colours. The relation is usable for computing the scoring function because then the first two arguments are given, and can be given (as required in this particular application) as tuples of colours.

The relation specified in section 5 is too large, but a goal specifying that the scoring function is to be computed happens to restrict the relation in the desired way. However, if we want to use a goal $\leftarrow MM(p,x,s)$ to solve for x the equation $f(p,x) = s$, with p and s given, then we cannot expect the desired result, because according to the specification in section 5, x can be a tuple containing arbitrary elements. However, if we would extend the specification so that indeed x is forced to consist of colours only, then would be able to use the modified specification to solve for x both $f(p,c) = x$ and $f(p,x) = s$. That is, we could then use the relation MM to play both sides of Mastermind.

Let us see how we can correct this deficiency in our previous specification of MM. The condition

```
not( *u = *v)
```

with u equal to a colour is satisfied by values for v which are arbitrary objects which are not necessarily colours. Hence we change occurrences of this condition, say, diff(u,v) and we add the clause

```
diff( *u, *v) ← colour( *u) & colour( *v) & not( *u = *v)
```

and specify also explicitly which colours exist by adding the clauses

```
colour(black). colour(blue). colour(green).
colour(red). colour(white). colour(yellow).
```

A specification of the relation MM, which is suitable for playing both the Codemaker's and the Codebreaker's parts, can be found as part of the complete Prolog program for Mastermind listed in the next section.

7. The complete program

We now have a Prolog program which can solve for x $f(p,c) = x$ by the goal statement

$\leftarrow MM(*p, *c, *x)$

and also can solve for x $f(p,x) = s$ by the goal statement

$\leftarrow MM(*p, *x, *s)$

We continue towards a complete program for Mastermind. As a first step we define the relation between a sequence of (probe, score)-pairs

$(p_1,s_1), \ldots, (p_i,s_i)$

and a *candidate code* p having the property that

$f(p_1,p) = s_1, \ldots, f(p_i,p) = s_i$

The desired relation is defined by

```
candcode(nil, * ).
candcode(( *p1. *s1). *ps, *p) ← mm( *p1, *p, *s1) & candcode( *ps, *p).
```

Let us call a *candidate solution* a sequence

$(p_1,s_1), \ldots, (p_i,s_i)$

of (probe, score)-pairs such that

$f(p_1,p_k) = s_1, \ldots, f(p_{k-1}, p_k) = s_{k-1},$ for $k = 2, \ldots, i-1.$

That is, each probe is a candidate code with respect to the preceding sequence of (probe, score)-
-pairs. A candidate solution is a *solution* if the last score has at least four black pegs, that is if the last probe is equal to the code.

For us it is important that a candidate solution be *extendable* to a solution. Of the property of being extendable we can say that

```
extendable(( * . ( * ++++ * )) . * ).
extendable( * cs) ← candcode( * cs, *cc) & score( * cc, * s)
                    & extendable(( * cc . * s) . * cs).
score( * p, * s) ← code( * c) & mm( * p, *c, * s).
```

A note on notation: Because each clause is, sepatately from the other clauses, universally quantified, a variable name is only meaningful within a clause. It follows that the name of a variable which occurs only once in a clause, is immaterial, and hence can be ommitted. Only the asterisk is written; the variable is anonymous. Conversely, each occurrence of an anonymous variable in a clause stands for a variable different from any other variable in the clause, anonymous or not.

With respect to a given candidate solution there are typically several possible candidate codes. The above simple definition of 'extendable' has the disadvantage that it does not extend with a best, but rather with any, candidate code. There is hence no guarantee that only reasonably short solutions are specified by 'extendable'. Our experience shows that with most codes 'extendable' gives a solution of length five. An exception was found with a code consisting of equal colours. D.E. Knuth was quoted [EEP] as having found that a solution of length five is always possible. In order to guarantee that our solutions do not exceed a given bound, we have restricted the above definition of 'extendable' to mean: extendable within the number of steps determined by an additional third argument.

The complete program is listed below in two parts. Only the part needed for the definition of 'extendable' is of interest from the point of view of relational programming. Yet a fairly large additional part if required for a program that interface with a client not familiar with its inner mechanisms. This part, labelled 'interactive manager' is also done in Prolog, though it is hardly an example of definitional programming. It has also been listed in full in order to show that for this kind of programming task Prolog is at least serviceable, although usually not particularly inspiring.

An exception is the way in which the backtracking of Prolog allows one to program a check on the correctness of input. For example, in PLAY it is desirable to check whether the *X produced by READ is correct. If not, CHECKSEED causes a complaint to appear and fails, so that backtracking causes READ to be reactivated, giving the user another opportunity for entering something.

In order to able to understand the interactive manager one has to know some of the built-in predicates of the Waterloo Prolog interpreter; see Appendix 1 for the relevant excerpts from [IOP]. See Appendix 2 for the control flow of the interactive manager.

```
OP(+,SUFFIX,150).        /*  + DEFINED AS SUFFIX OPERATOR  */
OP(=,RL,150).            /*  = DEFINED AS INFIX OPERATOR ASSOCIATING
                             FROM RIGHT TO LEFT  */

EXTENDABLE((*P.(*++++.*)).*,*+)
           <- WRITECH('THE CODE MUST BE: ') & CHECKCODE(*P0,*P) & WRITE(*P0).
EXTENDABLE(*CS,*N+) <- CANDCODE(*CS,*CC) & WRITECH('MY NEXT PROBE IS: ')
                     & CHECKCODE(*C,*CC) & WRITECH(*C) & WRITECH('; SCORE: ')
                     & SCORE(*CC,*S) & WRITESCORE(*S)
                     & *M+=*N & IS(*M,*MD) & WRITECH(*MD)
                     & WRITE(' TRIES TO GO')
                     & EXTENDABLE((*CC.*S).*CS,*N).

CANDCODE(NIL,*).
CANDCODE((*P1.*S1).*PS,*P) <- MM(*P1,*P,*S1) & CANDCODE(*PS,*P).

SCORE(*P,*S) <- CODE(*C) & MM(*P,*C,*S).

/*************************************************************************/
/* BEGINNING OF DEFINITION OF SCORING RELATION */

MM(*P,*C,*S1.*S2) <- BLACKS(*P,*C,*P1,*C1,*S1) & WHITES(*P1,*C1,*S2).

BLACKS(NIL,NIL,NIL,NIL,0).
BLACKS(*U.*P,*U.*C,*P1,*C1,*S+) <- BLACKS(*P,*C,*P1,*C1,*S).
BLACKS(*U.*P,*V.*C,*U.*P1,*V.*C1,*S) <- DIFF(*U,*V)
                                      & BLACKS(*P,*C,*P1,*C1,*S).

WHITES(NIL,*C,0).
WHITES(*U.*P,*C,*S+) <- DEL(*U,*C,*C1) & WHITES(*P,*C1,*S).
WHITES(*U.*P,*C,*S) <- NONMEM(*U,*C) & WHITES(*P,*C,*S).

/* DEL(U,Y,Y1) IF Y1 IS THE RESULT OF DELETING U FROM LIST Y */
DEL(*U,*U.*Y,*Y).
DEL(*U,*V.*Y,*V.*Y1) <- DIFF(*U,*V) & DEL(*U,*Y,*Y1).

/* NONMEM(U,V) IF U IS NOT A MEMBER OF LIST Y */
NONMEM(*U,NIL).
NONMEM(*U,*V1.*V) <- DIFF(*U,*V1) & NONMEM(*U,*V).

DIFF(*U,*V) <- COLOUR(*U) & COLOUR(*V) & NOT(*U=*V).

COLOUR(BLACK). COLOUR(BLUE). COLOUR(GREEN).
COLOUR(RED). COLOUR(WHITE). COLOUR(YELLOW).

*X=*X.

/* END OF DEFINITION OF SCORING RELATION */
/*************************************************************************/
```

Fig. 2: Main part of Mastermind program

```
/*INTERACTIVE MANAGER*/

PLAY <- WRITE('MASTERMIND AT YOUR SERVICE')
      & WRITE('ENTER AN ARBITRARY NUMBER BETWEEN 0 AND 16383')
      & READ(*X) & CHECKSEED(*X) & ADDAX(SEED(*X))
      & WRITECH('EXAMPLE FORMAT FOR ENTERING CODE: ')
      & WRITE('YELLOW.BLUE.WHITE.BLACK')
      & PLAY1.

PLAY1 <- WRITECH('DO YOU WANT TO MAKE OR BREAK CODES? ')
       & WRITE('ANSWER MAKE. OR ANSWER BREAK')
       & READ(*X) & CHECKMB(*X) & START(*X).

START(MAKE) <- / & WRITE('ENTER CODE; I PROMISE NOT TO LOOK') & READ(*C0)
             & CHECKCODE(*C0,*C) & ADDAX(CODE(*C)) & GENCODE(*P) & SCORE(*P,*S
             & WRITECH('MY FIRST PROBE IS: ')&CHECKCODE(*P0,*P) & WRITECH(*P0)
             & WRITECH('; SCORE: ') & WRITESCORE(*S)
             & EXTENDABLE((*P.*S).NIL,0+++++) & DELAX(CODE(*)) & ASK.

START(BREAK) <- GENCODE(*C) & ADDAX(CODE(*C))
              & WRITE('ENTER FIRST PROBE') & READ(*P0) & CHECKPRST(*P0,*P)
              & SCORE(*P,*S) & CONTBR(*S).

CONTBR(*++++.*) <- / & WRITE('YOU GOT IT') & DELAX(CODE(*)) & ASK.
CONTBR(*S) <- WRITECH('YOUR SCORE: ') & WRITESCORE(*S)
            & WRITE('ENTER NEXT PROBE OR TYPE STOP') & READ(*X0)
            & CHECKPRST(*X0,*X) & RESPONDTO(*X).

RESPONDTO(STOP) <- / & WRITECH('I ASSUME YOU GIVE UP; THE CODE IS: ')
                 & DELAX(CODE(*C)) & CHECKCODE(*C0,*C) & WRITE(*C0) & ASK.
RESPONDTO(YES) <- / & PLAY1.
RESPONDTO(NO) <- DELAX(SEED(*))
               & WRITE('MASTERMIND WAS PLEASED TO SERVE YOU')
               & WRITE('YOU ARE NOW RETURNED TO PROLOG') & EXIT.
RESPONDTO(*P) <- SCORE(*P,*S) & CONTBR(*S).

ASK <- WRITE('DO YOU WANT ANOTHER GAME? ANSWER YES. OR ANSWER NO')
     & READ(*X) & CHECKYN(*X) & RESPONDTO(*X).

/* CODE GENERATOR */
GENCODE(*U.*V.*W.*X.NIL) <- RANDOMCOLOUR(*U) & RANDOMCOLOUR(*V)
                          & RANDOMCOLOUR(*W) & RANDOMCOLOUR(*X).

RANDOMCOLOUR(*X) <- RANDNUM(*R) & REM(*R,6,*N) & SUM(*N,1,*N1)
                  & AX(COLOUR(*), COLOUR(*X),*N1).

/* R IS PREVIOUS, S IS NEXT RANDOM NUMBER */
RANDNUM(*S) <- DELAX(SEED(*R)) & PROD(*R,125,*X) & SUM(*X,1,*Y)
             & REM(*Y,16384,*S) & ADDAX(SEED(*S)).
```

Fig. 3: First part of Interactive Manager

```
/* CHECK WHETHER ARGUMENT IS CORRECT SEED FOR RANDOM-NUMBER
   GENERATOR
*/
CHECKSEED(*X) <- INT(*X) & GE(*X,0) & LE(*X,16383) & /.
CHECKSEED(*) <- COMPLAINT.

/* CHECK FOR 'MAKE' OF 'BREAK' */
CHECKMB(MAKE) <- /.
CHECKMB(BREAK) <- /.
CHECKMB(*) <- COMPLAINT.

/* CHECK FOR 'YES' OR 'NO' */
CHECKYN(YES) <- /.
CHECKYN(NO)  <- /.
CHECKYN(*) <- COMPLAINT.

/* CHECK FOR CODE OR PROBE AND CONVERSION TO OR FROM INTERNAL
   FORMAT, WHICH CONTAINS 'NIL'
*/
CHECKPRST(STOP,STOP) <- /.
CHECKPRST(*X,*Y) <- CHECKCODE(*X,*Y).

CHECKCODE(*U.*V.*W.*X,*U.*V.*W.*X.NIL)
       <- COLOUR(*U) & COLOUR(*V) & COLOUR(*W) & COLOUR(*X) & /.
CHECKCODE(*,*) <- COMPLAINT.

COMPLAINT <- WRITE('ERROR; TRY AGAIN') & FAIL.

WRITESCORE(*BLACKS.*WHITES) <- IS (*BLACKS,*X) & WRITECH(*X)
                             & WRITECH(' BLACK AND ') & IS(*WHITES,*Y)
                             & WRITECH(*Y) & WRITE(' WHITE').


/* CONVERSION FROM SUCCESSOR NOTATION TO DECIMAL NOTATION */
IS(0,0).
IS(*S+,*N1) <- IS(*S,*N) & SUM(*N,1,*N1).
```

Fig. 4: Second part of Interactive Manager

8. Related work

Sickel has investigated [INV] how to predict in general whether it is possible to compute a particular function from a relational definition with a given rule for selecting a goal.

A striking application of relational programming has been found by Colmerauer [GDM]. In his example of a compiler written in Prolog, the analyzer takes as input the source code and outputs a parse tree decorated with semantic information. The code generator takes such a tree as input and outputs object code. Both parts are written by Colmerauer as relations between strings and parse trees. The clauses defining the relation are rewrite rules in the traditional sense. For the analyzer the first argument is given; for the code generator the second argument is given. In this way it was possible to write the code generator as a set of rewrite rules, just as the analyzer was.

In [PRL] we showed that a logic program for quicksort could be inverted to a permutation generator by writing it as a relation between a possibly unsorted list and its sorted version.

9. Concluding remarks

It is widely accepted that definitional programming is more reliable and more productive in terms of human effort than imperative programming. It is also generally true that imperative programs are more productive in terms of processor time and memory space. Definitional programming has a promising future because computer processors and memories are expected to become considerably cheaper than are at present; also, it should be kept in mind that not nearly as much effort has been spent on efficient compilation of definitional languages as has been the case with imperative languages.

Of two approaches to definitional programming – functional and relational – the first has been explored much more intensively than the second. Lisp has been in use since about 1960 and was backed by massive and uninterrupted support from implementers and users, initially mainly at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Prolog arrived on the scene much later. Outside of Hungary, Prolog has been, at best, tolerated rather than supported. In addition to that, an entire category of applications, namely symbolic computation, has become Lisp territory; not because of an inherent superiority of functional over relational programming, but simply because Lisp was there first.

Because of the growing importance of definitional programming, it is now time to understand the relative merits of the functional and relational approaches. Far from presenting a comprehensive comparison, this paper has only attempted to contribute a small part which we expect to be relevant in such a comparison.

10. Acknowledgements

We owe a great debt of gratitude to Grant Roberts who made logic programming feasible in Waterloo. Roberts also suggested improvements to an earlier version of the Mastermind program. The suggestion of writing a program for Mastermind came from David Warren.

The Canadian National Science and Engineering Research Council has provided partial financial support.

11. References

[ABSET] E.W. Elcock, J.M. Foster, P.D.M. Gray, J.J.M. McGregor, and A.M. Murray: ABSET: A programming language based on sets. Machine Intelligence 6, B. Meltzer and D. Michie (eds.), Edinburgh University Press, 1971.

[ABSYS] J.M. Foster and E.W. Elcock: Absys 1: an incremental compiler for assertions, Machine Intelligence 4, B. Meltzer and D. Michie (eds.), 423-429. Edinburgh University Press, 1969.

[CDI] M.H. van Emden: Computation and Deductive Information Retrieval. E. Neuhold (ed.): Formal Description of Programming Concepts, North Holland, Amsterdam, 1978.

[CLP] D.H.D. Warren: Implementing Prolog-compiling predicate logic programs. DAI Research Reports 39 and 40. Dept. of Artificial Intelligence, University of Edinburgh, 1977.

[EFP] C. Wetherell: Etudes for programmers. Prentice-Hall, 1978.

[GDM] A. Colmerauer: Les grammaires de metamorphose; in L. Bolc (ed.): Natural Language Communication with Computers, Springer Lecture Notes in Computer Science, 1977.

[ICP] F.G. McCabe: Programmer's Guide to IC-Prolog. Dept. of Computation and Control, Imperial College, 1978.

[INV] S. Sickel: Invertibility of logic programs. Fourth Workshop on Automated Deduction. University of Texas at Austin, Feb. 1979.

[IOP] G.M. Roberts: An implementation of PROLOG; M.Sc. Thesis, Dept. of Computer Science, University of Waterloo, 1977.

[MOL] J.A. Robinson: A machine-oriented logic based on the resolution principle. J. ACM 12 (1965), 23-44.

[PLANNER] C. Hewitt: Planner: a language for manipulating models and proving theorems in a robot, Proc. First Int. Joint Conf. in Artificial Intelligence, pp. 295-301.

[PLPL] R.A. Kowalski: Predicate Logic as a programming language; Proc. IFIP 74, North Holland, 1974, 556-574.

[PPL] P. Szeredi: Prolog- a very high level language based on predicate logic. Proc. Second Hungarian Computer Science Conference, Budapest, July 1977.

[PRL] M.H. van Emden: Programming in resolution logic. Machine Intelligence 8 (eds. E.W. Elcock and D. Michie) by Ellis Horwood Ltd. and John Wylie, 1977.

12. Appendix 1: Some information on the built-in predicates of the Waterloo Prolog interpreter quoted from [IOP].

READ is a predicate with one or two arguments. The second argument is the optional file identifier. A term is read from the indicated file and unified with the first argument. The term must be delimited with the end of term character. If the end of the input file has been reached the predicate fails. If backtracking returns to the read then a read of the next term will be attempted. If the term read cannot be unified with the first argument or the format of the term is invalid then backtracking will cause a read of the next term to be attempted.

WRITE is a predicate with one or two arguments. The second argument is the optional file identifier. The term specified by the first argument is written on the indicated file. The term is delimited by the end of term character. The term is written using prefix, infix and suffix notation where appropriate, as indicated by the operator declarations at the time of writing.

WRITECH is a predicate with one or two arguments. The second argument is the optional file identifier. The first argument specifies a term which is formatted using the operator declarations (as for WRITE) and placed in the output buffer for the given file. If the buffer is filled then it is written to the given file (and emptied). If the buffer is partially filled then it is not written out.

There are several predicates which are included to provide the basic operations of integer arithmetic. Each of these predicates has three arguments. The first two are the input parameters and the last is the result parameter. The first two arguments must be integers. The appropriate integer function of the first arguments is unified with the third argument.

The arithmetic predicates are:

DIFF	– difference (subtraction)
PROD	– product
QUOT	– quotient
REM	– remainder
SUM	– sum

The database built-in predicates provide the facility for updating the database (i.e. the set of axioms in the active work space).

The ADDAX predicate is used to add an axiom to the database. It has one or two arguments. The first argument must be a valid axiom. It may be:

(a) a unit axiom. In this case it is a skeleton or an atom.

(b) a non-unit axiom. In this case it is of the form <head> ← <body>.
<head> must be a skeleton or atom.

The axiom specified by the first argument is added to the database. If a single argument is specified then the axiom is added after all other axioms with the same predicate name and number of arguments.

The DELAX predicate is used to delete an axiom from the database. It may be called with one or two arguments. The first argument is a term representing an axiom. The first argument may be:

(a) a unit axiom. In this case it is a skeleton or an atom.

(b) a non-unit axiom. In this case it is of the form <head> ← <body>
<head> must be a skeleton or an atom.

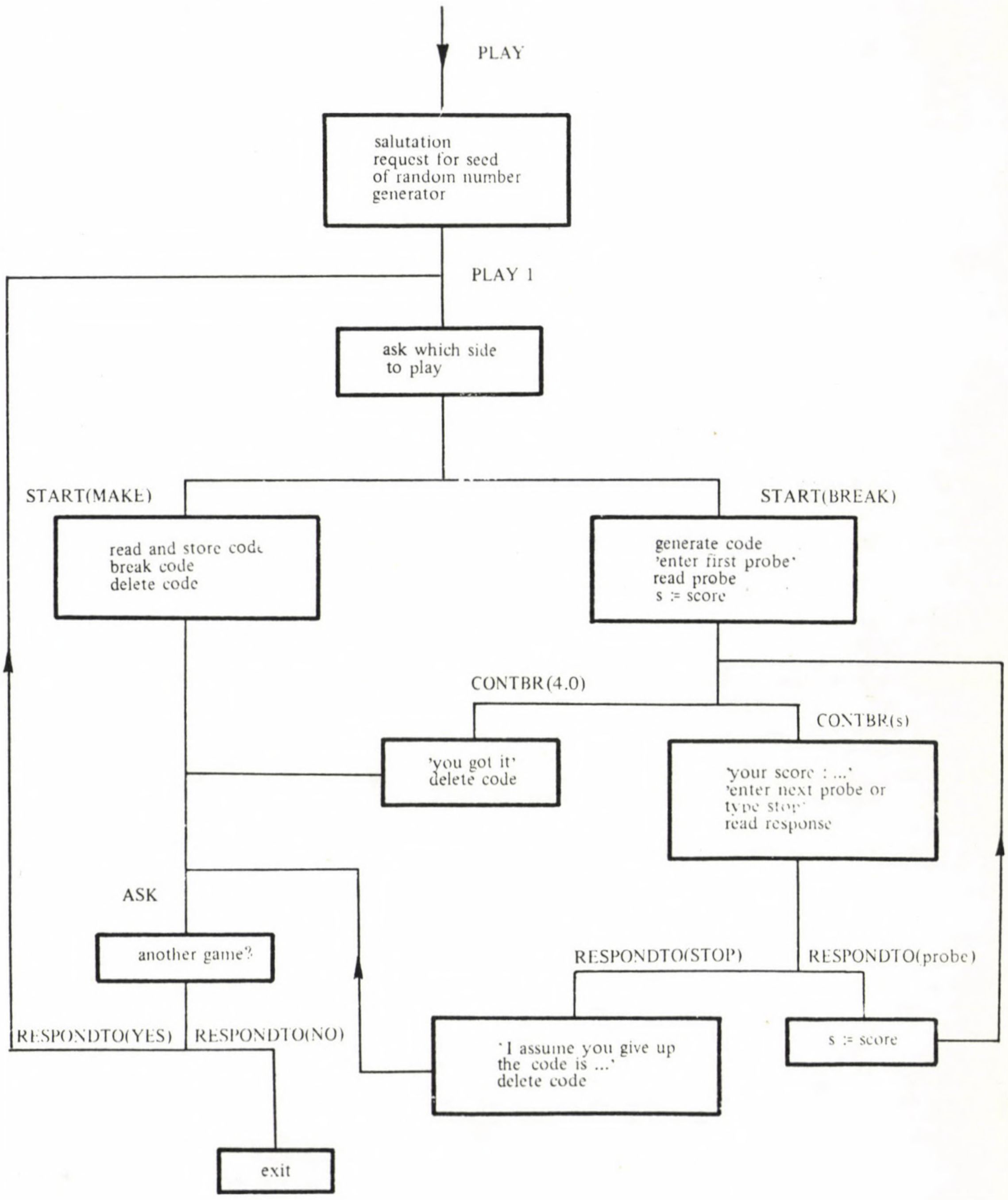
Thus the first argument specifies the name and number of arguments for the axiom to be deleted. If only one argument is specified then an attempt is made to unify the argument with each of the relevant axioms in the database. The axioms are selected in the order in which they appear in the database. If no axiom is found which is unifiable with the first argument then the predicate fails. If the unification succeeds for an axiom then the axiom is deleted and the predicate succeeds. If backtracking subsequently returns to this point then the predicate will fail, thus preventing accidental deletion of further axioms.

The AX predicate has two basic formats:

```
AX(<head>,<axiom>).
AX(<head>,<axiom>,<index>).
```

The AX predicate is used to retrieve axioms from the database. <head> is a model axiom head and may be a skeleton, an atom or a variable. If <head> is not a variable then it specifies a predicate name and number of arguments implicitly. The axioms for this name and number of arguments are retrieved. If an <index> is specified, then the *i*-th axiom that matches the <head> is unified with <axiom>, where *i* is the value of <index>.

13. Appendix 2: Control flow in the interactive manager



14. Appendix 3: An interactive session with the Mastermind Program.

```
WELCOME TO PROLOG 0.0
LOAD(MMIND)<-
<-play.
MASTERMIND AT YOUR SERVICE.
ENTER AN ARBITRARY NUMBER BETWEEN 0 AND 16383.
12345.
EXAMPLE FORMAT FOR ENTERING CODE: YELLOW.BLUE.WHITE.BLACK.
DO YOU WANT TO MAKE OR BREAK CODES? ANSWER MAKE. OR ANSWER BREAK.
break.
ENTER FIRST PROBE.
black.blue.green.red.
YOUR SCORE: 1 BLACK AND 0 WHITE.
ENTER NEXT PROBE OR TYPE STOP.
black.black.black.black.
YOUR SCORE: 2 BLACK AND 0 WHITE.
ENTER NEXT PROBE OR TYPE STOP.
black.black.white.white.
YOUR SCORE: 1 BLACK AND 1 WHITE.
ENTER NEXT PROBE OR TYPE STOP.
black.yellow.black.yellow.
YOU GOT IT.
DO YOU WANT ANOTHER GAME? ANSWER YES. OR ANSWER NO.
yes.
DO YOU WANT TO MAKE OR BREAK CODES? ANSWER MAKE. OR ANSWER BREAK.
make.
ENTER CODE; I PROMISE NOT TO LOOK.
red.white.bleu.yellow.
ERROR; TRY AGAIN.
so.you.have.been.looking.
ERROR; TRY AGAIN.
red.white.blue.yellow.
MY FIRST PROBE IS: WHITE.YELLOW.BLACK.RED; SCORE: 0 BLACK AND 3 WHITE.
MY NEXT PROBE IS: BLACK.BLACK.RED.WHITE; SCORE: 0 BLACK AND 2 WHITE.
3 TRIES TO GO.
MY NEXT PROBE IS: BLUE.RED.WHITE.YELLOW; SCORE: 1 BLACK AND 3 WHITE.
2 TRIES TO GO.
MY NEXT PROBE IS: RED.WHITE.BLUE.YELLOW; SCORE: 4 BLACK AND 0 WHITE.
1 TRIES TO GO.
THE CODE MUST BE: RED.WHITE.BLUE.YELLOW.
DO YOU WANT ANOTHER GAME? ANSWER YES. OR ANSWER NO.
no.
MASTERMIND WAS PLEASED TO SERVE YOU.
YOU ARE NOW RETURNED TO PROLOG.
PLAY<-
<-stop.
```

Computational Linguistics and Computer Languages Vol. XIII. 1979.

EFFICIENT RESOLUTION THEOREM PROVING IN THE PROPOSITIONAL LOGIC

R.Fiby, J.Sokol and M.Sudolský

Institute of Technical Cybernetics, Slovak Academy of Sciences,

Dúbravská cesta 5, 809 31 Bratislava, Czechoslovakia

ABSTRACT

Given a clause set C, it is shown here how to resolve upon any set P of atoms at once, using minimal unsatisfiable clause sets. Further, the satisfiability- decision strategy "resolve upon $P_1, \dots, P_n$ one after the other" is described. The efficiency of this very simple complete strategy is demonstrated by an example. In conclusion, remarks on a connection with the lock strategy and on a computer implementation are done. The strategy which described here for the propositional logic only, can be uplifted immediately to the first-order logic.

1. Introduction

The type "1" used as an index means "one".

Automatic theorem proving is a traditional field of Artificial Intelligence [22] which is applicable in automatic programming and question answering [10]. The first efforts in this field were embodied in [13, 23, 36] for instance. A great progress has been made after discovering the resolution rule [27]. To obtain efficient algorithms for theorem proving several resolution strategies were invented: semantic [5, 21, 26, 28-29, 31-32, 34, 38], merging [2-3, 25], lock [4], linear [2, 16, 17–20, 25, 33] and others [8–9, 12, 37, 39–40]. Nevertheless, their efficiency is not wholly satisfactory. The present resolution strategy seems to be more hopeful.
Strictly speaking, it is described here for unsatisfiability–decision because theorem proving and unsatisfiability–decision are equivalent [10].

The present strategy is based on the following observation: If P is a set of atoms appearing in a clause set C then the set $C((P))$ of all the clauses obtained from C by resolving upon all the elements of P at once is unsatisfiable iff C is unsatisfiable. To resolve upon at once, minimal unsatisfiable sets are considered. Such sets were encountered previously in another connection [2, 15].

The stategy which is outlined here implicitly, can be characterized by the statement "resolve upon $P_1, \dots, P_n$ one after the other". The efficiency of this very simple complete strategy is demonstrated by an example. In conclusion, remarks on a connection with the lock strategy and on a computer implementation are done.

The strategy was implemented in PDP–11/40 for the case when $P_1, \dots, P_n$ are one–element sets. In this case, the strategy is very suitable also for hand computation. The computational complexity of the strategy was not investigated. Remark that the computational complexity in the propositional logic was discussed in [6–7, 11, 24], for instance.

The strategy which is described here for the propositional logic only, can be uplifted immediately to the first-order logic, using for instance the Shostak theorem [30] which states: a clause set C is unsatisfiable (in the sense of the first-order logic) iff there is an unsatisfiable (in the sense of the propositional logic) general instance of C. Remark that automatic theorem proving in the propositional logic especially was investigated in a few papers only [1, 14, 35].

For the sake of greater clarity, literals and clauses (disjunctions of literals) are treated here more elementary than usually. The following preliminaries are done from the same reason.

Agreement. Throughout the paper:

1. A denotes a finite alphabet.

2. $+$ and $-$ denote two different letters from A.

3. $\boldsymbol{W}$ denotes a finite set of finite words over A; elements of $\boldsymbol{W}$ are called atoms.

4. $\widetilde{P}$ denotes the complement of a subset P of $\boldsymbol{W}$.

5. $|S|$ denotes the number of all the elements of a set S.

6. $\square$ denotes the empty clause.

Definition 1.

1. L is said to be a literal iff $L = +W$ or $L = -W$ for some atom W.

2. $\boldsymbol{C}$ is said to be a clause iff $\boldsymbol{C}$ is a set of literals.

3. A set I of literals is said to be an interpretation iff:

    1. $+W$ or $-W$ is from I for each atom W.
    2. There is no atom W such that $+W$ and $-W$ are from I.

4. An interpretation I is said to be a model of a clause set C iff each cluse from C contains a literal from I.

5. A clause set C is said to be satisfiable iff there is a model of C.

6. A clause set C' is said to be a consequence of a clause set C iff each model of C is a model of C'.

7. An unsatisfiable set C is said to be minimal iff $C - \{\boldsymbol{C}\}$ is satisfiable for each $\boldsymbol{C}$ from C.

Lemma 1.

A clause set C is unsatisfiable iff C contains a minimal unsatisfiable subset.

Proof.

1. Let C be unsatisfiable.

Suppose that each subset of C is not a minimal unsatisfiable set. Denote by m the number $|C|$. Evidently, $m \geqslant 2$. There are clauses $\boldsymbol{C}_1, \dots, \boldsymbol{C}_m$ such that:

1. $\boldsymbol{C}_1$ is from C and $C - \{\boldsymbol{C}_1\}$ is unsatisfiable.

2. $\boldsymbol{C}_2$ is from $C - \{\boldsymbol{C}_1\}$ and $C - \{\boldsymbol{C}_1, \boldsymbol{C}_2\}$ is unsatisfiable.

.

.

.

m. $\boldsymbol{C}_m$ is from $C - \{\boldsymbol{C}_1, \dots, \boldsymbol{C}_{m-1}\}$ and $C - \{\boldsymbol{C}_1, \dots, \boldsymbol{C}_m\}$ is unsatisfiable.

The first parts of these assertions imply the emptiness of $C - \{\boldsymbol{C}_1, \dots, \boldsymbol{C}_m\}$. However, the empty set of clauses is satisfiable. Contradiction.

2. Let C contain a minimal unsatisfiable subset. The unsatisfiability of C is evident.

Lemma 2.

Let $\boldsymbol{W} = \{W\}$. Then C is a minimal unsatisfiable set iff

$$C = \{\Box\} \text{ or } C = \{\{+W\}, \{-W\}\}.$$

Proof.

In the list of all the clause sets delete each satisfiable set. In the new list delete each clause set which contains another clause set as a proper subset. The remaining list consists of the clauses $\{\Box\}$, $\{\{+W\}, \{-W\}\}$.

Definition 2.

Let P be a subset of $\boldsymbol{W}$.

1. L is said to be a P-literal iff $L = +W$ or $L = -W$ for some W from P.

2. $\boldsymbol{C}$ is said to be a P-clause iff $\boldsymbol{C}$ is a set of P-literals.

3. The P-segment of a clause $\boldsymbol{C}$ is the set $\boldsymbol{C}[P]$ consisting of all the P-literals from $\boldsymbol{C}$.

4. The P-segment of a clause set C is the set $C[P]$ consisting of all $\boldsymbol{C}[P]$, where $\boldsymbol{C}$ runs all the elements of C.

Definition 3.

1. A clause $\boldsymbol{C}$ is said to be a tautology iff $+W$ and $-W$ are from $\boldsymbol{C}$ for some atom W.

2. A clause $\boldsymbol{C}$ from a clause set C is said to be a redundant of C iff there is a clause C^* from C such that $\boldsymbol{C}^*$ is a proper subset of $\boldsymbol{C}$.

3. $(\boldsymbol{C},K)$ is said to be a quasi–clause iff C is a clause and K is a clause set.

4. The quasi-clause closure of a clause set C is the quasi–clause set $\widetilde{C}$ consisting of all the pairs $(\boldsymbol{C}, \{ \boldsymbol{C} \})$, where $\boldsymbol{C}$ runs C.

2. *P*–resolvents

To resolve upon any set atoms at once, *P*-resolvents are introduced here. Theorem 1 points out the representativity of *P*-resolvents as regards consequences. At *P*-resolving, both satisfiability and unsatisfiability are preserved; it is stated in Theorem 2. The strategy "resolve upon $P_1, \ldots, P_n$ one after the other" is outlined implicitly in Theorem 3. Together with Lemma 1 and Lemma 2, Theorem 4 provides a powerful tool for finding all the minimal unsatisfiable subsets of a given clause set. Theorem 5 concerns "succesive resolving versus mass resolving".

Definition 4.

Let P be a subset of $\boldsymbol{W}$. The *P*-resolvent of a clause set C is the $\widetilde{P}$-clause set $C((P))$ defined, as follows: $\boldsymbol{C}$ is from $C((P))$ iff there are $\boldsymbol{C}_1, \ldots, \boldsymbol{C}_m$ from C such that:

1. $\boldsymbol{C}_r[P] \neq \boldsymbol{C}_s[P]$ for $r \neq s$.

2. $\{ \boldsymbol{C}_1[P], \ldots, \boldsymbol{C}_m[P] \}$ is a minimal unsatisfiable set.

3. $\boldsymbol{C} = \boldsymbol{C}_1[\widetilde{P}] \cup \ldots \cup \boldsymbol{C}_m[\widetilde{P}]$.

Theorem 1.

Let P be a subset of $\boldsymbol{W}$. A $\widetilde{P}$-clause set C' is a consequence of a clause set C iff C' is a consequence of $C((P))$.

Proof.

1. Let C' be a consequence of C.
Let M be an arbitrary model of $C((P))$. Denote by C^* the clause set defined as follows: $\boldsymbol{C}$ is from C^* iff $\boldsymbol{C}$ is from C and $\boldsymbol{C}$ does not contain any literal from $M[\widetilde{P}]$. Evidently, if $\boldsymbol{C}$ is from $C^*((P))$, then:

1. $\boldsymbol{C}$ does not contain any literal from $M[P]$.

2. $\boldsymbol{C}$ does not contain any literal from $M[\widetilde{P}]$.

It means that $\boldsymbol{C}$ does not contain any literal from M. Since M is also a model of $C^*((P))$ it means further that $C^*((P))$ is empty. Therefore, according to Definition 4, the set $C^*[P]$ does not contain any minimal unsatisfiable subset. Therefore, according to Lemma 1, the set $C^*[P]$ is satisfiable. Thus, there is a model M^* of $C^*[P]$. Evidently, $M^*[P] \cup M[\widetilde{P}]$ is a model of C. Therefore, $M^*[P] \cup M[\widetilde{P}]$ is a model of C'. Therefore, M is a model of C'.

2. Let C' be a consequence of $C((P))$.

Let M be an arbitrary model of C. The conditions 2–3 of Definition 4 imply that each clause from $C((P))$ contains a literal from M. Thus, M is a model of $C((P))$. Therefore, M is a model of C'.

Theorem 2.

Let P be a subset of $\boldsymbol{W}$. Then a clause set C is satisfiable iff $C((P))$ is satisfiable.

Proof.

1. Let C be satisfiable.

According to Theorem 1, the set $C((P))$ is a consequence of C. Therefore, $C((P))$ is statisfiable.

2. Let $C((P))$ be satisfiable.

Thus, there is a model M of $C((P))$. Define the clause set C^* as in the foregoing proof. As above, the following assertion can be proved: There is a model M^* of $C^*[P]$ such that $M^*[P] \cup M[\widetilde{P}]$ is a model of C. Thus, C is satisfiable.

Theorem 3.

. Let $P_1 \cup \ldots \cup P_n = \boldsymbol{W}$. Then:

1. A clause set C is satisfiable iff some of the sets

   $C,\ C((P_1)), \ldots, C((P_1)), \ldots, ((P_n))$

   is empty.

2. A clause set C is unsatisfiable iff some of the sets

   $C,\ C((P_1)), \ldots, C((P_1)) \ldots ((P_n))$

   contains $\square$ as an element.

Proof.

At beginning, note that $C((P_1))\ldots((P_n))$ is empty or consists of $\square$.

1. According to Theorem 1, the set C is satisfiable iff $C((P_1)),\ldots,((P_n))$ is satisfiable. The set $C((P_1))\ldots((P_n))$ is satisfiable iff $C((P_1))\ldots((P_n))$ is empty. However, $C((P_1))\ldots((P_n))$ is empty iff some of the sets C, $C((P_1)),\ldots,C((P_1))\ldots((P_n))$ are empty.

2. According to Theorem 1, the set C is unsatisfiable iff $C(P_1)),\ldots,((P))$ is unsatisfiable. The set $C((P_1)),\ldots,((P_n))$ is unsatisfiable iff $C((P_1,\ldots,((P_n))$ consists of $\square$. However, $C((P_1)),\ldots,((P_n))$ consists of $\square$ iff some of the sets C, $C(P_1)),\ldots,C((P_1)),\ldots,((P_n))$ contain $\square$ as an element.

Definition 5.

Let P be a subset of $\boldsymbol{W}$. The P–quasi–resolvent of a quasi–clause set C is the quasi–clause set $C((P))$ defined as follows: $(\boldsymbol{C},K)$ is from $C((P))$ iff there are $(\boldsymbol{C}_1,K_1),\ldots,(\boldsymbol{C}_m,K_m)$ from C such that:

1. $\boldsymbol{C}_r[P]\neq\boldsymbol{C}_s[P]$ for $r\neq s$.

2. $\{\boldsymbol{C}_1[P],\ldots,\boldsymbol{C}_m[P]\}$ is a minimal unsatisfiable set.

3. $\boldsymbol{C}=\boldsymbol{C}_1[\widetilde{P}]\cup,\ldots,\cup\,\boldsymbol{C}_m[\widetilde{P}]$ and $K=K_1\cup,\ldots,\cup K_m$.

Theorem 4.

Let C be a clause set and $P_1\cup,\ldots,\cup P_n=\boldsymbol{W}$.

Then:

1. If U is a minimal unsatisfiable subset of C then

   $(\square,U)$ is from $\overline{C}((P_1)),\ldots,((P_n))$.

2. If $(\boldsymbol{C},K)$ is from $\overline{C}((P_1),\ldots,((P_n))$ then $\boldsymbol{C}=\square$ and contains a minimal unsatisfiable subset of C.

Proof.

1. Let U be a minimal statisfiable subset of C. Using induction, one can show easily: If $1\leqslant i\leqslant n$, then:

   1. $\boldsymbol{C}$ is from $U((P_1)),\ldots,((P_i))$ iff there is K such that $(\boldsymbol{C},K)$ is from $\overline{U}((P_1)),\ldots,((P_i))$.

   2. If $(\boldsymbol{C},K)$ is from $\overline{U}((P_1)),\ldots,((P_i))$ then $\boldsymbol{C}$ is from $K((P_1)),\ldots,((P_i))$.

According to Theorem 3, the set $U((P_1)), \ldots, ((P_n))$ contains $\square$ as an element. Therefore, there is K such that $(\square, K)$ is from $\overline{U}((P_1)), \ldots, ((P_n))$. It implies that $\square$ is from $K((P_1)), \ldots, ((P_n))$. According to Theorem 3, the set K is unsatisfiable. Since K is a subset of U, it implies $K = U$. Thus, $(\square, U)$ is from $\overline{C}((P_1)), \ldots, ((P_n))$.

2. Let $(\boldsymbol{C}, K)$ be from $\overline{C}((P_1)), \ldots, ((P_n))$.

Using induction as above, one can show easily that $\boldsymbol{C}$ is from $K((P_1)), \ldots, ((P_n))$. Therefore, $\boldsymbol{C} = \square$. Further, according to Theorem 3 and Lemma 1, the set K contains a minimal unsatisfiable subset of C.

Theorem 5.

Let P_1, P_2 be subset of $\boldsymbol{W}$. Then:

1. $C((P_1))((P_2))$ contains $C((P_1 \cup P_2))$ as a subset.

2. Each clause from $C((P_1))((P_2))$ contains a clause from $C((P_1 \cup P_2))$ as a subclause.

Proof.

1. Let $\boldsymbol{C}$ be an arbitrary clause from $C((P_1 \cup P_2))$.

According to Definition 4, there are $\boldsymbol{C}_1, \ldots, \boldsymbol{C}_m$ from C such that:

1. $\boldsymbol{C}_r[P_1 \cup P_2] \neq \boldsymbol{C}_s[P_1 \cup P_2]$ for $r \neq s$.

2. $\{\boldsymbol{C}_1[P_1 \cup P_2], \ldots, \boldsymbol{C}_m[P_1 \cup P_2]\}$ is a minimal unsatisfiable set.

