XII

CL & CL

mputational 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

XII.

Budapest, 1978.

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

K U L T Ú R A

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

TAMÁS FREY
(1927 – 1978)

The world of science as well as our journal has suffered a distressing loss: Tamás Frey died on the 9th April 1978. Bot his life and his activites were closely connected with Hungarian history and science.

Ha was born 1927 in Pécs, and completed his grammar school studies there. In March 1944 he was arrested by the Gestapo, he was liberated in Mauthausen. He matriculated at the Faculty of Mechanical Engineering of the Technical University of Budapest in 1946, and in 1948 continued his studies at the Weak Current Section of the Faculty of Electrical Engineering. He took his degree in 1950 and began his scientific career at the Mathematical Department.

His intense mathematical interest lead him to the study of numerical and graphic methods, he defended his candidate's dissertation with the title "On some local theorems of the constructive function theory" in 1956. Since 1960 he worked as assistant professor, and from the early sixties he was active in the basic issues of computer science. His principal field of research covered automata theory, the theory of approximation, numerical algorithms and the theory of optimal processes, but he was involved in all important trends of international scientific life, so that he achieved results in mathematical statistics information theory, pattern recognition methods, ergodic theory and completion theory.

From 1962 he was caretaker director, from 1964 permanent director of the Computing Centre of the Hungarian Academy of Scienees (one of the predecessors of the Computer and Automation Institute of the Hungarian Academy of Sciences). His ability to obtain a comprehensive view, his acquaintance with his field of activity and his universal knowledge enabled him to develop this young institute to become a centre of applied mathematical research, promoting thereby the efficient application of the results of computer science.

In 1970 he was awarded the title of Doctor of Sciences for his dissertation "Automata, algorithms, their optimization and approximation", a synthesis of the outstanding achievements of his work.

He did not become divorced from educational work, and he actively participated to the very end in the teaching of mathematics and in the preparation and implementing of the educational reform. From 1968 he was Professor and Head of the Mathematical Department of the Faculty of Electrical Engineerig of the Technical University of Budapest. He was a member of several Hungarian and international scientific societies where he promoted the progress of science with the some generous activity as in his principal posts. Generations of mathematicians and engineers into full stature under his guidance, having been given careful and attentive assistance in their first attempts in personal interviews with him. His conception and his scientific imagination opened new vistas in the immense areas of sciences which more unexplored until then, and he surveyed them together with us.

A leading figure of our journal, as the Chairman of the Editorial Board he gave a definite shape to this scientific forum, yet as a true scientist not to be the very image of him, but of the state of the art. His oeuvre should be a model for all of us.

His brilliant personality and his power of scientific imagination carried everybody with him; however, during all his lifetime, he was fully aware of his responsibility for mathematics, both as teacher and scientist. Although his death cut off his numerous activities, considering his career in retrospect we find that it is not characterized by unrealized promises but by thoughts, the conceiving of which has clarified new prospects. This endowed his work with completion but not with finish, for nothing is finished in science. An activity to be continued and a scientist able to be regenerated is Tamás Frey.

Contents

A NEW APPROACH TO MODAL LOGIC

Imre Ruzsa

§1. The pioneering work of A.N. Prior

The investigations described in the present paper were initiated and motivated by some ideas of the late logician A.N. Prior. In Chapter *V* of his 1957 book [5] , he introduced a new modal propositional logic called System $\underline{Q}$. This system permits truthvalue-gaps, and hence, "true in all possible worlds" and "false in no possible world" fail to be synonymous therein. Using Kripke-style model structures (cf., e.g., in [3]), we can reformulate System $\underline{Q}$ as follows.

The language of $\underline{Q}$ contains the propositional connectives $\sim, \supset$ (negation, material conditional), two modal connectives $\underline{M}$ and $\underline{L}$ (possibility and strong necessity), parentheses, and a denumerable set P of sentence parameters. A pair $<W, V>$ is said to be an interpretation for P if W is a nonempty set (called the set of situations or possible worlds), and V is a two-place function from $P \times W$ into the set $\{0,1,2\}$. i.e., for $p \in P$, $w \in W$, the possible values of $V(p,w)$ are the numbers 0,1, and 2, where '0' and '1' stand for the usual truth values 'false' and 'true', respectively, whereas '2' stands for 'unstatable', i.e., '2' represents the truth-value-gap. According to Prior, $V(p,w) = 2$ means that the sentence represented by 'p' speaks of individuals which are absent or non-existent in the world w. Thus, Prior accepts here the primary view of Frege according to which sentences involving individual names without denotation are without truth value (c.f., e.g., [2]).