3. $\boldsymbol{C} = \boldsymbol{C}_1[\widetilde{P}_1 \cap \widetilde{P}_2] \cup \ldots \cup \boldsymbol{C}_m[\widetilde{P}_1 \cap \widetilde{P}_2]$.

Denote by C^* the set consisting of the clauses $\boldsymbol{C}_1, \ldots, \boldsymbol{C}_m$. According to Theorem 2, the set $C^*[P_1 \cup P_2]((P_1))$ is unsatisfiable. According to Lemma 1, the set $C^*[P_1 \cup P_2]((P_1))$ contains a minimal unsatisfiable subset. One can show easily that $C^*[P_1 \cup P_2]((P_1)) =$
$= C^*((P_1))[P_2]$. Thus, there are clauses $\boldsymbol{C}_1^*, \ldots, \boldsymbol{C}_n$ from $C^*((P_1))$ such that:

1. $\boldsymbol{C}_r^*[P_2] \neq \boldsymbol{C}_s^*[P_2]$ for $r \neq s$.

2. $\{\boldsymbol{C}_1^*[P_2], \ldots, \boldsymbol{C}_n^*[P_2]\}$ is a minimal unsatisfiable set.

Denote by C^* the clause $\boldsymbol{C}_1^*[\widetilde{P}_2] \cup \ldots \cup \boldsymbol{C}_n^*[\widetilde{P}_2]$ from $C^*((P_1))((P_2))$.
Let i be an arbitrary index from $1, \ldots, m$. The set $(C^* - \{\boldsymbol{C}_i\})[P_1 \cup P_2]$ is satisfiable because $C^*[P_1 \cup P_2]$ is a minimal unsatisfiable set. Therefore, according to Theorem 2, the set $(C^* - \{\boldsymbol{C}_i\})[P_1 \cup P_2]((P_1))$ is satisfiable. One can show easily that

$$(C^* - \{\boldsymbol{C}_i\})[P_1 \cup P_2]((P_1)) = (C^* - \{\boldsymbol{C}_i\})((P_2))((P_2)).$$

Therefore, according to Definition 4, the set $(C^* - \{\boldsymbol{C}_i\})\ ((P_1))\ ((P_2))$ is empty. It implies that $\boldsymbol{C}^*$ contains $\boldsymbol{C}_i[\widetilde{P}_1]\ [P_2]$ as a subclause. Because $\boldsymbol{C}_i[\widetilde{P}_1]\ [\widetilde{P}_2] = \boldsymbol{C}_i[\widetilde{P}_1 \cap \widetilde{P}_2]$, we have obtained that $\boldsymbol{C}^*$ contains $\boldsymbol{C}_1[\widetilde{P}_1 \cap \widetilde{P}_2], \ldots, \boldsymbol{C}_m[\widetilde{P}_1 \cap \widetilde{P}_2]$ as subclauses. Therefore, $\boldsymbol{C}$ contains $\boldsymbol{C}$ as a subclause. Because $\boldsymbol{C}^*$ is a subclause of $\boldsymbol{C}$, we obtain $\boldsymbol{C}^* = \boldsymbol{C}$. Thus, $\boldsymbol{C}$ is from $C^*((P_1))\ ((P_2))$, which is a subset of $C((P_1))\ ((P_2))$.

According to Definition 4, there are $\boldsymbol{C}_1^*, \ldots, \boldsymbol{C}_n^*$ from $C((P_1))$ such that:

1. $\boldsymbol{C}_r^*[P_2] \neq \boldsymbol{C}_s^*[P_2]$ for $r \neq s$.
2. $\{\boldsymbol{C}_1^*[P_2], \ldots, \boldsymbol{C}_n^*[P_2]\}$ is a minimal unsatisfiable set.
3. $\boldsymbol{C} = \boldsymbol{C}_1^*\ [\widetilde{P}_2] \cup, \ldots, \cup\ \boldsymbol{C}_n^*\ [\widetilde{P}_2]$.

Denote by C^* the clause set defined, as follows:

$\boldsymbol{C}^*$ is from C^* iff there are $\boldsymbol{C}_1, \ldots, \boldsymbol{C}_m$ from C such that:

1. $\boldsymbol{C}_r[P_1] \neq \boldsymbol{C}_s[P_1]$ for $r \neq s$.
2. $\{\boldsymbol{C}_1[P_1], \ldots, \boldsymbol{C}_m[P_1]\}$ is a minimal unsatisfiable set.
3. $\boldsymbol{C}_1[\widetilde{P}] \cup, \ldots, \boldsymbol{C}_m[\widetilde{P}_1]$ is some of the clauses $\boldsymbol{C}_1^*, \ldots, \boldsymbol{C}_n^*$.
4. $\boldsymbol{C}$ is some of the clauses $\boldsymbol{C}_1, \ldots, \boldsymbol{C}_m$.

Evidently, $C^*((P_1))$ contains $\boldsymbol{C}_1^*, \ldots, \boldsymbol{C}_n^*$ as elements. Thus, according to Definition 4, the set $C^*((P_1))\ [P_2]$ is unsatisfiable. One can show easily that $C^*((P_1))\ [P_2]$ is a subset of $C^*[P_1 \cup P_2]\ ((P_1))$. Thus, $C^*[P_1 \cup P_2]\ ((P_1))$ is unsatisfiable. Therefore, according to Theorem 2, the set $C^*[P_1 \cup P_2]$ is unsatisfiable. According to Definition 4, the set $C^*((P_1 \cup P_2))$ is not empty. Since each clause from $C^*((P_1 \cup P_2))$ is contained in $\boldsymbol{C}$, it means that $\boldsymbol{C}$ contains a subclause from $C^*((P_1 \cup P_2))$, which is a subset of $C((P_1 \cup P_2))$.

3. Reduced P-resolvents

Following the previous exposition, we describe here some important properties of reduced P-resolvents, which are obtained from P-resolvents by deleting tautologies and redundants. Using them, we improve the strategy "resolve upon $P_1, \ldots, P_n$ one after the other".

Definition 6.

Let P be a subset of $\boldsymbol{W}$. The reduced P-resolvent of a clause set C is the $\widetilde{P}$-clause se $C(P)$ defined as follows:

$\boldsymbol{C}$ is from $C(P)$ iff:

1. $\boldsymbol{C}$ is from $C((P))$.

2. $\boldsymbol{C}$ is not a tautology.

3. $\boldsymbol{C}$ is not a redundant of $C((P))$.

Theorem 6.

Let P be a subset of $\boldsymbol{W}$. A $\widetilde{P}$-clause set C' is a consequence of a clause set C iff C' is a consequence of $C(P)$.

Proof.

It follows immediately from Theorem 1 and the observation that $C(P)$ is a consequence of $C((P))$ and $C((P))$ is a consequence of $C(P)$.

Theorem 7.

Let P be a subset of $\boldsymbol{W}$. Then a clause set C is satisfiable iff $C(P)$ is satisfiable.

Proof.

It follows immediately from Theorem 2 and the observation that M is a model of $C(P)$ iff M is a model of $C((P))$.

Theorem 8.

Let $P_1 \cup, \ldots, \cup P_n = \boldsymbol{W}$. Then:

1. A clause set C is satisfiable iff some of the sets $C, C(P_1), \ldots, C(P_1), \ldots, (P_n)$ is empty.

2. A clause set C is unsatisfiable iff some of the sets $C, C(P_1), \ldots, C(P_1), \ldots, (P_n)$ contains as an element.

Proof.

It follows the proof of Theorem 3.

Definition 7.

Let P be a subset of $\boldsymbol{W}$. The reduced P-quasi-resolvent of a quasi-clause set C is the quasi-clause set $C(P)$ defined as follows:
(C,K) is from $C(P)$ iff:

1. (C,K) is from $C((P))$.

2. $\boldsymbol{C}$ is not a tautology.

3. If $(\boldsymbol{C}^*, K^*)$ is from $\boldsymbol{C}((P))$ and $\boldsymbol{C}^* = \boldsymbol{C}$, then K does not contain K^* as a proper subset.

Theorem 9.

Let C be a clause set and $P_1 \cup, \dots, \cup P_n = \boldsymbol{W}$.
Then:

1. If U is a minimal unsatisfiable subset of C then $(\square, U)$ is from $\overline{C}(P_1), \dots, (P_n)$.

2. If $(\boldsymbol{C},K)$ is from $C(P_1), \dots, (P_n)$ then $\boldsymbol{C} = \square$ and K is a minimal unsatisfiable subset of C.

Proof.

1. In the proof of Theorem 4, replace all the P-resolvents by P-resolvents without tautologies.

2. Let $(\boldsymbol{C},K)$ be from $\overline{C}(P_1), \dots, (P_n)$.

According to Theorem 4, $\boldsymbol{C} = \square$ and K contains a minimal unsatisfiable subset U of C. According to the same theorem, $(\square, U)$ is from $\overline{C}((P_1)), \dots, ((P_n))$. Therefore, $K = U$.

Theorem 10.

Let P_1, P_2 be subsets of $\boldsymbol{W}$. Then $C(P_1)(P_2) = C(P_1 \cup P_2)$.

Proof.

In both $C((P_1))((P_2))$ and $C((P_1 \cup P_2))$ delete all the tautologies and redundants. Theorem 5 implies that the remaining sets coincide, i.e. $C((P_1))(P_2) = C(P_1 \cup P_2)$; but $C((P_1))(P_2) = C(P_1)(P_2)$.

4. Examples

The improved strategy "resolve upon $P_1, \dots, P_n$ one after the other" is now demonstrated by Example 1. Finding all the minimal unsatisfiable subsets of a given clause set is demonstrated by Example 2. In Example 3 it is shown that in general $C((P_1))\ ((P_2)) \neq C((P_1 \cup P_2))$.

To handle clauses more easily, we represent here clauses by vectors over the set $\{+, 0, -\}$. This is possible because tautologies are ignored here. For instance, if $\boldsymbol{W} = \{W_1, \dots, W_7\}$ then clause $\{+W_1, W_2, +W_4, -W_5\}$ is represented by vector

[+ – 0 + – 0 0]. Similarly, we represent clauses sets by matrices over $\{+, 0, -\}$. In this representation clauses and rows are in the one–to–one correspondence determined by their enumerations. Note that our representation of clauses differs a somewhat from that of Yelowitz and Kandel [40].

Example 1.

Let $\boldsymbol{W} = \{W_1, \ldots, W_6\}$. We are given the clause set C, represented by the matrix

$$\begin{bmatrix} + & + & 0 & 0 & 0 & 0 \\ + & 0 & + & 0 & 0 & 0 \\ + & 0 & 0 & + & 0 & 0 \\ - & 0 & 0 & 0 & - & 0 \\ - & 0 & 0 & 0 & 0 & - \\ 0 & + & + & 0 & 0 & 0 \\ 0 & + & 0 & + & 0 & 0 \\ 0 & - & 0 & 0 & - & 0 \\ 0 & - & 0 & 0 & 0 & - \\ 0 & 0 & + & + & 0 & 0 \\ 0 & 0 & - & 0 & - & 0 \\ 0 & 0 & - & 0 & 0 & - \\ 0 & 0 & 0 & - & + & 0 \\ 0 & 0 & 0 & - & 0 & + \\ 0 & 0 & 0 & 0 & + & + \end{bmatrix}$$

to decide whether or not C is satisfiable.

Set $P_i = \{W_i\}$ for $i = 1, \ldots, 6$. Applying Definition 6 and Lemma 2, we obtain that $C(P_1), \ldots, C(P_1), \ldots, (P_6)$ are represented by the matrices

$$\begin{bmatrix} 0 & + & + & 0 & 0 & 0 \\ 0 & + & 0 & + & 0 & 0 \\ 0 & - & 0 & 0 & - & 0 \\ 0 & - & 0 & 0 & 0 & - \\ 0 & 0 & + & + & 0 & 0 \\ 0 & 0 & - & 0 & - & 0 \\ 0 & 0 & - & 0 & 0 & - \\ 0 & 0 & 0 & - & + & 0 \\ 0 & 0 & 0 & - & 0 & + \\ 0 & 0 & 0 & 0 & + & + \\ 0 & + & 0 & 0 & - & 0 \\ 0 & + & 0 & 0 & 0 & - \\ 0 & 0 & + & 0 & - & 0 \\ 0 & 0 & + & 0 & 0 & - \\ 0 & 0 & 0 & + & - & 0 \\ 0 & 0 & 0 & + & 0 & - \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & + & + & 0 & 0 \\ 0 & 0 & 0 & - & + & 0 \\ 0 & 0 & 0 & - & 0 & + \\ 0 & 0 & 0 & 0 & + & + \\ 0 & 0 & 0 & 0 & - & 0 \\ 0 & 0 & 0 & 0 & 0 & - \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 & 0 & + & + \\ 0 & 0 & 0 & 0 & - & 0 \\ 0 & 0 & 0 & 0 & 0 & - \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 & - & + & 0 \\ 0 & 0 & 0 & - & 0 & + \\ 0 & 0 & 0 & 0 & + & + \\ 0 & 0 & 0 & 0 & - & 0 \\ 0 & 0 & 0 & 0 & 0 & - \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & - \\ 0 & 0 & 0 & 0 & 0 & + \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Thus, according to Theorem 8, the set C is unsatisfiable.

Example 2.

Let $\boldsymbol{W} = \{ W_1, W_2 \}$. We are given the clause set C represented by the matrix

$$\begin{bmatrix} + & + \\ + & 0 \\ + & - \\ 0 & + \\ 0 & 0 \\ 0 & - \\ - & + \\ - & 0 \\ - & - \end{bmatrix}$$

to find all the minimal unsatisfiable subsets.

Set $P_i = \{ W_i \}$ for $i = 1,2$. The clause set $\overline{C}$ is represented by the matrix

$$\begin{bmatrix} + & + & \{1\} \\ + & 0 & \{2\} \\ + & - & \{3\} \\ 0 & + & \{4\} \\ 0 & 0 & \{5\} \\ 0 & - & \{6\} \\ - & + & \{7\} \\ - & 0 & \{8\} \\ - & - & \{9\} \end{bmatrix}$$

where the number i is written instead of the i-th row of matrix of C. Applying Definition 7 and Lemma 2, we obtain $\overline{C}(P_1)$, $\overline{C}(P_1)\,(P_2)$ represented by the matrices

$$\begin{bmatrix} 0 & + & \{4\} \\ 0 & 0 & \{5\} \\ 0 & - & \{6\} \\ 0 & + & \{1, 7\} \\ 0 & + & \{1, 8\} \\ 0 & + & \{2, 7\} \\ 0 & 0 & \{2, 8\} \\ 0 & - & \{2, 9\} \\ 0 & - & \{3, 8\} \\ 0 & - & \{3, 9\} \end{bmatrix} \qquad \begin{bmatrix} 0 & 0 & \{5\} \\ 0 & 0 & \{2, 8\} \\ 0 & 0 & \{4, 6\} \\ 0 & 0 & \{2, 4, 9\} \\ 0 & 0 & \{3, 4, 8\} \\ 0 & 0 & \{3, 4, 9\} \\ 0 & 0 & \{1, 6, 7\} \\ 0 & 0 & \{1, 6, 8\} \\ 0 & 0 & \{2, 6, 7\} \\ 0 & 0 & \{1, 3, 7, 9\} \\ 0 & 0 & \{1, 3, 8\} \\ 0 & 0 & \{2, 7, 9\} \end{bmatrix}$$

Thus, according to Theorem 9, the right-most column of the last matrix represents all the minimal unsatisfiable subsets of C.

Example 3.

Let $W = \{\mathbf{W}_1, \ldots, \mathbf{W}_n\}$. Let C be the clause set represented by the matrix

$$\begin{matrix} + & - & 0 & + \\ - & 0 & + & 0 \\ + & 0 & + & 0 \\ - & + & 0 & + \end{matrix}$$

and $P_1 = \{\mathbf{W}_1\}$, $P_2 = \{\mathbf{W}_2\}$. One can show easily that

$$C((P_1))\,((P_2)) \neq C((P_1 \cup P_2)).$$

5. Concluding remarks

We compare here the present strategy with the lock one [4]. Further, we describe such a clause representation which allows us to handle clauses efficiently by parallel computers. At last, computer P-resolving large clause sets is submitted.

Remark 1.

Denote by $\boldsymbol{W}_1, \ldots, \boldsymbol{W}_n$ all the elements of W.

1. Let i be an arbitrary index from $1, \ldots, n$. Assign to clauses $\{+\boldsymbol{W}\} \cup K_1$ and $\{-\boldsymbol{W}\} \cup K_2$ the clause $K_1 \cup K_2$ iff:

   1. i is the index of $\boldsymbol{W}$;

   2. If i_1 and i_2 are indices of any two atoms appearing in K_1 and K_2 correspondingly, then $i \leqslant i_1$ and $i \leqslant i_2$.

Call $K_1 \cup K_2$ the i-resolvent of $\{+\boldsymbol{W}\} \cup K_1$ and $\{-\boldsymbol{W}\} \cup K_2$.

2. Enumerate $+\boldsymbol{W}_1, -\boldsymbol{W}_1, \ldots, +\boldsymbol{W}_n, -\boldsymbol{W}_n$ by the integers $1, 1, \ldots, n, n$, correspondingly. With respect to this enumeration of literals, each i-resolvent is a lock resolvent and each lock resolvent is an i-resolvent for some i.

3. Let C be a clause set. A clause $\boldsymbol{C}$ is from $C((\{\boldsymbol{W}_i\}))$ iff: (1) $\boldsymbol{C}$ is from C and $\boldsymbol{W}_i$ does not appear in $\boldsymbol{C}$, or (2) $\boldsymbol{C}$ is the i-resolvent of some clauses $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ from C. Thus, the non-improved strategy "resolve upon $\{\boldsymbol{W}_1\}, \ldots, \{\boldsymbol{W}_n\}$ one after the other" is substantially the exhaustive and successive application of the lock resolution rule.

Remark 2.

Denote by $\boldsymbol{W}_1, \ldots, \boldsymbol{W}_n$ all the elements of W. Further, denote by B the Boolean algebra over $\{0, 1\}$. At last, denote by B_n the Cartesian product of n exemplars of $B \times B$. Assign to each clause $\boldsymbol{C}$ the element $\bar{\boldsymbol{C}}$ of B_n defined as follows:

If $1 \leqslant i \leqslant n$, then:

1. $\bar{\boldsymbol{C}}_i = 00$ iff: 1. $+\boldsymbol{W}_i$ is not from $\boldsymbol{C}$;
   2. $-\boldsymbol{W}_i$ is not from $\boldsymbol{C}$.

2. $\bar{\boldsymbol{C}}_i = 01$ iff: 1. $+\boldsymbol{W}_i$ is not from $\boldsymbol{C}$;
   2. $-\boldsymbol{W}_i$ is from $\boldsymbol{C}$.

3. $\bar{\boldsymbol{C}}_i = 10$ iff 1. $+\boldsymbol{W}_i$ is from $\boldsymbol{C}$;
   2. $-\boldsymbol{W}_i$ is not from $\boldsymbol{C}$.

4. $\bar{\boldsymbol{C}}_i = 11$ iff 1. $+\boldsymbol{W}_i$ is from $\boldsymbol{C}$;
   2. $-\boldsymbol{W}_i$ is from $\boldsymbol{C}$.

This assignement is an isomorphism from the Boolean algebra of all the clauses onto the Boolean algebra B_n. Under this assignment, clauses can be handled efficiently by $2n$ parallel elementary Boolean processors.

Remark 3.

We submit to produce the P-resolvent of a large clause set C as follows:

1. Represent C as the set S of all the pairs $(\boldsymbol{C}, K)$, where $\boldsymbol{C}$ is from $C[P]$ and K consists of all the clauses $\boldsymbol{K}$ from $C[\tilde{P}]$ such that $\boldsymbol{C} \cup \boldsymbol{K}$ is from C.

2. Find all the minimal unsatisfiable subsets of $C[P]$, using Theorem 9.

3. For each minimal unsatisfiable subset $\{\boldsymbol{C}_1, \ldots, \boldsymbol{C}_m\}$ of $C[P]$ such that $\boldsymbol{C}_r \neq \boldsymbol{C}_s$ for $r \neq s$:

   1. Find all $K_1, \ldots, K_m$ such that $(\boldsymbol{C}_1, K_1), \ldots, (\boldsymbol{C}_m, K_m)$ are from S.

   2. Include in $C((P))$ all the clauses $\boldsymbol{K}_1 \cup, \ldots, \cup \boldsymbol{K}_m$ where $\boldsymbol{K}_1$ runs $K_1; \ldots; \boldsymbol{K}_m$ runs K_m.

Remember that $\boldsymbol{C}$'s can be stored in the core memory, K's on the disks; the correspondence between $\boldsymbol{C}$'s and K's can be expressed by pointers from $\boldsymbol{C}$'s onto K's.

Acknowledgement

We acknowledge our colleagues E. Kostolanský, J. Mikloško, D. Ondruš, J. Šajda and I. Weigl for their kind encouragement and help during the preparation of this paper.

R e f e r e n c e s

[1] Amarel, S.: An approach to heuristic problem solving and theorem proving in the propositional calculus.
Proc. Conf. Systems and Computer Science (London, Ont., 1965).

[2] Anderson, R., and Bledsoe, W.W.: A linear format for resolution with merging and a new technique for establishing completeness. J. ACM 17 (1970), 525-534.

[3] Andrews, P.B.: Resolution with merging. J. ACM 15 (1968), 367-381.

[4] Boyer, R.S.: "Locking: A restriction of resolution." University of Texas at Austin Ph.D. Thesis (1971).

[5] Cantaralla, R.D.: Efficient semantic resolution proofs based upon binary semantic trees. Syracuse University Ph.D. Thesis (1969).

[6] Ceitin, G. S.: The complexity of a deduction in the propositional calculus. (Russian) Zap. Naučn. Sem. Leningrad. Otdel. Mat. Inst. Steklov. 8 (1968), 234-259.

[7] Ceitin, G.S., and Čubarjan, A.A.: Certain estimates of the lenghts of logical deduction in classical propositional calculus. (Russian) Dokl. Akad. Nauk Armjan. SSR 55 (1972), 10-12.

[8] Chang, C.L.: The unit proof and input proof in theorem proving. J. ACM 17 (1970), 698-707.

[9] Chang, C.L.: Theorem proving with variable constrained resolution. Information Sciences 4 (1972), 217-231.

[10] Chang, C.L., and Lee, R.C.T.: Symbolic logic and mechanical theorem proving. Academic Press, New York and London (1973).

[11] Chapin, E. W., Jr.: Measures of centrality and complexity for partial propositional calculi. Arch. Math. Logik Grundlagenforsch. 15 (1972), 7-18.

[12] Davis, M.: Eliminating the irrelevant from mechanical proofs. Proc. Symp. Appl. Mathem. 18 (1963).

[13] Davis, M. and Putnam, H.: A computing procedure for quantification theory. J. ACM 7 (1960), 201-215.

[14] Ehrenfreucht, A., and Orlowska, E.: Mechanical proof procedure for propositional calculus. Bull. Acad. Polon. Sci. Sér. Sci. Math. Astronom. Phys. 15 (1967), 25-30.

[15] Henschen, L., and Wos, L.: Unit refutation and Horn sets. J. ACM 21 (1974), 590-605.

[16] Kowalski, R., and Kuehner, D.: Linear resolution with selection function. Artificial Intelligence 2 (1971), 227-260.

[17] Loveland, D. W.: "Some linear Herbrand proof procedures: An analysis." Dept. Computer Sciences, Carnegie–Mellon University (1970).

[18] Loveland, D. W.: A linear format for resolution. Proc. IRIA Symp. on Automatic Demonstration (Versailles, 1970).

[19] Loveland, D. W.: A unifying view of some linear Herbrand proof procedures. J. ACM 19 (1972), 366-384.

[20] Luckham, D.: Refinement theorems in resolution theory. Proc. IRIA Symp. on Automatic Demonstration (Versailles, 1970).

[21] Meltzer, B.: "Theorem proving for computers: Some results on resolution and renaming." Computer J. 8 (1966), 341-343.

[22] Nilsson, N.J.:Artificial intelligence. Proc. IFIP Cong. (1974).

[23] Prawitz, D.: An improved proof procedure. Theoria 26 (1960), 102-139.

[24] Rabin, M.O.: Theoretical impediments to artifical intelligence. Proc. IFIP Cong. (1974).

[25] Reiter, R.: Two results on ordering for resolution with merging and linear format. J. ACM 18 (1971), 630-646.

[26] Reiter, R.: A semantically guided deductive system for automatic theorem proving. Proc. 3IJCAI (1973).

[27] Robinson, J.A.: A machine-oriented logic based on the resolution principle. J. ACM 12 (1965), 23-41.

[28] Robinson, J.A.: Automatic deduction with hyper-resolution. Internat. J. Computer Math. 1 (1965), 227-234.

[29] Robinson, J.A.: The generalized resolution principle. Machine Intelligence 3, Michie (ed.), Edinburgh University Press, Edinburgh (1968).

[30] Shostak, R.E.: On the role of unification in mechanical theorem proving. Acta Informatica 7 (1977), 319-323.

[31] Slagle, J.R.: Automatic theorem proving with renamable and semantic resolution. J. ACM 14 (1967), 687-697.

[32] Slagle, J.R., Chang, C.L., and Lee, R.C.T.: Completeness theorems for semantic resolution in consequence finding. Proc. 1IJCAI (1969).

[33] Slagle, J.R., and Norton, L.: Experiments with an automated theorem proving, having partial ordering rules. Division of Computer Research and Tech., National Inst. of Health, Bethesda, Maryland (1971).

[34] Szczerba, S.W.: Semantic method of proving theorems. Bull. Acad. Polon. Sci. Sér. Sci. Math. Astronom. Phys. 18 (1970), 507-512.

[35] Van Westrhenen, S.C.: Two programmes for the calculus of propositional logic. Automation in Language Translation and Theorem Proving, Commission of the European Communities, Brussels (1968).

[36] Wang, H.: Toward mechanical mathematics. IBM Journal 4 (1960), 2-22.

[37] Wos, L.T., Carson, D.G., and Robinson, G.A.: The unit preference strategy in theorem proving. Proc. AFIPS, Vol. 26 (1964).

[38] Wos, L.T., Robinson, G.A. and Carson, D. G.: Efficiency and completeness in the set of support strategy in theorem proving. J. ACM 12 (1965), 536-541.

[39] Yates, R., Raphael, B., and Hart, T.: Resolution graphs. Artificial Intelligence 1 (1970), 224-239.

[40] Yelowitz, L., and Kandel, A.: New results and techniques in resolution theory. IEEE Trans. Comp. C-25 (1976), 673-677.

Computational Linguistics and Computer Languages Vol. XIII. 1979.

A 2D TRANSITION FUNCTION DEFINITION LANGUAGE FOR A SUBSYSTEM OF THE CELLAS CELLULAR PROCESSOR SIMULATION LANGUAGE

T. Legendi

Research Group on Automata Theory Hungarian Academy of Sciences

Szeged, Hungary

Abstract

Cellular processors are based on a homogeneous set of *parallel* – working, locally interconnected and *locally controlled* base *cells.* For realization *two dimensional* sets from *microcells* (2-16 state automata, 20-100 gates of logic) are proposed. The design of cellprocessors and preparation of their software systems need simulation tools. A very important simulation subsystem ensures the definition, the minimization and the (virtual) execution of local control. (the local transition functions define the behaviour of the whole set of cells; as their execution is sequentialized, they may be interpreted as cellular microprograms).

A specialized adequate definition language is proposed, where topological, geometrical, data-flow features of a transition function are expressed in 2-dimensional microconfigurations, while the operations on the moving data are defined by expressions (referencing variables of the microconfigurations). The evaluation of this (sub) language is shown, too.

The main objectives of the whole research project for the design, implementation and programming of cellular procesors are outlined in the reference (Legendi 1977a).

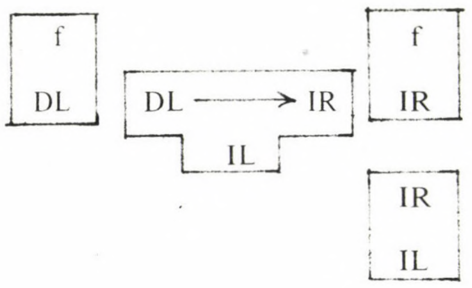
The motivations to define a new family of cellular processor simulation languages – rather than using the existing ones (Brender 1970; Baker – Herman 1970; Wu-Hung Lin 1972, Vollmar 1979) – and the structure of the proposed new languages are explained in detail in the references (Legendi 1975, 1977b, 1978b; Legendi – Molnár – Székely 1979).

This paper deals with the core of simulation systems – the local transition function (microprogram) definition sublanguage and mechanism.

There exist different approaches to the local function definition and execution system. It may consist of an external source language for function definition, a processor that transforms the source program into an internal representation which is used by a transition-executer program.

In extreme cases some components may be missing, or changing their function. For example for specific purpose it might be advantageous to define functions in algorithmic languages. In CELIA (Baker – Herman, 1970; Wu-Hung Liu 1972) FORTRAN rutines define and execute the local transition functions at the same time (with no internal function representation). For general purpose we find this method unsatisfactory – the cellular programmer is forced to think in some traditional sequential language being complicated and what is more, dangerous and in many cases ineffective to link user-written rutines to a system for common use.

So let us remain at the general scheme:



f – local function

DL – definition language

IR – internal representation

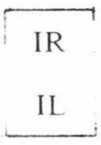
IL – implementation language

In our oppinion DL should be a highly problem-oriented, specialized and if possible, a high(er)-level language: this gives the opportunity for the cellular programmer to concentrate on his very topic and to write short, transparent, easy to read/debug/ modify programs.

The processor (DL → IR) should be intelligent – ensuring for the programmer a high-level input language while for the execution phase

it should generate a compressed, minimized size representation that executes fast at the same time. (The minimazation may be automatic, half-automatic, or programmed – e.g. user-directed.) These requirements are somewhat contradictory, it is not an easy task to find a good compromise to satisfy them.

The internal represantation of the local transition function is executed for each cell by the



program.

Thus the execution-phase needs minimal sized functions (especially, due to the inhomogeneous programming concept several dozens of functions should reside in the main core) and fast execution. In our oppinion, having fixed internal representation, fixed system rutines should do the job (for system safety and optimal execution time).

In summary, we propose, as a principle, a high-level function description language, an optimizing processor for it, standard internal representation and standard system rutines for the execution of the transitions.

How to evaluate these principles in practice?

As it will be shown, it was not a direct line, but a very logical way of evolution of a continuously expanding concrete language/system, that was developing during use, highly influenced by the advancing cellular programming experience and knowledge.

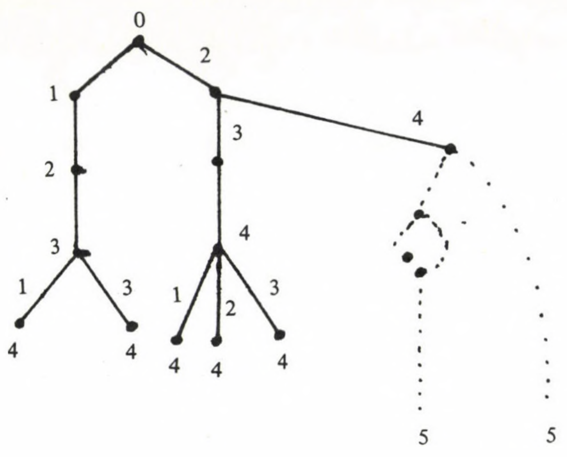
In the course of experimental work, different simple (see: Legendi, 1977b) and more complex definition languages were developed. (For the latter type, the TRANSCELL language serves as a good example it is a very specialized definition and user-directed minimization system; it has been developed as cross-software for a concrete 16 state specialized cell (see: Legendi 1978a, 1980a).)

The basis for the present system was the eldest, simplest term form. E.g. a function consists of a set of terms, one term contains the old state, the states of the neighbours (as conditions) and the new state is associated with the conditions. In the first implementation, the input language contained only terms, the internal representation was quite a big table containing only the new states, filled and searched in a fast way forming directly an address from the conditions and storing/retrieving there the corresponding new value.

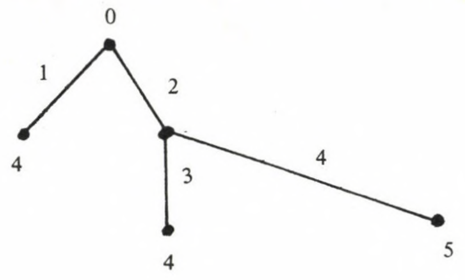
```
0 0000   ┌ . ┐
         │ . │
         │ . │
4 1234     5      =    4      1234     5
           .          old    neigh-   new
           .         state   bours    state
7777     └ . ┘
```

It was very easy to implement, fast in execution, but the input language was primitive, the table was enormous. Transition functons might be interpreted in a natural way as trees: the above table form is an internal representation of the whole tree. As in practice, we define only a relatively small part of the whole transition function, a logical decision was taken to use smaller tables: we should represent the (partial) tree of the transition function (in a two--dimensional table, for fast execution). This approach ensures the use of partial function tables (the original table was of fixed size, for complete functions). So the length of the table depends now on the concrete size of the function.

This representation gave a possibility for automatic minimization: if in some node we realize that all the leaves under it (new states) are identical, we can erase the lower level and write the new state into the node itself, thus making the tree and the table smaller and the execution faster.



the whole tree



the minimized tree

The disadvantage of this method is that in this case an "overdefinition" feature is obtained – originally undefined terms are made defined in this way. (So the system cannot alarm the programmer in case of using undefined terms during simulation time. It is uneffective to use the traditional function structure: state S turns into S_1 if C_1, S_2 if C_2 . . . otherwise S remains). In our earlier (and also in the advanced) cellular programming practice, however, it did not disturb the principle of using only explicitly defined terms/subfunctions. This system, including automatic minimization, had easily been implemented (see: Bolla, 1975), it was fast and memory saving, but with very low-level input (terms).

Everyday practice showed that in a lot of cases we had groups of terms which differed only in one (or more) states in the condition part; e.g. these groups could have been shortened using the OR-operation:

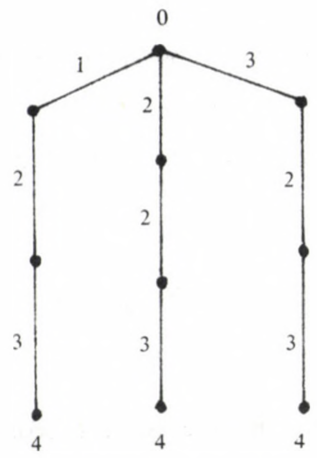
0 1 2 3 3 4	0 (1 2 3) 2 3 3 4	
0 2 2 3 3 4		(∗)
0 3 2 3 3 4	1 OR 2 OR 3	
old state neighbours new state		

0 1 2 3 1 4	0 (12) 2 3 (13) 4	
0 1 2 3 3 4		
		(∗)
0 2 2 3 1 4		
0 2 2 3 3 4		

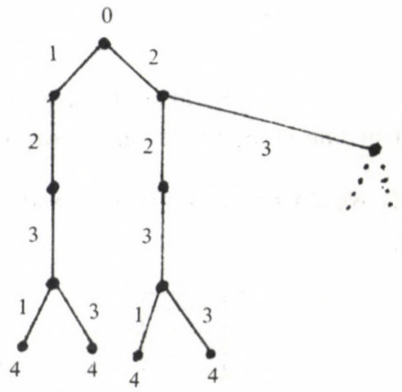
At the same time the internal representation influenced us to express the tree structure of the functions in the input language, too.

So, the following format had been used for the description of the local transition function (see: Sümeghy, 1977)

```
N0 = 0               N0 = 0
N1 = 1, 2, 3         N1 = 1, 2
N2 = 2               N2 = 2                ( * * )
N3 = 3               N3 = 3
  3*4                  1 * 4
                       3 * 4
                     N2 = 3
                       .
                       .
                       .
                       .
```



(the whole tree)



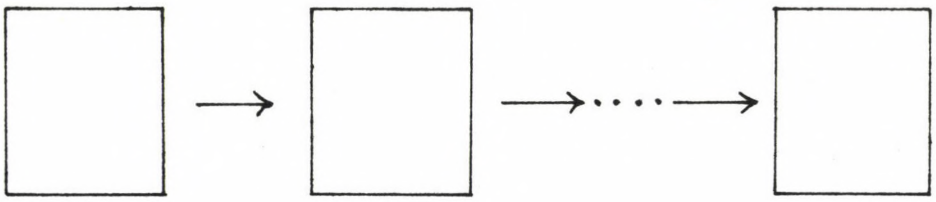
(the whole tree)

(In the case of backtrack, the upper part of the tree might have been omitted in the source code). Hence, this description form was more uniform with the internal representation and more compressed, too. Our first real cellular programs have been (and could have successfully been) coded in this system. (A relatively short program might correspond to some thousands of terms.)

To develop this system, small changes were introduced – to shorten the programs, a one-line form (as in *) was allowed, variables were used for storing the OR-ed groups of states, thus ensuring a shorter mode of writing in terms and the minimization of the table had been extended.

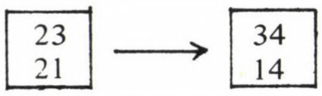
The first bigger and basic change was made after realizing that the description method, introduced in the reference (Legendi, 1977c) might be implemented as a computer input language (with relatively small modifications).

It is an open and in itself an interesting question in general how cellular algorithms could be described in the best way. It is very hard making them comprehensible and transparent since for different types of cellular algorithms very different description methods have been used (see: Vollmar 1976, Fáy 1978, Hermann 1973).
First, for the description of the *n*-step sorter (see: Appendix) the *microconfiguration concept* has been introduced. The basic idea is natural and simple: when doing cellular programming, our task is to evaluate some global transition function:

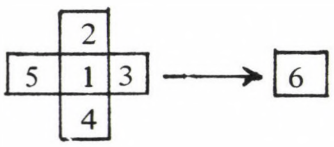


through finding the local function (system) which induces it. So our work is parallel decomposition in one step, from the whole space to individual cells.

It would be easier to descend only to the level of a group of cells (instead of individual cells).



This description has a lot of advantages. In its appearance it expresses the geometric behaviour of the cells much better than any linearized form. It means much more, than the traditional



geometric term form, because it unites more terms showing the common behaviour, *mutual* effect (on each other) of a group of cells. So it is not only a shorter form for more terms, but it is a much more adequate description tool.
(The sorter had originally been coded in more than 200 lines, here seven rules – microconfigurations – are enough, explicitly expressing the essence of the algorithm). Here, the concept of using variables becomes vital to preserve the transparency of the form.
So, at this level one microconfiguration may replace hundreds of terms and may express "microglobal" behaviour.

Still we have a very important problem: the sorter was a relatively simple case, in the sense that there is a need only for the continuous data-flow, no other operations are performed on the moving data.

Introducing the concept of *depending variables* and using expressions to describe the connection between them – e. g. making *expression-defined operations* possible during the moving of the data, we could solve the problem.

Earlier, in the new-value-part of a term or microconfiguration the use of variables (or sets of states for individual cells) had no meaning; one fixed state should have been defined. Now we may write:

$$\boxed{\begin{matrix} A & 5 \\ B & 6 \end{matrix}} \longrightarrow \boxed{\begin{matrix} 5 & C \\ 6 & D \end{matrix}} \qquad \begin{matrix} A = 0,\ 1,\ 2,\ 3 \\ B = 5,\ 6 \\ C = A + 2 \\ D = 2 \times A \end{matrix}$$

(here C depends on A,
here D depends on B)

for example.

This complex definition form is very adequate in the sense that geometrical/topological properties of the cellular program are defined by a geometrical tool (microconfigurations) while data transformations by expressions.

The Appendix shows the importance of these possibilities; in this way one microconfiguration with the corresponding expressions may replace hundreds of generalized or thousands of simple terms.

Our concrete practice with this new version of the processor showed that the write/debug cycle became nearly 10 times faster and the programs are much better self-documented.

The internal form remained as a two-dimensional table, containing the minimized tree structure of the (partial) transition function. New internal minimization algorithms were developed and implemented. (Average local transition functions need arrays with 200-600 elements.)

The details of the language are described in its User's Manual (see: Legendi, Molnár, Székely 1979), the series of new cellular algorithms, defined in this 2D language, will soon be published regularly.

Conclusion

The evolution of a special purpose (sub) language for the definition of cellular programs had been presented.

Different versions of the language processor were implemented (in FORTRAN-IV and PDP-8 assembly languages) and they were used for cellular programming.
Later versions proved to be more compact and adequate than the earlier ones.

Development of the sublanguage is in progress. Nowadays in cellprogramming, the bit--channel style becomes more important. (In this case the cells are treated as product-cells of two-state cells, for example an 8-state cell might be interpreted as

$$S_8 = S_2 \times S_2 \times S_2.)$$

In the case of the bit-channel style new tools, namely boolean variables and expressions should be incorporated into the (sub) language.

References

Baker – Hermann, 1970 CELIA – a cellular linear iterative array simulator. Proceedings of the Forth Conference on Application of Simulation, 1970, pp. 64-73.

Bolla, 1975 Minimized tree representation of cellular transition functions. Thesis. (in Hungarian) JATE, Szeged, 1975.

Brender, 1970. A Programming system for the simulation of cellular spaces. Ph. D. Thesis, The University of Michigan, Ann Arbor, 1970.

Fáy, 1978. Cellular design principles: a case study of maximum selection in CODD-ICRA cellular space. (detailed design) Computational Linguistics and Computer Languages, XII. pp. 165-231.

Hegedüs – Legendi – Pálvölgyi, 1978. INTERCELLAS User's Manual. Research Group on Automata Theory, Hungarian Academy of Sciences, 1978.

Herman – Liu, 1973. The daughter of CELIA, the French flag, and the firing squad. Simulation 21 (1973) 33-41.

Legendi, 1975. Simulation of cellular automata, the simulation language CELLAS. Conference on Simulation in medical, technical and economy sciences, Pécs 1975. pp. 100-106 (in Hungarian)

Legendi, 1977a. Cellprocessors in computer architecture. Computational Languistics and Computer Languages XI. pp. 147-167.

Legendi, 1977b. INTERCELLAS - an interactive cellular space simulation language. Acta Cybernetica, Tom. 3., Fasc. 3, pp. 261-267.

Legendi, 1977c Programming of cellular processors. "Cellular meeting" Braunshweig, June, 1977. Informatik-Berichte Nr. 7703 Technische Universität Braunschweig pp 53-56.

Legendi, 1978a. TRANSCELL – a cellular automata transition function and minimization language for cellular microprogramming. Computational Linguistics and Computer Languages, XII., 55-62.

Legendi, 1979. Cellular programs. A collection of implemented cellular algorithms. To appear in: "Parallel Processing" 1979/4 a periodical of the von Neumann Computer Science Society (in Hungarian).

Legendi, 1980a. TRANSCELL II – an advanced cellular processor microprogram definition and minimization language for a universal base cell. To appear in: Computational Linguistics and Computer Languages, XIV.

Legendi – Molnár – Székely, 1979. CELLAS User's Manual Research Group on Automata Theory, Hungarian Academy of Sciences, 1979.

Nishio, 1975. Real time sorting of binary numbers by one-dimensional cellular automata. Proceedings of the International Symposium on Uniformly Structured Automata and Logic.Tokyo, 1975.

Nishio, 1977 (ed.) Studies on Polyautomata, March 1977., Kyoto. (Reports on the results of the 1976 Research Group on "Polyautomata, their Structures and functions)

Sümeghy, 1977. Cellular transition function tree description language and its implementation. Thesis (in Hungarian), JATE, Szeged, 1977.

Vollmar, 1977. On two modified problems of synchronization in cellular automata. Acta Cybernetica, Tom. 3., Fasc. 4, pp. 293-300.

Vollmar, 1979. Algorithmen in Zellularautomaten. Teubner Studienbücher, Informatik, Stuttgart 1979.

Wu-Hung Liu, 1972. CELIA User's Manual. Dept. of Computer Science, State University of New York at Buffalo, Oct. 1972.

APPENDIX

Sorting binary numbers (Legendi, 1977c)

The task is to sort N binary integer numbers in growing order. (the solution may be used for more general similar problems, too). For the simplicity of the discussion an eight state homogeneous cellular space is supposed.

The algorithm to be implemented is parallel pairwise comparison and change (if it is needed) for having the pairs in valid order. Maximum N steps of (alternating) pairwise comparison/change are needed.

One pairwise comparison/change for two numbers is executed as explained in the following: the numbers are represented in the space in a natural way, one bit per one cell, using the states 0 and 1, each number is in consecutive cells of a raw, the numbers to be sorted are in consecutive raws, as indicated below:

```
. . .I  0  0  0  0  0  I  0  1  0 . . . 0 N          0,1,I,J,K,L,M,N,
. . .J  0  0  I  0  0  J  0  0 0 . . . 1 N
. . .I  0  0  J  0  0  I  0  1  0 .    1 N          represent
. . .J  0  0  I  0  0  J  1  1  1 .    1 N          the states
     . 0  0  J  0  0  I  1  1  0 .    0 N
    . 0  0  I  0  0  J  0  1  0 .    0 N
. . .I  0  0  J  0  0  I  0  0  1 .    0 N
. . .J  0  0  0  0  0  J  1  0  0. . . 1 N
```

An $\begin{smallmatrix} I \\ J \end{smallmatrix}$ pair of states ("operators") indicates that the leftmost investigated bit(s) of both numbers are equal.

$$\boxed{\begin{matrix} I & 0 \\ J & 0 \end{matrix}} \longrightarrow \boxed{\begin{matrix} 0 & I \\ 0 & J \end{matrix}} \qquad \boxed{\begin{matrix} I & 1 \\ J & 1 \end{matrix}} \longrightarrow \boxed{\begin{matrix} 1 & I \\ 1 & J \end{matrix}}$$

When the two numbers are in valid order (the first nonequal pair is $\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}$), no change will take place.

$$\boxed{\begin{matrix} I & 0 \\ J & 1 \end{matrix}} \longrightarrow \boxed{\begin{matrix} 0 & M \\ 1 & M \end{matrix}} \qquad \boxed{\begin{matrix} M & X \\ M & Y \end{matrix}} \longrightarrow \boxed{\begin{matrix} X & E \\ Y & M \end{matrix}} \qquad \boxed{\begin{matrix} M & F \\ M & N \end{matrix}} \dashrightarrow \boxed{\begin{matrix} N & 0 \\ N & 0 \end{matrix}}$$

x,y = 0,1 The state M memorizes the situation.

When the two numbers are not in valid order, the change is forced.

$$\boxed{\begin{matrix} I & 1 \\ J & 0 \end{matrix}} \rightarrow \boxed{\begin{matrix} 1 & K \\ 0 & L \end{matrix}} \qquad \boxed{\begin{matrix} x & K & v \\ y & L & w \end{matrix}} \rightarrow \boxed{\begin{matrix} y & v & K \\ x & w & L \end{matrix}} \qquad \boxed{\begin{matrix} K & N \\ L & N \end{matrix}} \rightarrow \boxed{\begin{matrix} N & 0 \\ N & 0 \end{matrix}}$$

$x,y,v,w = 0,1$ The pair of states $\begin{matrix} K \\ L \end{matrix}$ memorizes the situation.

Notice that the detection of the non-valid order and the first change cannot be executed in one step because of the restricted neighborhood condition.

It is remarkable that $n (n = 1,2, \dots , N)$ columns of operators can work in parallel, indepedently. Really, there is no need of waiting for the end of a pairwise comparison/change – the bit column on the left side of an operator is already in its right order (as to the last investigation) and can be compared/changed again, independently of the changes on its right side. At most, after $N + k$ steps (k is the number of bits of a number) the process will be complete.

Naturally it is not too practical that the numbers to be sorted and the operators are moving and that the needed area in the space is relatively large.

In a 16 state 4 neighbour homogeneous space the algorithm can be coded into a static area where an inside flow of operators is implemented in horizontal bit channels of the states whilst change in the vertical ones. The operator columns need not be prepared in a static way, they may be generated dynamically from an initial one.

Dividing decimal numbers by two

In many cases we use binary representation of numbers in a cellular space, (as in the above example. too).

It is in no way obligatory, however other representations (unary, decimal hexadecimal, residue number system, etc.) are frequently used, too (demonstrating the flexibility of cellular spaces).

The last example is one single operation on decimal numbers (one decimal digit per one cell) that may be used in many bigger processing elements.

The operations to be performed are *multiplication and divison by* special constants, by *two or five*. It is obvious but surprising that these operations might be executed in a very fast manner: *in one transition step,* independently of the length of the numbers. The reason is simple – these operations are *local*. For example in the case of multiplication by two, the new value of each digit depends only on itself and on the right neighbour digit. In the example, shown in detail (the division by two) the situation is similar – only, here the left neighbour digit determines the new value (together with the own value).

This small processing element (decimal/2) is used in converting from decimal to binary in decimal multiplication (together with multiplication by two), in multiplication/division by other special constants (sum and/or product of 2 and 5 etc.

```
****************************************************************************************

 SSSSSSSSSSS    ZZZZZZZZZZZZ              //      2222222222
SSSSSSSSSSSSS   ZZZZZZZZZZZZ             //      22222222222
SSS                    ZZZZ             //       222      222
SSS                   ZZZZ             //                2222
SSSSSSSSS            ZZZZ             //               2222
SSSSSSSSSSSSS       ZZZZ             //              2222
     SSSSSSSSS     ZZZZ             //             2222
           SSS    ZZZZ             //            2222
           SSS   ZZZZ             //             222
SSSSSSSSSSSSS   ZZZZZZZZZZZZ     //              22222222222
SSSSSSSSSSSS    ZZZZZZZZZZZZ    //               22222222222



                    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                    X                              X
                    X                              X
                    X       CELLAS PROCESSOR       X
                    X                              X
                    X                              X
                    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
                    X                              X
                    X       DATE =  07/11/79       X
                    X       TIME =  13/50/29       X
                    X       CORE =    92           X
                    X       DISC =   580           X
                    X                              X
                    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX


MAP=NO
BE,ZZ
      1.   %FUNCTION/MAX:10;/

            ***  MAX:  10,  SZSZ:   5, BELS:    0, DSI:   3  ***


      2.   VALTOZOK:*0:X(0,1,2,3,4,5,6,7,8,9),Y(0,1,2,3,4,5,6,7,8,9):

      3.   VALTOZOK:*0:W(5*X-X/2*10+Y/2);

      4.   IRELL:0;

      5.   MICRO:1*2:

      6.   X,Y,   $,W,/
100


      7.   %END;
           ***  SYNT:  0, OPTAR:  0, INT:  0, TESZT: -1, SZAM: 10, SERV: 32

  FELIR:  1, IRELL: -1   -1
```

```
LEPES=     0, MERET=   9 *   24, CSUCSOK=(   2,   2)(  10,  25)

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 9 3 7 6 0 0 0 0 0 0 7 9 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 2 1 9 8 0 0 0 0 0 0 3 5 4 9 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 6 7 1 8 0 0 0 0 0 0 2 7 4 3 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    DISP 0 1 0
    DO 10

LEPES=     1, MERET=   9 *   24, CSUCSOK=(   2,   2)(  10,  25)

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 4 6 8 8 0 0 0 0 0 0 3 9 5 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 9 9 0 0 0 0 0 0 1 7 7 4 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 3 3 5 9 0 0 0 0 0 0 1 3 7 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

LEPES=     2, MERET=   9 *   24, CSUCSOK=(   2,   2)(  10,  25)

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 2 3 4 4 0 0 0 0 0 0 1 9 7 5 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 5 4 9 0 0 0 0 0 0 0 8 8 7 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 6 7 9 0 0 0 0 0 0 0 6 5 5 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

LEPES=     3, MERET=   9 *   24, CSUCSOK=(   2,   2)(  10,  25)

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 7 2 0 0 0 0 0 0 0 9 8 7 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 2 7 4 0 0 0 0 0 0 0 4 4 3 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 8 3 9 0 0 0 0 0 0 0 3 4 2 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
LEPES=    4, MERET=   9 *   24, CSUCSOK=(   2,   2)( 10, 25)

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 5 8 6 0 0 0 0 0 0 0 4 9 3 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 3 7 0 0 0 0 0 0 0 2 2 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 4 1 9 0 0 0 0 0 0 0 1 7 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
LEPES=    5, MERET=   9 *   24, CSUCSOK=(   2,   2)( 10, 25)

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 2 9 3 0 0 0 0 0 0 0 2 4 6 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 6 8 0 0 0 0 0 0 0 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 2 0 9 0 0 0 0 0 0 0 0 8 5 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
LEPES=    6, MERET=   9 *   24, CSUCSOK=(   2,   2)( 10, 25)

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 4 6 0 0 0 0 0 0 0 1 2 3 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 3 4 0 0 0 0 0 0 0 0 5 5 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 4 0 0 0 0 0 0 0 0 4 2 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
LEPES=    7, MERET=   9 *   24, CSUCSOK=(   2,   2)( 10, 25)

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 7 3 0 0 0 0 0 0 0 0 6 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 7 0 0 0 0 0 0 0 0 2 7 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 5 2 0 0 0 0 0 0 0 0 2 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
LEPES=    8, MERET=  9 *  24, CSUCSOK=(  2,  2)( 10, 25)

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 3 6 0 0 0 0 0 0 0 0 3 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 1 3 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 2 5 0 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0


LEPES=    9, MERET=  9 *  24, CSUCSOK=(  2,  2)( 10, 25).

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 8 0 0 0 0 0 0 0 0 1 5 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 5 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 3 0 0 0 0 0 0 0 0 0 5 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0


LEPES=   10, MERET=  9 *  24, CSUCSOK=(  2,  2)( 10, 25)

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 9 0 0 0 0 0 0 0 0 0 7 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 3 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 6 0 0 0 0 0 0 0 0 0 2 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    STOP
```

```
 SSSSSSSSSSS    ZZZZZZZZZZZZ        0000          RRRRRRRRRRRR       TTTTTTTTTTTT
SSSSSSSSSSSSS   ZZZZZZZZZZZZ       000000         RRRRRRRRRRRRR      TTTTTTTTTTTT
SSS                     ZZZZ      000  000        RRR        RRR         TTTT
SSS                    ZZZZ      000    000       RRR        RRR         TTTT
SSSSSSSSS             ZZZZ      000      000      RRR        RRR         TTTT
SSSSSSSSSSSSS        ZZZZ       000      000      RRRRRRRRRRRRR          TTTT
     SSSSSSSS       ZZZZ        000      000      RRRRRRRRRRRR           TTTT
          SSS      ZZZZ          000    000       RRR    RRR             TTTT
          SSS     ZZZZ            000  000        RRR     RRR            TTTT
SSSSSSSSSSSSS   ZZZZZZZZZZZZ       000000         RRR       RRR          TTTT
SSSSSSSSSSS     ZZZZZZZZZZZZ        0000          RRR        RRR         TTTT

               1.  %FUNCTION/MAX:8;/

                    ***  MAX:   8,  SZSZ:   5, BELS:    0, USI:   3  ***


               2.  VALTOZOK:*0:I(2),J(3),K(4),L(5),M(6),N(7);

               3.  VALTOZOK:*0:X(0,1),Y(0,1),V(0,1),W(0,1);

               4.  VALTOZOK:*0:R(0,1,7);

               5.  VALTOZOK:*0:Q(2,3,4,5,6);

               6.  IRELL:0:

               7.  R,$$$$*R,

               8.  MICRO:1*2:

               9.

              10.  M,     X,          X,    M,

              11.

              12.  Q,     N,          N,    Q,                          /

              13.

              14.  MICRO:2*2:

              15.

              16.  I,     0,          0,    K,

              17.  J,     1,          1,    M,

              18.

              19.  I,     1,          1,    K,

              20.  J,     0,          0,    L,

              21.

              22.  I,     X,          X,    I,

              23.  J,     X,          X,    J,                 /
            ~~24~~.
              24.

              25.  MICRO:2*3:

              26.

              27.  X,     K,   V,         Y,    V,    K,

              28.  Y,     L,   W,         X,    W,    L,               /
            34.
              29.

              30.  %END:
```

	0	1	2	3	4	5	6	7
1	8	9	11	11	11	11	11	10
8	12	12	12	12	25	32	12	12
9	14	14	14	14	30	40	14	14
10	16	16	16	16	16	16	16	16
11	0	1	-1	-1	-1	-1	-1	7
12	23	18	13	13	13	13	13	13
13	0	0	0	22	0	45	6	0
14	21	24	15	15	15	15	15	15
15	1	1	1	20	1	46	6	1
16	17	17	17	17	17	17	17	17
17	7	7	0	0	0	0	0	7
18	0	0	19	22	4	45	6	0
19	6	6	6	6	6	6	6	6
20	6	3	1	1	1	1	1	1
21	1	1	4	20	31	46	6	1
22	3	5	0	0	0	0	0	0
23	0	0	2	22	4	45	6	0
24	1	1	2	20	4	46	6	1
25	26	27	13	13	13	13	13	13
26	28	28	28	28	4	45	0	28
27	29	29	29	29	4	46	1	29
28	0	0	0	0	0	0	0	0
29	1	1	1	1	1	1	1	1
30	26	27	15	15	15	15	15	15
31	4	4	4	4	4	4	4	4
32	33	34	35	35	35	35	35	35
33	36	36	37	36	38	45	39	36
34	36	36	39	36	38	45	39	36
35	36	36	36	36	36	45	39	36
36	0	1	0	0	0	0	0	0
37	0	1	2	2	2	2	2	2
38	0	1	4	4	4	4	4	4
39	0	1	6	6	6	6	6	6
40	41	42	43	43	43	43	43	43
41	44	44	38	44	38	46	39	44
42	44	44	37	44	38	46	39	44
43	44	44	44	44	44	46	39	44
44	0	1	1	1	1	1	1	1
45	5	5	0	0	0	0	0	0
46	5	5	1	1	1	1	1	1

The table form of the tree of the function.
(partially minimized table)

```
LEPES=   0  MERET=( 10, 32)  CSUCSPONTOK=(  2,  2),( 11, 33)    ELTOLAS=(  0,  0)              *** PRIM ***

    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    M 0 0 I 0 0 M 0 0 I 0 0 M 0 0 I 0 0 M 0 0 I 0 1 0 1 1 0 1 1 0 N
    I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 1 0 0 1 1 0 1 1 N
    J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 1 1 0 1 0 0 1 N
    I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 0 0 1 0 1 1 0 N
    J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 1 1 0 1 1 1 0 1 N
    I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 1 1 0 1 0 0 1 N
    J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 1 1 0 1 1 1 0 N
    M 0 0 J 0 0 M 0 0 J 0 0 M 0 0 J 0 0 M 0 0 J 0 1 1 1 0 0 1 0 0 N
 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
         CALLF 1 1 10 32 1
         DISP 0 1 0
         DO 25


LEPES=   1  MERET=( 10, 32)  CSUCSPONTOK=(  2,  2),( 11, 33)    ELTOLAS=(  0,  0)              *** PRIM ***

    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    0 M 0 0 I 0 0 M 0 0 I 0 0 M 0 0 I 0 0 M 0 0 I 1 0 1 1 0 1 1 0 N
    0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 1 0 0 1 1 0 1 1 N
    0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 1 1 0 1 0 0 1 N
    0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 0 1 0 1 1 0 N
    0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 1 1 0 1 1 1 0 1 N
    0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 1 1 0 1 0 0 1 N
    0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 1 1 0 1 1 1 0 N
    0 M 0 0 J 0 0 M 0 0 J 0 0 M 0 0 J 0 0 M 0 0 J 1 1 1 0 0 1 0 0 N
 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

          CELLAS PROCESSOR

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX



DATE        07/13/7[illegible]

TIME        12/18/3[illegible]
```

```
LEPES=   2  MERET=( 10, 32)  CSUCSPONTOK=(  2,  2),( 11, 33)    ELTOLAS=(  0,  0)

    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    0 0 M 0 0 I 0 0 M 0 0 I 0 0 M 0 0 I 0 0 M 0 1 I 0 1 1 0 1 1 0 N
    0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 1 J 0 0 1 1 0 1 1 N
    0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 1 1 0 1 0 0 1 N
    0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 1 0 1 1 0 N
    0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 1 K 1 0 1 1 1 0 1 N
    0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 L 1 1 0 1 0 0 1 N
    0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 M 1 1 0 1 1 1 0 N
    0 0 M 0 0 J 0 0 M 0 0 J 0 0 M 0 0 J 0 0 M 0 1 M 1 1 0 0 1 0 0 N
 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0


LEPES=   3  MERET=( 10, 32)  CSUCSPONTOK=(  2,  2),( 11, 33)    ELTOLAS=(  0,  0)

    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    0 0 0 M 0 0 I 0 0 M 0 0 I 0 0 M 0 0 I 0 0 M 1 0 I 1 1 0 1 1 0 N
    0 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 1 0 J 0 1 1 0 1 1 N
    0 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 1 K 1 0 1 0 0 1 N
    0 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 L 0 1 0 1 1 0 N
    0 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 1 K 0 1 1 1 0 1 N
    0 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 1 1 L 1 0 1 0 0 1 N
    0 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 1 M 1 0 1 1 1 0 N
    0 0 0 M 0 0 J 0 0 M 0 0 J 0 0 M 0 0 J 0 0 M 1 1 M 1 0 0 1 0 0 N
 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0


LEPES=   4  MERET=( 10, 32)  CSUCSPONTOK=(  2,  2),( 11, 33)    ELTOLAS=(  0,  0)

    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    0 0 0 0 M 0 0 I 0 0 M 0 0 I 0 0 M 0 0 I 0 1 0 0 1 K 1 0 1 1 0 N
    0 0 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 1 K 0 0 L 1 1 0 1 1 N
    0 0 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 L 0 1 K 0 1 0 0 1 N
    0 0 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 1 0 L 1 0 1 1 0 N
    0 0 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 1 0 K 1 1 1 0 1 N
    0 0 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 1 K 1 1 L 0 1 0 0 1 N
    0 0 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 L 1 1 M 0 1 1 1 0 N
    0 0 0 0 M 0 0 J 0 0 M 0 0 J 0 0 M 0 0 J 0 1 M 1 1 M 0 0 1 0 0 N
 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
LEPES=   5  MERET=( 10, 32)  CSUCSPONTOK=(  2,  2),( 11, 33)    ELTOLAS=(  0,  0)

   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 M 0 0 I 0 0 M 0 0 I 0 0 M 0 0 I 1 0 M 0 1 K 0 1 1 0 N
   0 0 0 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 K 1 1 L 1 0 1 1 N
   0 0 0 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 1 0 L 0 0 K 1 0 0 1 N
   0 0 0 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 1 I 1 1 L 0 1 1 0 N
   0 0 0 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 1 J 1 1 K 1 1 0 1 N
   0 0 0 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 1 K 0 0 L 1 0 0 1 N
   0 0 0 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 1 1 L 1 0 M 1 1 1 0 N
   0 0 0 0 0 M 0 0 J 0 0 M 0 0 J 0 0 M 0 0 J 1 1 M 1 0 M 0 1 0 0 N
10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
LEPES=   6  MERET=( 10, 32)  CSUCSPONTOK=(  2,  2),( 11, 33)    ELTOLAS=(  0,  0)

   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 0 M 0 0 I 0 0 M 0 0 I 0 0 M 0 1 K 0 0 M 1 0 K 1 1 0 N
   0 0 0 0 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 L 0 1 K 1 1 L 0 1 1 N
   0 0 0 0 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 1 K 0 0 L 1 1 K 0 0 1 N
   0 0 0 0 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 L 1 1 I 0 0 L 1 1 0 N
   0 0 0 0 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 1 1 J 0 1 K 1 0 1 N
   0 0 0 0 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 1 0 K 1 1 L 0 0 1 N
   0 0 0 0 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 1 I 1 1 L 0 1 M 1 1 0 N
   0 0 0 0 0 0 M 0 0 J 0 0 M 0 0 J 0 0 M 0 1 J 1 1 M 0 0 M 1 0 0 N
10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
LEPES=   7  MERET=( 10, 32)  CSUCSPONTOK=(  2,  2),( 11, 33)    ELTOLAS=(  0,  0)

   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 M 0 0 I 0 0 M 0 0 I 0 0 M 0 0 K 0 1 M 1 1 K 1 0 N
   0 0 0 0 0 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 1 0 L 0 1 K 0 0 L 1 1 N
   0 0 0 0 0 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 0 K 1 1 L 0 0 K 0 1 N
   0 0 0 0 0 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 1 1 L 1 0 I 1 1 L 1 0 N
   0 0 0 0 0 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 0 1 I 1 0 J 1 1 K 0 1 N
   0 0 0 0 0 0 0 I 0 0 J 0 0 I 0 0 J 0 0 I 0 1 J 1 1 K 1 0 L 0 1 N
   0 0 0 0 0 0 0 J 0 0 I 0 0 J 0 0 I 0 0 J 1 1 I 0 0 L 1 1 M 1 0 N
   0 0 0 0 0 0 0 M 0 0 J 0 0 M 0 0 J 0 0 M 1 1 J 1 0 M 0 1 M 0 0 N
10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
LEPES=   8  MERET=( 10, 32)  CSUCSPONTOK=(  2,  2),( 11, 33)    ELTOLAS=(  0,  0)

   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 M 0 0 I 0 0 M 0 0 I 0 0 M 0 0 K 1 1 M 0 1 K 0 N
   0 0 0 0 0 0 0 0 I 0 0 J 0 0 I 0 0 J 0 1 K 0 0 L 1 0 K 1 1 L 1 N
   0 0 0 0 0 0 0 0 J 0 0 I 0 0 J 0 0 I 0 0 L 1 1 K 1 0 L 1 0 K 1 N
   0 0 0 0 0 0 0 0 I 0 0 J 0 0 I 0 0 J 0 1 K 0 1 L 0 1 I 0 1 L 0 N
   0 0 0 0 0 0 0 0 J 0 0 I 0 0 J 0 0 I 0 0 L 1 1 I 0 1 J 0 0 K 1 N
   0 0 0 0 0 0 0 0 I 0 0 J 0 0 I 0 0 J 0 0 M 1 1 J 0 1 K 1 0 L 1 N
   0 0 0 0 0 0 0 0 J 0 0 I 0 0 J 0 0 I 0 1 M 1 0 M 1 1 L 1 1 M 0 N
   0 0 0 0 0 0 0 0 M 0 0 J 0 0 M 0 0 J 0 1 M 1 1 M 0 0 M 1 0 M 0 N
10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
LEPES=   9  MERET=( 10, 32)  CSUCSPONTOK=(  2,   2),( 11, 33)    ELTOLAS=(  0,   0)

    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 M 0 0 T 0 0 M 0 0 I 0 0 M 0 1 K 1 0 M 1 0 K N
    0 0 0 0 0 0 0 0 0 I 0 0 I 0 0 I 0 0 J 0 0 K 0 1 L 0 1 K 1 1 L N
    0 0 0 0 0 0 0 0 0 J 0 0 I 0 0 J 0 0 I 1 1 L 1 1 K 0 1 L 1 1 K N
    0 0 0 0 0 0 0 0 0 I 0 0 J 0 0 I 0 0 J 0 0 K 1 0 L 1 0 I 0 0 L N
    0 0 0 0 0 0 0 0 0 J 0 0 T 0 0 J 0 0 I 1 1 L 1 0 I 1 0 J 0 1 K N
    0 0 0 0 0 0 0 0 0 I 0 0 J 0 0 I 0 0 J 0 1 M 1 0 J 1 1 K 0 1 L N
    0 0 0 0 0 0 0 0 0 J 0 0 T 0 0 J 0 0 I 1 1 M 1 1 M 1 1 L 1 0 M N
    0 0 0 0 0 0 0 0 0 M 0 0 I 0 0 M 0 0 J 1 1 M 1 0 M 0 1 M 0 0 M N
10  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
LEPES=  10  MERET=( 10, 32)  CSUCSPONTOK=(  2,   2),( 11, 33)    ELTOLAS=(  0,   0)

    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 M 0 0 I 0 0 M 0 0 I 0 0 0 1 1 K 0 1 M 1 0 0
    0 0 0 0 0 0 0 0 0 0 I 0 0 J 0 0 I 0 0 J 1 0 0 1 0 L 1 1 K 0 N 0
    0 0 0 0 0 0 0 0 0 0 J 0 0 I 0 0 J 0 1 K 0 1 0 0 0 K 1 1 L 0 N 0
    0 0 0 0 0 0 0 0 0 0 I 0 0 J 0 0 I 0 0 L 1 1 0 1 1 L 0 0 I 1 N 0
    0 0 0 0 0 0 0 0 0 0 J 0 0 I 0 0 J 0 1 K 0 1 0 0 1 1 0 0 J 1 N 0
    0 0 0 0 0 0 0 0 0 0 I 0 0 J 0 0 I 0 0 L 1 1 0 0 1 J 1 0 K 1 N 0
    0 0 0 0 0 0 0 0 0 0 J 0 0 I 0 0 J 0 1 I 1 0 0 1 1 M 1 1 L 0 N 0
    0 0 0 0 0 0 0 0 0 0 M 0 0 J 0 0 M 0 1 J 1 1 0 0 0 M 1 0 M 0 N 0
10  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
LEPES=  11  MERET=( 10, 32)  CSUCSPONTOK=(  2,   2),( 11, 33)    ELTOLAS=(  0,   0)

    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 M 0 0 I 0 0 M 0 0 M 0 1 M 0 0 K 1 1 M N 0
    0 0 0 0 0 0 0 0 0 0 0 I 0 0 J 0 0 I 0 1 M 1 1 K 1 1 L 1 0 K N 0
    0 0 0 0 0 0 0 0 0 0 0 J 0 0 I 0 0 J 0 0 K 0 0 L 1 1 K 1 0 L N 0
    0 0 0 0 0 0 0 0 0 0 0 I 0 0 J 0 0 I 1 1 L 1 1 K 0 0 L 0 1 I N 0
    0 0 0 0 0 0 0 0 0 0 0 J 0 0 I 0 0 J 0 0 K 1 0 L 1 0 M 0 1 J N 0
    0 0 0 0 0 0 0 0 0 0 0 I 0 0 J 0 0 I 1 1 L 1 0 M 1 1 M 1 1 K N 0
    0 0 0 0 0 0 0 0 0 0 0 J 0 0 I 0 0 J 1 1 I 0 1 M 1 1 M 0 0 L N 0
    0 0 0 0 0 0 0 0 0 0 0 M 0 0 J 0 0 0 1 1 J 1 0 M 0 1 M 0 0 M N 0
10  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
LEPES=  12  MERET=( 10, 32)  CSUCSPONTOK=(  2,   2),( 11, 33)    ELTOLAS=(  0,   0)

    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 I 0 0 M 0 0 M 1 0 M 1 1 K 1 N 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 J 0 0 I 1 1 M 0 1 K 0 1 L 0 N 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 I 0 0 J 1 0 K 1 1 L 0 1 K 0 N 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 J 0 1 K 0 1 L 0 0 K 1 0 L 1 N 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 I 0 0 L 1 1 K 1 1 L 0 0 M 1 N 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 J 0 1 I 0 1 L 0 1 M 1 1 M 0 N 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 I 0 1 J 1 0 M 1 1 M 1 0 1 1 N 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 J 0 1 M 1 1 M 0 0 M 1 0 M 0 N 0 0
10  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
LEPES=  13  MEPET=( 10, 32)  CSUCSPONTOK=(  2,  2),( 11, 33)    ELTOLAS=(  0,  0)

   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 M 0 0 I 0 0 M 0 1 M 0 1 M 1 1 K N 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 I 0 0 J 0 1 I 1 0 M 1 0 K 1 0 L N 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 J 0 0 I 0 1 J 1 1 K 1 0 L 0 0 K N 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 I 0 0 J 0 0 K 0 0 L 1 1 K 1 1 L N 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 J 0 0 I 1 1 L 1 1 K 0 0 L 0 1 M N 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 I 0 0 J 1 0 M 1 0 L 1 1 M 1 0 M N 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 J 0 0 I 1 1 M 0 1 M 1 1 M 0 1 M N 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 M 0 0 J 1 1 M 1 0 M 0 1 M 0 0 M N 0 0
10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
LEPES=  14  MERET=( 10, 32)  CSUCSPONTOK=(  2,  2),( 11, 33)    ELTOLAS=(  0,  0)

   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 M 0 0 I 0 0 M 1 0 M 1 1 M 0 N 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 I 0 0 J 1 1 I 0 1 M 0 1 K 1 N 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 J 0 0 I 1 1 J 0 1 K 0 0 L 1 N 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 I 0 0 J 1 0 K 1 1 L 0 1 K 0 N 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 J 0 1 I 0 1 L 0 0 K 1 0 L 1 N 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 I 0 1 J 0 1 M 1 1 L 1 1 M 0 N 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 J 0 1 1 1 0 M 1 1 M 1 0 M 1 N 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 M 0 1 J 1 1 M 0 0 M 1 0 M 0 N 0 0 0
10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
LEPES=  15  MERET=( 10, 32)  CSUCSPONTOK=(  2,  2),( 11, 33)    ELTOLAS=(  0,  0)

   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 M 0 0 M 0 1 M 0 1 M 1 0 M N 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 I 0 1 M 1 0 I 1 0 M 0 1 K N 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 J 0 1 I 1 0 J 1 0 K 1 1 L N 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 I 0 1 J 1 1 K 1 0 L 0 0 K N 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 J 1 0 1 0 0 L 1 1 K 1 1 L N 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 I 1 0 J 1 1 M 0 1 L 1 0 M N 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 J 1 1 I 0 1 M 1 1 M 0 1 M N 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 M 1 1 J 1 0 M 0 1 M 0 0 M N 0 0 0
10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
LEPES=  16  MERET=( 10, 32)  CSUCSPONTOK=(  2,  2),( 11, 33)    ELTOLAS=(  0,  0)

   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 M 0 0 M 1 0   1 1 M 0 N 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 I 1 1 M 0 1 1 0 0 M 1 N 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 J 1 1 I 0 1 J 0 1 K 1 N 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 M 1 1 J 0 1 K 0 0 L 1 N 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 M 0 0 M 1 1 L 1 1 K 0 N 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 I 0 1 M 1 0 M 1 1 L 0 N 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 J 1 0 M 1 1 M 1 0 M 1 N 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 M 1 1 M 0 0 M 1 0 M 0 N 0 0 0 0
10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
LEPES=  17  MERET=( 10, 32)  CSUCSPONTOK=(  2,   2),( 11, 33)     ELTOLAS=(  0,  0

    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 M 0 1 M 0 1 M 1 0 M N 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 I 1 0 M 1 0 I 0 1 M N 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 J 1 0 I 1 0 J 0 1 K N 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 M 1 0 J 1 0 K 1 1 L N 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 M 0 1 M 1 1 L 1 0 K N 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 M 1 1 M 0 1 M 1 0 L N 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 M 0 1 M 1 1 M 0 1 M N 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 M 1 0 M 0 1 M 0 0 M N 0 0 0 0
10  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0


LEPES=  18  MERET=( 10, 32)  CSUCSPONTOK=(  2,   2),( 11, 33)     ELTOLAS=(  0,  0

    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 M 1 0 M 1 1 M 0 N 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 I 0 1 M 0 0 I 1 N 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 J 0 1 I 0 0 J 1 N 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 M 0 1 J 1 1 K 1 N 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 M 1 1 M 0 1 L 0 N 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 M 1 0 M 1 1 M 0 N 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 M 1 1 M 1 0 M 1 N 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 M 0 0 M 1 0 M 0 N 0 0 0 0 0
10  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0


LEPES=  19  MERET=( 10, 32)  CSUCSPONTOK=(  2,   2),( 11, 33)     ELTOLAS=(  0,  0

    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 M 0 1 M 1 0 M N 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 I 1 0 M 0 1 I N 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 J 1 0 M 0 1 J N 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 M 1 1 M 1 1 K N 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 M 1 0 M 1 0 L N 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 M 0 1 M 1 0 M N 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 M 1 1 M 0 1 M N 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 M 0 1 M 0 0 M N 0 0 0 0 0
10  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0


LEPES=  20  MERET=( 10, 32)  CSUCSPONTOK=(  2,   2),( 11, 33)     ELTOLAS=(  0,  0

    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 M 1 1 M 0 N 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 I 0 0 M 1 N 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 J 0 0 M 1 N 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 M 1 1 M 0 N 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 M 0 1 M 1 N 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 M 1 1 M 0 N 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 M 1 0 M 1 N 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 M 1 0 M 0 N 0 0 0 0 0 0
10  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
LEPES=  21  MERET=( 10, 32)  CSUCSPONTOK=(  2,  2),( 11, 33)    ELTOLAS=(  0,  0)

   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 M 1 0 M N 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 I 0 1 M N 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 J 0 1 M N 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 M 1 0 M N 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 M 1 1 M N 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 M 1 0 M N 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 M 0 1 M N 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 M 0 0 M N 0 0 0 0 0 0
10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0


LEPES=  22  MERET=( 10, 32)  CSUCSPONTOK=(  2,  2),( 11, 33)    ELTOLAS=(  0,  0)

   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 M 0 N 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 I 1 N 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 J 1 N 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 M 0 N 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 1 M 1 N 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 1 M 0 N 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 0 M 1 N 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 0 M 0 N 0 0 0 0 0 0 0
10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0


LEPES=  23  MERET=( 10, 32)  CSUCSPONTOK=(  2,  2),( 11, 33)    ELTOLAS=(  0,  0)

   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 M N 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 I N 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 J N 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 0 M N 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 1 1 M N 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 1 0 M N 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 0 1 M N 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 0 0 M N 0 0 0 0 0 0 0
10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
LEPES=  24  MERET=( 10, 32)  CSUCSPONTOK=(  2,  2),( 11, 33)    ELTOLAS=(  0,  0)

    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 N 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 N 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 N 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 0 N 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 1 1 N 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 1 0 N 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 0 1 N 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 0 0 N 0 0 0 0 0 0 0 0
10  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0


LEPES=  25  MERET=( 10, 32)  CSUCSPONTOK=(  2,  2),( 11, 33)    ELTOLAS=(  0,  0)

    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 N 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 N 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 N 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 0 N 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 1 1 N 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 1 0 N 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 0 1 N 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 0 0 N 0 0 0 0 0 0 0 0
10  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
        .STOP
```

D. MATHEMATICAL SEMANTICS

Computational Linguistics and Computer Languages Vol. XIII. 1979.

THERE ARE GENERAL RULES FOR SPECIFYING SEMANTICS: Observations of Abstract Model Theory

I. Sain
Theoretical Laboratory, Institute for Co-ordination of Computer Techniques, Budapest, Hungary

Introduction

One of the central themes of "General Semantics" and "Theory of Languages with Semantics" is called *Abstract Model Theory* (AMS classification code: 03695). Here the central problem is to define the notion of a *language with semantics* (sometimes it is called "a logic" or "a language" or "a model theoretic system"). Concrete examples of such languages are: Classical first order logic, Higher order logic, Modal logic, Intensional logic, Logic of Actions, Programming Languages, Languages for reasoning about programs etc. For early works on Abstract Model Theory see: "Universal Grammar by R. Montague 1969", Andréka-Gergely--Németi 72, §IV of 73a, 74a, 77, 78, Dahn 73, Gergely 74, 77, Lindström 74, Makowsky 73, Németi 76, Gergely–Szabolcsi 79, Andréka–Németi 76, Németi–Sain 78 etc. Cf. also Barwise 77. Motivations for Abstrtact Model Theory can be found in Pask 76, Gergely 73 – 77, Szabolcsi 78, Sain 78, Andréka–Gergely–Németi 72,74. Problem 9 of Makowsky 65, Dahn 79.

A central problem in all the quoted works is: *"How to define the general notion of a language with semantics?* " If the notion is too broad then our results will be *irrelevant* to languages, if it is too narrow then in our investigations we shall *ignore* many important languages. The second choice was taken by *Makowsky* and *Barwise* who postulated in their definition of a language that the models (i.e. interpretations or "possible worlds") of a language are always classical first order structures. By this decision languages with Kripkestyle semantics or Intensional semantics are excluded, cf. Dahn 73, 78, Andréka–Dahn–Németi 76, Montague 73 etc.. The Andréka–Gergely–Németi team (in the following AGN team) *tried to avoid such restrictions.* They were motivated by Dahn 73, 78, Montague 73, Pask 76 etc..

Def. IV. 1. in AGN 73a as well as the definition of a language (sometimes called "a logic") in AGN 73a, 74a, 77, 78, Andréka–Németi 75, 76, 77, 79, Németi 76 are general enough The results obtained are applicable to an impressively broad spectrum of languages and logics indeed (including the ones in Dahn 73, 78, Montague 73, Parikh 78, Pratt 78, Hayes 71, Banachowski et al 77, Andréka–Németi–Sain 79, 79a etc.).

All these works nicely illustrate the observation: "In our age *Concept Formation* is the most difficult task in science" (cf. Gergely–Németi 71). Namely throughout all these works there is a struggle for finding an adequate concept of a "language with semantics". Since 1972 on the works of the AGN team are dominated with this problem. Many ideas were intuitively explained, illustrated by examples, approximated by implicite definitions, the explicite definitions and rigorous expositions of which, however, were stated as a task of the future.

The present work proposes a possible solution for this task. While trying to solve this task, we found that by using a *little* Set Theory, we can formalise ideas explicitely the impliciteness of which was a central problem in the above quoted works. § 1 of the present work sums up the notions of set theory we are going to use.

The backbone of the present work is § 2.1. (However, to understand, interpret and apply § 2.1 adequately, it is necessary to read the rest of the paper.) In § 2.1 the notion of a language with semantics is rigorously defined, and then restricting conditions are gradually introduced, each of which serves *to exclude irrelevant* "situations". In each step it is carefully investigated that only irrelevant situations be excluded. At the end we arrive at a fairly general notion of a language (Def. 2.6) for which meaningful abstract model theoretic theorems can be proved, e.g. Thm. 2.1. As a byproduct, a kind of methodology for defining languages with semantics is obtained. (By using this methodology, one can foresee the consequences of the decisions one is making during defining a new language.) Also methodolgy is proposed for: *Specifying semantics of languages.* A specification of the semantics of a language L is a "definition" i.e. it is a text written in some *metalanguage.* This text (serving to present the semantics of our language L in question) is called here the *presentation of* L and is denoted by $<\sigma, \mu, \kappa>$ cf. Def. 2.3. Before and after Def. 2.3. several different existing presentations of languages are surveyed, compared, and some consequences (pleasant and unpleasant) of the decisions made in the presentations are observed. Finally in Def. 2.6. some general rules are proposed which could and should be followed in specifying the semantics of any language. (At least the results of this paper seem to suggest so.) Theorem 2.1. shows some consequences of following or breaking these rules.

The reason for our using a little set theory here might be the fact that the presentation of a language is written in some *metalanguage.* Metalanguage might lead to metamathematics and a possible version of matemathematics is Set Theory.

The results of § 2.1. are applied among others to languages used in Computer Science and Programming theory. Such are e.g. Classical first order logic, Higher order logic, Languages for reasoning about programs and program schemes, Dynamic Logic, Logic of Actions etc. .

We shall arrive at the conclusion that in the case of more complex languages (Higher order ones. Languages for reasonong about programs, Logic of Actions etc.) the incompleteness results have been misinterpreted. They do *not* prove that these languages are incomplete. Instead they prove that certain mathematical models of these languages i.e. certain presentations of these languages are anomalous! This simply means that the mathematical modelling of certain phenomena is more difficult than that of others. I.e.: the mathematical modelling of certain more complex languages needs more time, more carefulness, deaper considerations, and more "revisions". Ad-hoc very natural looking common-sense definitions might imply hidden paradoxes, vicious circles, and other nonwanted nonsensical consequences. The reason for this is nothing special: it is simply the increase in comlexity which prevents our common-sense reasoning from seeing the consequences of some decisions. In short: the mathematical theory of the more complex languages (some mentioned above) is still *in the phase of concept formation.* All this shows that today common-sense reasoning should give way to scientific reasoning also in the phase of concept formation.

As an example, the field of semantics of languages for reasoning about programs will be investigated which is full of contraversies. Applying our criteria to this field, two main approaches will be distinguished here: the standard and the nonstandard ones. The nonstandard approach was initiated in Andréka–Németi 78, Andréka 78, Gergely–Úry 78, Andréka–Németi–Sain 78, 79, 79a. Applying our results to the relationship between the standard and nonstandard semantics, we shall find that nonstandard semantics is the result of a deeper, more careful analysis of the phenomenon under consideration, and accordingly, it is a more faithful model of the real situation we want to understand.

In the methodolgy or rules proposed here the main theme is expliciteness. By the quotation on the front page of ALGOL 60 report: ”If something can be said at all then it can be said explicitely too, and if something cannot be said then we must consider its nonstandard models as well.”

1. Basic definitions, notations, conventions

In this paragraph we shall recall some definitions, notations, and conventions commonly used in *set theory* and *model theory.*

0 denotes the *empty set.*

ω denotes the set of *natural numbers.*

A_B denotes the set of all functions from A into B,
i.e. ${}^{A}B \overset{df}{=} \{ f : F$ maps A into $B \}$, see Monk 76 p. 7 line 12.