For compound formulas f,g, the valuation rules are defined inductively as follows:

(a) $V(f,w) = 2$ iff (if and only if) for some parameters p of f, $V(p,w) = 2$.

(b) If $V(f,w) \neq 2$, and $V(g,w) \neq 2$, then:
$V(\sim f,w) = 1 - V(f,w)$;
$V(f \supset g,w) = 0$ if $V(f,w) = 1$ and $V(g,w) = 0$, in other cases, $V(f \supset g,w) = 1$;
$V(\underline{M}f,w) = 1$, if for some $v \in W$, $V(f,v) = 1$; in other cases, $V(\underline{M}f,w) = 0$;
$V(\underline{L}f,w) = 0$ if for some $v \in W$, $V(f,v) \neq 1$, in other cases, $V(\underline{L}f,w) = 1$.

A set α of formulas is said to be satisfiable iff there exists an interpretation $<W, V>$ such that for some $w \in W$, $V(f,w) = 1$ holds for all $f \in \alpha$. The other semantical notions (unsatisfiability, consequence) are defined in the canonical way. Let us note that the unsatisfiability of the unit set $\{\sim f\}$ does not mean that f is "always true", i.e. that for all $<W, V>$, and for all $w \in W$, $V(f,w) = 1$ holds, for the case $V(f,w) = 2$ is not excluded. Rather, the unsatisfiability of $\{\sim f\}$ only means that f is "never false", i.e., that the case $V(f,w) = 0$ is excluded. Thus, it is better to use the formulation "f is irrefutable" (instead of "f is valid") for the case $\{\sim f\}$ is unstatisfable.

One sees immediately that $\underline{Q}$ is an $\underline{S5}$-like modal system in which the relation R of accessibility is universal in the sense that $R = W \times W = W^2$. Obviously, the notion of interpretation may be re-defined as follows: An interpretation (for a set P of parameters) is a triple $<W,R,V>$ where W and V are as earlier, and R is an equivalence relation on W (i.e., $R \subseteq W^2$ is reflexive, symmetric, and transitive). Of course, in the valuation rules for $\underline{M}f$ and $\underline{L}f$, the clause "for some $v\epsilon W$" is to be replaced by "for some $v\epsilon W$ such that $<w,v>\epsilon R$". By varying the conditions on R, one gets $\underline{Q}$-variants of other well-known modal systems ($\underline{S4}$, $\underline{B}$, $\underline{T}$, etc.).

Let us emphasize that in System $\underline{Q}$, $\sim\underline{M}\sim p$ and $\underline{L}p$ may have different truth values in certain worlds. This means that the impossibility of $\sim p$ is not identified with the (strong) necessity of p. We shall use "$\underline{N}p$" as abbreviation for "$\sim\underline{M}\sim p$" without confusing it with "$\underline{L}p$".

A sentence of the form "$\underline{L}(p \supset p)$" expresses that "$p \supset p$" is "always true", i.e., that p is never unstatable. Prior used the abbreviation "$\underline{S}p$" for "$\underline{L}(p \supset p)$" with the intuitive meaning "p is universally statable". Now, "$\underline{L}p$" can be expressed by means of $\underline{M}$ and $\underline{S}$ as follows:

$$"\underline{L}p" = "\sim(\underline{S}p \supset \underline{M} \sim p)" = "\underline{S}p \;\&\; \underline{N}p".$$

Thus, $\underline{S}$ can be used as a primitive (instead of $\underline{L}$).