$\cup A \overset{df}{=} \{ a : \exists b (a \epsilon b \epsilon A) \}$.

$A \cup B \overset{df}{=} \cup \{ A, B \}$.

$Sb(A)$ denotes the set of all subsets of the set A.

$<a_1, \ldots , a_n>$ denotes the sequence of sets $a_1, \ldots , a_n$ $(n < \omega)$ in this order (cf. Notation following Def. 1.4).

$A \times B \overset{df}{=} \{ (a,b) : a \epsilon A, \ b \epsilon B \}$.

$Do(f)$ and $Rg(f)$ denote the *domain* and the *range* of the function f. *I.e.*

$Do(f) \overset{df}{=} \{ x : (x,y) \epsilon f \}$ and

$Rg(f) \overset{df}{=} \{ y : (x,y) \epsilon f \}$.

d denotes a *similarity type* (see Def. 1.4).

F_d denotes the *set* of *first order formulas of type* d (see Def. 1.5). Note that F_d is a set (as opposed to proper class).

M_d denotes the *class of classical models* (relational structures) *of type* d, see Def. 1.6. Note that since M_d is a class, its elements are sets by definition.

A *classical model* (an element of M_d for some type d) is denoted by an underlined capital like

$\underset{\sim}{A}$, $\underset{\sim}{T}$, or $\underset{\sim}{D}$

and its *universe* is denoted by the same capital without underlining. E.g.:

A is the universe of $\underset{\sim}{A}$,

T is the universe of $\underset{\sim}{T}$,

D is that of $\underset{\sim}{D}$.

If $A \epsilon M_d$ for some type d and $<R,m> \epsilon d$ i.e. $d(R) = m$ then the *denotation of* R *in* $\underset{\sim}{A}$ is denoted by

$\underset{\sim}{A}_R$ (cf. Monk 76 p. 194).

$\underset{\sim}{\omega}$ denotes $<\omega, +, \cdot, 0,1>$, the *standard model of arithmetic.*

By a *valuation of the variables into a model* $\underset{\sim}{A}$ a function

$q : \omega \to A$ is understood, see e.g. Monk 76 p. 195.

Let $\underset{\sim}{A} \epsilon M_d$, $\varphi(x_1, \ldots, x_n) \epsilon F_d$. Let further $q : \omega \to A$ be an arbitrary valuation of the variables into $\underset{\sim}{A}$ and let $q(x_1) = a_1, \ldots, q(x_n) = a_n$ $(a_1, \ldots, a_n \epsilon A)$. Then

$\underset{\sim}{A} \models \varphi(x_1, \ldots\ldots, x_n)[q]$ is understood in the usual sense (see Def. 1.8), and it is abbreviated by

$\underset{\sim}{A} \models \varphi[a_1, \ldots, a_n]$, cf. Chang – Keisler p. 27–28.

$(\exists\,!\,x)\,\varphi(x) \overset{df}{=} [(\exists x)\varphi(x) \wedge (\forall x \,\forall y)\ ([\varphi(x) \wedge \varphi(y)] \to x = y)]$

where x and y are variables and $\varphi \epsilon F_d$ for some similarity type d.

If d is the similarity type of the *first order language of ZFC set theory* (see Def. 1.7) then we shall use a modified notation for F_d and M_d. The first order language of ZFC set theory contains only one binary relation ϵ. I.e., its similarity type is $\{(\epsilon,2)\}$. For convenience, we shall write:

F_ϵ instead of $F_{\{(\epsilon,2)\}}$, and

M_ϵ instead of $M_{\{(\epsilon,2)\}}$.

Note that F is a set and so are the elements of M.

ZFC denotes the set of axioms of Zermelo-Frankel Set Theory with the Axiom of Choice *formulated in the language* F_ϵ, see e.g. Chang – Keisler 73 p. 507-508. Thus $ZFC \subseteq F_\epsilon$, and *ZFC* is a *set* (as opposed to proper class). Cf. Def. 1.7 and Remark 1.4.

Throught this work we shall work in *Zermelo – Frankel Set Theory with the Axiom of Choice* (cf. e.g. Chang – Keisler 73 p. 507, Devlin 73 p. 2-3, Takeuti – Zaring 71). We shall call this simply "*Set Theory*" for brevity.

Recall the notion of a *real world* (V,ϵ) of Set Theory from Chang – Keisler 73 bottom of p.476, Devlin 73 bottom of p. 3. It consists of a class V and a "binary relation" ϵ defined on this class V such that the axioms of Set Theory (cf. above) are true in (V,ϵ) in the usual sense. V is called the class of all sets and ϵ is called "element of".

We shall work in a *fixed but arbitrary* real world (V,ϵ). By a *set* we shall automatically understand an element of V. *Throughout* this work we shall assume that there is a *fixed* (V,ϵ). All our statements will refer to this fixed (but otherwise unknown) real world (V,ϵ).

We shall often refer to the "*Language of Set Theory*" cf. Takeuti – Zaring 71 p. 4. Set Theory itself is a collection of formulas of this "language", and by a *formula* φ of Set Theory we shall understand a formula of this Language of Set Theory as defined in Takeuti – Zaring 71 p. 4. If φ is a formula of Set Theory then it is meaningful to say that " φ is true in the world (V,ϵ)". This will be abbreviated by "$(V,\epsilon) \models \varphi$".

Recall that every statement of mathematics can be formulated in the language of Set Theory. If $\varphi(x_1, \ldots, x_n)$ is a formula of Set Theory and $a_1, \ldots, a_n$ are sets, i.e. elements of V, then

$\varphi(a_1, \ldots, a_n)$ is said to be true

iff

$(V,\epsilon) \models \varphi[a_1, \ldots, a_n]$.

(See Chang – Keisler 73 p. 476-477.)

DEFINITION 1.1.

Let $S \subseteq V$ be an arbitrary class.

S is called *definable in Set Theory*

iff

there exists a formula $\varphi(x_1, \ldots, x_n)$ of Set Theory such that

$$S = \{ \langle a_1, \ldots, a_n \rangle \epsilon V : \varphi(a_1, \ldots, a_n) \}$$

or in more detail:

$$S = \{ <a_1, \ldots, a_n> \epsilon V : (V, \epsilon) \models \varphi[a_1, \ldots, a_n] \} .$$

To stress "certain points" we shall refer to the above property by saying "S is *explicitely* definable in Set Theory".

DEFINITION 1.2. (Devlin 73, Sacks 72, Takeuti – Zaring 71 etc.)

A class $W \subseteq V$ is called *transitive*

iff

for every $y \epsilon W$ and for every $x \epsilon y$ also $x \epsilon W$ holds.

DEFINITION 1.3. (Chang – Keisler 73 p. 475, Devlin 73, Hinman 78 p. 215)

$L(\omega) \subseteq V$ is defined to be the class of all ***hereditarily finite elements*** of V:

$b \epsilon L(\omega)$ iff the smallest transitive class $W \subseteq V$ containing b is finite.

Note that $L(\omega) \epsilon V$.

DEFINITION 1.4. (Sacks 72 § 2 p. 11)

By a *similarity type* we understand a function

$d : \Omega \to \omega$ such that $\Omega \subseteq L(\omega)$. I.e.:

A function (set of pairs) d is called a similarity type

iff

$Rg(d) \subseteq \omega$ and $Do(d) \subseteq L(\omega)$.

NOTATION

Whenever $a_0, \ldots, a_n$ are sets (i.e. $a_0, \ldots, a_n \epsilon V$), the symbol $<a_0, \ldots, a_n>$ denotes the function $s : (n+1) \to \{ a_0, \ldots, a_n \}$ such that $s(0) = a_0, \ldots, s(n) = a_n$.

I.e. $Do(<a_0, \ldots, a_n>) = n + 1$ and
$<a_0, \ldots, a_n> (i) = a_i$ for every $i \leqslant n$.
$<a_0, \ldots, a_n>$ is said to be a *sequence* of lenght $n + 1$.

To distinguish between $<a,b>$ and (a,b), we recall that the latter is $(a,b) = \{ a \{ a,b \} \}$, while the former is

$$<a,b> = \{(0,a),(1,b)\} = \{\{0,\{0,a\}\}, \{1,\{1,b\}\}\}.$$

DEFINITION 1.5. (Chang – Keisler 73 p. 22)

Let $d : \Omega \to \omega$ be a finite similarity type. I.e. let $\Omega \epsilon L(\omega)$.

Let $\Lambda, \neg, \exists$ be different and fixed elements of $L(\omega)$. It is always supposed that $\Lambda, \neg, \exists \cap \Omega = 0$.

The *set of first order formulas* F_d of type d is defined as the smallest set for which the following (i) and (ii) hold.

(i) For every $R \epsilon \Omega$,
$<R, v_1, \ldots, v_n> \epsilon F_d$ whenever $d(R) = n$ and $v_1, \ldots, v_n \epsilon \omega$.

(ii) If $\varphi, \psi \epsilon F_d$ where $Do(\varphi) = n+1$ and $Do(\psi) = m+1$ and $\varphi = <s_0, \ldots, s_n>$ and $\psi = <t_0, \ldots, t_m>$ then also

$< \Lambda, s_0, \ldots, s_n, t_0, \ldots, t_m> \epsilon F_d$,
$<\neg, s_0, \ldots, s_n> \epsilon F_d$, and for every $i \epsilon \omega$
$< \exists, i, s_0, \ldots, s_n> \epsilon F_d$.

Note that $F_d \subseteq L(\omega)$ because clearly

$$F_d \subseteq \{ {}^n(\Omega \cup \{\Lambda, \neg, \exists\} \cup \omega) : n \epsilon \omega \}.$$

Thus $F_d \epsilon V$.

NOTATION

Whenever $\varphi, \psi \epsilon F_d$, the shorthand $(\varphi \Lambda \psi)$ refers to the formula $< \Lambda, s_0, \ldots, s_n, t_0, \ldots, t_m> \epsilon F_d$ where $\varphi = <s_0, \ldots, s_n>$ and $\psi = <t_0, \ldots, t_m>$. Similarly for $\neg\varphi$ and $\exists v_i \varphi$.

REMARK 1.1.

It is easy to see that the elements of F_d are the first order formulas of similarity type d in the *usual sense* (cf. Monk 76, Chang – Keisler 73, Andréka – Gergely – Németi 75 etc.). A formula is a finite sequence or *string of symbols* from the alphabet $(\Omega \cup \{\Lambda, \neg, \exists\} \cup \omega)$. A sequence or string of lenght $n \epsilon \omega$ is a function $\varphi : n \to (\Omega \cup \{\Lambda, \neg, \exists\} \cup \omega)$etc.

REMARK 1.2.

Observe that, for every finite similarity type d, by definition $d \epsilon L(\omega) \epsilon V$ further $F_d \subseteq L(\omega) \epsilon V$. Thus $F_d \epsilon V$. Further there exists a formula $\sigma_d(v)$ of set theory such that $F_d = \{ \varphi \epsilon V : (V, \epsilon) \models \sigma_d[\varphi] \}$ i.e.: F_d is defined by σ_d. Cf. Chang – Keisler 73 bottom of p. 28. This latter statement *will* be proved in Thm. 2.3.

DEFINITION 1.6. (Chang – Keisler 73 p. 20, Monk 76 Def. 11.1 p. 194)

Let $d : \Omega \to \omega$ be a similarity type. Recall that $\Omega \subseteq L(\omega)\epsilon V$. Then the *class* M_d *of all models of type* d is defined as:

$$M_d = \{ (A,F)\epsilon V : F \text{ is a function, } Do(F) = \Omega, \text{ and for every } (R,n)\epsilon d,\ F(R) \subseteq {}^nA \ \} .$$

REMARK 1.3.

Observe that for every similarity type $d\epsilon L(\omega)\epsilon V$ we have $M_d \subseteq V$, further there is a set theoretic formula $\mu_d(x)$ such that

$$M_d = \{ \underset{\sim}{A}\epsilon V : (V,\epsilon) \models \mu_d[\underset{\sim}{A}] \} .$$

This latter statement about the existence of an explicite definition $\mu_d(x)$ will be proved in Thm. 2.3. Thus M_d is a definable class in V. The elements of M_d i.e. the so called *models of type d* are *sets* i.e. they are elements of V.

DEFINITION 1.7.

Throughout we suppose that there is a fixed element ϵ' of $L(\omega)\epsilon V$. E.g. $\epsilon' = \{\{\{0\}\}\}$. Further we suppose that ϵ' is distinct from "everything else" i.e. $\epsilon' \notin (\{ \wedge, \neg, \exists \} \cup \omega)$. The similarity type $\{ (\epsilon', 2)\}$ will be called the *type of ZFC set theory*. Throughout F_ϵ denotes $F_{\{(\epsilon', 2)\}}$ and M_ϵ denotes $M_{\{(\epsilon',2)\}}$.

Observe that $F_\epsilon \subseteq L(\omega)\epsilon V$ and $M_\epsilon \subseteq V$. Thus F_ϵ is a set and so are the elements of M_ϵ. In other words, F_ϵ and the elements of M_ϵ are elements of V. ZFC denotes a special subset of F_ϵ (cf. Chang – Keisler 73 p. 507), namely the set of all axioms of ZFC Set Theory *formulated in the language* F_ϵ. Therefore $ZFC \subseteq F_\epsilon$ and $ZFC\ \epsilon V$. Thus ZFC is a *set*.

REMARK 1.4.

In metamathematical investigations the set $F_\epsilon \epsilon V$ is often said to be "*the code* of the Language of Set Theory" inside of the world (V,ϵ). Similarly, the set $ZFC\ \epsilon V$ is often said to be *the code* of "the collection of Axioms of Set Theory" inside of (V,ϵ). We shall drop these long names, we shall call F_ϵ "the language of ZFC". Further, we shall write *everywere* "ϵ" instead of "ϵ'".

DEFINITION 1.8 (Chang – Keisler 73 p. 27-29, Monk 76 Def. 11.5 p. 196)

Let $d\epsilon L(\omega)$ be a similarity type.

(1) The validity relation $\models\ \subseteq (M_d \times F_d)$ of classical first order language is defined in the usual way, see Chang – Keisler 73 p. 29, Monk 76 Def. 11.5.

Let $\underset{\sim}{A}\epsilon M_d$ and $\varphi\epsilon F_d$ be arbitrary. (Recall that $\underset{\sim}{A}\epsilon V$ automatically holds.) The statement $\underset{\sim}{A} \models \varphi$ is pronaunced as "φ is *valid* in the model $\underset{\sim}{A}$".

Sometimes $\models$ will also be used as a ternary relation $\models \subseteq (M_d \times F_d \times {}^{\omega}V)$ (cf. Monk 76 Def. 11.5 p. 196): Let $q \epsilon {}^{\omega}A$ be arbitrary. Then $\underset{\sim}{A} \models \varphi[q]$ denotes that "φ is true in $\underset{\sim}{A}$ under the *valuation* q of the variables".

(2) Let further $Th \subseteq F_d$ be arbitrary. (Recall from Remark 1.2 that $Th \,\epsilon V$.) $Md(Th)$ denotes the class of all models of Th, i.e.

$$Md(Th) \overset{df}{=} \{ \underset{\sim}{A} \epsilon M_d : \underset{\sim}{A} \models Th \} .$$

Observe that $Md(Th) \subseteq V$ always holds.

REMARK 1.5.

We shall see in Thm. 2.3. that there exists a set theoretic formula $\kappa_d(x,y)$ such that

$$\models \; = \{ (\underset{\sim}{A},\varphi) \epsilon V : (V,\epsilon) \models \kappa_d[\underset{\sim}{A},\varphi] \} ,$$

i.e., for any $\underset{\sim}{A},\varphi \epsilon V$ we have:

$$\underset{\sim}{A} \models \varphi \quad \text{iff} \quad (V,\epsilon) \models \kappa_d[\underset{\sim}{A},\varphi].$$

We shall often use $Md(ZFC)$. Clearly $Md(ZFC) \subseteq M_\epsilon \subseteq V$. I.e., the modes of the theory $ZFC(\subseteq F_\epsilon)$ are *sets* (elements of v V).

REMARK 1.6.

Note that every element b of $L(\omega)$ has a name $\overline{b} \epsilon F_\epsilon$ such that $\overline{b}$ denotes b in (V,ϵ). E.g., the ordinal $2 \epsilon L(\omega)$ has the name $\overline{2} = \{ 0, \{ 0 \} \}$ ϵF_ϵ, see Takeuti – Zaring 71 p. 10 Defs 4.1, 4.2, 5.1. To check this, it is enough to see that for every $n \epsilon \omega$ there is a "ZFC-name" $\overline{n} \epsilon F_\epsilon$ such that $\overline{n}$ denotes n in the world (V,ϵ). It is well known that Peano's Arithmetic (PA) is a subtheory of ZFC. Thus everything that is expressible in PA is also expressible in ZFC. All natural numbers have names (called numerals) in PA. Thus they also have names in ZFC i.e. in F_ϵ. The reason for this is that the elements of F_ϵ were defined by *recursion along* ω.

REMARK 1.7.

Throughout this work we shall *tacitly assume* that the set of axioms ZFC *is consistent* i.e. the real world (V,ϵ) is such that there exists a model of ZFC inside of (V,ϵ). (For models of ZFC cf. e.g. Chang – Keisler 73 p. 83, Devlin 73 p. 14 line 6, p. 16 line 8, Hinman 78 p. 214.)

NOTATION

Let ψ be some mathematical text, e.g. "the Continuum Hypothesis is true" or "x denotes the smallest infinite cardinal". Then obviously there exists a formula of F_ϵ which expresses ψ i.e. which is the rigorous translation of ψ. This formula will be denoted by 'ψ'. (Cf. Devlin 73 p. 3 line 7 bottom up.) Since $F_\epsilon \subseteq L(\omega)\epsilon V$, '$\psi$'$\epsilon L(\omega)\epsilon V$.

Examples:

1.) Consider the mathematical statement $x = 1$. Now, its translation '$x = 1$' is the following formula:

$$\forall y(y\epsilon x \leftrightarrow \forall z(z\not\epsilon y)).$$

Clearly '$x = 1$' is a typical element of F_ϵ.

2.) Let φ be an arbitrary first order formula, i.e. $\varphi\epsilon F_d$ for some fixed $d\epsilon V$. Then '$\models \varphi$' is a formula in F_ϵ expressing the claim that the formula φ is valid in every relational structure $\underset{\sim}{A}\epsilon M_d$ i.e. in every model of F_d. Thus $\varphi\epsilon F_d$ for some similarity type d but '$\models \varphi$' ϵF_ϵ. To see this in more detail: Suppose that '$y\epsilon F_d$' is $\sigma_d(y)$ further suppose that '$x\epsilon M_d$' is $\mu_d(x)$ and '$x \models y$' is $\kappa_d(x,y)$. (Of course then $\sigma_d(y)$, $\mu_d(x)$, $\kappa_d(x,y)\epsilon F_\epsilon$. Also for the *ZFC*-name $\overline{\varphi}$ of $\varphi\epsilon L(\omega)$ we have $\overline{\varphi}\epsilon F_\epsilon$ by Remark 1.6.) Now, the formula $\forall x[\mu_d(x) \to \kappa_d(x,\overline{\varphi})]$ is an element of F_ϵ. Further observe that '$\models \varphi$' *is* the set theoretic formula:

$$\forall x[\mu_d(x) \to \kappa_d(x,\overline{\varphi})].$$

TERMINOLOGY about COMPUTABILITY

We shall use the word "recursive" as a synonim for "computable" or "effective". I.e. the adjective "recursive" will be applied not only to subsets of ω but also to subsets of $L(\omega)$ (see Def. 1.3.). (To stress the point: there is a nonempty recursive set $R \subseteq L(\omega)$ such that $R \cap \omega = 0$.) This "abuse" of wors is based on well estabilished results and habits of the Theory of Computability. There is only one place in this work in which we shall strictly avoid this sloppiness. Namely, *inside* of the proof of Thm. 2.1. only *subsets of* ω are called recursive while subsets of $L(\omega)$ are called "Turing-decidable". But this is only inside that one proof. *Anywhere else* in the text Turing-computable functions are called recursive even if they are outside of ω.

2. The concept of a language with semantics

2.1. Fundamental considerations

First of all we shall give a rather general concept of a language *with semantics*. Our definition below originates from Abstract Model Theory, cf. Makowsky 73, Lindström 74 p. 133 ("Abstrac Logic"), Németi – Sain 78 Def.1., Sain 78 §2 p. 10-14, Gergely 77, Németi 76. Brawise 77 p. 45 §5.6., Monk 76 Def.26.19. p. 417 (weak General Logic), Dahn 79.

DEFINITION 2.1 (The notion of a langue)

$\mathbb{L}$ is defined to be a *language* (with semantics)

iff

$\mathbb{L}$ is a triple

$\mathbb{L} = \langle S,M,k \rangle$ such that:

– S is a *set*,
– M is a *class*, and
– k is a *function* with domain $S \times M$, i.e. there exists a class R such that

$$k : S \times M \to R.$$

CONVENTIONS, REMARKS about Def. 2.1.

Note that by Def. 2.1. every triple consisting of a set, a class, and a function defined on their Cartesian product is a language.

If $\mathbb{L} = \langle S,M,k \rangle$ is a language then we use the following names for its parts S, M, and k:

– S is called the *syntactic language*, in short the *syntax of* $\mathbb{L}$. (Its members are often called expressions, sentences, or formulas.)

– M is called the class of *models* (or possible worlds or possible interpretations) of $\mathbb{L}$
– k is called the *meaning function* of $\mathbb{L}$, i.e.

$$k : S \times M \to \text{"Meanings"},$$

and for every $\varphi \in S$ and $\mathfrak{G} \in M$ we say that $k(\varphi, \mathfrak{G})$ is *the meaning* of φ in the model $\mathfrak{G}$. I.e. $k(\varphi, \mathfrak{G})$ is the meaning of the syntactic expression or sentence or whatever in the world or interpretation or model $\mathfrak{G}$. Other outhors often use the word "*denotation*" instead of meaning for $k(\varphi, \mathfrak{G})$.

Examples 2.1.

First we give examples that are *not* intuitive, they only illustrate what the above Def. 2.1. says word by word.

Ord denotes the class of all ordinals.

1.) <Ord, Ord, +> is *not* a language.
It is true that $+ : \text{Ord} \times \text{Ord} \to \text{Ord}$ is a function since the addition is defined on ordinal numbers but the first member of the triple is *not* a set.

2.) $<\omega$, Ord, $+>$ is a language:
ω is the *set* of all finite ordinals, Ord is a class and $+ : \omega \times \text{Ord} \to \text{Ord}$ is a function. So it does satisfy our Def. 2.1.

3.) $<\omega,\omega, +>$ is a language since sets are also classes.

4.) $< \{0,1\} , \{2,3,4\} , +>$ is a language.

5.) Let t be the similarity type of arithmetic

$<+, \cdot , 0, 1>$.

Let Eq be the set of all equations of type t. Let ALG be the class of all algebras of type t. E.g. $<\omega, + , \cdot , 0,1> \in ALG$. If $\mathfrak{G}$ is an algebra and $e \in Eq$ is an equation then let $k(e, \mathfrak{G})$ be the set of all solutions of the equation e in the algebra $\mathfrak{G}$. Now clearly $<Eq, ALG, k>$ is a language in the sense of Def. 2.1.

6.) Let t be as above. Let Terms be the set of all terms of type t without variable symbols. I.e. $(1 + 1 + 1) \in$ Terms but $(x + 1) \notin$ Terms.

For any term $\tau \in$Terms and algebra $\mathfrak{G}$ let $q(\tau, \mathfrak{G})$ be the element of the universe of $\mathfrak{G}$ denoted by τ. E.g. $q((1 + 1), <\omega, +,\cdot, 0,1>) = 2 \in \omega$. (Recall that if there are no variable symbols in a term τ then it denotes an element of the universe of the algebra $\mathfrak{G}$.)

Now <Terms, ALG, q> is a language.

The above examples were rather naive and ad-hoc. We only wanted to illustrate what *is* said and what *is not* said in Def. 2.1. The following examples are more intuitive.

Example 2.2.

1.) Define the *first order language of type* d as

$$\mathbb{L}_d \overset{df}{=} <F_d, M_d, \models>$$

where

– F_d is the set of first order formulas of type d, cf. Def. 1.5. ;

– M_d is the class of classical models of type d, cf. Def. 1.6.;

– $\models$ is the validity realtion defined in Def. 1.8.

$\mathbb{L}_d$ is a language in the sense of Def. 2.1. To see this, recall that

– $F_d \epsilon V$ is a *set*,

– $M_d \subseteq V$ is a *class*, and

– $\models$ is a ternary relation between classical models of type d, first order formulas of type d, and valuations of the variables.

Every ternary relation can be conceived of as a binary function. Thus, $\models$ can be conceived of as a function defined on $F_d \times M_d$ as follows:

For every formula $\varphi \epsilon F_d$ and model $\underset{\sim}{A} \epsilon M_d$ the meaning $\models(\varphi, \underset{\sim}{A})$ of φ in $\underset{\sim}{A}$ is defined to be the set of all valuations satisfying φ in $\underset{\sim}{A}$ i.e.:

$$\models(\varphi, \underset{\sim}{A}) \overset{df}{=} \{ q \epsilon\, {}^{\omega}A : A \models \varphi[q] \},$$

cf. Monk 76 Def. 11.5. p. 196. Thus

$$\models : F_d \times M_d \rightarrow \text{"Meanings"}$$

is a meaning function as it was required in Def. 2.1.

2.) *First order Modal language of type* d is defined as:

$$ML_d = \langle MF_d, KM_d, \text{meaningof} \rangle$$

where:

– MF_d is the set of all first order *modal* formulas of type d (one new unary sentential connective $\diamond$ is added to F_d);

– KM_d is the class of first order *Kripke* models of type d. (Note that for every similarity type t we have $KM_d \nsubseteq M_t$ i.e. Kripke models are *different* from classical models.)

– For every $\varphi \epsilon MF_d$ and $\mathfrak{M} \epsilon KM_d$ we define meaningof $(\varphi, \mathfrak{M})$ to be the set of all valuations into $\mathfrak{M}$ making φ true. (Note that a valuation into $\mathfrak{M}$ contains also the choice of a possible world in $\mathfrak{M}$ among others: Roughly speaking, $\mathfrak{M}$ is a partial order $\langle M, \leqslant \rangle$ where $M \subseteq M_d$. The elements of M are called possible worlds in $\mathfrak{M}$.)

Now clearly

$$\text{meaningof} : (MF_d \times KM_d) \rightarrow V$$

and ML_d satisfies Def. 2.1. of a language.

Observe that ML_d does not satisfy the definition given in Makowsky 73, Barwise 77, because the elements of KM_d are *not* classical structures (neither are they algebras, algebraic systems, many-sorted classical models, or anything the like).

For further examples of languages $<S,M,k>$ which do *not* satisfy any definition postulating that M be a class of *classical structures* see Dahn 73, 78, 79 Gallin 75, Montague 73 etc.

Example 2.3. (Def. 4 of Gergely – Szőts 78)

$\mathbb{P}_d \overset{df}{=} <$Program schemes of type d, M_d, traceof$>$

where traceof$(p,\underset{\sim}{A})$ is the *set of all traces* of the program scheme p in the model $\underset{\sim}{A}\epsilon M_d$, is a language in the sense of Def. 2.1.

In more detail:

P_d denotes the set of *program schemes of type* d. P_d is defined as in Manna 74, Andréka – Németi 78, Andréka – Németi 78a, Gergely – Úry 78 p.72 Def. 5.2.
E.g. let t be the similarity type of arithmetic. Then the following sequence is in P_t i.e. it is a program scheme of type t.

$<(0 : y_0 \leftarrow 0),$

$(1 : \text{IF } y_0 = y_1 \text{ THEN } 4),$
$(2 : y_0 \leftarrow y_0 + 1),$
$(3 : \text{IF } y_1 = y_1 \text{ THEN } 1),$
$(4 : \text{HALT})>.$

Now we shall quote the definition of an *ω-trace* of a program scheme $p\epsilon P_d$ in a model $\underset{\sim}{A}\epsilon M_d$ from Andréka 78, Andréka – Németi 78, Andréka - Németi - Sain 79, Gergely – Szőts 78. Sain 79, Gergely – Szőts 78.

Definition (of an ω-trace)

Note: Intuitive explanation of the definition comes immediately after the definition. Cf. also "Definitions" in §3.

Let $p\epsilon P_d$ be an arbitrary program scheme of type d. We shall denote the parts of p as

$$p = <(i_0 : u_0), \ldots, (i_n : u_n), (i_{n+1} : \text{HALT})>.$$

Let $\{y_1, \ldots, y_m\}$ contain all the variables occurring in p. Let $\underset{\sim}{A}\epsilon M_d$ be arbitrary.

Then by an *ω-trace* of p in $\underset{\sim}{A}$ we understand a sequence

$s = <s_0, \ldots, s_m>$ such that $s_0, \ldots, s_m \epsilon\ {}^{\omega}A$ and (i), (ii) below are satisfied:

(i) $s_0(0) \overset{df}{=} i_0$.

(ii) Suppose $z \epsilon \omega$ and $s_0(z) = i_k$.
If $k = n + 1$ then $\forall_j\ (s_j(z) = s_j\ (z + 1))$,
else (1) and (2) below hold:

(1) If $\underline{u_k = "y_w \leftarrow \tau"}$ then $s_0(z + 1) = i_{k+1}$ and

for every $0 < j \leqslant m$,

$$s_j(z + 1) = \begin{cases} \tau[s_1(z), \ldots, s_m(z)]_{\underset{\sim}{A}} & \text{if } j = w \\ \\ s_j(z) \text{ otherwise} \end{cases}$$

(2) If $\underline{u_k = "\text{IF } \chi \text{ THEN } \nu"}$ then

$s_j(z + 1) = s_j(z)$ for every $0 < j \leqslant m$, and

$$s_0(z + 1) = \begin{cases} \nu \text{ if } \underset{\sim}{A} \models \chi[s_1(z), \ldots, s_m(z)] \\ \\ i_{k+1} \text{ otherwise} \end{cases}$$

Let $s_0, \ldots, s_m \epsilon\ {}^{\omega}A$. We define $\underline{\underset{\sim}{A} \models p[s_0, \ldots, s_m]}$ to hold

iff

$s = \langle s_0, \ldots, s_m \rangle$ is an ω-trace of p in $\underset{\sim}{A}$. $\underline{\underset{\sim}{A} \models p[s]}$ denotes the same.

Also if $s \epsilon\ {}^{\omega}({}^{\omega}A)$ then:

$$\underset{\sim}{A} \models p[s] \text{ iff } (\exists m \epsilon \omega)\ \underset{\sim}{A} \models p[s(0), \ldots, s(m)].$$

END of Definition

Remark

An ω-trace of p in $\underset{\sim}{A}$ is a sequence $\langle s_0, \ldots, s_m \rangle$ such that for every $j \leqslant m$, s_j is a function $s_j : \omega \to A$ from "time" ω into "data values" A. If $z \epsilon \omega$ and $0 < j \leqslant m$, then $s_j(z)$ is the *value* of the variable y_j at time point z. We use y_0 as "*the control variable*" of p. I.e. $s_0(z)$ is considered to be the "value of the control or execution" at time point $z \epsilon \omega$. Thus $s_0(z)$ is supposed to be a "*label*" in the program p. Note that the labels of $p = \langle (i_0 : u_0), \ldots, (i_n : u_n), (i_{n+1} : \text{HALT}) \rangle$ are $i_0, \ldots, i_{n+1}$.

The sequence $<s_0, \ldots, s_m>$ is the *history of an execution* of p in $\underset{\sim}{A}$ along the "time axis" $\underset{\sim}{\omega} = <\omega, +, \cdot, 0,1>$.

End of Remark

Now, *ω-traceof* is defined to be a function:

$$\omega\text{-traceof} : P_d \times M_d \to \text{"Meanings"}$$

such that for every program scheme $p \epsilon P_d$ and model $\underset{\sim}{A} \epsilon M_d$, the meaning of p in $\underset{\sim}{A}$ is defined to be:

$$\omega\text{-traceof}(p,\underset{\sim}{A}) \overset{df}{=} \{ s \epsilon^{(m+1)}({}^{\omega}A) : \underset{\sim}{A} \models p[s] \} \quad .$$

Then

$$\mathbb{P}_d = <P_d, M_d, \omega\text{-trace of}>$$

is a language in the sense of Def. 2.1.

Example 2.4.

Let $Th \subseteq F_d$ be a fixed theory.

Notation: $\overline{x} = <x_0, \ldots, x_m>$ where $|\overline{x}| \overset{df}{=} m + 1 \quad (m \epsilon \omega)$.

Define

$$P_d' = \{ \varphi(\overline{x}, \overline{y}) \epsilon F_d : |\overline{x}| = |\overline{y}| \epsilon \omega, \; Th \models (\forall \overline{x} \; \exists ! \overline{y}) \varphi(\overline{x},\overline{y}) \} \quad .$$

Let $\underset{\sim}{A} \epsilon M_d$, $\varphi(\overline{x},\overline{y}) \epsilon P_d'$, and $|\overline{x}| = m + 1$. Now $<s_0, \ldots, s_m> \epsilon^{(m+1)}({}^{\omega}A)$ is called an *ω-trace of φ in* $\underset{\sim}{A}$ iff

$$(\forall i \epsilon \omega) \; \underset{\sim}{A} \models \varphi[s_0(i), \ldots, s_m(i), s_0(i + 1), \ldots, s_m(i + 1)],$$

and the function *ω-traceof'* with domain $P_d' \times M_d$ is defined as:

$$\omega\text{-traceof}'(\varphi(\overline{x},\overline{y}),\underset{\sim}{A}) \overset{df}{=} \{ s \epsilon^{(m+1)}({}^{\omega}A) : s \text{ is an } \omega\text{-trace of } \varphi \text{ in } \underset{\sim}{A} \} \; .$$

Now,

$$<P_d', Md(Th), \omega\text{-traceof}'>$$

is a language.

Note that this language is practically the same as the language $\mathbb{P}_d$ in Example 2.3. (More precisely, $\mathbb{P}_d$ is *"recursively reducible"* to a language of this kind. Cf. Andréka – Gergely – Németi 77 § 2.1. p. 13.)
What is more unusual, the following is also a language:

$$<P_d', M_d, \omega\text{-traceof}'>$$

but as opposed to the first one, this second language is more similar to nondeterministic languages.

The folowing is a simplified version of Def. 2.1. (Cf. Makowsky 73, Németi 76, Gergely 77, and unit 1.1. of Gergely – Vershinin 78.)

DEFINITION 2.2.

$\mathbb{L}$ is defined to be a *language* with semantics (in the simplified sense)

iff

$\mathbb{L}$ is a triple

$\mathbb{L} = <S,M, \models>$ such that:

– S is a *set*,

– M is a *class,* and

– $\models$ is a ***binary relation*** between elements of S and M i.e. $\models \subseteq M \times S$.

If $\mathbb{L} = <S,M, \models>$ is a language in the sense of Def. 2.2. then $\models$ is called the *validity relation* of $\mathbb{L}$. For any $\varphi \epsilon S$ and $\mathfrak{G} \epsilon M$, if $\mathfrak{G} \models \varphi$ then we say that φ is valid in $\mathfrak{G}$.

REMARK 2.1.

The language as defined in Def. 2.2. is a special case of the one as deffined in Def. 2.1. To see this observe that $\models$ can be conceived of as a meaning function $\models : S \times M \rightarrow \{0,1\}$.

CONVENTION

Unless otherwise specified, by a language we shall understand one of the simplified version in the sense of Def. 2.2.

REMARK 2.2.

Languages in this sense were investigated in Gergely – Úry 78 Ch. 4 p. 59-70. Cf also p. 26. of Gergely – Úry 78, Gergely – Vershinin 78 unit 1.1., Németi 76, and Németi – Sain 77.

Example 2.5.

The first order language $\mathbb{L}_d = \langle F_d, M_d, \models\rangle$ of type d is a language in the sense of Def. 2.2. as well. Namely, the (usual first order) validity relation $\models$ can be conceived of as a binary relation between first order formulas and classical models of type d as follows (see also Def. 1.8.):
For any $\varphi \in F_d$ and $\underset{\sim}{A} \in M_d$,

$$\underset{\sim}{A} \models \varphi \qquad \text{iff} \qquad \models(\varphi, \underset{\sim}{A}) = {}^{\omega}A$$

(see the definition of the meaning function

$$\models\ : F_d \times M_d \to Sb({}^{\omega}A)$$

in Example 2.2.).
Cf. also Gergely – Vershinin 78 unit 3.2. p. 527.

Example 2.6.

As further examples for Def. 2.2. above, recall that in the case of classical higher order languages of a fixed similarity type d the models are the usual first order models i.e. elements of M_d while the formulas are allowed to contain new "higher order" variable symbols too.

For every $n \in \omega$ and similarity type d, the *classical n-th order language* $\mathbb{L}^n_d$ of *type* d (with the classical semantics, see e.g. Monk 76 p. 491, p. 493, Andréka – Gergely – Németi 73,75) is an example for a language as defined in Def. 2.2.

NOTATIONS

Let $n \in \omega$. If $\mathbb{L}^n_d$ is the classical *n-th order language of type d*, then $\mathbb{L}^n_d$ will be denoted by the triple

$$\mathbb{L}^n_d = \langle F^n_d, M_d, \overset{n}{\models}\rangle.$$

E.g. $\langle F^2_d, M_d, \overset{2}{\models}\rangle$ ddenotes the clasical second order language of type d.

In the case of classical first order language we shall omit the indices 1 (as we did already in the case of the first order language of set theory in §1). I.e.:

$$\langle F^1_d, M_d, \overset{1}{\models}\rangle = \mathbb{L}_d = \langle F_d, M_d, \models\rangle.$$

Example 2.7.

Further examples for languages in the sense of Def. 2.2.: (1), (2), and (3) below are *languages for reasoning about programs* (cf. §2.3. and §3).

(1) $\mathbb{D}_d = <P_d \times F_d, M_d, \models>$

as defined in Andréka – Németi 78, 78a, and Gergely – Úry 78 Def. 9.6. p. 125.

(2) $\mathbb{T}\mathbb{D} = <[(P_d \times F_d) \cup TF_d], TM_d, \models>$

as defined in §3 of the present work. The definition of $\mathbb{T}\,\mathbb{D}_d$ can also be found in Andréka – Németi – Sain 78, 79, 79a. Cf. also § 2.3. here.

(3) $\mathbb{D}_d^{\omega} = <P_d \times F_d, M_d, \overset{\omega}{\models}>$

as defined in Manna 74 Ch. 4, Andréka – Németi 78, 78b, Andréka – Németi – Sain 79, Gergely – Szőts 78, Gergely – Úry 78.

(4) First order language *with valuations*:

Define

$$M_d' \overset{df}{=} \{ (\underset{\sim}{A},q) : \underset{\sim}{A} \epsilon M_d \text{ and } q \epsilon {}^{\omega}A \} .$$

Let $\varphi \epsilon F_d$.

Now we define $(\underset{\sim}{A},q) \mathrel{\dot{\models}} \varphi$ to hold iff $\underset{\sim}{A} \models \varphi[q]$,
cf. Example 2.2.

Clearly $<F_d, M_d', \dot{\models}>$ is a language in the sense of Def. 2.2., since $\dot{\models} \subseteq (F_d \times M_d')$.

(5) *The language of program schemes of type d*:

Let $M_d'' \overset{df}{=} \{ (\underset{\sim}{A},q) : \underset{\sim}{A} \epsilon M_d \text{ and } q \epsilon {}^{\omega}({}^{\omega}A) \}$.

First recall from Example 2.3. that for every program scheme $p \epsilon P_d$, $\underset{\sim}{A} \epsilon M_d$, and $q \epsilon^{\omega}({}^{\omega}A)$,

$\underset{\sim}{A} \models p[q]$ iff q is an ω-trace of p in $\underset{\sim}{A}$.

Then we define

$$(A,q) \mathrel{\ddot{\models}} p \quad \text{iff} \quad [q \epsilon^{\omega}({}^{\omega}A) \text{ and } q \text{ is an } \omega\text{-trace of } p \text{ in } \underset{\sim}{A}].$$

Now:

$<P_d, M_d'', \ddot{\models}>$ is a language in the sense of Def. 2.2.

(6) *Modal language with Kripke semantics*:

$$ML'_d \overset{df}{=} <MF_d, KM_d, \models>$$

where MF_d and KM_d are as in Example 2.2(2). (MF_d is the set of first order modal formulas and KM_d is the class of Kripke models of type d.)

$\models$ is a binary relation $\models \subseteq KM_d \times MF_d$ defined as Kripke did, see Dahn 73, 78.

(7) *Intuitionistic language of type* d:

$$IL_d \overset{df}{=} <F_d, KM_d, \models>$$

where $\models \subseteq (F_d \times KM_d)$ is defined as in Andréka – Dahn – Németi 76 interpreting "$\neg$" of F_d as intuitionistic negation.

(8) *Intensional language of type* d:

$INL_d = <IF_d, IM_d, \models>$ as defined in Gallin 75, Montague 73, again satisfies Def. 2.2.

Note that for any similarity type t we have:

$KM_d \not\subseteq M_t$, $IM_d \not\subseteq M_t$, therefore any definition of a language postulating that the interpretations be classical structures *excludes* the last three examples (even if many-sorted structures are allowed!).

MOTIVATIONS 2.1.

We want to make restrictions on the general notion of a language given in Def.2.1. to make the "mathematical model" $<S,M,k>$ of a language more defined i.e. we want to fit it closer to our intuition about languages and their mathematical modelling. As a first step, we want to exclude situations that are not relevant to our intuition based on our knowledge of some kinds of languages, e.g. classical first order languages, programming languages, natural languages etc.

One can imagine several mathematical tools for this purpose. E.g. in Németi 76, Andréka – Németi 76, Andréka – Gergely – Németi 78 this tool was category theory. In Andréka – Gergely – Németi 73, 77, Németi – Sain 78, Andréka – Németi 75, 79 this tool was Universal Algebra (Universal Algebraic Logic). The features of the notion of a language with semantics we are trying to concentrate on *now* require a certain amount of *Set Theory*. There are cartain aspects of the notion of a language with semantics which *can* be formulated if we use some Set Theory. (We are not able to make these "aspects" or "features" explicite and precise without it but this does not mean that they cannot.) The amount of Set Theory needed is not much e.g. the notion of a formula and a model of Set Theory. The notions needed will be recalled during the text.

The idea of the present approach originates from Sacks 72. *There* (cf. Sacks 72 e.g. p. 2. and p. 22) it is stressed that the basic notions of classical first order model theory are *absolute* in the set theoretic sense: line 7 of p.2. in Sacks 72 reads as: "The central notions of model theory are absolute, and absoluteness, unlike cardinality , is a logical concept. That is why model theory does not founder on that rock of undecidability, the Generalized Continuum Hypothesis." It is also mentioned in Sacks 72 that "real logical notions do not depend on set theoretical hypotheses". This suggested our present notions of a language's being "absolute" and being "*stable*" (see Def. 2.6. later.) Absoluteness of a language will be defined in a subsequent paper.

END of Motivations.

Our first requirement for a language $\mathbb{L} = \langle S,M,k\rangle$ will be to be able to *define* $S,M,$ and k in Set Theory. Such a "definable" language will be called a *well presented language.* The reason for requiring this is *not only* the obvious one (a mathematical model of a language *has* to be defined some way) but there are also deaper considerations about M and k being reasonably coherent semantics to S or not.

DEFINITION 2.3. (Németi – Sain 78 Def. 2.)

A language $\mathbb{L} = \langle S,M,k\rangle$ is *well presented* if (1) – (3) below hold.

1.) $S \subseteq L(\omega)$, S is definable in Set Theory, and S is recursively enumerable. More precisely, we require the following (a) – (c):

a.) $S \subseteq L(\omega)$ and $S \epsilon V$.

b.) There exists a set theoretic formula $\sigma(x) \epsilon F_\epsilon$ such that $S = \{ a \epsilon V : \sigma(a) \}$.

(c) $(\forall x \epsilon S)\ ZFC \models$ '$x \epsilon S$'.
Here we note that '$x\epsilon S$' ϵF_ϵ exists since S is definable and x is also definable because $x \epsilon L(\omega)$: By $x\epsilon L(\omega)$, there exists a name $\bar{x} \epsilon F_\epsilon$ of x in (V,ϵ), cf. Remark 1.6, and then $\sigma(\bar{x})$ is the formula in F_ϵ denoted by '$x\epsilon S$' (according to the notational convention at the end of § 1). *)

2.) M is a *class* definable in set theory. I.e.: there is a set theoretic formula $\mu(x)\epsilon F_\epsilon$ such that M is the collection of all such sets $a\epsilon V$ for wich $\mu(a)$ is true:

$$M \overset{df}{=} \{ a\epsilon V : \mu(a) \} .$$

3.) k is a *function* (class of pairs) definable in set theory. I.e., there is a set theoretic formula $\kappa(x,y,z)\epsilon F_\epsilon$ such that $k(\subseteq V)$ is the class of those triples $\langle a_1, a_2, a_3\rangle$ of sets for wich $\kappa(a_1, a_2, a_3)$ is true, $a_1 \epsilon S$, $a_2 \epsilon M$, and a_3 is an arbitrary set:

$$k \overset{df}{=} \{ \langle a_1, a_2, a_3\rangle \epsilon V : \kappa(a_1, a_2, a_3), a_1 \epsilon S, a_2 \epsilon M, a_3 \epsilon V \}.$$

If $\mathbb{L} = \langle S,M,k\rangle$ is well presented then we shall use the following terminologies:

" $\mathbb{L}$ is well presented (or presented) by $\langle\sigma,\mu,\kappa\rangle$ ",

"$\langle\sigma,\mu,\kappa\rangle$ presents $\mathbb{L}$ ",
"S is presented by σ",

"M is presented by μ",

"k is presented by κ".

* By means of results given in e.g. Monk 76, one can prove that $S \subseteq L(\omega)$ and $S = \{a\epsilon V : \sigma(a)\}$ imply that $(\forall x\epsilon S)\ ZFC \models$ '$x\epsilon S$' is really equivalent to the fact that S is recursively enumerable. Cf. Cor. 14.13 in Monk 76.

Example 2.8.

Let $f : \omega \to \omega$ be a function *undefinable* in ZFC set theory. Suppose f is not even parametrically definable. Such a function exists since there are only countably many definable functions. Then

$$<\omega, \omega, f>$$

is a language in the sense of Def. 2.2. but it is *not well presented.*

Example. 2.9.

The following languages are *well presented*:

1.) For any similarity type d, the *first order language* $\mathbb{L}_d$ *of type* d *is well presented.* This is claimed in Chang – Keisler 73 as well, see p. 28. line 5 bottom up there. We shall prove it in Thm. 2.3. In the proof of Thm. 2.3. we shall construct formulas $\sigma_d, \mu_d, \kappa_d \in F_\epsilon$ such that $\mathbb{L}_d$ is well presented by $<\sigma_d, \mu_d, \kappa_d>$. In Hinman 78. p. 217. a set theoretic formula definiting the first order validity relation (corresponding to our κ_d) is denoted by "Mod".

2.) Let d be an arbitrary similarity type and let $n \epsilon \omega$. Then the *classical n-th order language*

$$\mathbb{L}^n_d = <F^n_d, M_d, \overset{n}{\models}>$$

is well presented.

References on these languages:

– second order language of type d: Monk 76. p. 491, Andréka – Gergely – Németi 75, Gergely – Vershinin 78 unit 3.1;

– n-th order language of type d: Monk 76 p. 493, Andréka – Gergely – Németi 73.

3.) <Modal formulas, Kripke models, $\models$ >.
See Andréka – Dahn – Németi 76.

4.) <Algorithmic logic formulas, Models, $\models$ >.
See Banachowski at al 77.

5.) <Program schemes, Models, "Traces">.
See Andréka 78, Andréka – Németi – Sain 78, 79, Gergely – Szőts 78, Gergely – Úry 78.

6.) <Program schemes, Models, "Computed functions">.
See Gergely – Úry 78, Manna 74.

7.) <Program schemes, $\begin{matrix}\text{Continuous}\\ \text{Algebras}\end{matrix}$, "Least fixed points">.

See Courcelle – Guessarian 77, Goguen at al 77.

DEFINITION 2.4

We define the set of *tautologies* of a well presented language $\mathbb{L} = <S,M, \models>$ as follows:

$$TA_{\mathbb{L}} \overset{df}{=} \{ \varphi\epsilon S : M \models \varphi \} .$$

More precisely, if $\mathbb{L}$ is presented by $<\sigma, \mu,\kappa>$, then

$$TA_{\mathbb{L}} \overset{df}{=} \{ \varphi\epsilon S : (V,\epsilon) \models \forall x[\mu(x) \rightarrow \kappa(x, \overline{\varphi})] \}$$

where $\overline{\varphi}\ \epsilon F_{\epsilon}$ is the name of $\varphi\ \epsilon S$.

DEFINITION 2.5. (Gergely – Úry 78 p. 60)

A well presented language $\mathbb{L} = <S,M,\models>$ is called *complete*

iff

$TA_{\mathbb{L}}$ is recursively enumerable. [*)]

Remark 2.3.

Our notion of completeness is called "weak completeness" in Monk 76 p.204.

NOTIVATIONS 2.2.

We shall recall later from textbooks on Set Theory that L denotes a subclass of V i.e. $L \subseteq V$ such that (L, ϵ) is also a "model of Set Theory" i.e. (L,ϵ) is a possible world of Set Theory if (V, ϵ) is one. L is usually called the constructible universe. (See Def. 2.7.) CH abbreviates the Continuum Hypothesis.

Suppose $(V,\epsilon) \not\models$ CH i.e. $V \neq L$. Let

$\underset{\sim}{A}\epsilon L\subseteq V$ be infinite. ($\underset{\sim}{A}\epsilon M_d$ and $|A| \geqslant \omega$.) We shall see the existence of a (fairly simple) formula $\varphi\epsilon F_d^3$ such that the meaning of φ in $\underset{\sim}{A}$ does not depend on φ and $\underset{\sim}{A}$ but it depends on their set theoretical "surrounding". I.e. if we change the set theoretical world arround $\underset{\sim}{A}$ then the meaning of φ in $\underset{\sim}{A}$ changes. E.g.:

$(V,\epsilon) \models$ ' $\underset{\sim}{A} \models \varphi$ ' but

$(L,\epsilon) \models$ ' $\underset{\sim}{A} \not\models \varphi$'.

[*] More precisely, $TA_{\mathbb{L}}$ is Turing-enumerable, cf. the proof of Thm. 2.1.

(If we started with $(V,\epsilon) \models CH$ then we can *reverse* the procedure by considering $(L,\epsilon) \models$ '$\underset{\sim}{A} \not\models \varphi$' to be our starting point and "build a sufficiently large V around" L such that $(V,\epsilon) \models$' $A \models \varphi$'.)

All these will be proved later about higher order languages. The proof will be especially transparent for $\mathbb{L}^3_d = \langle F^3_d, M_d, \overset{3}{\models} \rangle$. The above quoted existence of $\varphi \epsilon F^3_d$ such that for every infinite $\underset{\sim}{A} \epsilon M_d$ $(V,\epsilon) \models$' $\underset{\sim}{A} \models \varphi$' but $(L,\epsilon) \models$' $\underset{\sim}{A} \not\models \varphi$' means that $\varphi \epsilon F^3_d$ does not speak about its models $A \epsilon M_d$ but is speaks about *something else* namely their set theoretic surrounding (V,ϵ). We shall argue that this means that there was a *mistake* somehow in the definition of $\mathbb{L}^3_d$ since a language should speak about its models and not about something *else*. Of course such a language $\mathbb{L}^3_d$ can not be complete or anything if it speaks about (V,ϵ) or other fancy metaobjects instead of speaking about its own models. We shall argue that the "linguistic and logical intuition" about a language $\mathbb{L} = \langle S,M, \models \rangle$ requires the syntax $\varphi \epsilon S$ to speak about its interpretations or models $\mathfrak{G} \epsilon M$ and not to speak about anything else.

(We shall also argue that with sufficient care and time in all intuitively reasonable cases the definition of $\mathbb{L}$ can be modified and elaborated as to meet the above requirements.)

DEFINITION 2.6.

Let the language $\mathbb{L} = \langle S,M, \models \rangle$ be well presented. Then we define:

1.) $\mathbb{L}$ *is stable*

iff

for every $\varphi \epsilon S$, if $\models \varphi$ is true then it is also true in every model $\underset{\sim}{W} \epsilon M_\epsilon$ of ZFC. (Recall that $M_\epsilon \subseteq V$ and thus $\underset{\sim}{W} \epsilon V$.)
More precisely:

$\mathbb{L}$ is stable

iff

for every $\varphi \epsilon S$ such that $\models \varphi$ it is also true that in every model $\underset{\sim}{W} \epsilon Md(ZFC)$ we have $\underset{\sim}{W} \models$ ' $\models \varphi$' .

2.) *In other words*:

$\mathbb{L}$ *is stable*

iff

$(\forall \varphi \epsilon S)[\models \varphi$ implies $ZFC \models$' $\models \varphi$'].

3.) *I.e.* for every valid formula φ of $\mathbb{L}$ it should be a *"mathematical truth"* that φ is valid.

4.) Now we give a more detailed and *more precise* formulation of the above defined notion of *stability of languages.*

To this end *recall* our convention that "$\models \varphi$ is true" means "$(V,\epsilon) \models$' $\models \varphi$' ". (See §1 before Def. 1.1.) *Recall* that the language $\mathbb{L} = \langle S,M, \models \rangle$ is well presented by *set theoretic formulas* $\sigma(y)$, $\mu(x)$, and $\kappa(x,y)$ iff

$$S = \{ y\epsilon V : \sigma(y) \} = \{ y\epsilon V : (V,\epsilon) \models \sigma[y] \}, \text{ etc, see Def. 2.3.}$$

Let $\Theta(y)$ denote the set theoretic formula

$$\forall x[\mu(x) \rightarrow \kappa(x,y)].$$

Clearly $\Theta(y)$ is a set theoretic formula in one free variable y. Therefore, for any $b\epsilon V$, the statement $(V,\epsilon) \models \Theta[b]$ is meaningful. (It says that the set b has the property Θ. It is either true or not.)

Recall that $S \subseteq L(\omega)$ and therefore for every $\varphi\epsilon S$ there exists a "ZFC -name" $\overline{\varphi}\epsilon F_\epsilon$ which singles out φ from (V,ϵ). I.e. $ZFC \vdash \exists ! x(x = \overline{\varphi})$, etc. See Remark 1.6. Observe that for every $\varphi\epsilon S$ we have $\Theta(\overline{\varphi})\epsilon F_\epsilon$. Now

$\mathbb{L}$ *is stable*

iff

there exists a presentation $\langle\sigma,\mu,\kappa\rangle$ of $\mathbb{L}$ such that for every $\varphi\epsilon S$ (*i*) implies (ii) below.

(i) $(V,\epsilon) \models \Theta[\varphi]$.
(ii) $ZFC \models \Theta(\overline{\varphi})$.

The more precise formulation of (ii) reads as:

(ii)' $(V,\epsilon) \models$' $ZFC \models \Theta(\overline{\varphi})$'.

An equivalent formulation of (ii) is the following:
(ii)" $(V,\epsilon) \models$' For every $\underset{\sim}{W}\epsilon Md(ZFC)$ $\underset{\sim}{W} \models \Theta(\overline{\varphi})$'.

Or equivalently:

For every $\underset{\sim}{W}\epsilon M_\epsilon$ if $(V,\epsilon) \models$' $W \models ZFC$' then also $(V,\epsilon) \models$' $\underset{\sim}{W} \models \Theta(\overline{\varphi})$'.

Example 2.10.

The following two languages are *not stable.*

1.) Denote by Eq the set of *Diophantine equations* (automatic formulas of the language of $\underset{\sim}{\omega}$).
Let the language D be defined as follows:

$$D \overset{df}{=} <Eq, {}^{\omega}\omega, \models>$$

where for every $e \epsilon Eq$ and $h \epsilon {}^{\omega}\omega$ we define

$$h \models e \quad \text{iff} \quad \underset{\sim}{\omega} \not\models e[h].$$

I.e. h is a D-model of e if h is not a solution of e in $\underset{\sim}{\omega}$.

Clearly $\models e$ iff e has no solution in $\underset{\sim}{\omega}$.

I.e.:

e is a tautology of the language D

iff

$\underset{\sim}{\omega} \not\models \exists\, \overline{y}\, e(\overline{y})$.

Now we claim that D *is not stable.*

Proof:

By Davis 73, Hilbert's tenth problem is unsolvable. This implies the existence of a Diophantine equation $e(\overline{y})$ such that

$$(V,\epsilon) \not\models \text{'}\underset{\sim}{\omega} \models \exists\, \overline{y}\, e(\overline{y})\text{'} ,$$

but for some model $\underset{\sim}{W} \epsilon Md(ZFC)$ (inside of (V,ϵ)) we have

$$\underset{\sim}{W} \models \text{'}\underset{\sim}{\omega} \models \exists\, \overline{y}\, e(\overline{y})\text{'} .$$

I.e. $\models e$ is true in (V,ϵ) but the same $\models e$ is false in some model $\underset{\sim}{W}$ of ZFC (inside of (V,ϵ)). By the definition of $\models$, this means that the language D is not stable.

OED

2.) Define the language L as follows:

$$L = <F,\omega, \models >$$

where

– F is the set of all first order formulas of the language of $\underset{\sim}{\omega}$ in one free variable x;

– For every $n \epsilon \omega$ and $\varphi(x) \epsilon F$ we define

$$n \vDash \varphi \quad \text{iff} \quad \underset{\sim}{\omega} \vDash \varphi\,[n].$$

We clain that L *is not stable.*
The proof is left to the reader.

3.) Note that the languages $\mathbb{D}_d$ and $\mathbb{T}\,\mathbb{D}_d$ are stable while $\mathbb{D}_d^{\omega}$ is not stable (see Example 2.7.).

THEOREM 2.1.

Let $\mathbb{L} = \langle S, M, \vDash \rangle$ be well presented. Then:

$\mathbb{L}$ is copmlete iff $\mathbb{L}$ is stable.

PROOF

First we recall some notions about computability in $L(\omega)$:
The followings are definable in $L(\omega)$:

1.) a function $f : L(\omega) \to L(\omega)$ is *Turing-computable,*

2.) a set $H \subseteq L(\omega)$ is *Turing-enumerable,*

3.) a set $H \subseteq L(\omega)$ is *Turing-decidable.*

(Recall that the above notions restricted to $\omega \subseteq L(\omega)$ are:

1.) recursive function

2.) recursively enumerable set

3.) recursive (decidable) set.)

In this proof we strictly distinguish between "recursive" and "Turing-computable" etc. Cf. the "Terminology about computability" in § 1.

We shall prove the theorem in two steps:

1.) We prove that if $\mathbb{L}$ is complete then it is stable.
2.) We suppose that $\mathbb{L}$ is stable and we give a calculus for $\mathbb{L}$.

PART (1) of the proof

In the first part of the proof we shall use the following lemmas:

LEMMA 2.1.

Let the language $\mathbb{L} = \langle S,M, \models\rangle$ be well presented by the set theoretic formulas $\sigma(y)$, $\mu(x)$, $\kappa(x,y)$.

Define:

$$\Theta(y) \overset{df}{=} \forall x[\mu(x) \rightarrow \kappa(x,y)] \wedge \sigma(y).$$

Then

$$\mathbb{L} \text{ is complete} \quad \text{iff}$$

$$\{ a \epsilon L(\omega) : \Theta(a) \} \quad \text{is Turing-enumerable.}$$

PROOF of Lemma 2.1.

By Def. 2.5.,

$$\mathbb{L} \text{ is complete} \quad \text{iff}$$

$$\{ \varphi \epsilon S : (V,\epsilon) \models \forall x[\mu(x) \rightarrow \kappa(x,\overline{\varphi})] \} \quad \text{is Turing-enumerable.}$$

Now observe that

$$\{ \varphi \epsilon S : (V,\epsilon) \models \forall x[\mu(x) \rightarrow \kappa(x, \overline{\varphi})] \} = \{ a \epsilon L(\omega) : \Theta(a) \}$$

since $S \subseteq L(\omega)$ and S is presented by $\sigma(y)$.

QED Lemma 2.1.

Recall from Monk 76 that a set $R \subseteq L(\omega)$ is called *syntactically definable* iff there exists a $\psi \epsilon F_\epsilon$ such that both (i) and (ii) below hold:

(i) For every $x \epsilon L(\omega)$,

$$x \epsilon R \quad \text{iff} \quad ZFC \vdash \psi(\overline{x})$$

(ii) For every $x \epsilon L(\omega)$,

$$x \epsilon R \quad \text{iff} \quad (V,\epsilon) \models \psi[x].$$

Note that the present notion is strictly stronger than the one introduced in Def. 1.1. Namely, if a set $R \subseteq L(\omega)$ is "syntactically definable" then it is also "definable" in the sense of Def. 1.1. . But there are definable subsets of $L(\omega)$ which are not syntactically definable. (The same applies to having a name: a set has a name iff it is definable in the sense of Def. 1.1., cf. Remark 1.6.)

LEMMA 2.2.

$R \subseteq L(\omega)$ is Turing-enumerable

iff

R is syntactically definable.

Proof: is at the end of this part (1) of the proof.

Now *suppose* that $\mathbb{L} = <S,M, \models>$ is *complete.* We prove that $\mathbb{L}$ is stable. Since $\mathbb{L}$ is well presented, there are set theoretic formulas $\sigma(y)$, $\mu(x)$, $\kappa(x,y) \epsilon F_\epsilon$ such that $\mathbb{L}$ is presented by $<\sigma,\mu,\kappa>$.
Define $\Theta(y) \epsilon F_\epsilon$ as follows:

$$\Theta(y) \overset{df}{=} \forall x[\mu(x) \to \kappa(x,y)] \ \wedge \ \sigma(y).$$

By Lemma 2.1. the completeness of $\mathbb{L}$ implies that

$$\{ a \epsilon L(\omega) : \Theta(a) \} \text{ is Turing-enumerable.}$$

By Lemma 2.2., this implies that $\{ a \epsilon L(\omega) : \Theta(a) \}$ is syntactically definable i.e. *there exists* a formula $\psi \epsilon F_\epsilon$ such that:

(*) $(y \epsilon \{ \dot{a} \epsilon L(\omega) : \Theta(a) \}$ iff $ZFC \vdash \psi(\bar{y})$ iff $(V,\epsilon) \models \psi[y])$.

Now we introduce a new set theoretic formula $\kappa'(x,y)$ such that $<\sigma,\mu,\kappa'>$ presents the language $\mathbb{L}$.

$$\kappa'(x,y) \overset{df}{=} (\kappa(x,y) \vee [\psi(y) \ \wedge \ \mu(x)]).$$

Clearly $\models \ = \ \{ (x,y) \epsilon V : (V,\epsilon) \models \kappa'[x,y] \}$ by (*).
This can be seen as follows:

Suppose $x \models y$. Then $\kappa(x,y)$ and therefore $\kappa'(x,y) V[\psi(y) \ \wedge \ \mu(x)]$.
Suppose $\kappa'(x,y)$. Then $\kappa(x,y)$ or $\psi(y) \wedge \mu(x)$. Now either $\psi(y) \wedge \mu(x)$ or not. Suppose $(\psi(y) \wedge \mu(x))$ i.e. $(V,\epsilon) \models (\psi[y] \wedge \mu[x])$. By (*) this implies $\Theta(y)$. Since $\mu(x)$, this implies $\kappa(x,y)$ i.e. $x \models y$. Thus $<\sigma,\mu,\kappa'>$ presents $\mathbb{L}$.
Let $\Theta'(y) = \forall x[\mu(x) \to \kappa'(x,y)]$. Since $<\sigma,\mu,\kappa'>$ is a presentation of $\mathbb{L}$ to prove that $\mathbb{L}$ is stable, it is enough to show that for every $\varphi \epsilon S$, if $\models \varphi$ then $ZFC \models \Theta'(\bar{\varphi})$ by Def. 2.6.
Let $\models \varphi$. Then obviously $\Theta(\bar{\varphi})$ is true in (V,ϵ). By (*), $ZFC \vdash \psi(\bar{\varphi})$. Therefore by Gödel's Completeness Theorem and the definition of Θ' we have $ZFC \models \Theta'(\bar{\varphi})$. This clearly implies that for every model $\underset{\sim}{W} \epsilon M_\epsilon$ of ZFC, $\Theta'(\bar{\varphi})$ is true in $\underset{\sim}{W}$ i.e. ' $\models \varphi$ ' is true in $\underset{\sim}{W}$. Remains to prove Lemma 2.2. (During the above proof we had to use only "one direction" of Lemma 2.2.)

PROOF of Lemma 2.2.

Here we prove only one direction which was needed in the proof (1): We prove that if $R \subseteq L(\omega)$ is Turing-enumerable *then* there exists a $\psi \epsilon F_\epsilon$ such that for every $x \epsilon L(\omega)$,

$$(x \epsilon R \text{ iff } ZFC \vdash \psi(\overline{x}) \text{ iff } (V,\epsilon) \models \psi[x]).$$

The following lemma is known:

Lemma 2.3.

A relation $R \subseteq {}^n\omega$ $(n \epsilon \omega)$ is recursively enumerable

iff

there exists a formula $\varphi \epsilon F_\epsilon$ such that: for every n-tuple $<x_0, \ldots, x_{n-1}> \epsilon {}^n\omega$,

$$[(<x_0, \ldots, x_{n-1}> \epsilon R \text{ iff } PA \vdash \varphi(\overline{x}_0, \ldots, \overline{x}_{n-1})) \text{ and}$$
$$(<x_0, \ldots, x_{n-1}> \epsilon R \text{ iff } \underset{\sim}{\omega} \models \varphi(\overline{x}_0, \ldots, \overline{x}_{n-1}))].$$

(PA abbreviates the set of axioms of Peano Arithmetic.) I.e., R is recursively enumerable iff there exists a formula $\varphi \epsilon F_\epsilon$ such that R is both syntactically and semantically definable by φ in Peano Arithmetic.

Proof

To see that Lemma 2.3. is true, see Monk 76 Cor. 14.13 p.251. and the proof of Prop. 14.8. there.

QED Lemma 2.3

A staightforward consequence of Lemma 2.3. is the following:

Lemma 2.4.

A relation $R \subseteq {}^n\omega$ $(n \epsilon \omega)$ is recursively enumerable

iff

there exists a formula $\varphi \epsilon F_\epsilon$ such that for every n-tuple $<x_0, \ldots, x_{n-1}> \epsilon {}^n\omega$,

$$[<x_0, \ldots, x_{n-1}> \epsilon R \text{ iff } ZFC \vdash \varphi(\overline{x}_0, \ldots, \overline{x}_{n-1}) \text{ iff }$$
$$(V,\epsilon) \models \varphi(\overline{x}_0, \ldots, \overline{x}_{n-1})].$$

Proof

To see that Lemma 2.4. is a consequence of Lemma 2.3 , observe that PA is a subtheory of ZFC.

QED Lemma 2.4.

Now we quote a lemma from the Theory of Algorithms.

Lemma 2.5.

Let $G \subseteq \omega$ be a recursive (decidable) set. Let the function g be such that both

$$g : L(\omega) \rightarrowtail\!\!\twoheadrightarrow G \subseteq \omega$$

and its inverse

$$g^{-1} : G \rightarrowtail\!\!\twoheadrightarrow L(\omega)$$

are bijective and Turing-computable.
Let further $R \subseteq L(\omega)$ be arbitrary.
Denote $g^{*}(R) = \{ g(r) \epsilon L(\omega) : r \epsilon R \}$.

Then:

R is Turing-enumerable

iff

$g^{*}(R)$ is recursively enumerable.

Proof: For a proof cf. e.g. Monk 76 Thm 3.39 p. 56.

QED Lemma 2.5.

The following lemma is also well known:

Lemma 2.6.

a.) There exists a set $G \subseteq \omega$ (called Gödel numbers) and a function $g : L(\omega) \to G$ (called Gödel numbering function) such that G and g satisfy the conditions of Lemma 2.5. I.e.:

$G \subseteq \omega$ is recursive, and

g and g^{-1} are bijective and Turing-computable.

b.) Further well known properties of the Gödel numbering function g are:

(i) g is syntactically definable, i.e. there is a $\gamma \epsilon F_{\epsilon}$ such that for every $x,y \epsilon L(\omega)$:

$(g(x) = y$ iff $ZFC \vdash \gamma(\overline{x},\overline{y})$ iff $(V,\epsilon) \models \gamma[x,y])$.

(ii) $ZFC \vdash$' g is a function' i.e.
$ZFC \vdash (\forall x \; \exists ! \, y)\, \gamma(\overline{x},\overline{y})$.

QED Lemma 2.6.

For Gödel numbers see e.g. Monk 76 p. 52, 72.

Now we prove Lemma 2.2. using Lemmas 2.3.– 2.6. above. Let $R \subseteq L(\omega)$ be Turing-enumerable. Denote the Gödel numbering function by g . $g.: L(\omega) \rightarrowtail\!\!\!\twoheadrightarrow G \subseteq \omega$. By Lemmas 2.5. and 2.6., the set

$$g^* \, (R) = \{ g(x) : x \epsilon R \} \subseteq G$$

is recursively enumerable.

Then by Lemma 2.4. there exists a formula $\varphi \epsilon F_{\epsilon}$ such that

0.) $(z \epsilon g^* \, (R)$ iff $ZFC \vdash \varphi(\overline{z})$ iff $(V,\epsilon) \models \varphi[z])$.

Let $x \epsilon R$ be arbitrary. Then $g(x) \epsilon g^*(R)$. Thus by (0):

1.) $ZFC \vdash \varphi(\overline{g(x)})$ and $(V,\epsilon) \models \varphi[g(x)]$
where $\overline{g(x)}$ is the ZFC -name of $g(x) \epsilon g^*(R)$.

Define

$$\psi(x) \overset{df}{=} (\exists \; y \epsilon \omega)[\gamma(x,y) \wedge \varphi(y)]$$

where γ defines g. $\psi(\overline{x}) \epsilon F_{\epsilon}$.

We prove that

$$(x \epsilon R \text{ iff } ZFC \vdash \psi(\overline{x}) \text{ iff } (V,\epsilon) \models \psi[x]).$$

a.) *Suppose* that $x \epsilon R$. Then by Lemma 2.6. (b) (i),

2.) $ZFC \vdash \gamma(\overline{x},\overline{g(x)})$ and $(V,\epsilon) \models \gamma[x,g(x)]$.

1.) and 2.) above imply $ZFC \vdash \psi(\overline{x})$ and $(V,\epsilon) \models \psi[x]$.

b.) *Suppose* that

3.) $ZFC \vdash \psi(\overline{x}) = (\exists \, y \epsilon \omega)[\gamma(\overline{x},y) \wedge \varphi(y)]$.

5. Recall that $\Theta(b) \epsilon F_\epsilon$. To the classical first order language $\langle F_\epsilon, M_\epsilon, \models \rangle$ there is enclosed an algorithm Alg_ϵ that decides whether any element $\pi \epsilon L(\omega)$ is a proof of $\Theta(b)$ from ZFC or not since ZFC is a Turing-decidable subset of F_ϵ. Let Alg contain this Alg_ϵ. Thus finally Alg decides whether c is a proof of $\Theta(b)$ from ZFC or not, it prints out "YES" if c is a proof of $\Theta(b)$ from ZFC and it prints out "NO" otherwise. Then Alg stops.

The above algorithm Alg satisfies the conditions $(***)$: For every input $(\varphi,p) \epsilon L(\omega)$ it stops in finitely many steps, it prints out "YES" if p is a (σ,μ,κ)-proof of φ and it prints out "NO" otherwise. Thus the above Alg is a Turing-machine that *decides* the set of (σ,μ,κ)-proofs as a subset of $L(\omega)$. Further, by $(**)$ above (which is a straightforward consequence of stableness of $\mathbb{L}$), for every $\varphi \epsilon S$, there exists a (σ,μ,κ)-proof of φ iff $\models \varphi$.

I.e., Alg is a *complete and sound calculus* (cf. Gergely – Úry 78 Ch. 4) for the language $\mathbb{L}$.

So far we gave a proof concept to the language $\mathbb{L}$ using the hypothesis that $\mathbb{L}$ is stable and that we are given the stable presentation $\langle \sigma,\mu,\kappa \rangle$ of $\mathbb{L}$ and we constructed a decision algorithm for the set of proofs. By Monk 76 this implies that $TA_{\mathbb{L}}$ is Turing--enumerable. Therefore $\mathbb{L}$ is *complete* in the sense of Def. 2.5.

QED of PART (2) of the proof.

QED Thm. 2.1.

2.2. Languages well known in logic

THEOREM 2.2.

Let d be a similarity type. Then the first order language

$\mathbb{L}_d = \langle F_d, M_d, \models \rangle$ of type d is stable.

Proof

Thm. 2.2. is a consequence of completeness of $\mathbb{L}_d$, *Thm.* 2.1., and the following Thm. 2.3.

QED Thm. 2.2.

THEOREM 2.3.

Let d be a similarity type. Then the first order language

$\mathbb{L}_d = \langle F_d, M_d, \models \rangle$ of type d is well presented.

(Cf. Remarks 1.2, 1.3, and 1.5)

Proof

We shall give a proof for a finite similarity type d. The proof for the case $|d| = \omega$ is quite similar. We shall give two formulas: $\sigma_d(y)$, $\kappa_d(x,y) \epsilon F_\epsilon$ by which F_d and $\models$ can be presented. The presentation of M_d is left to the reader.

1.) Presentation of F_d:

Recall that $F_d \subseteq L(\omega)$ and $F_d \epsilon V$. It is well known that F_d is recursively enumerable, moreover, it is recursive, cf. Monk 76 Prop. 10.15 p. 168, and Remark 1.1. To satisfy condition 1.) of Def. 2.3., remains to give a formula $\sigma_d(y) \epsilon F_\epsilon$ which defines F_d, i.e., for which:

$$(V,\epsilon) \models (F_d = \{ y : \sigma_d(y) \}).$$

First we define:

$$\delta(J) \overset{df}{=} [J \subseteq L(\omega) \wedge J = \{ (\varphi \wedge \psi), (\neg \varphi), (\exists x_n \varphi), R(x_{i_1}, \ldots, x_{i_m}): (R,m) \epsilon d, <i_1, \ldots, i_m> \epsilon {}^m\omega, n \epsilon \omega, \varphi,\psi \epsilon J \}].$$

$\delta(J) \epsilon F_\epsilon$, since $L(\omega)$ can be explicitely defined in ZFC set theory, i.e. $L(\omega)$ can be defined by a formula of F_ϵ.
Now let $\sigma_d(y) \overset{df}{=} (\forall J)[\delta(J) \rightarrow y \epsilon J]$. $\sigma_d(y) \epsilon F_\epsilon$. To see that $\sigma_d(y)$ defines F_d, one can easily show that:

$$ZFC \vdash (\exists ! J)\, \delta(J)$$

by using the facts that the elements of $L(\omega)$ are finite, $L(\omega)$ is transitive, moreover, every element of $L(\omega)$ is contained in a finite transitive set.

$ZFC \vdash (\exists ! J)\, \delta(J)$ and the definitions of $\delta(J)$ and $\sigma_d(y)$ imply that

$$(V,\epsilon) \models (F_d = \{ y : \sigma_d(y) \})$$

which was to be proved.

2.) Presentation of $\models$:

Suppose that M_d is presented by $\mu_d(x) \epsilon F_\epsilon$, i.e.

$$M_d = \{ \underset{\sim}{A} \epsilon V : (V,\epsilon) \models \mu_d [\underset{\sim}{A}] \} .$$

Recall the following notations:

If $\underset{\sim}{A} \epsilon M_d$ then its universe is denoted by A. If $(R,m) \epsilon d$ then the denotation of R in $\underset{\sim}{A}$ is denoted by $\underset{\sim}{A}_R$. Note that A and $\underset{\sim}{A}_R$ can be explicitely defined. Now we shall

5. Recall that $\Theta(b)\epsilon F_\epsilon$. To the classical first order language $<F_\epsilon, M_\epsilon, \models>$ there is enclosed an algorithm Alg_e that decides whether any element $\pi\epsilon L(\omega)$ is a proof of $\Theta(b)$ from *ZFC* or not since *ZFC* is a Turing-decidable subset of F_ϵ. Let Alg contain this Alg_e. Thus finally Alg decides whether c is a proof of $\Theta(b)$ from *ZFC* or not, it prints out "YES" if c is a proof of $\Theta(b)$ from *ZFC* and it prints out "NO" otherwise. Then Alg stops.

The above algorithm Alg satisfies the conditions $(***)$: For every input $(\varphi,p)\epsilon L(\omega)$ it stops in finitely many steps, it prints out "YES" if p is a (σ,μ,κ)-proof of φ and it prints out "NO" otherwise. Thus the above Alg is a Turing-machine that *decides* the set of (σ,μ,κ)-proofs as a subset of $L(\omega)$. Further, by $(**)$ above (which is a straightforward consequence of stableness of $\mathbb{L}$), for every $\varphi\epsilon S$, there exists a (σ,μ,κ)-proof of φ iff $\models \varphi$.

I.e., Alg is a *complete and sound calculus* (cf. Gergely – Űry 78 Ch. 4) for the language $\mathbb{L}$.

So far we gave a proof concept to the language $\mathbb{L}$ using the hypothesis that $\mathbb{L}$ is stable and that we are given the stable presentation $<\sigma,\mu,\kappa>$ of $\mathbb{L}$ and we constructed a decision algorithm for the set of proofs. By Monk 76 this implies that $TA_{\mathbb{L}}$ is Turing--enumerable. Therefore $\mathbb{L}$ is *complete* in the sense of Def. 2.5.

QED of PART (2) of the proof.

QED Thm. 2.1.

2.2. Languages well known in logic

THEOREM 2.2.

Let d be a similarity type. Then the first order language

$\mathbb{L}_d = <F_d, M_d, \models>$ of type d is stable.

Proof

Thm. 2.2. is a consequence of completeness of $\mathbb{L}_d$, *Thm.* 2.1., and the following Thm. 2.3.

QED Thm. 2.2.

THEOREM 2.3.

Let d be a similarity type. Then the first order language

$\mathbb{L}_d = <F_d, M_d, \models>$ of type d is well presented.

(Cf. Remarks 1.2, 1.3, and 1.5)

Proof

We shall give a proof for a finite similarity type d. The proof for the case $|d| = \omega$ is quite similar. We shall give two formulas: $\sigma_d(y)$, $\kappa_d(x,y) \epsilon F_\epsilon$ by which F_d and $\models$ can be presented. The presentation of M_d is left to the reader.

1.) Presentation of F_d:

Recall that $F_d \subseteq L(\omega)$ and $F_d \epsilon V$. It is well known that F_d is recursively enumerable, moreover, it is recursive, cf. Monk 76 Prop. 10.15 p. 168, and Remark 1.1. To satisfy condition 1.) of Def. 2.3., remains to give a formula $\sigma_d(y) \epsilon F_\epsilon$ which defines F_d, i.e., for which:

$$(V,\epsilon) \models (F_d = \{ y : \sigma_d(y) \}).$$

First we define:

$$\delta(J) \overset{df}{=} [J \subseteq L(\omega) \wedge J = \{ (\varphi \wedge \psi), (\neg\varphi), (\exists x_n \varphi), R(x_{i_1}, \ldots, x_{i_m}): (R,m) \epsilon d, \langle i_1, \ldots, i_m \rangle \epsilon {}^m\omega, n \epsilon \omega, \varphi, \psi \epsilon J \}].$$

$\delta(J) \epsilon F_\epsilon$, since $L(\omega)$ can be explicitely defined in ZFC set theory, i.e. $L(\omega)$ can be defined by a formula of F_ϵ.
Now let $\sigma_d(y) \overset{df}{=} (\forall J)[\delta(J) \rightarrow y \epsilon J]$. $\sigma_d(y) \epsilon F_\epsilon$. To see that $\sigma_d(y)$ defines F_d, one can easily show that:

$$ZFC \vdash (\exists ! J)\, \delta(J)$$

by using the facts that the elements of $L(\omega)$ are finite, $L(\omega)$ is transitive, moreover, every element of $L(\omega)$ is contained in a finite transitive set.

$ZFC \vdash (\exists ! J)\, \delta(J)$ and the definitions of $\delta(J)$ and $\sigma_d(y)$ imply that

$$(V,\epsilon) \models (F_d = \{ y : \sigma_d(y) \})$$

which was to be proved.

2.) Presentation of $\models$:

Suppose that M_d is presented by $\mu_d(x) \epsilon F_\epsilon$, i.e.

$$M_d = \{ \underset{\sim}{A} \epsilon V : (V,\epsilon) \models \mu_d [\underset{\sim}{A}] \} .$$

Recall the following notations:

If $\underset{\sim}{A} \epsilon M_d$ then its universe is denoted by A. If $(R,m) \epsilon d$ then the denotation of R in $\underset{\sim}{A}$ is denoted by $\underset{\sim}{A}_R$. Note that A and $\underset{\sim}{A}_R$ can be explicitely defined. Now we shall give a formula $\kappa_d(x,y)\epsilon F_\epsilon$ which defines $\models$, i.e. for which:

$$\models \;=\; \left\{ (\varphi, \underset{\sim}{A}) : (V,\epsilon) \models \kappa_d[\underset{\sim}{A},\varphi],\ \varphi\epsilon F_d, \underset{\sim}{A}\epsilon M_d \right\}.$$

Recall that for any *set* A the corresponding set $U\left\{ {}^nA : n < \omega \right\}$ is explicitely definable in set theory. So is ${}^nA$ for any $n\epsilon\omega$. Now we shall define a formula $\gamma(x,y)\epsilon F_\epsilon$.

Intuitively: $\gamma(\underset{\sim}{A},T)$ will express that T is the set of all "valuated formulas" true in $\underset{\sim}{A}$. I.e. the elements of T will be formula-valuation pairs (φ,a) instead of only formulas φ.

More precisely: For any model $\underset{\sim}{A}\epsilon M_d$ and any set T, we want $\gamma(\underset{\sim}{A},T)$ to hold iff $T = \left\{ (\varphi,\bar{a}) : \underset{\sim}{A} \models \varphi[\bar{a}] \text{ and } \bar{a}\,\epsilon\, {}^nA \text{ for some } n\epsilon\omega. \right\}$

The definition of the formula $\gamma(x,y)$:

$$\begin{aligned}\gamma(\underset{\sim}{A},T) \overset{df}{=} [T \subseteq (F_d \times U\{ {}^nA : n < \omega \}) \wedge \\ \wedge\, T = \{ &<(\psi\wedge\chi), \bar{a}>, <\neg\varphi,\bar{a}>, <(\exists x_r \varphi_1),\bar{a}>, \\ &<R(x_{i_1}, \ldots, x_{i_m}),\bar{a}>: \\ &\psi,\chi,\varphi,\varphi_1 \epsilon F_d,\ (\exists\ n < \omega)\ \bar{a}\,\epsilon\,{}^nA, \\ &(<\psi,\bar{a}>\epsilon T \ \wedge\ <\chi,\bar{a}>\epsilon T), \quad <\varphi,\bar{a}> \notin T, \\ &(\exists\ b\epsilon A)\ (\varphi_1(a_0, \ldots, a_{r-1}, b, a_{r+1}, \ldots, a_n))\epsilon T, \\ &[(\forall_j \leqslant m) i_j < n \ \wedge\ <a_{i_1}, \ldots, a_{i_m}>\epsilon \underset{\sim}{A}_R] \}].\end{aligned}$$

Now let:

$$\kappa_d(\underset{\sim}{A},\varphi) \overset{df}{=} \forall T[(\gamma(A,T) \rightarrow (\exists\ n\epsilon\omega)\ (\forall\bar{a}\,\epsilon\,{}^nA)<\gamma,\bar{a}>\epsilon T) \wedge \mu(\underset{\sim}{A})].$$

It is easy to show that

$$ZFC \vdash (\forall\underset{\sim}{A})[\mu(\underset{\sim}{A}) \rightarrow (\exists\, !T)\gamma(\underset{\sim}{A},T)].$$

This implies that $\models$ is defined by $\kappa_d(x,y)$.

QED Thm. 2.3.

Recall the definition of a constructive universe:

DEFINITION 2.7. (Chang-Keisler 73 p. 475, Devlin 73, Hinman 78 p. 215.)

For all ordinals α,

1.) If $\alpha = 0$ then $L(\alpha) \overset{df}{=} 0$.