One might hope that a quantified extension of System $\underline{Q}$ provides a first-order modal logic in which the problem of individual names in modal contexts gets a satisfactory solution: and this is the very philosophical significance of System $\underline{Q}$. (As it is known, Kripke's modal semantics does not solve this problem since free occurrences of individual terms are excluded therein. See, e.g., [4].) In some of his papers, Prior mentioned the possibility of such an extension, but he never gave complete semantics for a first-order extension of $\underline{Q}$. However, some informal principles of Prior are worthy of mention.

If a formalized sentence involves an individual term which is without denotation in a certain world w, then this sentence must be unstatable (without classical truth value) in w. Then, the formal semantics must contain rules to this effect.

In checking informal reasonings, we follow Quine's advice according to which one has only to exhibit structures which are relevant in the inference. Thus it may happen that we use "px" for "x is taller than Jones", i.e., we may use one-place predicate parameter for representing dyadic predicate whenever one of its arguments is fixed throughout. Now, if our Jones is absent in the world w, then every sentence involving p must be unstatable in w. Hence, the formal semantics must contain rules which permit some predicate parameters to be "degenerate" in certain worlds, and prescribe that a sentence involving a predicate parameter degenerate in the world w must be unstatable in w. – We note here that one can assume other grounds which make a predicate degenerate in a world. E.g., one can assume that the predicate 'red' is degenerate on the universe of integers – although 'red' involves no induvidual terms.

Furthermore, in Prior's view, the first-order extension of $\underline{Q}$ must be a "free logic" in which "$\exists x(x = x)$" is refutable, i.e., the domain of some worlds may be empty. (See. e.g. [6], pp. 157-159.)

Having axiomatized versions of the (propositional) $\underline{Q}$ (see e.g. in [1] and [7]), Prior proposed in 1968 an axiomatized-deductive calculus as a quantified extension of $\underline{Q}$ (cf. [6]cited above). As was shown by the present author, this attempt of Prior was unsuccessful (see [8], §0, pp. 53-54) inasmuch as "$\exists x(x = x)$" is deducible from his axioms.

My own investigations in this field led me to the conclusion that the quantified extension of System $\underline{Q}$ – satisfying Prior's intention as far as possible – must obey the following two principles:

(i) "Not for all x is false that" and "for some x is true that" must be distinguished analogously as "$\sim \underline{L} \sim p$" and "$\underline{M}p$" are distinguished. This means that one has to introduce a truly existential quantifier "$\exists x$" which is stronger than "$\sim \forall x \sim$".

(ii) Rigid and non-rigid individual terms are strongly distinguished.

In my earlier paper [8], four "Prior type" first-order modal systems (analogous to $\underline{S5}$, $\underline{S4}$, the "Brouwerian system" $\underline{B}$, and the Feys-von Wright system $\underline{T}$) were introduced. These systems are primarily defined as semantical ones. Their primitive logical signs are $\sim, \supset, =$ (identity), $\forall$, $\exists$, $\underline{M}$, and $\underline{S}$. Empty universes are not excluded, and hence, "$\exists x(x = x)$" is refutable (but "$\forall x(x = x)$" and "$\sim \forall x \sim (x = x)$" are irrefutable) in these systems. In general all the logical truths of classical first–order logic (with identity) remained irrefutable (provided "$\sim \forall x \sim$" is not replaced by the strong existential quantifier "$\exists x$"). On empty universes, the logical truths are neither true nor false but unstatable.

In my cited paper, rigid individual terms are represented by individual parameters, and non--rigid ones by (free) individual variables. And here we find the main insufficiency of that approach. For, generally speaking, non-rigid individual names are descriptions, and the descriptions of a language depend on the predicates of that language. Representing them by independent signs, this dependence remains hidden. Hence, my paper [8] provides only a provisional solution of the problem of the behaviour of individual names in modal contexts. In the final solution of this problem, descriptions must be permitted in the object language.

The present paper is a short report on my new approach to the solution of the outlined problem. In the next section a formal semantics for six first-order modal systems will be introduced. These systems are $\underline{Q}$-like versions of the classical modal calculi $\underline{S5}$, $\underline{S4}$, $\underline{B}$, $\underline{T}$, $\underline{D}$ (the deontic weakening of $\underline{T}$), and $\underline{K}$ (Kripke's minimal system). Their common language is based on the same primitives as in [8] plus the descriptor ($\underline{I}$). Descriptions are permitted without any restriction. E.g., modal connectives and descriptions may occour inside descriptions, free variables occurring in a description may be quantified from outside etc. The object language does not contain a "zero entity".