2.) If $\alpha > 0$ is a limit ordinal then

$$L(\alpha) \overset{df}{=} \bigcup_{\beta < \alpha} L(\beta).$$

3.) If $\alpha = \beta + 1$, then

$$L(\alpha) \overset{df}{=} \left\{ X \subseteq L(\beta) : X \text{ is definable in } L(\beta) \right\} .$$

($L(\alpha)$ has been defined already for the special case $\alpha = \omega$, cf. Def. 1.3.)
A set x is said to be *constructible* iff there exists an α such that $x \epsilon L(\alpha)$.
The set of all constructible sets is called the *constructible universe* and is denoted by L.

$$L \overset{df}{=} \bigcup_{\alpha} L(\alpha).$$

Note that $L \subseteq V$ is a *class.*

REMARK 2.6.

Recall that (L,ϵ) is a "model of ZFC", i.e. $(L,\epsilon) \models ZFC$.
Also recall that $(L,\epsilon) \models CH$ (CH denotes the Continuum Hypothesis), cf. Chang – Keisler 73 p. 477.

On the basis of the proofs of the following Thm. 2.4. and Thm. 2.5. we would like to argue that higher order logics are incomplete *because* they are not stable. (Cf. also Motivations 2.2.)

By Thm. 2.1. and the incompleteness of higher order languages we know that the third and second order languages are not stable. In the following Thm.2.4. and Thm. 2.5. we shall prove these facts directly by constructing a third order formula ψ and a second order formula φ such that the set theoretic statements ' $\overset{3}{\models} \psi$ ' and ' $\overset{2}{\models} \varphi$ ' will be true in the real world (V,ϵ) but they will be false in some $\underset{\sim}{W} \epsilon Md(ZFC)$. In the proof of Thm. 2.4. we shall *explicitely show* that the meaning of ψ in $\underset{\sim}{A}$ does not depend on ψ and $\underset{\sim}{A}$ (but instead on something else). See the comments inside of the proof of Thm. 2.5. and Motivations 2.2.

THEOREM 2.4.

Let d be a similarity type. Then the third order language $<F_d^3, M_d, \overset{3}{\models}>$ is not stable.

Proof

Let the formula φ be defined as follows:

We shall use in φ the following symbols:

P is a third order variable symbol standing for unary relations of unary relations;

F is a third order variable symbol standing for unary functions defined on unary relations;

X and Y are second order variables standing for unary relations;

x and y are first order variables.

Now let

$$\varphi \overset{df}{=} \neg(\exists P \forall F \left\{\exists Y \forall X(P(X) \rightarrow F(X) \neq Y) \wedge \right.$$
$$\exists Y[P(Y) \wedge \forall X(\exists x \forall y(X(y) \leftrightarrow x = y) \rightarrow$$
$$\left. \rightarrow F(X) \neq Y)]\right\}).$$

Clearly $\varphi \epsilon F^3_d$ for every type d.

Basic Observation: The following observation reveals the intuitive reason responsible for the incompleteness of higher order languages.

Let $\underset{\sim}{W} \epsilon Md(ZFC)$. Let $\underset{\sim}{A} \epsilon W$ be such that $\underset{\sim}{A} \epsilon M_d$ and $|\underset{\sim}{A}| = \omega$ holds in $\underset{\sim}{W}$ as well as in (V,ϵ). Now clearly

$$\underset{\sim}{W} \models \text{'} \underset{\sim}{A} \overset{3}{\models} \psi \text{'} \quad \text{iff} \quad \underset{\sim}{W} \models CH$$

for $\varphi \epsilon F^3_d$ defined above.

Suppose that $\underset{\sim}{W} \models CH$ and $(V,\epsilon) \not\models CH$ or the other way round. (It is known that both are possible.) Then either $\underset{\sim}{W} \models \text{'} \underset{\sim}{A} \overset{3}{\models} \varphi \text{'}$ and $(V,\epsilon) \models \text{'} \underset{\sim}{A} \overset{3}{\models} \neg \varphi \text{'}$ or the other way round.

Now on the basis of the above observation we shall prove the theorem. The reader is invited to do this himself before reading what follows.

It is easy to write a formula $\psi \epsilon F_d^3$ by using φ such that $M_d \overset{3}{\models} \psi$ iff CH. I.e.:

$$(V,\epsilon) \models \text{'} \overset{3}{\models} \psi \text{'} \quad \text{iff} \quad (V,\epsilon) \models CH.$$

Intuitively: '$\underset{\sim}{A} \models \psi$' should say $(|A| = \omega \Rightarrow \underset{\sim}{A} \models \varphi)$.
In more detail:It is easy to construct a $\chi \epsilon F^2_d$ such that $\underset{\sim}{A} \models \chi$ iff $|A| = \omega$.
Now we define $\psi \overset{df}{=} (\chi \rightarrow \varphi)$.
Similarly, there exists a $\psi\text{'} \epsilon F^3_d$ such that

$$M_d \overset{3}{\models} \psi\text{'} \quad \text{iff} \quad \neg CH.$$

Namely, $\psi\text{'} \overset{df}{=} (\chi \rightarrow \neg\varphi)$.

Recall from §1. Remark 1.7. that throughout this work we tacitly assume that ZFC is consistent i.e. there exists a model $\underset{\sim}{W} \epsilon Md(ZFC)$.
Recall that (V,ϵ) is arbitrary but fixed. We distinguish two cases:

$$\text{either } (V,\epsilon) \models CH \quad \text{or} \quad (V,\epsilon) \models \neg CH.$$

1.) Suppose $(V,\epsilon) \models CH$.
By Cohen's result, cf. e.g. in Chang – Keisler 73 p. 44 line 11, there exists a model $\underset{\sim}{W} \epsilon Md(ZFC)$ such that $\underset{\sim}{W} \models \neg CH$. Then $(V,\epsilon) \models \text{'} \overset{3}{\models} \psi \text{'}$ and $W \not\models \text{'} \overset{3}{\models} \psi \text{'}$.

2.) Suppose $(V,\epsilon) \models \neg CH$.
Since ZFC is consistent, there exists $\underset{\sim}{W}\epsilon Md(ZFC)$. We can suppose that $\underset{\sim}{W} \models \neg CH$ i.e. $\underset{\sim}{W} \models \psi'$. Recall the notion of a constructive universe (Def. 2.7.). Let L^W denote the constructive universe *inside of* $\underset{\sim}{W}$. Denote "element of" in $\underset{\sim}{W}$ by E. Recall that $(L^W,E) \models ZFC$. By Thm. 7.4.4. of Chang – Keisler 73.p. 477, $(L^W,E) \models CH$. Observe that $L^W \epsilon V$ (though it *is* a class in W). Thus $(L^W,E)\epsilon M_\epsilon$. This completes the proof of the fact that ψ' is not stable.

QED Thm. 2.4.

THEOREM 2.5.

Let d be a similarity type. Then the second order language $<F_d^2, M_{d},\overset{2}{\models}>$ of type d is not stable.

Proof

Let ν be a number theoretic formula i.e. a first order formula in the language of $\underset{\sim}{\omega}$.

Let $P,T,$ and S be three second order variable symbols, P and T denoting binary functions and S denoting unary ones. I.e., P,T , and S are function variable symbols. Let z be a first order variable symbol.

Let $\nu' \overset{df}{=} \nu(+/P\ , \cdot/T, \text{succ}/S, 0/z)$ be the second order formula obtained from ν by replacing every occurence of $+$ by P , " $\cdot$ " by T, the sucessor by S, and the zero by z. I.e. ν' is a number theoretic formula in which addition is denoted by P (plus), multiplication is denoted by T(times) etc.

Denote PA' the Peano Axioms without the induction scheme. Observe that PA' is finite. Let $\pi\alpha'$ be the single formula obtained from PA' by conjunction. I.e. $\pi\alpha' = \Lambda\ (PA')$. Let

$$\pi\alpha = \pi\alpha'(+/P\ , \cdot/T\ , \text{succ}/S\ , 0/z).$$

I.e. $\pi\alpha(P,T,S,z)$ is a formula expressing the Peano Axioms without induction such that P denotes addition, T denotes multiplication, etc.
Note that $\pi\alpha$ begins by saying:

$$\forall x[S(x) \neq z] \ \Lambda\ \forall x \forall y[S(x) = S(y) \to x = y].$$

This implies that z, $S(z)$, $S(S(z))$ etc. are all distinct. Let U be a second order variable symbol denoting unary relations. Let $\pi\alpha^U$ be the relativised version of $\pi\alpha$ to U. (Cf. Chang – Keisler 73.). I.e. the first order quatifiers $\forall x$ and $\exists x$ are replaced by $(\forall x \epsilon U)$ and $(\exists\ x\epsilon U)$. Similarly ν^U denotes the relativised version of ν' to U.

Let φ denote the following formula:

$$(\forall U,P,T,S\forall z)[(U(z)\ \land\ \forall x[U(x)\rightarrow U(S(x))]\ \land\ \pi\alpha^U(P,T,S,z))\rightarrow \nu^U(P,T,S,z)].$$

φ is a second order formula, $\varphi\epsilon F_d^2$ moreover $\varphi\epsilon F_\emptyset^2$. I.e. φ is contained in every second order language. Let $\underset{\sim}{A}\epsilon M_d$ be arbitrary but infinite. (I.e. $|A|\geqslant\omega$.) Now clearly $\underset{\sim}{A}\models\varphi$ iff $\underset{\sim}{\omega}\models\nu$. More precisely, $\underset{\sim}{A}\models\varphi$ iff $(V,\epsilon)\models$'$\underset{\sim}{\omega}\models\nu$'.

Clearly for every number theoretic formula ν the corresponding $\varphi_\nu\epsilon F_0^2$ can be produced such that $\underset{\sim}{A}\models\varphi_\nu$ iff $\underset{\sim}{\omega}\models\nu$. This means that when speaking about an infinite model $\underset{\sim}{A}\epsilon M_d$ in its second order language F_d^2 then we can define full Number Theory inside of $\underset{\sim}{A}$ and *pretend* that we are speaking about $\underset{\sim}{A}$. But we are not speaking about $\underset{\sim}{A}$ at all instead we are speaking about $\underset{\sim}{\omega}$ which is completely independent of A. Such a language *cannot* be complete since its formulas $\varphi\epsilon F_d^2$ are *not* speaking about its models $\underset{\sim}{A}\epsilon M_d$ but they are speaking about something else. Validity of φ_ν in $\underset{\sim}{A}$ depends on the validity of ν in $\underset{\sim}{\omega}$. Thus $M_d\models\varphi_\nu$ iff $\underset{\sim}{\omega}\models\nu$.

Now let ν be a number theoretic closed formula such that $\underset{\sim}{\omega}\models\nu$ and $ZFC\not\models$'$\underset{\sim}{\omega}\models\nu$'. I.e. there exists a model $\underset{\sim}{W}\epsilon M_\epsilon\subseteq V$ of ZFC such that $\underset{\sim}{W}\not\models$'$\underset{\sim}{\omega}\models\nu$'. (Such a ν exists, e.g. let $e(\bar{y})$ be a Diophantine equation without solution and let ν be $\forall\bar{y}\neg e(\bar{y})$. Etc.) Let $\varphi=\varphi_\nu$. φ is a tautology ($\overset{2}{\models}\varphi$) iff $(V,\epsilon)\models$'$\underset{\sim}{\omega}\models\nu$'. We chose ν such that $\underset{\sim}{\omega}\models\nu$. Thus $\overset{2}{\models}\varphi$. I.e. $(V,\epsilon)\models$'$\overset{2}{\models}\varphi$'. But there exist $\underset{\sim}{W}\epsilon Md(ZFC)\subseteq V$ such that $\underset{\sim}{W}\not\models$'$\underset{\sim}{\omega}\models\nu$'. Therefore $\underset{\sim}{W}\not\models$'$M_d\overset{2}{\models}\varphi$' i.e. $W\not\models$'$\models\varphi$' i.e. φ is not a second order tautology inside of $\underset{\sim}{W}$. Therefore $<F_d^2, M_d, \overset{2}{\models}>$ is *not* stable (for every choice of the type d).

All this was straightforward since the formulas $\varphi_\nu\epsilon F_d^2$ do not speak about their models $\underset{\sim}{A}\epsilon M_d$ but they speak about something else namely $\underset{\sim}{\omega}$. φ_ν is valid in $\underset{\sim}{A}$ iff ν is valid in $\underset{\sim}{\omega}$. Now if we change the set theoretic world (V,ϵ) such that the validity of ν in ω changes then the validity of φ_ν in $\underset{\sim}{A}$ changes too! The validity of φ_ν in $\underset{\sim}{A}$ changes without changing $\underset{\sim}{A}$ or φ_ν but only changing the set theoretic world around them!

QED Thm. 2.5.

Denote by $\mathbb{H}^n_d = <F^n_d, M^H_d, \overset{n}{\models}_H>$ the Henkin-type n-th order language. Note that the syntax of $\mathbb{H}^n_d$ coincides with that of $\mathbb{L}^n_d$, the semantics, however, is different. (Cf. Henkin 50, Monk 76 p. 493.)

THEOREM 2.6.

For every similarity type d and $n<\omega$, the language $\mathbb{H}^n_d$ is stable.

THEOREM 2.7

(i) The Honking-type intensional model theory, as defined in Gallin 75, is stable.

(ii) The non Henkin-type intensional model theory, as defined in Gallin 75 and in papers of R. Montague, is not stable.

2.3. Languages for reasoning about programs

Descriptive Programming Languages in the sense of Gergely – Úry 78. p. 79:

In the followings let d be an arbitrary similarity type. Recall from Gergely – Szőts 78 Def.2, Gergely – Úry 78. p. 81. Def. 5.12, Andréka – Németi 78, Andréka – Németi – – Sain 79 p. 2, Manna 74 Chap. 4.1., or Andréka – Németi – Sain 78 § 1 the definition of the *Classical Language*

$\mathbb{D}_d^\omega \overset{df}{=} \langle P_d \times F_d, M_d, \overset{\omega}{\models} \rangle$ *of Program Verification* of type d. Recall that:

– P_d denotes the set of program schemes of type d (cf. Example 2.3.).

– If $\underset{\sim}{D} \epsilon M_d$ and $(p,\psi) \epsilon P_d \times F_d$ then $\underset{\sim}{D} \overset{\omega}{\models} (p,\psi)$ iff the program scheme p is partically correct w.r.t. ψ in the model $\underset{\sim}{D}$ for standard traces.

$\mathbb{D}_d^\omega$ is called Classical *Descriptive Programming Language* on p. 79. of Gergely – Úry 78.

THEOREM 2.8

The Classical Language of Program Verification

$\mathbb{D}_d^\omega = \langle P_d \times F_d, M_d, \overset{\omega}{\models} \rangle$ is *not stable.*

More precisely:

There exist a finite similarity type d, a model $\underset{\sim}{W} \epsilon Md(ZFC)$ of ZFC and a statement $(p,\psi) \epsilon P_d \times F_d$ such that:

$(V,\epsilon) \models$ '$\overset{\omega}{\models} (p,\psi)$' while $\underset{\sim}{W} \models$ '$\overset{\omega}{\not\models} (p,\psi)$' .

Proof

This is a special case of Thm. 4.2.

QED Thm. 2.8.

THEOREM 2.9.

1.) For every similarity type d and $Th \subseteq F_d$ satisfying the conditions of Thm. 4.1., the language

$^{\omega}\mathbb{D}_d^{Th} = <P_d \times F_d, Md(Th), \overset{\omega}{\models}>$ is not stable.

2.) Even reasonable restrictions* of the language

$^{\omega}\mathbb{D}_d^{Th} = <P_d \times F_d, Md(Th), \overset{\omega}{\models}>$ are not stable.

Proof.

Thm. 2.9. is a special case of Thm. 2. of Andréka – Németi – Sain 79. Cf. also Sain 79. §4 Thm. 4.2. and Thm. 2.9. there.

QED Thm. 2.9.

REMARK

The Classical Language of Program Verification $\mathbb{D}_d^{\omega}$ just happens to be an ω-logic in the sense of Barwise 77. p. 42.

Now recall the notion of a nonclassical *Descriptive Programming Language* from Def. 9.6. in Gergely – Úry 78. p. 125. and from the beginning of the following §3.

In §3 a (nonclassical) Descriptive Programming Language $\mathbb{T}\mathbb{D}_d$ of type d will be defined in detail. Here we use that definition, therefore the reader is kindly asked to have a look at §3. The parts of that Descriptive Programming Language $\mathbb{T}\mathbb{D}_d$ are denoted the following way:

$$\mathbb{T}\mathbb{D}_d \overset{df}{=} <[TF_d \cup (P_d \times F_d)],\ Md(Th), \models>$$

where:

– TF_d is the syntax of the classical three sorted first order language $\pi\mathbb{L}_d = <TF_d, TM_d, \models>$ of a certain similarity type td to be defined in §3;

– P_d and F_d are as above:

– $Th \subseteq TF_d$ is an *arbitrary* theory on the syntax TF_d. Thus $Md(Th) \subseteq TM_d$;

– $\models$ in the language $\mathbb{T}\mathbb{D}_d$ denotes an extension of the validity relation of the language $\mathbb{T}\mathbb{L}_d$: The definition of "$\mathfrak{M} \models (p, \psi)$" can also be found in §3.

* "Reasonable restrictions" are understood in the sence of Def. 3.1. and Gergely – Úry 78 Def. 4.4,L.4.5, and p. 97-100 Defs 7.5., 7.8, etc. . Further 4.4, 4.5. etc. of Gergely – Úry 78 explain *how and why* we use restrictions of languages i.e. completeness is investigated w.r.t. a reasonable subset E of the syntax.

" $\mathfrak{M} \models (p,\psi)$ " is pronaunced as: "the program p is partially correct w.r.t. ψ in the model $\mathfrak{M}$ ".

The letter "T" in $\mathbb{T}\mathbb{D}$, TF, TM and td serves to express that we are trying to treat *time* explicitely in this language. Cf. p. 118. of Gergely – Úry 78.

THEOREM 2.10.

a.) The Descriptive Programming Language

$$\mathbb{T}\mathbb{D}_d = <[TF_d \cup (P_d \times F_d)], TM_d, \models >$$

is *stable.*

b.) Moreover, for every recursively enumerable subset Th of TF_d, the language

$$\mathbb{T}\mathbb{D}_d^{Th} = <[TF_d \cup (P_d \times F_d)], \quad Md(Th) , \models >$$

is *stable.*

Proof

Thm. 2.1. and Thm. 3.2. imply both (a) and (b).

Remark: By using the proof of Thms 2.2., 2.3, and 3.2. one can construct a direct proof of the present Thm. 2.10.

QED Thm. 2.10.

THEOREM 2.11.

The Descriptive Programming Language $\mathbb{D}_{Ax}^{\overset{\models}{\xi}}$, as defined in Def. 9.6. of Gergely – Úry 78 p. 125. is stable, whenever Ax' is recursively enumerable.

Proof

Thm. 2.11. can be proved by using p. 133. of Gergely – Úry 78 or equivalently using Thm. 3.4. here. The latter says that $\mathbb{D}_{Ax'}^{\overset{\models}{\xi}}$ is strongly equivalent with the language $\pi\mathbb{D}_d^{Pax'}$, more precisely, with $<P_d \times F_d, Md(Pax'),\models >$. ($Pax'$ is defined above Thm. 3.4.) Therefore Thm. 2.4. (b) completes the proof. A detailed proof will be supplied later.

QED Thm. 2.11.

Expressive Programming Languages in the sense of Gergely – Úry 78:

In Gergely – Úry 78 the language $\mathbb{E}_d^{\omega} = <[F_d \cup (P_d \times F_d) \cup P_d], M_d, \overset{\omega}{\models} >$

is called standard *Expressive Programming Language.* Note that here for $p \epsilon P_d$, $\underset{\sim}{D} \epsilon M_d$, and $q \epsilon {}^{\omega}D$ the statement

$$\underset{\sim}{D} \overset{\omega}{\models} p[q] \text{ holds} \quad \text{iff}$$

q is a trace of p in $\underset{\sim}{D}$. I.e. $\overset{\omega}{\models}$ is essentially a 3-ary relation for P_d.

THEOREM 2.12.

(i) The standard Expressive Programming Language is *not stable,* and for any $Th \subseteq F_d$ satisfying the conditions of Thm. 4.1., the Expressive Programming Language

$$<[F_d \cup (P_d \times F_d) \cup P_d], Md(Th), \overset{\omega}{\models}>$$

is *not stable.*

(ii) For every Turing-enumerable $Th \subseteq TF_d$ the Expressive Programming Language

$$\mathbb{T}\mathbb{E}_d^{Th} = <[TF_d \cup (P_d \times F_d) \cup P_d], Md(Th), \models >$$

is *stable.*

Proof of (i) is in Andréka – Németi –Sain 79.

Proof of (ii) is in Andréka – Németi – Sain 79a, 79b, Sain 79.

QED Thm. 2.12.

2.4. Logic of Actions (Processlogic, Dynamic Logic etc.)

The above summarised results on Languages for Reasoning about Programs D_d, $\mathbb{T}\mathbb{D}_d$, $\mathbb{E}_d$, $\mathbb{T}\mathbb{E}_d$ together with the concepts of § 3, Andréka–Németi–Sain 78, 79, 79a, 79b, Gergely – Úry 78, Sain 79 may help us to continue the program outlined in Andréka – Gergely – Németi 74, Hayes 70, 71, McCarthy – Hayes 69, Gergely 73, 74, Pask 76, Ecsedi – Tóth 78 to develop a really applicable *Logic of Actions* for A.I. purposes and for the theory of Problem Solving Systems, cf. also Štepánková – Havel 76. A careful reading of Andréka – Gergely – Németi 74 and related works reveal that the requirements postulated there were *not* satisfied until now (in the literature) but they can be satisfied now on the base e.g. of Thm. 3.2. and related results and constructions. Cf. also Pratt 78, 79 and Parikh 78.

3. Nonstandard model theory for program schemes

On languages for reasoning about programs

Here we try to develop a natural semantic framework for programs and statements about programs. The need for such a framework was explained in the introduction of Gergely – Úry 78. (It was called Expressive Programming Language there.) In trying to understand the "Programming Situation", its languages, their meanings etc., the first question is how an interpretation or model of a program or program scheme p should look like. The classical approach (Manna 74, Ianov 60) says that an interpretation or model of a program scheme is a relational structure $\underset{\sim}{D}$ consisting of all the possible *data values.* The program p containes variables, say, "y". The more ambitious version of existing programming theory calls y an "*identifier*". Anyway, the classical approach says that y denotes elements of $\underset{\sim}{D}$ just as variables in classical first order logic do. Now we argue that the identifier y does *not* denote elements of $\underset{\sim}{D}$ but rather y denotes some kinds of "locations" or "addresses" which may *contain* different data values (i.e. elements of $\underset{\sim}{D}$) at different points z of time T. (For detailed accounts of this idea of locations and their contents in programming see Andréka – –Németi–Sain 78,79b.) Thus there is a set I of locations, a set T of time points, and a function ext: $I \times T \to D$ which tells for every location $s \epsilon I$ and time point $z \epsilon T$ what the content $ext(s,z) \epsilon D$ of location s is at time point z. Of course, this content is a data value i.e. it is an element of D. Time has a structure too ("later than" etc.) and data values have structure too, thus we have structures $\underset{\sim}{T}$ and $\underset{\sim}{D}$ over the sets T and D of time points and possible data values respectively. Therefore a model or interpretation for programs p is a four-tuple $\mathfrak{M} = \langle \underset{\sim}{T}, \underset{\sim}{D}, I, ext \rangle$ where $\underset{\sim}{T}$ and $\underset{\sim}{D}$ are the time structure and data structure resp., I is the set of location and $ext : I \times T \to D$ is the "content of . . . at time . . . " function.

Consider e.g. the statement "$y = y + 1$" which frequently occurs in programs. If y denotes elements of $\underset{\sim}{D}$ then the interpretation of "$y = y + 1$" is not very natural. However, if y denotes a location $s \epsilon I$ then "$y = y + 1$" means that the content of the location s changes during time T. Now, it is easy to imagine that when reasoning about really complex control structures of new programming languages, this difference in expliciteness and naturalness might have practical importance.

Of course when specifying the semantics of a programming language P we may have ideas about how an interpretation $\mathfrak{M}$ of P may look like and how it may not look. These ideas may be expressed in the form of axioms about $\mathfrak{M}$. E.g. we may postulate that $\underset{\sim}{T}$ of $\mathfrak{M}$ has to satisfy the Peano Axioms of arithmetic.

These axioms are easy to express since a closer investigation of $\mathfrak{M}$ as defined above, reveals that it is a model of classical 3-sorted logic (the sorts being T, D, and I). Thus the axioms can be formed in calssical 3-sorted logic in a convenient manner to express all our ideas or postulates about the semantics of the programming language P under consideration. A detailed exposition of this framework for reasoning about programs can be found in this paragraph. Cf. also Gergely – Úry 78 Part III.

The semantics of programming $\mathbb{T}\mathbb{D}$ given in §3 here is a result of a careful analysis of the Programming Situation in which efforts were made to make the mathematical model $\mathbb{T}\mathbb{D}$ of the Programming Situation to be as "faithful" and "explicite" or "natural" as possible. We have the impression that in formation of the classical semantics $\mathbb{D}^\omega$ of programming (Manna 74, Ianov 60 etc.) this point of explicitness or faithfulness was neglected. The results of § 2 and § 3 seem to reveal certain practical consequences of this difference in explicitness between $\mathbb{D}^\omega$ and $\mathbb{T}\mathbb{D}$. It turned out in §2 that certain mathematical language concepts including $\mathbb{D}^\omega$ are *anomalous*. It does not mean, however, that the original language would be anomalous at all. Only the mathematical model of the original language is such and a careful study of the situation might lead to a new healthier mathematical model. (I.e. real understanding of the situation in question might help to avoid the anomalies.) In the preceding §2 we tried to provide some tools for such "careful investigations". (Of course , the real methodology for such investigations is nonexistent yet, the present §2 is only a trial in this direction.)

DEFINITIONS

Now to every classical (one-sorted) similarity type d (see Def. 1.4.) we define an associated *3-sorted similarity type td*. About many-sorted logic and its model theory see Monk 76 p. 483 and Barwise 77 p. 42.

As before, d is an arbitrary similarity type. Let t denote the similarity type of *Peano Arithmetic* and let t be disjoint from d. The type td is defined as follows: There are *3 sorts of td*: $\bar{t},\bar{d},\bar{i}$ called "time", "data", and "intensions" respectively.

The *operation symbols of td* are the following:

the operation symbols of d,
the operation symbols of t, and
an additional operation symbol "ext".

The *sorts (or arities) of the operation symbols of td*: The operation symbols of t go from sort $\bar{t}$ to sort $\bar{t}$. The operation symbols of $\bar{d}$ go from sort $\bar{d}$ to sort $\bar{d}$. The operation symbol "ext" goes from sort $(\bar{i},\bar{t})$ to sort $\bar{d}$. I.e., "ext" has two arguments, the first is of sort $\bar{i}$, the second is of sort $\bar{t}$ and the result or value of "ext" is of sort $\bar{d}$.
Now the definition of the 3-sorted type td is completed.

$$\mathbb{T}\mathbb{L} = \langle TF_d, TM_d, \models \rangle$$

denotes the 3-sorted language *of type td,* see* Monk 76. p. 483.
In more detail:

(i) TM_d is the class of all *models of type td,* see Monk 76. Def. 29. 27. I.e. a model $\mathfrak{M} \epsilon TM_d$ has

1. *three universes* throughout denoted by T,D and I of sorts $\bar{t}, \bar{d},$ and $\bar{i}$ respectively,

2. operations "$T^n \to T$" originating from the type t, operations "$D^n \to D$" originating from d, and an operation ext: $I \times T \to D$.

Roughly speaking, we could say that $\mathfrak{M}$ consist of structures $\underset{\sim}{T} \epsilon M_t, \underset{\sim}{D} \epsilon M_d$, and an additional operation ext $: I \times T \to D$. Therefore we** shall use the sloppy notation:

$\mathfrak{M} \overset{df}{=} <\underset{\sim}{T}, \underset{\sim}{D}, I, ext>$ for elements of TM_d.

(ii) TF_d is the set of first order (3-sorted) formulas of type td. Roughly speaking, we can say that F_t and F_d are contained in TF_d, and there are additional terms of the form "$ext(y,\tau)$" where τ is a term of type t and y is a *variable of sort* $\bar{i}$. Further, "$ext(y,\tau)$" is defined to be a term of sort $\bar{d}$.

(iii) $\models \subseteq (TM_d \times TF_d)$ is the usual, see Monk 76. p. 484.

Now we the *meaning of program schemes* $p \epsilon P_d$ in the 3-sorted models $\mathfrak{M} \epsilon TM_d$. Let $p \epsilon P_d$ be a fixed program scheme. Let $y_1, \dots, y_m$ be all the variables accurring in p. Let $\mathfrak{M} \epsilon TM_d$ be fixed. Recall that I is the universe of sort $\bar{i}$ of $\mathfrak{M}$.

A *trace of* p in $\mathfrak{M}$ is a sequence $<s_0, \dots, s_m> \epsilon ^{(m+1)}I$ of elements of I satifying ($*$) below. (I.e. a trace of p in $\mathfrak{M}$ is a sequence of $\bar{i}$-sorted elements of $\mathfrak{M}$). To formulate ($*$), observe that if $s \epsilon I$ then "$ext(s, -)$" is a function $<ext(s,z) : z \epsilon T>$ from T into D. We shall use y_0 as "*the controll variable*" of p. I.e. $ext(s_0,z)$ is considered to be the "value of the controll or execution" at time point z. Thus "$ext(s_0,z)$" is supposed to be a "*label*" in the program scheme p.

* By a little abuse of notations we could write: $T\mathbb{L}_d \overset{df}{=} L_{td}, TF_d \overset{df}{=} F_{td}$, and $TM_d \overset{df}{=} M_{td}$.

** This abuse of notation is taken from Barwise 77. p. 42.

* The sequence $<ext(s_0, -), \dots, ext(s_m, -)>$ of functions should be a *history of an execution* of p in $\underset{\sim}{D}$ along the "time axis" $\underset{\sim}{T}$.

The only difference from the classical definition (see the definition of an ω-trace in Example 2.3., in Manna 74, Andréka – Németi – Sain 78, Def. 2 in Gergely – Szőts 78, Gergely – – Úry 78) of a trace of p in $\underset{\sim}{D}$ is that now the "time axis" of execution is not necessarily $<\omega, s, +, \cdot, 0, 1>$ but, instead, it is $\underset{\sim}{T}$.

Condition (*) above can be made precise by replacing ω with $\underset{\sim}{T}$ in the classical definition, see Andréka – Németi 78, Andréka – Németi – Sain 78, Gergely – Úry 78.

The trace $<s_0, \dots, s_m>$ of p in $\mathfrak{M}$ *terminates* if $ext(s_0, z)$ is the label of the HALT statement, for some $z \epsilon T$. If the trace $<s_0, \dots, s_m>$ terminates at time $z \epsilon T$ then its *output* is $<ext(s_1, z), \dots, ext(s_m, z)>$.
Now we define for $\psi \epsilon F_d$:
$\mathfrak{M} \models (p, \psi)$ to hold iff for every *terminating* trace of p in $\mathfrak{M}$ the output satisfies ψ in $\underset{\sim}{D}$. Cf. Def. 8 of Gergely – Szőts 78, Andréka – Németi 78, Andréka – Németi – Sain 78.

For an arbitrary theory $Th \subseteq TF_d$ the consequence relation $Th \models (p, \psi)$ is defined in the usual way.

THEOREM 3.1.

Let $Th \subseteq TF_d$ be an arbitrary recursively enumerable set of formulas. Then the set

$$\{ (p,\psi) \epsilon P_d \times F_d : Th \models (p, \psi) \}$$

is recursively enumerable.

Proof

This Thm. 3.1. is a consequence of Thm. 3.2.

QED Thm. 3.1.

The following theorem solves Problems 1, 2, and 3 of Andréka – Németi – Sain 78 and generalises Thm. 11.4. of Gergely – Úry 78 (cf. the restriction $PA \subseteq Ax'$ at the beginning of §9 there and Thm. 1. of Andréka – Németi – Sain 78. We define the *Descriptive Programming Language* $\mathbb{T}\mathbb{D}_d$ as follows:

$$\pi \mathbb{D}_d = <[TF_d \cup (P_d \times F_d)], TM_d, \models>.$$

THEOREM 3.2.

Denote $TS_d \overset{df}{=} (P_d \times F_d) \cup TF_d$. The Descriptive Programming Language

$\mathbb{T}\mathbb{D}_d = \langle TS_d, TM_d, \models \rangle$ is *strongly complete,* i.e. for every recursively enumerable set $Th \subseteq S_d$, the set $\{ \rho \epsilon S_d : Th \models \rho \}$ of its consequences is recursively enumerable. Specially: $\{ (p,\psi) \epsilon P_d \times F_d : Th \models (p,\psi) \}$ is recursively enumerable. Further, the language $\mathbb{T}\mathbb{D}_d$ is *compact.* Moreover, in the proof of the present theorem we gave a *strongly complete calculus* (inference system) for the Descriptive Programming Language $\mathbb{T}\mathbb{D}_d$.

Proof

The idea of the proof is: to reduce or translate the language $\langle TS_d, TM_d, \models \rangle$ to the *complete* language $\langle TF_d, TM_d \models \rangle$ by a computable function $\Theta : TS_d \to TF_d$ such that: for every $\rho \epsilon S_d$ and every $\mathfrak{M} \epsilon TM_d$:

$$\mathfrak{M} \models \rho \quad \text{iff} \quad \mathfrak{M} \models \Theta(\rho) \quad \text{and} \quad \Theta(\rho) = \rho \quad \text{if} \quad \rho \epsilon TF_d.$$

The proof goes similarity to Dahn 73, 78, and Andréka – Gergely – Németi 77 p. 13 § 2.1., see def. of "L_1 is recursively reducible to L_2" there. A detailed proof can be found in Andréka – Németi – Sain 79b and in Sain 79.

QED Thm. 3.2.

The execute programs in arbitrary elements of TM_d might look counter-intuitive. However, we may require the theory $Th \subseteq TF_d$ to contain a certain fixed set $Ax \subseteq TF_d$ of axioms expressing all the intuitive requirements about *time* and about *processes* "happening in time". After having done so, there is nothing wrong with executing programs in models $\mathfrak{M} \epsilon TM_d$ of Th since $\mathfrak{M} \models Ax$ and Ax does contain all our intuitive ideas about time, processes etc. (Basically the same was done by Henkin when he defined the new semantics for higher order logic and, at least in Computer Science, everybody is satisfied with his system, see e.g. Robinson 69, 69a, Pietrzykowski 73.)

To illustrate this here, we define a set $Ax \subseteq TF_d$ of axioms of the above kind.

DEFINITION OF THE THEORY Ax:

Roughly speaking, Ax is nothing but the Peano Axioms for the sort $\bar{t}$. However, in our present syntax TF_d, variables of sort $\bar{t}$ may occure in formulas which contain symbols of sort $\bar{d}$ and $\bar{i}$ as well. Well, the *induction axioms* must be stated for these formulas "of mixed sort", too. Namely:
Let $\varphi(z) \epsilon TF_d$ such that z is a variable of sort $\bar{t}$. Then we define φ^* to be the induction formula:

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

Now the induction axioms are:

$$IA \overset{df}{=} \{ \varphi^* : \varphi(z) \epsilon TF_d \text{ and } z \text{ is of sort } \bar{t} \}.$$

Clearly $IA \subseteq TF_d$, since if $\varphi(z)\epsilon TF_d$ and z is a variable of sort t then $\varphi(0)$, $\varphi(z+1)\epsilon \mathrm{TF}_d$ because 0 and $z+1$ are terms of sort t.

Let PA denote the Peano axioms for the sort $\bar{t}$ (recall that t is the similarity type of arithmetic).
Now we define:

$$Ax \overset{df}{=} PA \cup IA.$$

Denote by Axe the set Ax together with the axiom of extensionality. I.e.

$$Axe \overset{df}{=} Ax \cup \left\{ \forall y_1 y_2 [\forall x(ext(y_1,x) = ext(y_2,x)) \rightarrow y_1 = y_2] \right\}.$$

THEOREM 3.3. (Uniquness of traces)

Let $p\epsilon P_d$ and $\mathfrak{M} \models Axe$ ($\mathfrak{M}\, \epsilon TM_d$) be arbitrary. Then for a fixed input $q\epsilon\, {}^{\omega}D$, p has *at most one trace* in $\mathfrak{M}$ with input q.

Proof: can be found in Andréka – Németi – Sain 79a, 79b and in Sain 79.

QED Thm. 3.3.

References

Andréka, H. 78: A characterisation of Floyd provable programs. To appear in Proc. Coll. Logic in Programming, Salgótarján, 1978, Collog. Math. Soc. J. Bolyai – North – Holland.

Andréka, H. – Dahn, B. – Németi, I. 76: On a proof of Shelah. Bull. de l'Academie Polonaise des Sciences, Vol. XXIV, No.1, 1976. pp.1-7.

Andréka, H. – Gergely, T. – Németi, I. 72: Problem oriented hierarchy of Languages and hierarchy oriented Logic. KFKI-72-46, Budapest, 1972 (in Hungarian).

Andréka, H. – Gergely, T. – Németi, I. 73: On some problems of n-th order languages. (in Hungarian). Matematikai Lapok, Vol. XXIV, No. 1-2, 1973. pp. 63-94.

Andréka, H. Gergely, T. – Németi, I. 73a: Purely algebraic construction of first order logics. Publ. Central Res. Inst.for Physics H.A.S., KFKI-73-71, 1973.

Andréka, H. – Gergely, T. – Németi, I. 74: On the theory of problem solving systems. (in Hungarian: Magasszintű mesterséges intelligencia tudásreprezentációjának eszközei). Preprint, 1974.

Andréka, H. – Gergely, T. – Németi, I. 74a: On Universal Algebraic Construction of Logics. Publ. Central Res. Inst. for Physics H.A.S., KFKI-74-41, 1974.

Andréka, H. – Gergely, T. – Németi, I. 75: Easily Comprehensible Mathematical Logic and its Model Theory. Publ. Central Res. Inst. for Physics H.A.S., KFKI-75-24, 1975.

Andréka, H. – Gergely, T. – Németi, I. 77: On Universal Algebraic Construction of Logics. Studia Logica, XXXVI, 1-2. Wroclaw, 1977, pp. 9-47.

Andréka, H. – Gergely, T. – Németi, I. 78: An approach to Abstract Model Theory. J. Symb. Logic, 1978.

Andréka, H. – Németi, I. 75: A simple, purely algebraic proof of the completeness of some first order logics. Algebra Universalis Vol. 5, 1975, p. 8-15.

Andréka, H. – Németi, I. 76: Generalisation of variety and quasivariety concept to partial algebras through category theory. Preprint No. 5/1976, Math. Inst. H.A.S. 1976. To appear in Dissertationes Mathematicae (Rozprawy).

Andréka, H. – Németi, I. 77: On Universal Algebraic Logic. Preprint, Math. Inst. H.A.S., 1977.

Andréka, H. – Németi, I. 78: Completeness of Floyd Logic. Bull. Sec. Logic Vol. 7 No.3, pp. 115-127. Polish Acad. Sci., 1978.

Andréka, H. – Németi, I. 79: On systems of varieties definable by schemes of equations. Preprint, Math. Inst. H.A.S., 30/1978. To appear in Algebra Universalis.

Andréka, H. – Németi, I. 79a: Solutions of Problems 2.3 and 2.11. of Henkin–Monk–Tarski 71. Math. Inst. H.A.S. Preprint, 1979.

Andréka, H. – Németi, I. 79b: $\mathrm{UfUpDc}_\alpha = \mathrm{UfUpLf}_\alpha$ in Cylindric Algebra Theory. Math. Inst. H.A.S. Preprint, 1979.

Andréka, H. – Németi, I. – Sain, I. 78: Classical manysorted model theory to turn negative results on program schemes to positive. Preprint, 1978.

Andréka, H. – Németi, I. – Sain, I. 79: Completeness Problems in Verification of Programs and Program Schemes. Proc. Coll. Math. Foundations of Comp. Sci. 1979. Olomouc, Springer-Verlag, Lecture Notes in Comp. Sci. 74, pp. 208-218.

Andréka,H.. – Németi, I. – Sain, I. 79a: Henkin-type semantics for programs and program schemes to turn negative results to positive. In L. Budach (ed): Fundamentals of Computation Theory FCT'79, Akademie–Verlag Berlin 1979, pp. 18-24.

Andréka, H. – Németi, I. – Sain, I. 79b: A Complete Logic for Reasoning about Programs via Nonstandard Model Theory. Math. Inst. H.A.S. Preprint, 1979.

Banachowski, L. – Kreczmar, A. – Mirkowska, G. – Rasiowa, H. – Salwicki, A. 77: An introduction to algorithmic logic. Banach Center Publications Vol. 2, 1977.

Barwise, J. 77: Handbook of Mathematical Logic. North-Holland, 1977.

Chang, C.C. – Keisler, H.J. 73: Model Theory. North-Holland, 1973.

Courcelle, B. – Guessarian, I. 77: On some classes of interpretations. IRIA, Repport de Racherce No. 253, 1977.

Dahn, B.I. 73: Generalized Kripke-Models. Bul. de l'Aadémie Polonaise des Sciences, Série des Sciences math., astr. et phys., Vol. XXI, No. 12, 1973, pp 1073-1077.

Dahn, B.I. 78: Contributions to the Model Theory for Nonclassical Logics. Zeitschrift für Math. Logic und Grundlagen d. Math. 20 (1074), pp. 473-479.

Dahn, B.I. 79: Pradikatenkalküle der ersten Stufe für Kripkemodelle und Metrische Strukturen. Dr. sc. nat. Dissertation, Berlin, 1979.

Davis, M. 73: Hilbert's tenth problem is unsolvable. Amer. Math. Monthly 80, 1973, pp. 233-269.

Devlin, K.J. 73: Ascepts of Constructibility. Lecture Notes in Math. 354, Springer-Verlag, 1973.

Ecsedi – Tóth, P. 78: Intensional Logic of Actions. CL and CL, XII, 1978. pp. 31-44.

Frege, G. 92: Uber Sinn und Bedeutung. Zeitschrift für Philosophie und philosophische Kritik 100: 25-50, 1892.

Gallin, D. 75: Intensional and higher order modal logic, with applications to Montague semantics. North-Holland, Elsevier, 1975.