§3 is devoted to some problems of the descriptions. §4 treats the so-called relevance relations which are important with respect to the axiomatizations of the systems. Finally, §5 contains the syntactico-axiomatized versions of our systems.

As a short report, this paper contains no proofs. The omitted proofs and other details may be found in [9] (to appear).

§2. The Q systems: Semantics

[Throughout in this paper, we assume a normal set theory (in particular, we assume that $U \epsilon U$ does not hold for any set U), and we use the standard set-theoretical notation. The set of the natural numbers will be denoted by 'ω'. A natural number n may be assumed as the set of natural numbers less than n. Thus, '0' denotes both the empty set and the number nought. For all sets U, $U^0 = \{0\} = 1$, and $U^{n+1} = U^n \times U$. Empty sequences will be identified by 0. We use " $\{U,u\}$ " as abbreviation for "$U \cup \{u\}$ ".]

2.1. **Q-modal languages**

2.1.1. **Definition.** By a _Q-modal language_ we mean a quintuple $<C,X,P,T,F>$ where C, X, P, T, and F are sets satisfying the following conditions (i) to (v).

(i) C is the set of the _logical constants_ of the language. It contains ten distinct symbols; namely: Left and right parentheses, the connectives $\sim, \supset, \underline{M}$, and $\underline{S}$ (negation, conditional, possibility, and necessary statability, respectively), the quantifiers V and $\exists$, the sign of identity (=) and the descriptor ($\underline{I}$). I.e.,

$$C = \{ (\ ,\ ), \sim, \supset, V, \exists, =, \underline{I}, \underline{M}, \underline{S}, \} .$$

(ii) X is a denumerably infinite set of symbols called (individual) _variables;_ $C \cap X = 0$.

(iii) P is an at most denumerably infinite set of symbols called _parameters._ More specifically:

$$P = \{V\} \cup A \cup (\cup_{n \epsilon \omega} P_n);$$

$P \cap C = \{V\}$; $P \cap X = 0$; for all n, $A \cap P_n = 0$; and for all m,n if $m \neq n$, then $P_m \cap P_n = 0$. – The set A is at most denumerably infinite, its members are called _individual parameters_ (briefly; _in-parameters)._ For all $n \epsilon \omega$, P_n is at most denumerably infinite, its members are called _n-place predicate parameters_ (briefly: _pred$_n$-parameters:_ in the case $n = 0$, _sentence parameters)._

(iv) Finite and nonempty sequences formed from the members of $C \cup X \cup P$ will be called _expressions_ of the language. Certain expressions will be called terms, others will be called formulae. T and F are the sets of the terms and the formulae, respectively, of the language. The simultaneous inductive definition of terms and formulae is given below in 2.1.2.

(v) Certain occurrences of a variable $x \epsilon X$ in a term or a formula will be called free occurrences of x. The definition of T and F in 2.1.2 contains also the inductive definition of the free occurrences of variables in terms and formulae.

Comment. The case $P = \{ \forall \}$ is not excluded. We include $\forall$ in P for convenience; the advantages of this device will be seen in 2.2.

2.1.2 **The definition of terms and formulas.** In what follows, double inverted commas will be used as quasi-quotation signs (instead of Quine's corners). If 'f' and 'g' represent expressions (of a language), then

$$\text{"}fg\text{"}$$

represents the concatenation of the expressions represented by 'f' and 'g' (in this order). However, double inverted commas will be omitted frequently (if no misunderstanding can arise by their omission).

The sets T and F are the smallest subsets of the set of the expressions of a Q-language satisfying the following conditions (i) to (vii).

(i) $X \cup A \subseteq T$. Any occurrence of a variable x in a variable or in an in-parameter is a free one. (Of course, if $y \epsilon X$, then x occurs in y iff $x = y$, and if $a \epsilon A$, then x does not occur in a.)

(ii) If $t_1, t_2 \epsilon T$, then "$t_1 = t_2$"ϵF. An occurrence of $x \epsilon X$ in "$t_1 = t_2$" is a free one iff it is a free one in t_1 or in t_2.

(iii) For all $n \epsilon \omega$: If $p \epsilon P_n$, and $t_1, \ldots, t_n$ are in T, then "$pt_1 \ldots t_n$"ϵF. (In the case $n = 0$, $p \epsilon F$ simply). An occurrence of $x \epsilon X$ in "$pt_1 \ldots t_n$" is a free one iff it is a free one in t_1, or ... or in t_n.

(iv) If $x \epsilon X$, $f \epsilon F$, and f involves some free occurrences of x, then "$\underline{I}xf$"ϵT. No occurrence of x in "$\underline{I}xf$" is a free one. If $y \neq x$, $y \epsilon X$, then an occurrence of y in "$\underline{I}xf$" is a free one iff it is a free one in f.

(v) If $x \epsilon X$, and $f \epsilon F$, then "$\forall xf$" and "$\exists xf$" are in F. No occurrence of x in these formulas is a free one. If $y \neq x$, $y \epsilon X$ then the free occurrences of y in "$\forall xf$" and in "$\exists xf$" are exactly the same as those in f.

(vi) If $f \epsilon F$, then "$\sim f$", "$\underline{M}f$" and "$\underline{S}f$" are in F. An occourrence of a variable x in these formulas is a free one iff it is a free one in f.

(vii) If f, g are in F, then "$(f \supset g)$"ϵF. The free occurrences of a variable x in "$(f \supset g)$" are the same as those in f and in g.

Formulae defined by (ii) and (iii) will be called atomic formulae. Terms defined by (iv) will be called desriptions.

Comment. "Vacuous" descriptions are excluded by (iv); i.e., if f involves no free occurrences of x, then "$\underline{I}xf$" is not a term.

2.1.3. **Further definitions and conventions.** In what follows, we shall use (in most cases, without particular reference) the following letters (with or without subscripts) as metavariables: 'x', 'y', and 'z' for (individual) variables, 'a', 'b' and "c" for in-parameters, 'p' for (arbitrary) parameters, 't' for (arbitrary) terms, 'f', 'g', and 'h' for formulae.

(i) A formula f (a term t) is said to be open if it involves some free occurrences of some variables; in the contrary case, it is said to be closed. Closed formulae are called sentences. – Subformulae of a formula are defined as usually, by adding that if "$\underline{I}xg$" occurs in f then the subformulae of g are subformulae of f.

(ii) An occurrence of the quantifier $\forall$ in a formula f is said to be effective iff is followed by "xg" where g is a subformula of f, $x \epsilon X$, and g involves some free occurrences of x.

(iii) By the parameters of a formula f we mean the parameters involved in f with the restriction that $\forall$ is a parameter of f iff f involves some effective occurences of $\forall$. By the parameters of a description "$\underline{I}xg$"we mean the parameters of g.

(iv) Convections for omitting parentheses. Outermost parentheses will be omitted. As a further convention, we write "$f \supset g \supset h$" instead of "$f \supset (g \supset h)$". – We use dots (as spacemarkers) to help the reading of schemata where the formation rules do not prescribe parantheses. E.g., we shall write "$\underline{I}x.f$", "$\exists x.x = \underline{I}y.f$", and "$\underline{M}.a = b. \supset f$" instead of "$\underline{I}xf$", "$\exists xx = \underline{I}yf$", and "$\underline{M}a = b \supset f$", respectively.

(v) We shall use the signs $\&$, $\vee$, $\equiv$, $\underline{N}$, and $\underline{L}$ as abbreviations, accordirg to the following contextual definitions.
For all formulae f,g:

$$"f \,\&\, g" =_{df} "\sim (f \supset \sim g)", \quad "f \vee g" =_{df} "\sim f \supset g",$$

$$"f \equiv g" =_{df} "(f \supset g) \,\&\, (g \supset f)",$$

$$"\underline{N}f" =_{df} "\sim\underline{M}\sim f", \qquad "\underline{L}f" =_{df} "\underline{N}f \,\&\, \underline{S}f",$$

Furthermore, we use the abbreviation

$$'\underline{E}^0' \text{ for } "\exists x.x = x"$$

As this abbrevation suggest, we "identify" f and g whenever g is obtained from f by a correct re-namig of its bound variables.