Gergely, T. 73: Mathematical Foundations of General System Theory. Proc. Conf. on System Theory 1973. Sopron, J. Neumann Comp. Sci. Society. Also: A mesterséges intelligencia kutatás logikai eszközeiről. Rendszerelméleti Konferencia'73, Problémamegoldási szekció, Sopron, 1973. 34-42.

Gergely, T. 74: Lectures held at General Systems Research LTD, London, 1974.

Gergely, T. 77: Role of a General Language Concept in the Construction of an Abstract Cognition Theory. In J. Rose and C. Bilcin (Eds.) Modern Trends in Cybernetics and Systems 2 (Springer-Verlag, N.Y. 1977) pp. 131-152.

Gergely, T. 78: May the theory of programming be first order? In the collection of abstracts of the Conference "Logic in Programming" Salgótarján, 1978.

Gergely, T 79: On the unity between Cybernetics and General System Theory. Kibernetes, 1979, Vol. 8, pp. 45-49.

Gergely, T. – Németi, I. 71: Az általános rendszerelmélet formalizálásának és alkalmazásának logikai alapjai. Rendszer Kutatás, Közgazdasági és Jogi Könyvkiadó, 1973.

Gergely, T. – Szabolcsi, A. 79: On the backfround of model theoretic semantics. To appear in CL and CL, XIII, 1979.

Gergely, T. – Szőts, M. 78: On the incompleteness of proving partial correctness. Acta Cybernetica, Tom. 4, Fasc. 1, Szeged, 1978.

Gergely, T. – Ury, L. 78: Mathematical Programming Theories. SZÁMKI Preprint.

Gergely, T. – Vershinin, K.P. 78: Model theoretical investigations of theorem proving methods. Notre Dame J. of Formal Logic, Vol. XIX, No. 4, 1978, pp. 523-542.

Goguen, J.A. – Thatcher, J.W. – Wagner, E.G. – Wright, J.B. 77: Initial Algebra Semantics and continuous Algebras. J. Association Comp. Mach., Vol. 24, No.1, 1977, pp. 68-95.

Hayes, P.J. 70: Robotologic. Machine Intelligence 5, Edinburgh Univ. Press, 1970, pp. 534-554.

Hayes, P.J. 71: A logic of actions. Machine Intelligence 6, Edinburgh Univ. Press, 1971, pp. 495-520.

Henkin, L. 50: Completeness in the theory of types. J. of Symbolic Logic, Vol.15 (1950), pp. 81-91.

Hinman, P.G. 78: Recursion–Theoretic Hierarchies. Springer-Verlag, 1973.

Ianov, Y.I. 60: The Logical Schemes of Algorithms. In Problems of Cybernetics, Vol. 1, pp. 82-140, Pergamon Press, N.Y. (English translation).

Kutschera, F. von 76: Einführung in die intensionale Semantik. De Gruyter Studienbuch, 1976.

Lindström, P. 74: On characterizing elemntary logic. Logical Theory and Semantics Analysis pp. 129-146, D. Reidel Publ. Co., Dortrecht-Holland, 1974.

Makowsky,J.A.73: On the axiomatic theory of model theoretic languages. Logc Semester'73, Warsaw.

Makowsky, J.A.75: Model Theory and Applications. Centro Internazionale Matematico Estivo, Edizioni Cremonese, Roma, 1975, pp. 122-150.

Manna, Z. 74: Mathematical Theory of Computation. McGraw Hill, 1974.

McCarthy, J. – Hayes, P.J. 69: Some Philosophical Problems from the Standpoint of Articial Intelligence. Machine Intelligence 4, Edinburgh Univ. Press, 1969, pp. 463-502.

Monk, J.D. 76: Mathematical Logic. Springer-Verlag, 1976,

Montague, R. 73: The proper treatment of quantification in ordinary English. Approaches to natural languages, Dortrecht, 1973, pp. 221-242.

Németi, I. 76: From hereditary classes to varieties in abstract model theory and partial algebra. Beiträge zur Algebra und Geometrie 7 (1978), pp. 69-78.

Németi, I. – Sain, I. 77: Cone injectivity and some Birkhoff-type theorems in categories. To appear in Proc. Coll. Univ. Alg. Esztergom 1977, Colloq. Math. Soc. J. Bolyai – North-- Holland

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, Collog. Math. Soc. J. Bolyai – North-Holland.

Parikh, R. 78: A completeness result for a propositional dynamic logic. M.I.T. /LCS/TM-106, 1978.

Pask, G. 76: Conversation Theory: Applications in Education and Epistemology. Elsevier, N.Y., 1976.

Pratt, V.R. 78: A Practical Decision Method for Propositional Dynamic Logic. Proc. 10th ACM Symp. on Theory of Computing, 1978, pp. 326-337.

Pratt, V.R. 79: Dynamic Algebras. MIT Preprint, 1979. To appear in the volume of invited papers of 6th Intl. Conf. Logic.

Pietrzykowski, T. 73: A complete mechanisation of second-order type theory. JACM 1973, Vol. 20, No.3.

Robinson, J.A. 69: Mechanizing Higher-Order Logic. Machine Intelligence 4, 1969.

Robinson, J.A. 69a: A note on mechanizing higher order logic. Machnie Intelligence 5, 1969.

Sacks, G.E. 72: Saturated Model Theory. Math. Lecture Notes Series, W.A. Benjamin, Inc., Massachusetts, 1972.

Sain, I. 78: Model Theoretic and Universal Algebraic methods in Sementics. (in Hungarian). Preprint, 1978.

Sain, I. 79: Abstract Model Theory and Completeness of Languages. Part II of SZKI Preprint, March 1979.

Sánthâné, T.E. – Szőts, M. 79: Report on Conference on Logic in Programming, Salgótarján, 1978. Preprint, 1979. Appeared in Hungarian translation: Számitástechnika, Vol. X, No. 2-3, 1979.

Szabolcsi, A. 78: A természetes nyelv szemantikájának modellelméleti kezelése. Dissertation, 1978. Budapest.

Štepánková, O. – Havel, I.M. 76: A logical theory of robot problem solving. Artificial Intelligence 7, 1976, pp. 129-161.

Takeuti, G. – Zaring, W.M. 71: Introduction to Axiomatic Set Theory. Springer–Verlag, 1971.

Computational Linguistics and Computer Languages Vol. XIII. 1979.

APPLICATIONS OF UNIVERSAL ALGEBRA, MODEL THEORY, AND CATEGORIES IN COMPUTER SCIENCE (Survey and Bibliography)

A.Hajnal and I. Németi
Mathematical Institute Hungarian Academy of Sciences
Budapest, Hungary

In the last eight years universal algebra and model theory together with its categorical versions received an ever increasing application in the field of computer science, more precisely in the study of semantics of programming languages and in the methodology of proving properties of programs. Real progress in these applications of universal algebra was started by Burstall – Landin [69], Thatcher [67], [70], Montague [70a], Thatcher – Wright [68]. The concepts and tools of universal algebra turned out to be flexible enough to be adapted to the new and rather complex situations arising from (at least a kind of) computer programming. In 1972 Robin Milner proved the correctness of a compiler by using results of universal algebra, cf. Milner – Weyrauch [72a]. By today applications of universal algebra and its categorical version went so far that e.g. in the recent volume:

– "Fundamentals of Computer Science" (/LNCS 56/, Springer Verlag, 1977) section B is nearly entirely based on universal algebra and categories; in 1975 Springer Verlag published a volume with title "Category Theory Applied to Computation and Control" /LNCS 25/; at the recent conference on "semantics of programming" in Sophia Antipolis (France) universal algebra and lattice theory were generally accepted tools, etc.

Some of the main directions of research are surveyed briefly below. A deeper survey is Goguen [79].

1. The theory of tree automata is almost entirely based on universal algebra, cf. Thatcher [67], [70], [73], Thatcher – Wright [68], Alagić [75], Baker [73], Brainerd [67], [68], [69], Doner [70], Elienberg – Wright [67], Engelfriet [75], Ferenci [76], Gécseg [77], Gécseg – Horváth [76], Gécseg – Tóth [77], Karpinski [73], Levi – Joshi [73], Magidor – Moran [69], Mezei – Wright [67], Shepard [69], Steinby [77], Yeh [71].

2. Universal algebric theory of the denotational semantics of context free languages was started by Goguen et al [75a] (p. 75), Montague [70a], Andréka – Gergely – Németi [74], [77] independently, and was generalised to context sensitive languages by Kaphengst – Reichel [77]. Syntax is treated as a free algebra (if there are several syntactical categories then heterogeneous) and interpretations are special homomorphisms from this free algebra into a prespecified class of algebras. The algebraic properties of this class of algebras determine the semantic properties of the language, cf. Goguen [74b], Goguen et al [74b], [75a], Kaphengst – Reichel [71], [77], Montague [70], Andréka – Gergely – Németi [77], Wagner et al [76],

Letitschewski [68], Glushkov – Zeitlin – Letitschewski [75]. Németi – Sain [78] is a detailed introduction. For a related work cf. Bloom [76], Rattray – Rus [77].

3. The theory of quasivarieties and universal Horn classes serve as foundations for new non-procedural programming languages, cf. Andréka – Németi [76a], [76b], Battoni – Melloni [73], Colmerauer [75], van Emden [74], [75a], [75b], [77], van Emden – Kowalski [74], Kowalski [73], [74], [76], Roussel [75], E. Tóth [76], Warren [77], Bruynooghe [76]. Kaphengst – Reichel [71] (equoids), Reichel [78] treat quasivarieties applied to computer science.

4. Universal algebra together with model theory are used in the "most denotational" semantics and also in investigating such widely used program proving methods as Floyd's one, cf. Manna [69], [74], Manna – McCarthy [70], Montague [70b], Rasiowa [73], [77], Scala [71], Burstall [69], van Emden – Kowalski [74], Floyd [67], Hayes [71], [77], Janseen -van Emde Boas [77], Andréka – Németi [77], [78], Gergely – Szőts [78], Abrahamson [78], Bowen [78], DeMillo [75], Harel [77], [78], Pratt [77], Harel – Pratt [77], Harel – Meyer – Pratt [77], Fisher – Ladner [77], Andréka – Németi – Sain [78], [79], Gergely – Úry [78], [78a], [79], [79a]. Terminology: algebraic systems in the sense of Malcev, i.e. models in the sense of Tarski, Chang – Keisler etc. are called interpretations in computer science, especially in the theory of interpreted program schemes, cf. Brand [76] p. 205, Manna [74] p. 244, Courcelle – Guessarian [77] p. 2, Scala [71], Abramson [78], etc.

5. Categorical versions of universal algebra are used in the fixed point approaches to semantics (originating from Dana Scott), cf. Lehmann [76], Lehmann – Smyth [77], Adamek – Koubek [77a], Advanced seminar on semantics [77], Arbib [76], [77], Day [75], Goguen et al [76b], Meseguer [77], Obtułowicz [77], Smyth [76a], [76b], Tiuryn [77a], [77b], [78], [79], [79a], Wagner et al [76], Wand [75a], Plotkin – Smyth [77]. About model theoretic treatment of fixed point semantics see Section 3 of Gergely – Úry [78].

6. Categorical versions of universal algebra (triplets, algebraic theories etc.) are the foundations of the so-called "unified automation theory" which deals with systems, fuzzy automata, deterministic automata etc. in a unified frame (and solves important engineering problems e.g. realisation problems). Chap. IV of Manes [76] is a good (but not too fresh) survey. Other references are Adamek [74], [75], [76], [76b], Adamek – Koubek [77b], [78], Adamek – Trnková [77a], [77b], Alagić [75], Anderson et al [76], Arbib – Manes [72], [74a] [75a] – [75g], [77], Bainbridge [72], Beckman [70], Budach [75], Budach – Hoehnke [75], Ehrig [72] – [74], Ehrig et al [75] – [77], Elgot [71], [75] [77], [78], Elgot et al [76], Ginali [76], Give'on [70], Goguen [71] – [77], Goguen et al [73] – [77], Hoehnke [77], Koubek – Reiterman [75], [78], Manes [76], Rine [71], [74], Trnková [74] – [77], Trnková – Adamek [77], [78], Trnková et al [75], [79], Wand [72], [75b], Wiweger [73], Buslenko – Simonov [76], [77], Skornjakov [74], Vainstein – Osetinskij [77].

7. The universal algebraic theory of abstract data types originates from the recognition that: a specification of abstract data types is nothing but a definition of a class of (heterogeneous) algebras. Further an implementation of this specification is correct if it is a free algebra of this class. The problem of the existence of free algebras belongs to universal algebra (and is not completely solved). Methodology of proving correctness is obtained from the universal algebraic methodology of proving freeness of algebras in a class. Since quasivarieties always have free algebras, they are a central tool in data type theory cf. Thatcer – Wagner – Wright [76], [78]. Why and how data types are universal algebras and why existing universal algebra theory is relevant to their study is explained in more detail in Goguen – Thatcher – Wagner [76], also cf. Zilles [75], Guttag [75], [76], Goguen et al [75d], Andréka – Németi [75]. Kaphengst – Reichel developed a refined notion of free algebra while working on the fundamentals of a universal algebra of partial algebras. Many authors consider the latter to be more adequate to data type theory and computer science in general, cf. e.g. Kaphengst – Reichel [71], [77], Reichel [78b], Hoehnke [77], [78], Andréka – Németi [76c]. The literature of universal algebraic theory of abstract data types is too broad to be covered here; but some further randomly chosen examples are: Goguen [75a], [77], [77a], [79], Zilles [74], Plotkin – Smyth [77], Burstall – Goguen [77], [79], Rosenberg [76], Guttag – Horowitz – Musser [76]. A deeper survey is Goguen [79].

8. The free magma approach to semantics of programming languages originating from France is also based on universal algebra.

Here F is a similarity type in the universal algebraic sense and a class of "F – magma" – s is a class of universal algebras of type F cf. Courcelle – Guessarian [77] p. 8-9, Guessarian [76]p. 192. Def. 2. [78], Berry – Courcelle [76] p. 170, Nivat [75], Berry – Lévy [77] p. 15. Trees play a central role where trees are "terms" or "polynomial symbols" of universal algebra, i.e. the elements of the free algebra of a similarity class are called trees cf. Berry [77] p. 15. Infinite trees are kinds of infinitely long polynomial symbols, cf. e.g. Tiuryn [77a]. The free magma approach is strongly related to the works listed in 2. and 7. cf. e.g. Goguen et al [75a] and Tiuryn [77b], [79a]. A whole branch of semantics associates infinite trees i.e. infinite terms to programs (and associates finite trees i.e. terms to program specifications cf. Burstall – Goguen [77], [78]), Nivat [72], [75], [78], Tiuryn [77], [79], Goguen et al [75a], Berry – Courcelle [76] p. 171 etc. The free complete F -magma is the free algebra (in the universal algebraic sense) of a class of complete partially ordered universal algebras (called sometimes interpretations), cf. e.g. Arnold [77], Bloom [76a], Arnold – Nivat [77], Berra – Courcelle [76], Berry – Lévy [77], Courcelle – Guessarian [77], Courcelle – Nivat [76], Nivat [72], [75], [78].

9. We note that beyond the scope of this survey there are many other interesting applications of universal algebra and categories, e.g. an application in General Systems Theory is outlined in Goguen – Varela [78].

The following bibliography is not intended to be complete. In order to keep size manageable we only take samples from each main "direction" known to us. Throughout, LNCS abbreviates volume of the series "Lecture Notes in Computer Science" published by Springer Verlag (Berlin – Heidelberg – New York).

Abramson, F. G. [78]: Interpolation Theorems for Program Schemata. *Information and Control 36,* 1978, 217-233.

Abstract Software Specifications (Advanced course on . . .) Jan. 1979. Dept. Comp. Sci. Techn. Univ. Denmark, Copenhagen.

Adámek, J. [74]: Free algebras and automata realizations in the language of categories. *Commun. Math. Univ. Carolinae 15,* 1974, 589-602.

Adámek, J. [75]: Automata and categories: Finiteness contra minimality, in: *Mathematical Fundations of Computer Science,* LNCS 32, 1975, 160-166.

Adámek, J. [76a]: Cogeneration of algebras in regular categories. *Bull. Aust. Math. Soc. 15,* 1976, 355-370.

Adámek, J. [76b]: Cogeneration and minimal realization. *Commun. Math. Univ. Carolinae 17,* 3, 1976, 609-614.

Adámek, J. [77]: Realization theory for automata in categories. *J. Pure Appl. Algebra 9,* 1977, 281-296.

Adámek, J. – Koubek, V. [77a]: Remarks on Fixed Points of Functors, in: *Fundamentals of Computer Science,* ed. by G. Goos, J. Hartmanis, LNCS 56, Springer Verl., 1977, 199-206.

Adámek, J. – Koubek, V. [77b]: Functorial Algebras and Automata. *Kybernetika 13,* 4, 1977, 245-260.

Adámek, J. – Koubek, V. [app]: Algebras and Automata over a Functor. To appear in *Kybernetika.*

Adámek, J. – Trnková, V. [77a]: On languages accepted by machines in the category of sets, in: *Mathematical Foundations of Computer Science,* LNCS 53, 1977, 523-531.

Adámek, J. – Trnková, V. [77b]: Recognizable and Regular Languages in a Category, in: *Fundamentals on Computer Science,* LNCS 56, 1977, 206-212.

Adámek, J. – Trinková, V. [78]: Varietors and machines. A survey paper. *COINS Tech. Rep. Univ of Massachusetts.* (Submitted to *Algebra Universalis.*)

Advanced Seminar on Semantics. Sophia–Antipolis, France, Sept. 1977. (Notes xeroxed on the spot).

Aiello, L. – Attardi, G. – Prini, G. [77]: Towards a more declarative programming style. In *Working conf. on Formal description of Programming Concepts,* IFIP, 1977, 5.1- 5.16.

Alagić, S [73]: Algebraic aspects of ALGOL 68. *COINS Technical Report 73B-5,* Computer and Information Science, Univ. of Mass. Nov. 1973.

Alagić, S. [75]: Categorical theory of tree processing, in: *Category theory applied to Computation and Control* LNCS 25, 1975, 65-73.

Alagić, S. – Arbib, M.A. [78]: The design of well-structured and correct programs, Springer Verl. 1978.

Algebraische Methoden u. ihre Anwendungen in der Automaten Theorie. Proc. Semin., April, 1976. in Weissig. (Available from N. J. Lehman, Techn. Univ. Dresden.)

Anderson, B. D. O. – Arbib, M. A. – Manes, E. G. [76]: Foundations of System Theory: Finitary and Infinitary Conditions, ed. Beckmann – Künzi, Lecture Notes in Economics and Mathematical Sciences 115, Springer Verl. 1976.

Anderson, E. R. – Belz, F. C. – Blum, E. [76]: SEMINOL 73. *Acta Informatica 6,* 109-131.

Andréka, H. – van Emden, M. H. – Németi, I. [79]: Greates fixed point semantics for the programming language PROLOG, (manuscript.)

Andréka, H. – Gergely, T. – Németi, I. [74], [77]: On Universal Algebraic Construction of Logics. *Studia Logica XXXVI,* 1-2, 1977, 9-47.

Andréka, H. – Németi, I. [75]: On applications of universal algebra in computer science. *Számológép Kiskönyvtár 75*, 1975, 145-153. (In Hungarian)

Andréka, H. – Németi, I. [76a]: The generalized completeness of Horn predicate-logic as a programming language. *D.A.I. Research Report No. 21,* Department of Artificial Intelligence, Univ. of Edingburgh, March. 1976. Also in: *Acta Cybernetica 4,* 1, 1978, 3-10.

Andréka, H. – Németi, I. [76b]: On the adequateness of predicate-logic programming. *AISB European Newsletter,* 23, July 1976, 30-32.

Andréka, H. – Németi, I. [76c]: Generalisation of variety and quasivarietyconcept to partial algebras through category theory. Preprint of *Math. Inst. Hung. Acad. Sci.* 5, March 1976, 1-85.

Anréka, H. – Németi, I. [77]: On completeness of systems for program proving. *Math. Inst. Hung. Acad. Sci.* 1977. 1-110 (In Hungarian)

Andréka, H. – Németi, I. [78]: A characterisation of Floyd provable programs. Preprint of *Math. Inst. Hung. Acad. Sci.* Febr. 1978.

Anréka, H. – Németi, I. [78a]: Completeness of Floyd Logic. (Abstract). *Bulletin of the Section of logic,* Wroclaw, 7, 3, 115-121.

Andréka, H. – Németi, I. – Sain, I. [78]: To verify programs within or without logic? To appear in: *MFCS'79,* Olomuc.

Andréka, H. – Németi, I. – Sain, I. [79]: Classical many-sorted model theory to turn negative results on program schemes to positive. Preprint, Budapest, 1979.

Apt, K. R. [78]: Equivalence of operational and denotational semantics for a fragment of PASCAL. In: *Formal Description of Programming Concepts,* ed. E.J. Neuhold, 1978, 139-163.

Apt, K. R. [78a]: A sound and complete Hoare-like system for a fragment of PASCAL. Preprint of the *Mathematical Centre,IW 96/78,* Amsterdam, 1978, 1-59.

Arbib, M. A. [76]: Categorical Notes on Scott's theory of computation. *Proc. of a meeting at Dortmund University,* Bericht Nr. 37, Dortmund, 1976. 5-8.

Arbib, M. A. [77]: Free Dynamics and Algebraic Semantics, in: *Fundamentals on Computer Scienc,* LNCS 56, 1977, 212-228.

Arbib, M. A. – Manes, E. G. [72]: Decomposable machines and simple recursion. *Computer and Information Sciences Technical Report 72B-2,* Univ. of Mass., 1972.

Arbib, M. A. – Manes, E. G. [74a]: Foundations of systems theory: Decomposable machines. *Automatica 10,* 1974, 285-302.

Arbib, M. A. – Manes, E. G. [74b]: Machines in a category: An expository introduction. *SIAM Review 16,* 1974 , 163-192.

Arbib, M. A. – Manes, E. G. [75a]: Adjoint machines, state behaviour machines and duality. *J. Pure Appl. Algebra 6,* 1975, 313-344.

Arbib, M. A. – Manes, E. G. [75b]: Arrows, Structures and Functors, Acad. Press, 1975.

Arbib, M. A. – Manes, E. G. [75c]: Fuzzy machines in a category. *Bull. Austral Math. Soc. 13,* 1975, 169-210.

Arbib, M. A. – Manes, E. G. [75d]: Basic concepts of category theory applicable to computation and control, in: *Category Theory Applied to Computation and Control,* LNCS 25, 1975, 1-35.

Arbib, M. A. – Manes, E. G. [75e]: A cetegorist's view of automata and systems, ibid. 51-65.

Arbib, M. A. – Manes, E. G. [75f]: Fuzzy morphisms in automata theory, ibid. 80-87.

Arbib. M. A. – Manes, E. G. [75g]: Time-Varying Systems, ibid. 87-92.

Arbib, M. A. – Manes, E. G. [77]: Intertwined recursion, tree manipulations and linear systems. *COINS Techn. Rep. 77-13,* Univ. of Mass. 1977.

Arbib, M. A. – Manes, E. G. [78]: Partially additive categories and computer semantics. *COINS Technical Report 78-12,* Unif. of Mass. Amherst, June 1978, 1-39.

Arnold, A. [77]: Systemes d'équations dans le magmoide. Ensembles rationnels et algebriques d'arbres. These d'État, Lille, 1977.

Arnold, A. – Dauchet, M. [76]: Bi-transductions de forets. in: *Automata, Languages and Programming. Third International Colloquium at the Univ. of Edinburgh,* ed. S. Michaelson, R. Milner. Edinburgh Univ. Press, 1976, 74-87.

Arnold, A. – Dauchet, M. [77]: Theorie des magmoides. *Publ. du Laboratorie de Calcul,* Lille, 1977.

Arnold, A. – Nivat, M. [77]: Non-Deterministic Recursive Program Scemes, in: *Fundamentals on Computer Science,* LNCS 56, 1977, 12-22.

Arnold, A. – Nivat, M. [78]: Algebraic Sementics of Non-Deterministic Recursive Program Schemes. *Lab. Informatique Theoretique et Programmation Rep. No 78-4,* 1978.

Arnold, A. – Nivat, M. [78a]: The metric space of infinite trees: Algebraic and topological properties. *IRIA Res. Rep. No 323.*

Bachus, I. [78]: Can programming be liberated from the von Neumann Style? A functional style and its algebra of programs. *CACM 21,* 8, 1978, 613-641.

Bainbridge, E. S. [72]: A Unified Minimal Realization Theory with Duality for Machines in a Hyperdoctrine. Diss., Univ. of Michigan, 1972.

Baker, B. S. [73]: Tree transductions and families of tree languages. *Hardvard Univ., Center for Research in Computing Technology, TR9-73,* 1973.

de Bakker, J. W. [76]: Semantics and termination of nondeterministic recursive programs, in: *Automata Languages and Programming. Third International Colloquium at the Univ. of Edinburgh,* ed. S. Michaelson, R. Milner, Edinburgh Univ. Press, 1976, 435-478.

de Bakker, J. W. – Meertens, L. G. L. [72]: On the completeness of the inductive assertion method. *Mathematical Centre Report IW 12,* 1972, Amsterdam.

Bertol, W. [74]: Algebraic complexity of machines. *Bull. Acad. Pol. Sci.* 8. 1974, 851-856.

Battani, G. – Meloni, H. [73]: Interpreteur du langage de programmation PROLOG. Groupe d'Intelligence Artificielle, Marseille, Luminy, 1973.

Beckman, F. S. [70]: Categorical notions and duality in automata theory. *IBM Thomas J. Watson Research Center Research Report RC 2977,* Yorktown Heights, July 1970.

Bekić, H. [71]: Definable operations in general algebra and the theory of automata and flowchatrts. *Report of IBM Laboratory,* Vienna, 1971.

Benson, D. B. [70]: Syntax and Semantics: A categorical view. *Information and Control 17,* 1970, 145-160.

Benson, D. B. [74]: An Abstract machine theory for formal language parsers. *Acta Informatica 3,* 1974, 187-202.

Benson, D. B. [75]: The Basic algebraic structures in categories of derivations. *Information and Control 28,* 1975, 1-29.

Berry, G. – Courcelle, B. [76]: Program equivalence and canonical forms in stable discrete interpretations, in: *Automata Languages and Programming. Third International Colloquium at the Univ. of Edinburgh,* ed. S. Michaelson, R. Milner, Edinburgh Univ. Press, 1976. 168-189.

Berry, G. – Lévy, J. [77]: Minimal and optimal computations of recursive programs. *IRIA preprint No 233,* 1974, 1-54. To appear in: *JACM.*

Bertoni, A. – Mauri, G. – Torelli, M. [77]: An algebraic approach to problem solution and problem semantics. In: MFCS 1977, LNCS 53, 253.

Birkhoff, G. – Lippson, J. D. [74]: Universal Algebra and Automata, in: *AMS Symp. Pure Math. 25,* 1974, 41-51.

Blikle, A. [72]: Equational Languages. *Information and Control* 1972, 134-147.

Blikle, A. [73]: An algebraic approach to programs and their computations, in: *Proc. Symp. and Summer School on the Mathematical Foundations of Computer Science,* High Tatras, Czechoslovakia, 1973.

Blikle, A. [77]: An analysis of program by algebraic means. MFCS, Warsaw, 1974, Banach Center Publications Vol. 2, 1977.

Blikle, A. [78]: Specified programming. To appear in: *Proc. of the Int. Conf. on Mathematical Studies of Information Processing,* Kyoto, Aug. 24-26, 1978.

Bloom, S. L. [76]: Projective and Inductive Generation of Abstract Logics. *Studia Logica XXXV, 3,* 1976. 249-255.

Bloom, S. L. [76a]: Varieties of ordered algebras. *J. Comput. and Syst. Sci. 13,* 1976, 200-212.

Bloom, S. L. [prep.]: Algebraic and graph theoretic characterizations of structured flowchart schemes. (In preparation.)

Bloom, S. L. – Elgot, C. C. [74]: The existence and construction of free iterative theories. *IBM Thomas J. Watson Center,* Research Report *RC 4937,* 1974; also in: *J. Comput. and Syst. Sci. 12,* 3, 1976, 305-318.

Bloom, S. L. – Elgot, C. C. – Wright, J. B. [78]: Solution of the iteration equation and extensions of the scalar iteration operation. *IBM Res. Rep. RC 7029,* 1978. 1-37.

Bloom, S. L. – Ginali, S. – Rutledge, J.D. [77]: Scalar and vector iteration. *J. Comput. and Syst. Sci. 14,* 1977, 251-256.

Blum, E. K. [69]: Towards a Theory of Semantics and Compilers for Programming Languages. *J. Comput. and Syst. Sci. 3,* 1969, 248-274.

Blum, E. K. [71]: Semantics of Programming Languages. IFIP W. G. 3.2, Bulletin, April 1971.

Blum, E. K. [73]: Formalization of Semantics of Programming Languages, in: *Theorie des Algorithmes, des Languages et da la Programmation,* Seminaries IRIA, 1973.

Blum, E. K. – Estes D. R. [77]: A generalisation of the homomorphism concept. *Algebra Universalis 7,* 1977, 143-161.

Bobrow, L. S. – Arbib, M. A. [74]: Discrete Mathematics, in *Applied Algebra for Computer and Information Science,* Washington, Hemisphere Publ., 1974.

Boudol, G. [75]: Langages polyadiques algébriques. Théorie des schémas de programme: sémantique de l'appel par valeur. These 3^0 cycle, Paris, 1975.

Brainerd, W. S. [67]: Tree generating systems and tree automata. Ph. Diss., Purdue Univ., 1967.

Brainerd, W. S. [68]: The minimization of tree automata. *Information and Control 13,* 1968, 484-491.

Brainerd, W. S. [69]: Tree generation regular systems. *Information and Control 14,* 1969, 217-231.

Brand, D. [76]: Proving Programs Incorrect. In: *Automata, Languages, and Programming,* Edinburgh Univ. Press, 1976. 201-227.

Brown, F. M. [78]: A semantic theory of logic programming. *D. A. I. Res. Rep. No 51,* Univ. of Edinburgh, 1978.

Brynooghe, M. [76]: An interpreter for predicate logic programs. P. 1. *Report CW 10, Applied Mathematics and Programming Division, Katholieke Univ,* Leuven, Oct. 1976.

Budach, L. [75]: Automata in additive categories with applications to stochastic linear automata, in: *Category Theory applied to Computation and Control,* LNCS 25, 1975, 119-136.

Budach, L. – Hoehnke, H. J. [75]: Automaten und Funktionen, Berlin, 1975.

Burstall, R. M. [69]: Formal description of program structure and semantics in first order logic. in: *Machine Intelligence 5,* ed. B. Meltzer, D. Michie, Edinburgh Univ. Press, 1969, 79-98.

Burstall, R. M. [72a]: An algebraic description of programs with assertions, verification and simulation, in: *Proc. ACM Conf. on Proving Assertions about Programs,* Las Cruces, New Mexico, 1972, 7-14.

Burstall, R. M. [72b]: Some techniques for proving correctness of programs which alter data structures, in: *Machine Intelligence 7,* ed. B. Meltzer, D. Michie, Edinburgh Univ. Press, 1972, 23-50.

Burstall, R. M. [77]: Design considerations for a functional programming language. *Proc. of Infotech State of the Art Conf.,* Copenhagen, 1977.

Burstall, R. M. – Goguen, J. A. [77]: Putting theories together to make specifications, in: *Proc. of Fifth Int. Joint Conf. on Artifical Intelligence,* MIT, Cambridge, Mass., 1977, 1045-1058.

Burstall, R. M. – Landin P. J. [69]: Programs and their proofs: An algebraic approach, in: *Machine Intelligence 4,* ed. M. Meltzer, D. Michie, Edinburgh Univ. Press, 1969. 17-43.

Burstall, R. M. – Thatcher, J. W. [75]: The algebraic theory of recursive program schemes, in: *Proc. AAAS Symp. on Category Theory Applied to Computation and Control,* LNCS 25, 1975, 126-131.

Buslenko, N. P. – Simonov, V. M. [76]: On the categorical representation of dynamic systems. *Programirovanie* 5, 1976. (In Russian).

Buslenko, N. P. [77]: On the categorical desription of complex systems. *Programirovanie* 1, 1977, 82-94. (In Russian).

Categorical and Algebraic Methods in Computer Science and System Theory. 2nd Workshop, Dortmund, 1978. (In preparation)

Category Theory Applied to Computation and Control. LNCS 25, ed. E. G. Manes, Springer Verl., 1975.

Chirica, L. [78]: Proof of correctness of a compiler by algebraic semantics. Theses. *UCLA Report, UCLA-EMG- 7697.*

Chirica, L. – Martin, D. F. [76]: An algebraic formulation of Knuthian semantics, in: *Proc. 17th Annual IEEE Symp. Found. Comp. Sci.*, Houston, Texas, 1976. 127-136.

Clark, K. – Sickel, S. [77]: Predicate logic: a calculus for the formal derivation of programs. *Proc. IJCAI-77.* Conference, 1977.

Colloquium on Mathematical Logic in Programming. Sept. 10-15 1978, Hungary. Proceedings of Colloquia Mathematica Societatis János Bolyai. (in preparation)

Colmerauer, A. [75]: Les grammaires de metamorphose. Group d'Intelligence Artificielle, Marseille-Luminy, Nov. 1975.

Constable, R. L. [77]: On the theory of Programming logics, in: *Proc. ACM STOC 9,* 1977, 269-285.

Cook, S. A. [78]: Soundness and completeness of an axiom system for program verification. *SIAM J. on Computing,* 7, 1. 1978. 70-91.

Courcelle, B. [78]: Equational theories and equivalences of programs. IRIA, 1978.

Courcelle, B. – Guessarian, I. [77]: On some classes of interpretations. *Rapport de Recherce No 253, IRIA,* 1977.

Courcelle, B. – Nivat, M. [76]: Algebraic families of interpretations. *17th Symp. Found. Comp. Sci.,* Houston 1976. Also in: *IRIA Rapp. Rech. No 189,* 1976.

Courcelle, B. – Nivat, M. [78]: The algebraic semantics of recursive program schemes.*Proc. 7th MFCS,* Zakopane, LNCS 64, Springer Verl. 1978.

Courcelle, B. – Raoult, J. C. [78]: Completions of Ordered Magmas. IRIA, 1978.

Cousot, P. – Cousot R. [77]: Static determination of dynamic properties. In: *Working Conf. on Formal Description of Progr. Concepts.* IFIP, 1977, 12.1-12.40.

Damm, W. [77]: Higher type program schemes and their tree languages, in: *Proc. 3rd GI Conf. Theoretical Computer Science,* LNCS 48, Springer Verl. 1977, 51-72.

Damm, W. – Fehr, E. [78]: On the power of self application and higher type recursion, in: *Proc. 5th ICALP,* LNCS 62, Springer Verl. 1978, 177-191.

Damm, W. – Fehr, E. – Indermark, K [78]: Higher type recursion and self application as control structures, in: *Formal Description of Progr. Concepts, Proc. IFIP,* 1977.

Davidson, D. – Hartman, G. [72]: Semantics of natural language. Rediel Publ., 1972, 769.

Davis, R. [69]: Universal coalgebra and categories of transition systems. *Math. Syst. Theory 4,* 1969, 91-95.

Day, A. [75]: Filter monads, continuous lattices and closure systems. *Can. J. Math. XXVII.* 1975. 50-59.

DeMillo, R. [75]: Non-definability of certain semantic properties of programs. *Notre Dame J. Formal Logic 16,* 1975, 583-590.

van Dijk, T. A. [77]: Action description. Paper for the *Coll. le discours descriptif,* Urbino, Italy, 1977.

Doner, J. E. [70]: Tree acceptors and some of their applications. *J. Comput. and Syst. Sci. 4,* 1970, 406-451.

Dubinsky, A. [75]: Computation on arbitrary algebras. Queen LNCS 37, 1975. 319-341.

Eder, G. [76]: A PROLOG-like interpreter for non-Horn clauses. Dept. of A. I., Univ. of Edinburgh, Sept. 1976.

Ehrich, H. D. [77]: Algebraic semantics of Type Definitions and Structured Variables, in: *Fundamentals on Computer Science,* LNCS 56, 1977, 84-98.

Ehrich, H. D. [77a]: Algebraische Spezifikation von Datenstrukturen. *Proc. Workshop "Graphentheoretische Konzepte in der Informatik"*, ed. J. Mühlbacher, Hansen--Verl. München, 1977.

Ehrich, H. D. [78]: Extensions and implementations of abstract data type specifications. Abt. Informatik, Univ. Dortmund, Germany.

Ehrich, H. D. – Lohberger, V. G. [78]: Parametric Specification of Abstract Data Types, Parameter Substitution, and Graph Replacements. In: *Proc. Workshop "Graphentheoretische Konzepte in der Informatik"*, Hanser-Verl. München, 1978.

Ehrigh, H. [72]: Automata theory in monoidal categories, in: *Proc. Tagung über Kategorien,* Mathematisches Forschungsinstitut, Oberwolfach, 1972, 12-15.

Ehrigh, H. [73]: Axiomatic theory of systems and systematics. *Report 73-05, Technische Universität Berlin,* 1973.

Ehrigh, H. [74]: Universal Theory of Automata: A categorical approach. Teubner Studienbücher Informatik, 1974.

Ehrig, H. – Kreowski, H. J. [75]: Power and initial automata in pseudoclosed categories, in: *Category Theory Applied to Computation and Control,* LNCS 35, 1975, 144-150.

Ehrig, H. – Kreowski, H. J. – Padawitz, P. [77]: Some remarks concerning correct specification and implementation of abstract data types. *Technische Universität Berlin, Bericht Nr. 77-13,* 1977.

Ehrig, H. – Kreowski, H. J. – Padawitz, P. [78]: Stepwise Specification and Implementation of Abstract Data Types. *Proc. 5th Int. Coll. Automata Lang. and., Progr.*, Udine, Italy.

Ehring, H. – Kühnel, W. – Pfenderl, M. [75]: Diagram characterization of recursion, in: *Category Theory Applied to Computation and Control,* LNCS 35, 1975, 137-144.

Eilenberg, S. – Wright, J. [67]: Automata in General Algebras. *Information and Control 11,* 1967, 452-470.

Elgot, C. C. [71]: Algebraic theories and program schemes, in: *Symposium on Sementics of Algorithmic Languages,* ed. E. Engeler, Springer Verl., 1971, 71-88.

Elgot, C. C. [75]: Monadic computations and iterative algebraic theories. *IBM Research Report RC 4564,* Oct. 1973. Also in: *Proc. of Logic Colloquium '73* North-Holland, 1975, 230.

Elgot, C. C. [77]: Some "geometrical" categories associated with flowchart schemes. *IBM Res. Rep. RC-6534,* 1977, 1-8.

Elgot, C. C. [78]: A representative strong equivalence class for accessible flowchart schemes. Prepared for *Int. Conf. on Math. Studies of Information Processing,* 1978, Kyoto, Japan.

Elgot, C. C. – Bloom, S. L. – Tindell, R. [76]: On the algebraic structure of rooted trees. *IBM RC 6320,* 1976. Also in: *I. Comput. and Syst. Sci. 16*, 3, 362-396.

van Emde Boas, P. [78]: The connection between model logic and algorithmic logics. *Dept. of Math. Univ. of Amsterdam, R 78-02,* May 1978, 1-15.

van Emde Boas, P. – Janssen, T. M. V. [78]: Montague Grammar and Programming Languages. *ITW/VPW,* University of Amsterdam, 1978.

van Emden, M.H. [74]: First-order predicate logic as a high-level program language. *MIP-R-106, Scool of Artifical Intelligence, Univ. of Edinburgh,* May, 1974.

van Emden, M. H. [75a]: Programming with resolution logic. *Research Report CS-75-30, Dept. of Computer Science, Univ. of Waterloo,* Ontario, Canada, Nov. 1975.

van Emden, M. H. [75b]: A representation of flowgraphs in first order predicate logic. *Dept. of Computer Science, Univ. of Waterloo,* Waterloo, Ontario, Canada Febr. 1975.

van Emden, M. H. [77]: Relational equations, grammars and programs. *Research Report CS-77-17, Dept. of Computer Science, Univ. of Waterloo,* Ontario, Canada June 1977.

van Emden, M. H. – Kowalski, R. A. [74]: The semantics of predicate logic as a programming language. *Memo No 73, School of Artificial Intelligence, Univ. of Edinburgh,* Feb. 1974. Also in: *MIP-R-103, Dept. of Machine Intelligence, Univ. of Edinburgh.*

Engeler, E. [74]: Algorithmic Logic. *Foundations of Computer Science,* ed. J. W. de Bakker, Mathematical Centre Tract, MCT 63, Amsterdam, 1974.

Engelfriet, J. [75]: Bottom-up and top-down tree transformations: a comparison. *Mathematical Systems Theory 9,* 1975, 198-231.

Engelfriet, J. [75a]: Tree automata and tree grammars. *DAIMI Report FN-10, Univ. of Aarhus,* Aarhus, Denmark, 1975.

Engelfriet, J. – Schmidt, E. M. [78]: IO and OI. *J. Comput. and Syst. Sci. 15,* 3, and *16,* 1, 67-99.

Ferenci, F. [76]: A new representation of context-free languages by tree automata. *Foundations of Control Engineering 1,* 1976, 217-222.

Floyd, R. W. [67]: Assigning meaning to programs. *Proc. Symposium on Applied Mathematics. XIX.* ed. J. T. Schwartz, Am. Math. Soc., Providence, R. I., 1967.

Fischer, M. J. – Ladner, R. E. [77]: Propositional modal logic of programs. *TR-77-02-02, Univ. of Washington,* Febr. 1977.

Fried, J. [77]:Simulations of Pawlak machines and fuzzy morphisms of partial algebras. *Commun. Math. Univ. Carolinae 18,* 2, 1977, 343-351.

Fundamentals on Computer Science. Ed. G. Goos, J. Hartmanis, LNCS 56, Springer Verl., 1977.

Gallin, D. [75]: Intensional and Higher-Order Modal Logic, North-Holland – American Elsevier, N. Y. 1975.

Gécseg, F. [77]: Universal Algebras and Tree Automata, in: *Fundamentals on Computer Science,* LNCS 56, 1977, 98-113.

Gécseg, F. – Horváth, G. [76]: On representation of tree and context-free languages by tree automata. *Foundations of Control Engineering,* 1976, 161-168.

Gécseg, F. – Steinby, M. [77]: Algebraic Theory of Tree-Automata. JATE, Szeged, Hungary (In Hungarian).

Gécseg, F. – Tóth, E. P. [77]: Algebra and logic in theoretical computer scence. In: MFCS 1977, SLNCS 53. 78-93.

Gergely, T. – Szabolcsi, A, [78]: How to do things with model theory. 1978. (manuscript).

Gergely, T. – Szőts, M. [78]: On the incompleteness of proving partial correctness. *Acta Cybernetica 4,* 1, 1978, 45-57.

Gergely, T. – Ury, L. [78]: Mathematical Theories of Programming 1978, (manuscript).

Gergely, T. – Ury, L. [78a]: On the notions of completeness in programming theory. *Proceedings of the Colloquium on Logic in Programming,* 1978. *(to be appeared).*

Gergely, T. – Ury, L. [79]: On the non-deterministic programming (submitted to *MFCS' 79*).

Gergely, T. – Ury, L. [79]: Operational semantics of parallelism, (submitted to the *Symposium on semantics of concurrent computation).*

Gergely, T. – Vershinin, K. P. [75]: Proof by analoghy. *KFKI-75-79,* Budapest, Hungary, 1975. (in Russian)

Gergely, T. – Vershinin, K. P. [78]: Model theoretical investigation of theorem proving methods. *Notre Dame J. Form. Log. 19,* 1978, 523-542.

Ginali, S. [76]: Iterative algebraic theories, infinite trees and program schemata. Diss., Dept. of Mathematics University of Chicago, 1976.

Give'on, Y. – Arbib, M. A. [68]: Algebra Automata II.: The Categorical Framework for Dynamic Analysis. *Information and Control 12,* 1968. 346-370.

Give'on, Y. [70]: A categorical review of algebra, automata and system theories. *Symposia Mathematica 4,* Instituto Nazionale di Alta Matematica, Academic Press, 1970.

Glushkov, V. M. – Letichevskij, A. B. [73]: Theory of discrete transformers. Selected questions of algebra and logic. Novosibirsk, 1973. (In Russian)

Glushkov, V. M. – Tseytlin, G. E. – Yushchenko, E. L. [74]: Algebras, Languages, Automata. Kiev, Naukova Dumka, 1974.

Goguen, J. A. [71]: Discrete-time machines in closed monoidal categories I. *Institute for Computer Research, Quarterly Report No. 30,* Aug. 1971, The Univ. of Chicago. Also in: *J. Comput. and Syst. Sci. 10,* 1975. 1-43.

Goguen, J. A. [72a]: Minimal realization of machines in closed categories. *Bull. Am. Math. Soc.* 1972, 777-783.

Goguen, J. A. [72b]: On homomorphisms, simulations, correctness, subroutines and termination for programs and program schemes, in *Proc. 13th IEEE Symp. on Switching and Automata Theory* 1972, 52-60. (See also Goguen [74a] for a revised version of this paper.)

Goguen, J. A. [72c]: On mathematics in computer science education. *IBM Thomas Watson Research Center, Research Report No RC 3899,* 1972.

Goguen, J. A. [73]: Realization in universal. *Math. Syst. Theory 6,* 1973, 359-374.

Goguen, J. A. [74a]: On homomorphisms, correctness, termination, unfoldments and equivalence of flow diagram programs. *J. Comput. and Syst. Sci. 8,* 1974, 333-365.

Goguen, J. A. [74b]: Semantics of computation, in: *Category Theory Applied to Computation and Control,* ed. M. A. Arbib, E. G. Manes, Univ. of Mass. Press, Amherst, 1974, 234-239. Also in: LNCS 25, 1975, 151-163.

Goguen, J. A. [74c]: Set-theoretic correctness proofs. *UCLA Reports on semantics and theory of computation, Report No. 1,* 1974.

Goguen, J. A. [75a]: Correctness and equivalence of data types, in: *Proc. Conf. on Alg. S. Th.,* Udine, Italy, 1975. Springer Verl., 1976, 352-358.

Goguen, J. A. [75b]: Some remarks on Data Structures, UCLA, Los Angeles, Calif. 1975. (Manuscript)

Goguen, J. A. [75c]: On fuzzy robot planning. *Proc. U.S. – Japan Joint Conf. on Fuzzy sets and Appl.,* 1975.

Goguen, J. A. [75d]: Objects. *Int. J. General Systems 1,* 1975, 237-243.

Goguen, J. A. [77]: Abstract errors for abstract data types, in: *Proc. of IFIP Working Conf. on Formal Description of Programming Concepts,* ed. J. Dennis, MIT, 1977. 21.1-21.32.

Goguen, J. A. [77a]: Algebraic Specification. To appear in: *Research Directions in Software Technology,* ed. P. Wagner, MIT Press, 1977.

Goguen, J. A. [78]: Some design principles and theory for OBJ-O a language for expressing and executing algebraic specifications of programs, in: *Proc. Int. Conf. Math. Studies of Inf. Processing,* Kyoto, Japan, 429-475.

Goguen, J. A. [79]: Some ideas in algebraic semantics. Dept. Comp. Sci., UCLA, 1979. (Preprint.)

Goguen, J. A. – Burstall, R. M. [78]: Some fundamental properties of algebraic theories: A tool for semantics of computation. Preprint, 1978. Submitted to *Theor. Comput. Sci.*

Goguen, J. A. [79]: Semantics of CLEAR. Preprint, 1979, 1-60.

Goguen, J. A. – Meseguer, J. [77]: Correctness of recursive flow diagram program, in: MFCS, LNCS 53, 580-595.

Goguen, J. A. – Thatcher, J. W. – Wagner, E. G. [76]: An initial algebra approach to the specification, correctness and implementation of abstract data types. *IBM Thomas J. Watson Research Center,* Yorktown Heights, NY., *Research Report RC 6487,* 1976. To appear in: *Current Trends in Programming methodology 3,* Data Structuring, ed. R. T. Yeh, Prentice Hall, 1977.

Goguen, J. A. – Thatcher, J. W. – Wagner, E. G. – Wright, J. B. [73]: A junction between computer science and category theory: I. Basic Definitions and Examples, P. 1. *IBM Research Report RC 4526,* Sept. 1973.

Goguen, J. A. – Thatcher, J. W. – Wagner, E. G. – Wright, J. B. [74]: Factorisation, congruences and the decomposition of automata and systems. *IBM Research Report RC 4934,* July 1974. Also in: *Proc. Second Annual Symp. on Mathematical Foundations of Computer Science,* Warsaw, Poland LNCS 28, 1975. 33-45.

Goguen, J. A. – Thatcher, J. W. – Wagner, E. G. – Wright, J. B. [74b]: Initial algebra semantics. Extended abstract. *IBM Research Report RC 4865,* May 1974. Also in: *Proc. 15th IEEE Symp. on Switching and Automata Theory,* 1974, 64-77. Full paper: *IBM Research Report RC 5243,* Jan. 1975.

Goguen, J. A. – Thatcher, J. W. – Wagner, E. G. – Wright, J. B. [75a]: Initial algebra semantics and continuos algebras. Ext. version of Goguen et al [74b]: *IBM Research Report RC 5701,* Now. 1975. Also in: *JACM 24,* 1, 1977, 68-95.

Goguen, J. A. – Thatcher, J. W. – Wagner, E. G. – Wright, J. B. [75b]: Parallel realization of systems, using factorizations and quotients in categories. *IBM Research Report RC 5668,* Oct. 1975, Also to appear in: *J. Franklin Inst.*

Goguen, J. A. – Thatcher, J. W. – Wagner, E. G. – Wright, J. B. [75c]: Introduction to categories, algebraic theories and algebras. *IBM Research Report RC 5369,* April 1975.

Goguen, J. A. – Thatcher, J. W. – Wagner, E. G. – Wright, J. B. [75d]: Abstract data types as initial algebras and the corectness of data representations, in: *Proc. Conference on Computer Graphics, Pattern Recognition and Data Structures,* Beverly Hills, CA, 1975, 89-93.

Goguen, J. A. – Thatcher, J. W. – Wagner, E. G. – Wright, J. B. [76a]: A junction between computer science and category theory, I.: Basic concepts and examples P. 2. *IBM Thomas J. Watson Research Center, Research Report RC 6908,* 1976.

Goguen, J. A. – Varela, F. [78]: Systems and distinctions: duality and complementarity. Submitted to *Int. J. General Systems.*

Gordon, M. J. C. [75]: Towards a semantic theory of dynamic binding. *AI Memo 265, Stanford Univ.,* Stanford, Calif., 1975.

Grabowski, M. [78]: Full Weak Second-Order Logic versus Algorithmic Logic. *Proc. Math. Logic. in Programming,* Hungary, 1978.

Guessarian, I. [76]: Semantics equivalence of program schemes and its syntactic characterization. *Automata Languages and Programming, Third Int. Coll. at the Univ. of Edinburgh,* 1976, 189-201.

Guessarian, I. [77]: Tests and their syntactic characterisation.*Theoretical Computer Science 11,* 2, 1977, 133-156. (In French).

Guessarian, I. [78]: Some applications of algebraic semantics. MFCS, ed. Winkowski, Springer Verl.,1978. 257-266.

Guttag, J. V. [75]: The specification and application to programming of abstract data types. *Univ. of Torornto, Computer Systems Res. Group TR-CSRG-59,* Sept. 1975.

Guttag, J. V. [76]: Abstract data types and the development of data stuctures. Suppl. to Proc. Conf. on Data Abstraction, Definition, and Structure. *SIGPLAN Notices 8,* March 1976.

Guttag, J.V. – Horowitz, E. – Musser, D. R. [76]: Abstract data types and software validation. *Report ISI/RR-76-48. Information Sciences Inst.,* Marina de Rey, CA, 1976.

Hájek, P. – Havránek, T. [78]: Mechanising hypothesis formation. Springer Verl., 1978.

Harel, D. [78]: Arithmetical completeness in logics of programs. *Proc. ICALP,* Udine 1978. To appear in: LNCS, Springer Verl.

Harel, D. – Meyer, A. R. – Pratt, V. R. [77]: Computability and Completeness in Logics of Programs, in: *Proc. ACM STOC 9,* 1977, 261-268.

Harel, D. – Pratt, V. R. [77]: Nondeterminism in Logics of Programs, in: *Proc. ACM STOC 9,* 1977, 203-213.

Hatcher, W. S. – Rus, T. [76]: Context-free algebra. *J. Cybernetics 6,* 1976, 65-77.

Hayes, P. J. [71]: A logic of action. *Machine Intelligence 6,* Edinburgh Univ. Press, 1971.

Hayes, P. J. [77]: In defence of logic. Essex Univ., Colchester, U. K. 1977. *A. I. Conference at Univ. of Massachusetts* 1977.

Henessy, M. – Ashcroft, E. A. [76]: The semantics of nondeterminism, in: *Automata Languages and Programming. Third International Colloquium at the University of Edinburgh,* 1976, 478-494.

Hoare, C. A. R. – Lauer, P. [74]: Consistent and complementary formal theories of the semantics of programming languages. *Acta Informatica 3,* 1974, 135-153.

Hobbs, J. R. – Rosenschein, S. J. [78]: Making Computational Sense of Montague's Intensional Logic. *Artificial Intelligence 9,* 1978, 287-306.

Hoehnke, H. J. [75]: Synthesis and complexity of logical systems, in: *Category Theory Applied to Computation and Control,* LNCS 25, 1975, 170-174.

Hoehnke, H. J. [76a]On partial algebras. Preprint. Berlin, 1976. (To be published)

Hoehnke, H. J. [76b]: On partial recursive definitions. Preprint. Berlin, 1976.

Hoehnke, H. J. [77]: On partial recursive definitions and programs, in: *Fundamentals on Computer Science,* LNCS 56, 1977, 260-275.

Hoehnke, H. J. [78]: On partial algebras. To appear in *Proc. Coll. Univ. Alg.,* Esztergom, Hungary.

Hyland, J. M. E. [app]: A syntactic characterization of the equality in some models for the lambda calculus. To appear in: *J. London Math. Soc.*

Igarashi, S. [72]: Admissibility of fixed point induction in first order logic of typed theories. *Memo AIM-168, Computer Science Dept. Stanford Univ.,* Stanford, Calif., 1972.

Janicki, R. [app]:An algebraic characterization of concurrency relations. To appear in: *Information Processing Le tters.*

Janssen, T.M.V. [76]: A computer program for Montague grammar, in: *Proc. Amsterdam Coll. Coll. Montague grammar and related topics,* 1976, 154-176.

Janssen, T. M. V. [78]: Logical investigations of PTQ arising from programming requirements. *Math. Centr. ZW 117/78,* Amsterdam.

Janssen, T. M. V. – van Emde Boas, P. [77a]: On the proper treatment of referencing, dereferencing, and assigment, in: *Proc. 4th ICALP Conf.,* Turku, 1977. LNCS 52., 282-300.

Janssen, T. M. V. – van Emde Boas, P. [77b]: The expressive power of intensional logic in the semantics of programming languages, in: *Proc. 6th MFCS Symp.,* LNCS 53, ed. J. Gruska, Springer Verl., 1977, 303-311.

Kaphengst, H. – Reichel, H. [71]: Algebraische Algorithmentheorie. *Wiss. Inf. und Berichte, Nr. 1, Reihe A,* VEB Kombinat ROBOTRON, 1971.

Kaphengst, H. – Reichel, H. [72]: Operative Theorien und Kategorien von operativen Systemen. *Studien zur Algebra und ihren Anwendungen,* Berlin, Akademie Verl., 1972.

Kaphengst, H. – Reichel, H. [77]: Initial Algebraic Semantics for Non Context-free Languages, in: *Fundamentals on Computer Science,* LNCS 56, 1977, 120-127.

Karpinski, M. [73]: Free structure tree automata I. Equivalence. *Bull. Acad. Polon. Sci., Ser. Sci. Math. Astronom. Phys.* 21, 1973, 441-446.

Kekeliya, G. M. – Kissanov, G. E. – Tseytlin, G. E. [74]: Realization of Algorithmic algebraical means in homogeneous systems. *Kybernatica* 5, 1974, 29-35. (In Russian).

Kfoury, D. J. [72]: Comparing algebraic structures up to algorithmic equivalence, in: *Automata Languages and Programming,* ed. M. Nivat, North-Holland, Amsterdam, 1972, 253-263.

Kfoury, D. J. – Park, D. M. R. [75]: On the termination of Program schemas. *Information and Control 29,* 1975, 243-251.

Kotov, V. E. [78]: An algebra for parallelism based on Petri nets. Ed. J. Winkowski, MFCS 1978, Springer Verl., 39-55.

Koubek, V. – Reiterman, J. [75]: Automata and Categories – Input processes, in: *Mathematical Foundations of Computer Science,* LNCS 32, 1975, 280-286.

Koubek, V. – Reiterman, J. [78]: Categorial constructs of free algebras, colomits, and completions of partial algebras. Preprint. Prague, 1978.

Kowalski, R. [73]: Predicate logic as programming language. *Memo No 70, Dept. of Computational Logic, School of Artifical Intelligence, Univ. of Edinburgh,* Nov. 1973.

Kowalski, R. [74]: Logic for problem solving. *Memo No 75, Dept. of Computational Logic, School of Artificial Intelligence, Univ. of Edinburgh,* March, 1974.

Kowalski, R. [76]: Algorithm = Logic + Control. Dept. of Computing and Control, Imperial College. London, Nov. 1976.

Kupka, I. [77]: Partial algebras for representing semantics of information processing. Theses. Univ.Hamburg, 1977.

von Kutshera, F. [74]: Intensional semantics of natural language, in: *Logic Conf,* Kiel, 1974. Lecture Notes in íathematics 499. 445-459.

von Kutschera, F. [76]: Einführung in die instensionale Semantik (Grundlagen der Kommunikation) W. de Gruyter, Berlin, 1976.

Kühnel, W. – Meseguer, J. – Pfender, M. – Sols, I. [77]: Primitive r ecursive algebraic theories and program schemes. *Bull. Austral. Math. Soc. 17,* 1977, 207-233.

Landin, P. J. [69]: A Program Machine Symmetric Automata Theory, in: *Machine Intelligence 5,* Edinburgh Univ. Press, 1969, 99-120.

Lehmann, D. J. [76]: Categories for fixpoint semantics. *Theory of Computation Report No 15, Dept. of Computer Science, Univ. of Warwick.* Also in: *Proc. 17th Annual Symp. Foundations of Computer Science,* IEEE, 1976.

Lehmann, D. J. – Smyth, B. M. [77]: Data types. Preprint. Dept. of Computer Science, Univ. of Warwick, Coventry, Great Britain.

Les .arbres en algébre et en programmation. Troisiéme Colloque, Lille, Febr. 1978. Org.: Arnold, A., Dauchet, M. Jacob, G.

Letichevskij, A. A. [68]: Syntax and Semantics of Formal Languages. *Kybernetica* 4, 1968. (In Russian).

Letichevskij, A. A. [73]: Pratical methods of recognition of equivalence of discrete transformers and program schemes. *Kybernetica*, Kiev, 4, 1973.

Levi, L. S. – Joshi, A. K. [73]: Some results in tree automata. *Math. Syst. Theory 6,* 1973. 334-342.

Levy, M. R. [78]: Data Types with Sharing and Circularity. Ph. D. Thesis. *Report. CS-78-26,* Faculty of Math. Univ. of Waterloo, Ontario, CA, 1978.

Lipski, W. J. [76]: Informational systems with incomplete information, in: *Automata Languages and Programming. Third International Colloquium at the Univ. of Edinburgh,* 1976, 120-131.

Lipski, W. J. [77]: Informational systems: semantics issues related to incomplete information. P. 1. *PRACE CO PAN. CC PAS Reports 275,* Warszawa, 1977.

Lipski, W. J. [77a]: On the logic of incomplete information. *ICS PAS Reports 300.*

Liskov, B. H. – Berzins, V. [77]: An appraisal of program specifications. To appear in: *New Directions in Software Techn. MIT Memo 141-1,* MIT Press.

Liskov, B. H. – Zilles, S. N. [74]: Programming with abstract data types. Proc. of ACM Symp. on Very High Level Languages, *SIGPLAN Notices 9,* 1974, 50-59.

Littrich, G. – Merzenich, W. [77]: Nets over manysorted operator domains and their semantics, in: *Fundamentals on Computer Science,* LNCS 56, 1977, 240-245.

Litvintchouk, S. D. – Pratt, V. R. [77]: A proof-checker for dynamic logic. *Res. Rep. Mass. Inst. of Techn.,* Camridge, June 1977.

Lloyd, C, [72]: Some concepts of universal algebra and their application to computer science. *CSWP-1, Computing Centre, Univ. of Essex,* 1972.

Magidor, M. – Moran, G. [69]: Finite automata over finite trees. *Technical Report 30, Hebrew Univ.,* Jerusalem, Israel, 1969.

Manes, E. G. [76]: Algebraic theories. Chap. IV. GTM 26, Springer Verl., 1976.

Manna, Z. [69]: The correctness of programs. *J. Comput. and Syst. Sci. 3,* 1969, 119-127.

Manna, Z. [74]: Introduction to Mathematical Theory of Computation. McGraw-Hill, New York, 1974.

Manna, Z. – McCarthy, J. [70]: Properties of programs and partial function logic, in: *Machine Intelligence 5,* Edinburgh Univ. Press, 1970, 27-37.

Manna, Z. – Pnueli, A. [70]: Formalization of properties of functional programs. JACM *17,* 1970, 555-569.

Manna, Z. – Shamir, A. [78]: The convergence of functions tofixed points of recursive definitions. *Theoretical Computer Science 6,* 2, 1978, 109-142.

Manna, Z. – Vuillemin, J. [72]: Fixpoint approach to the theory of computation. *CACM 15,* 1972, 528-536.

Markowsky, G. [76]: Chain-complete posets and directed sets with applications. *Algebra Universalis 6,* 1976, 53-68.

Markowsky, G. [77]: Categories of chain-complete posets. *Theoretical Computer Science 4,* 1977, 125-235.

Markowsky, G. – Rosen, B. K. [76]: Bases for chain-complete posets. *IBM J. Res. Development 20,* 2, 1976.

Márkusz, Zs. – Szőts, M. [78]: On Semantics of Programming Languages Defined by Universal Algebraic Tools. *Proc. Math. Logic in Programming,* Hungary, 1978.

Mathematical Foundations of Computer Science. Ed: J. Winkowski, LNCS 64, 1978.

Mazurkiewicz, A. [77]: Concurrent Program Schemes and their Interpretations. *DAIMI-PB-78, Aarhus Univ.,* 1977.

Meitus, V. J. [74]: Abstract prescription of formal language syntax. *Dokl. A. N. SSSR, 216,* 2, 1974, 261-263.

Meitus, V. J. [78]: Transforming categories and finite transformers. I-II. *Kybernetica* 2, 1978, 15-18.

Meitus, V. J. – Vershinin, K. P. [74]: Computable categories and functors. *Dokl. A. N. SSSR, 216,* 1, 1974.

Meseguer, J. [77]: On Order-Complete Universal Algebra and Enriched Functorial Semantics, in: *Fundamentals on Computer Science,* LNCS 56, Springer Verl., 1977, 294-302.

Meseguer, J. [78]: Completions, factorizations, and colimits for ω-posets. *Semantics and Theory of Computation Rep. No 13* UCLA 1978.

Mezei, J. – Wright. J. B. [67]: Algebraic automata and context free sets. *Information and Control 11,* 1967, 3-29.

Milne, R. E. [74]: The formal semantics of computer languages and their implementations. Ph. D. Theses, Cambridge, Univ., Cambridge, England, 1974. Also in: *Technical Monograph PRG-13, Oxford Univ, Computing Lab., Programming Research Group.*

Milner, R. [70]: Algebraic theory of computable polyadic functions. *Computer Sci. Memo. 12, University College,* Swansea, 1970.

Milner, R. [71]: An algebraic definition of simulation between programs. *Stanford Artifical Intelligence Project, Memo AIM-142, Computer Science Dept. Report No CS-205, Stanford Univ.*, Febr. 1971.

Milner, R. [73]: Processes: A mathematical model of computing agents. *Proc. Colloq. In Math. Logic.* Bristol, England, 1973.

Milner, R. [77]: Fully abstract models of typed lambda calculi. *Theoretical Computer Science* 4, 1977, 1-23.

Milner, R. [77a]: Flowgraphs and Flow Algebras. *Dept. of Comp. Sci. Report CSR-5-77,* Edinburgh.

Milner, R. [78]: Synthesis of communicating behaviour. University of Edinburgh, 1978. (manuscript)

Milner, R. [78a]: Algebras for Communicating Systems. *University of Edinburgh, Dept. of Comp. Sci. Internal Report CSR-25-78,* 1978, 1-18.

Milner, R. – Weyrauch, R. [72a]: Proving Compiler Correctness in a Mechanised Logic, in: *Machine Intelligence 7,* 1972, 51-72.

Milner, R. – Weyrauch, R. [72b]: Program semantics and correctness in a mechanised logic. *First USA-Japan Computer Conference,* 1972, 384-392.

Mirkowska, G. [77]: Algorithmic logic and its applications in the theory of programs I-II. *Fundamenta Informaticae, 1,* 1, 1977, 1-7; 147-167.

Mirkowska, G. [78]: Model existence theorem in algorithmic logic with nondeterministic programs. Preprint. Univ. Warsaw, 1978.

Montague, R. [70a]: Universal grammar. *Theorie* 36, 1970, 373-398.

Montague, R. [70b]: Pragmatics and intensional logic. *Synthese* 22, 1970, 68-94.

Morris, F. L. [72]: Correctness of translations of programming languages- an algebraic approach. *Stanford Artificial Intelligence Project Memo AIM-174,* 1972.

Mosses, P. [77]: Making denotational semantics less concrete. *Dept. of Computer Science, Aarhus Univ.,* Denmark. Extended Abstract of a presentation at the *Workshop on Semantics of Programming Languages,* Bad Honnef, Germany, 1977. To appear in *Univ. of Dortmund Technical Report.*

Németi, I. – Sain, I. [78]: Connections between algebraic logic and initial algebra semantics of CF languages. *Proc. Coll. Math. Logic in Programming,* Hungary, 1978.

Nivat, M. [72]: Languages algebrigue sur le magma libre et sémantique des schémas de programme, in: *Automata Languages and Programming,* ed. M. Nivat, North Holland, 1972, 293-308.

Nivat, M. [75]: On the interpretations of polyadic recursive program schemes. *Symposia Mathematica XV,* Instituto Nationale di Alta Matematica, Italy, 1975, 225-281.

Nivat, M. [78]: Infinite words, infinite trees and infinite computations of recursive programs. *Algebra and Applications 11th Semester of the Stefan Banach International Mathematical Center,* Warsaw 1978.

Obtułowicz, A. [77]: Functorial Semantics of the Type Calculus, in: *Fundamentals on Computer Science,* LNCS 56, 1977, 302-308.

Obtułowicz, A. – Wiweger, A. [78]: Functional interpretation of λ-terms. *Proc. Math. Logic in Progr.,* Hungary, 1978.

Ono, H. [76]: On the representability of the termination and the partial correctness of program schemes. *J. of Tsuda College 8,* 1976, 37-52.

Ono, H. – Nakahara, H. [77]: On the termination of program schemes. *J. of Tsuda College 9,* 1977, 43-48.

Oppen, D. C. [75]: On logic and program verification. *Tech. Rep. 82, Dept. of Computer Science, Univ. of Toronto,* 1975.

Pask, G. [76]: Conversation Theory. Elsevier Press, Amsterdam-New York, 1976.

Pasztor, A. [79]: Surjections in the category of chain complete posets, continuos universal algebras and related structures used in algebraic semantics of programming. Submitted to *FCT 79,* Berlin.

Patrikh, R. [78]: Completeness result for a propositional dynamic logic. *Lab. Comput. Sci.,* MIT, *MIT/LCS/TM-106.*

Plotkin, G. D. [76]: A Powerdomain Construction. *SIAM J. on Computing 5,* 3, 1976, 452-488.

Plotkin, G. D. [78]: T^{ω} as a universal domain. *J. Comput. and Syst. Sci. 17,* 1978, 209-236.

Plotkin, G. D. – Smyth, M. [77]: The category theoretic solution of recursive domain equations, in: *Foundations of Computer Science,* 1977. Extended version: *Univ. Edinburgh, D. A. I. Rep. No 60,* 1978.

Pnuelli, A. [77]: The temporal logic of programs. *Proc. IEEE FOCS 18,* 1977, 46-57.

Popplestone, R. J. [78]: Relational Programming, in: *Machine Intelligence 9.* Ellis Horwood, -Chichester, Sussex.

Pratt, V. R. [77]: Semantic Considerations on Floyd-Hoare Logic. *Proc. ACM STOC 9,* MIT, Cambridge, 1977, 109-121.

Proc. of the Int. Workshop on Semantics of Programming Languages. Bad Honnef, 1977, ed. V. Claus, K. Indermark etc. Univesrität Dortmund.

Pultr, A. [75]: Closed categories of fuzzy sets. *Proc. Conf. Automaten u. Algorithmentheorie,* Weissig, 1975.

Rasiowa, H. [73]: On ω^+-valued algorithmic logic and related problems. Supplement to *Proc. Symp. and Summer School on MFCS,* High Tatras, Czehoslovakia, 1973. Also in: *cc PAS Rep. 150,* 1974.

Rasiowa, H. [77]: Algorithmic logic. *PRACE IPI PAN. ICS PAS Rep 281,* Warsaw 1977.

Rattray, C. M. I. [75]: Structure Algebrique, Compilation et Programmes. *Univ. Sci. Grenoble. Lab. Informatique.* Parts I and II. Part II appeared in: *Proc. Eurocomp. Conf. Softw. Eng.* ONLINE, London, 1976.

Rattray, C. M. I. – Rus, T. [77]: Hash-hierarchy, a matehematical device for computer system modelling. *Proc. 1st Int. Symp. on Math. Modelling,* Missouri, 1977, 1-15.

Redyko, V. N. [73]: Definitorial algebras and algebra of languages. *Kybernetica 4,* 1973. (In Russian)

Redyko, V. N. [76]: Theoretic definitorial aspects of languages. In: *Problems of Cybernetics 31,* Nauka, 1976. (In Russian).

Reichel, H. [78]: Fundamentals of an algebraic theory of computation (summary). Preprint. *Banach Center of Math.,* Warsaw, 1978.

Reichel, H. [78a]: Limit-colimit doctrines in computer science. Algebra and Applications, Banach Center semester, 1978. To appear in *Banach Center Publications.*

Reichel, H. [78b]: Algebraic specifications of abstract data types. Algebra and Applications Banach Center Semester, 1978. To appear in *Banach Center Publications.*

Reiterman, J. [77]: A more categorical model of universal algebra, in: *Fundamentals on Computer Science,* LNCS 56, 1977, 308-314.

Reynolds, J. [72]: Notes on a lattice-theoretic approach to the theory of computation. *Systems and Information Science Dept., Syracuse Univ.,* Syracuse, New York, 1972.

Reynolds, J. [77]: Semantics of Domain of Flow Diagrams. *JACM 24,* 3, 1977, 484-503.

Ricci, G. [73]: Cascades of tree automata and computations in universal algebras. *Math. Syst. Theory 7,* 1973, 201-218.

Rine, D. C. [71]: A Categorical Charachterization of General Automata. *Information and Control 19,* 1971, 30-40.

Rine, D. C. [74]: A category theory for programming languages. *Mathematical Systems Theory 7,* 4, 1974, 304-317.

de Roever, W. P. [74]: Operational mathematical and axiomatized semantics for recursive procedures and data structures. *Mathematical Centre Report ID 1/74,* 1974.

Rosenberg, A. L. [76]: Universal data objects are trees. *IBM Res. Rep. RC-5872,* Febr. 1976.

Roussel, P. [75]: PROLOG: Manual de reference et d'utilisation. Groupe d'Intelligence Artificielle, Marsseille-Luminy, Sept. 1975.

Rus, T. [76]: Context-free algebra: A Mathematical device for compiler specification. LNCS 45, Springer Verl., 1976.

Sain, I. [78]: On model theoretic and universal algebraic methods in semantics of computation. Preprint. Budapest, 1978. (In Hungarian).

Salwicki, A. : On the equivalence of FS-expressions and programs. *Bull. Acad. Polon. Sci. Ser. Math. Astronom. Phys. 18,* 275-278.

Scala, H. J. [71]: Graphschemata und ihre Eigenschaften eine Anwendung der Modelltheorie. *Angewandte Informatik,* 1, 1971.

Schönfeld, W. [78]: Application of relation algebras in computer science. Univ. Stuttgart. Also in: Equations in finite relation algebras. *Not. Amer. Math. Soc. 24,* 1977. A-254.

Scott, D. [71]: The lattice of flow diagrams. *Symp. on Semantics of Algorithmic Languages,* ed. E. Engeler. LNCS 188, New York, Springer Verl., 1971, 311-366.

Scott, D. [72]: Mathematical concepts in programming language semantics, *AFIPS Conf. Proc. 40,* 1972, 225-234.

Scott, D. [75]: Combinatorics and classes, lambdacalculus and computer science theory. *Proc. Rome Symp.* LNCS 37, Berlin, Springer Verl., 1975, 1-26.

Scott, D. [76]: Data types as lattices. *SIAM J. on Computability,* 1976.

Scott, D. [77]: Logic and programming languages. *CACM 20,* 9, 1977, 634-641.

Scott, D. [app]: Lattice-theoretic models for the lambda-calculus. (To appear).

Scott, D. – Strachey, C. [71]: Toward a mathematical semantic for computer languages. *Proc. Symp. on Computers and Automata,* Polytechnic Inst. of Brooklyn, *21,* 1971, 19-46.

Segerberg, K. [77]: Completeness Theorem in Modal Logic of Programs. *Not. Amer. Math. Soc.* 1977. A-552.

Shapard, C. D. [69]: Languages in general algebras. *Proc. ACM Symp. on Theory of Computing,* 1969, 155-163.

Sheperdson, J. C. [75]: Computation over abstract structures: Serial and parallel procedures and Friedman's effective definitional schemes. *Logic Coll. '73,* ed. H. E. Rose, J. C. Sheperdson, Amsterdam, North-Holland, 1975, 445-513.

Skornyakov, L. A. [74]: On Algebraic Automata. *Kybernetica 2,* 1974, No 31-34.

Smyth, M. B. [76a]: Category-Theoretic Solution of Recursive Domain Equations. *Theory of Computation Rep. No 14, Dept. of Computer Science, Univ. of Warwick,* 1976. Also in: *J. of Comp. Systems Sci. 16,* 1, 1978.

Smyth, M. B. [76b]: Powerdomains. *Theory of Computation Report No 12, Dept. of Computer Science, Univ. of Warwick,* 1976. Also in: *Mathematical Foundations of Computer Science,* LNCS 45, 1976, 537-543.

Steinby, M [77]: On algebras as tree automata. *Universal algebras, Colloquia Mathematica János Bolyai Societatis,* North-Holland, 1977. (To appear.)

Steinby, M. [77a]: On the structure and realizations of tree automata. *Second Coll. sur les Arbres en Algébre et en Programmation,* Lille, 1977.

Strachey, C. [72]: Varieties of programming Languages. *Proc. International Computing Symp.,* Venice, 1972, 222-233.

Szabolcsi, A. [78]: Modeltheoretic treatement of the semantics of natural languages. Theses. Budapest, 1978. (In Hungarian).

Thatcher, J. W. [67]: Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. *J. Comput and Syst. Sci. 1,* 1967, 317-322.

Thatcher, J. W. [70]: Generalized^2 sequential machine maps. *J. Comput and Syst. Sci. 4,* 1970. 339-367.

Thatcher, J. W. [73]: Tree automata – an informal survey. *Currents in Computing,* ed. A. V. Aho, New Jersey, Prentice Hall, 1973.

Thatcher, J. W. – Wagner, E. G. – Wright, J. B. [76]: Specification of abstract data types using conditional axioms. *IBM Res. Rep. RC-6214,* 1976, 1-17.

Thatcher, J. W. – Wagner, E.G. – Wright, J. B. [78]: Data type specification: parametrization and the power of specification techniques. *Proc. of 10th SIGACT Annual Symp. on Theory of Computing,* 1978.

Thatcher, J. W. – Wright, J. B. [68]: Generalized finite automata with an application on a decision problem of second-order logic. *Math. Syst. Th. 2,* 1968, 57-81.

Theoretical Computer Science. Darmstadt, March 1977. Ed. H. Tzschach, H. Waldschmidt, H. K. G. Walter. LNCS 48, 1977.

Thomason, R. H. [74]: □ed/Formal philosophy. Selected papers of Richard Montague, Yale Univ. Press, 1974.

Tiuryn, J. [77a]: Fixed-points and algebras with infinitely long expressions. P. 1. Regular algebras. *PRACE IPI PAN. ICS PAS REPORTS 284,* Warsawa, 1977. Also in: SLNCS 53, 513-523.

Tiuryn, J. [77b]: Fixed Points and Algebras with Infinitely Long Expressions, II, in: *Fundamentals Computer Science,* LNCS 56, 1977, 332-340. Also in: *ICS PAS Reports 311,* Warsawa, 1978.

Tiuryn, J. [app]: Ordered regular algebras and rational algebraic theories (To appear.)

Tiuryn, J. [78]: Unique fixed points as least fixed points. In: *Schrifter z. Informatic u. Angew. Math. Rheinisch-Westfälische Techn. Hochschule, Aachen,* Bericht No 49.

Tiuryn, J. [79]: Continuity problems in the power-set algebra of finite trees. *4th Workshop on Trees in Algebra and Progr.,* Lille, 1979.

Tiuryn, J. [79a]: Connection between regular algebras and rational algebraic theories. *Proc. 2nd Workshop categorical and algebraic methods in comp. sci. and system theory,* Dortmund.

E. Tóth, P. [76]: Horn logic and its applications in computer science. Master thesis. Eötvös L. Univ., Budapest, 1976. (In Hungarian).

E. Tóth, P. [78]: Intensional Logic of Actions. *CL & CL XII,* 1978. 31-45.

Trees in Algebra and Programming. Conf. in Lille. Febr. 1979. (Proc. to appear.)

Trnková, V. [74]: On minimal realizations of behaviour maps in categorial automata theory. *Commun. Math. Univ. Carolinae 15,* 1975, 555-566.

Trnková, V. [75a]: Minimal realizations for finite sets in categorical automata theory. *Commun. Math. Univ. Carolinae 16,* 1975, 21-35.

Trnková, V. [75b]: Automata and categories, in: *Mathematical Foundations of Computer Science,* LNCS 32, 1975, 138-152.

Trnková, V. [77]: Relational Automata in a Category and their Languages, in: *Fundamentals on Computer Science,* LNCS 56, 1977, 340-358.

Trnková, V. [app]: General theory of relational automata. To appear in: *Fundamenta Informaticae.*

Trnková, A. – Adámek, J. [77]: Minimal realizations is not universal. Technische Univ. Dresden, Sektion Math. *Vorträge zur Automatentheorie* 21/1977. (To appear.)

Trnková, V. – Adámek, J. [78]: Analyses of languages acepted by varietor machines in category. Univ. Carol.

Trnková, V. – Adámek, J. – Koubek, V. – Reiterman, J. [75]: Free algebras, input processes and free monads. *Commun. Math. Univ. Carolinae 16,* 1975, 339-351.

Trnková, V. et al [79]: On languages recognisable in varieties of universal algebras and tree--group functors. Charles Univ. Prague. (Manuscript.)

Tseytlin, G. E. [74]: On Criteria of Infinite Generation in Universal Algebras, *Kybernetica 3, 1974,* 46-51.

Tseytlin, G. E. [74a]: Homogeneous structures and modified Post Systems. *Proc. Int. Symp. on Discrete Systems IFAC -74,* Riga, 1974, 239-247.

Tseytlin, G. E. [75]: The theory of the modified Post Algebras and multidimensional automata structures. MFCS, Springer Verl., 1975.

Tucker, J. V. [78]: Computing in algebraic systems. Math. Inst. Univ. Oslo. 1978. Preprint.

Turner, R. [75]: An algebraic theory of formal languages. MFCS, ed. I. Bečvar, Springer Verl., 1975.

Turner, R. [78]: An algebraic theory of formal languages. Extended version. Preprint. Dept. Comp. Sci. Univ. Essex, Colchester, England. 1978.

Vainstein, F. S. – Osentinskij, N. I. [77]: On the theory of complex systems. *Programirovanie* 2, 1977, 76-84.

Vershinin, K. P. [73]: On the connection between formal languages to describe mathematical theories and axiomatic systems of the set theory. *Kybernetica,* (Kiev) 4, 1973. (In Russian)

Vershinin, K. P. [75]: On the notion of text correctness in the language TL, in: *Mathematical aspects of the theory of intelligent machines.* Kiev, 1975. 61-70. (In Russian)

Wagner, E. G. [71a]: Languages for defining sets in arbitrary algebras. *Proc. 12th IEEE Symp. on Switching and Automata Theory,* East Lansing, Mich., 1971.

Wagner, E. G. [71b]: An algebraic theory of recursive definitions and Thoery of Computing. Shaker Heights, Ohio, 1971.

Wagner, E. G. [73]: From algebras to programming languages. *Proc. 5th ACM Symp. on Theory of Computing,* Austin, Texas, 1973.

Wagner, E. G. [74]: Notes on categories, algebras and programming languages. From a course of lectures given at Queen Mary College, 1974.

Wagner, E. G. – Thatcher, J. W. – Wright, J. B. [77]: Free continuous theories. *IBM Th. Watson Research Center RC-6906, 1977.*

Wagner, E. G. – Thatcher, J. W. – Wright, J. B. [78]: Programming Languages as Mathematical Objects. To appear in *Proc. MFCS,* 1978, 1-25.

Wagner, E. G. – Wright, J. B. – Goguen, J. A. – Thatcher, J. W. [76]: Some fundamentals of order-algebraic semantics. *IBM Research Report RC 6020,* June 1976. Also in: *Proc. of Fifth Int. Symp. on Math.Found.of Comp. Sci.*, Gdansk, Poland LNCS 45, 153-168.

Wand, M. [72]: Closure under program schemes and reflective subcategories. MIT, Artificial Intelligence Lab. Memo, 1972.

Wand, M. [75a]: Fixed-point constructions in orderenriched categories. *Techn. Rep. 23, Computer Science Dept., Indiana Univ.,* Apr. 1975. To appear in *Theor. Comput. Sci. 1979.*

Wand, M. [75b]: An algebraic formulation of the Chomsky hierarchy, in: *Category Theory Applied to Computation and Control,* LNCS 25, 1975, 209-214.

Wand, M. [77]: Final Algebra Semantics and Data Type Extensions. *TR 65, Computer Sci. Dept. Indiana Univ.,* 1977.

Warren, D. H. D. [77]: Logic Programming and Compiler Writing. *Report No 44. Dept. of Artificial Intelligence, Univ. of Edinburgh,* 1977.

Weyrauch, R. W. – Milner, R. [72]: Program correctness in a mechanized logic. *Proc. of the first USA-JAPAN Computer Conference,* 1972, 384-390.

Winkowski, J. [78]: An algebraic approach to non-sequential computations. *ICS PAS Report 312,* Warsaw, 1978.

Wiweger, A. [73]: On coproducts of automata. *Bull. Acad. Polon. Sci. 21,* 1973, 753-758.

Wojdylo, B. [77]: Many-sorted algebras and their application in computer science. Institute of Mathematics, Nicholas Copernicus Univ., Torun, Poland.

Working Conference on Formal Description of Programming Concepts. Preprints of technical papers. IFIP, 1977, Saint Andrews, New Brunswick.

Wright, J. B. – Thatcher, J. W. – Wagner, E. G. – Goguen, J. A. [76]: Rational algebraic theories and fixed-point solutions, in: *Proc. IEEE 17th Symp. on Found. of Comp. Sci.*, Houston, Texas, 1976, 147-158.

Wright, J. B. – Wagner, E. G. – Thatcher, J. W. [77]: A uniform approach to inductive posets and inductive closure. *IBM Res. Rep. RC-6817,* 1977, 1-20.

Zhuravlev, Y.J. [77] [78]: Correct algebras on the sets of non-correct heuristic algorithms. I. II. III. *Kybernetica* (Kiev) No 4, 6, 1977; 1978, 2, (In Russian).

Zilles, S. N. [74]: Algebraic Specifications of Data Types. *Computation Structures Group Memo 119, MIT,* Cambridge, Mass., 1974.

Zilles , S. N. [75]: An introduction to data algebras. Working draft paper. IBM Research, San Jose, Sept. 1975.

Yeh, R.T. [71]: Some structural properties of generalized automata and algebras. *Mathematical Systems Theory 5,* 1971, 306-318.