(vi) Substitution of terms. If f is any formula, then the expression "$f(t_1/t_2)$" will denote the formula obtained from f by substituting t_1 for every free occurrence of t_2 in f. If t_1 is a variable, then we assume – in order to avoid difficulties – that it does not occur in f.

2.2. **Interpretations**

2.2.1. **Definition.** By an interpretation of a Q-language $<C,X,P,T,F>$ we mean a quadruple $I = <W,R,U,\varphi>$ where W and U are nonempty sets, $R \subseteq W^2$, and φ is a function with domain P satisfying the following conditions (i) to (iii).

(i) $\varphi(\digamma) \epsilon (\underline{P}(U))^W$, i.e., $\varphi(\digamma)$ is a function from W into $\underline{P}(U)$ (the set of all subsets of U). The value of $\varphi(\digamma)$ at $w \epsilon W$ will be denoted by $\dot{\varphi}(\digamma)_w$.

(ii) If $a \epsilon P$ is an in-parameter, then $\varphi(a) \epsilon U$.

(iii) If $p \epsilon P$ is a pred$_n$-parameter ($n \epsilon \omega$), then $\varphi(p) \epsilon (\underline{P}(U^n \cup \{U\}))^W$. The value of $\varphi(p)$ for $w \epsilon W$ will be denoted by $\varphi(p)_w$.

Comments. W may be called the set of indices labelling possible situations or "worlds". (A convenient assumption: W is a set of ordinals.) R may be conceived as representing the accessebility relation between the possible worlds. U may be called the universe or the domain of individuals. Finally, φ is said to be the reference function which determines the reference (the denotation) of all parameters in all possible worlds. As it will be shown, if p is a predicate parameter, then the presence of U in $\varphi(p)_w$ will mean that p is "degenerate" in the world labelled by w in the sense that any sentence containing p is without truth value at w.

2.2.2. Given an interpretation $I = <W,R,U,\varphi>$ of a Q-language, we say that a parameter $p \epsilon P$ is degenerate at $w \epsilon W$ (according to φ) iff

(i) $p = \digamma$, and $\varphi(\digamma)_w = 0$, or

(ii) $p = a$ is an in-parameter, and $\varphi(a) \notin \varphi(\digamma)_w$, or

(iii) p is a predicate parameter, and $U \epsilon \varphi(p)_w$.

We say that φ pre-degenerates a formula f (a term t) at w iff a parameter of f (of t) is degenerate at w (according to φ).

2.2.3. Given an interpretation I of a Q-language, by an assignment we mean a function V from X into U, i.e., $V \epsilon U^X$. – For $u \epsilon U$, $x \epsilon X$, and $V \epsilon U^X$, V^x_u denotes the assignment which is the same as V except (at most) that $V^x_u(x) = u$.

We say that V pre-degenerates a formula f (a term t) at $w \epsilon W$, iff either for some x occurring free in f (in t), $V(x) \notin \varphi(\digamma)_w$, or else φ pre-degenerates f (t) at w (in the sense of 2.2.2).

2.3. **Valuation rules**

Let I be an interpretation of a Q-language, and let V be an assignment. We define (inductively) for all $f \epsilon F$, and for all $t \epsilon T$, the value of f (of t) at w according to V, denoted by $V(f)_w$ (by $V(t)_w$). As we shall see, $V(t)_w \epsilon U \cup \{U\}$, and $V(f)_w \epsilon \{0,1,2\} = 3$. The definitions run from "(Val.Var.)" to "(Val.$\exists$)" below.

(Val.Var.) For $x \epsilon X$, $V(x)_w = V(x)$.

(Val.In) If a is an in-parameter, then $V(a)_w = \varphi(a)$.

(Val.I) If there is exactly one $u \epsilon U$ such that

$V^x_u(f)_w = 1$, then $V(\underline{I}x.f)_w = u_0$, where $\{u_0\} = \{u \epsilon U: V^x_u(f)_w = 1\}$.

In other cases, $V(\underline{I}x.f)_w = U$. (Here f must involve some free occurrences of x.)

(Val.=) If $V(t_1)_w = V(t_2)_w \epsilon \varphi(\digamma)_w$, then $V(t_1 = t_2)_w = 1$. If $\{V(t_1)_w, V(t_2)_w\} \not\subseteq \varphi(\digamma)_w$, then $V(t_1 = t_2)_w = 2$. In other cases, $V(t_1 = t_2)_w = 0$.

(Val.Pr) If p is a pred$_n$-parameter $(n \epsilon \omega)$, $U \not\epsilon \varphi(p)_w$, $V(t_i)_w = u_i$ $(1 \leqslant i \leqslant n)$, $<u_1, \ldots, u_n> \epsilon \varphi(p)_w$, and $\{u_1, \ldots, u_n\} \subseteq \varphi(\digamma)_w$, then $V(pt_1 \ldots t_n)_w = 1$. If $U \epsilon \varphi(p)_w$, or if $\{u_1, \ldots, u_n\} \not\subseteq \varphi(\digamma)_w$, then $V(pt_1 \ldots t_n)_w, = 2$. In other cases, $V(pt_1 \ldots t_n)_w = 0$. (In the case $n = 0$: $V(p)_w = 1$ iff $\varphi(p)_w = \{0\} = 1$; $V(p)_w = 2$ iff $U \epsilon \varphi(p)_w$; otherwise $V(p)_w = 0$.)

(Val.~) $V(\sim f)_w = 2$ if $V(f)_w = 2$. In other cases, $V(\sim f)_w = 1 - V(f)_w$.

(Val.M) If $V(f)_w \neq 2$, and for some $v \epsilon W$, both $<w,v> \epsilon R$ and $V(f)_w = 1$ hold, then $V(\underline{M}f)_w = 1$. If $V(f)_w = 2$, then $V(\underline{M}f)_w = 2$. In other cases, $V(\underline{M}f)_w = 0$.

(Val.S) If $V(f)_w \neq 2$, and for some $v \epsilon W$, both $<w,v> \epsilon R$ and $V(f)_w = 2$ hold, then $V(\underline{S}f)_w = 0$. If $V(f)_w = 2$, then $V(\underline{S}f)_w = 2$. In other cases, $V(\underline{S}f)_w = 1$.

(Val. ⊃) If $V(f)_w = 2$, or if $V(g)_w = 2$, then $V(f \supset g)_w = 2$. If $V(f)_w = 1$, and $V(g)_w = 0$, then $V(f \supset g)_w = 0$. In other cases, $V(f \supset g)_w = 1$.

(Val.∀) If for all $u \epsilon U$, $V^x_u(f)_w = 2$, then $V(\forall x.f)_w = 2$. If for some $u \epsilon U$, $V^x_u(f)_w = 0$, then $V(\forall x.f)_w = 0$. In other cases, $V(\forall x.f)_w = 1$.

(Val.∃) If $V(\forall x.f)_w = 2$, then $V(\exists x.f)_w = 2$ except when the following conditions hold: $\exists x.f$ involves no descriptions, and V does not pre-degenerate it at w(cf. 2.2.3). – If for some $u \epsilon U$, $V^x_u(f)_w = 1$, then $V(\exists x.f)_w = 1$. In the remaining cases, $V(\exists x.f)_w = 0$.

Comments.

(i) The numbers 0, 1, and 2 as values of formulae represent the falsity, the truth, and the truth-valuelessness, respectively; and U as a possible value of a description expresses the failure of the denotation.

(ii) If V pre-degenerates a formula f (a description $\underline{I}x.f$) at w, then $V(f)_w = 2$ ($V(\underline{I}x.f)_w = U$). If f involves no descriptions, then $V(f)_w = 2$ iff V pre-degenerates f at w.

(iii) It is easy to see that $V(\exists x.f)_w$ and $V(\sim \forall x \sim f)_w$ "almost" coincide. Particularly, if $\varrho(\forall)_w \neq 0$, then they coincide for all f. But if $\varrho(\forall)_w = 0$, then they may differ for some f, e.g., for $f =$ "$x = x$". For, in this case, $V(\sim \forall x \sim .x = x)_w = V(\forall x.x = x)_w = 2$, but $V(\underline{E}^0)_w = V(\exists x.x = x)_w = 0$. Thus, "$\exists x$" is somewhat stronger than "$\sim \forall x \sim$". If x has no free occurrences in f, then $V(\forall x.f)_w = V(\exists x.f)_w = V(f)_w$.

(iv) As (Val.Pr) shows, if for some $i\,(1 \leqslant i \leqslant n)$, $V(t_i)_w \notin \varrho(\forall)_w$, then $V(pt_1 \ldots t_n)_w = 2$ even in the case $<V(t_1)_w, \ldots, V(t_n)_w> \epsilon \varrho(p)_w$. This is in contrast to Kripke's modal semantics, and causes some difference concerning the so-called Barcan formulas. (See (18) and (19) in 2.5.2, and (8) . . . (11) in 2.5.8.)

One can see easily that if $V_1(x) = V_2(x)$ for all x occurring free in a formula f, then $V_1(f)_w = V_2(f)_w$; and similarly for terms instead of formulae. Hence, if f is a sentence (a closed formula), then $V(f)_w$ does not depend on V. Thus, we write $\varrho(f)_w$ as the value of f at w (according to ϱ). Similarly, if t is a closed term, we write $\varrho(t)_w$ as the value of t at w (according to ϱ).

2.4. **Satisfiability, consequence, irrefutability**

2.4.1. An interpretation $I = <W,R,U,\varrho>$ is said to be

a $\underline{Q5}$-interpretation iff R is an equivalence relation (on W),
a $\underline{Q4}$-interpretatipn iff R is reflexive and transitive,
a $\underline{QB}$-interpretation iff R is reflexive and symmetric,
a $\underline{QT}$-interpretation iff R is reflexive,
a $\underline{QD}$-interpretation iff for all $w \epsilon W$, there is a $v \epsilon W$ such that $<w,v> \epsilon R$ holds.

Finally, any interpretation is said to be a QK-interpretation.

2.4.2. Let $<C,X,P,T,F>$ be a $\underline{Q}$-language, and let α be a set of sentences of this language. We say that the interpretation $I = <W,R,U,\varrho>$ of this language satisfies the set α iff for some $w \epsilon W$, $\varrho(f)_w = 1$ holds for all $f \epsilon \alpha$.

For $\underline{J} = \underline{K},\underline{D},\underline{T},\underline{B},\underline{4},\underline{5}$, we say that α is QJ-satisfiable iff there is a $\underline{QJ}$-interpretation which satisfies α, and we say that the sentence f is $\underline{QJ}$-satisfiable iff the unit set $\{f\}$ is $\underline{QJ}$-satisfiable. A sentence f is said to be a QJ-consequence of the set (of sentence) α iff $\{\alpha, \sim f\}$ is $\underline{QJ}$-unstatisfiable (i.e., is not $\underline{QJ}$-statisfiable), and f is said to be QJ-refutable iff $\sim f$ is $\underline{QJ}$-satisfiable. We say that f is QJ-irrefutable iff $\sim f$ is $\underline{QJ}$-unsatisfiable (i.e., if f is a $\underline{QJ}$-consequence of the empty set of sentences). – We may say that a sentence f is QJ-valid iff whenever $<W,R,U,\varrho>$ is a $\underline{QJ}$-interpretation of a $\underline{Q}$-language containing all parameters of f, then for all $w \epsilon W$, $\varrho(f)_w = 1$ holds. It is obvious that f is $\underline{QJ}$-valid iff f is $\underline{QJ}$-irrefutable, and f involves neither parameters nor descriptions. The sentences "$\underline{E}^0 \supset \underline{E}^0$" and "$\underline{SE}^0$" are examples of $\underline{QJ}$-valid sentences (for all $\underline{J}$).

By the previous definitions, we may speak of the sementical $\underline{QJ}$-systems. The following diagram shows the interrelations between these systems:

$$\begin{tikzcd}
 & & & \underline{Q4} \arrow[dr] & \\
\underline{QK} \arrow[r] & \underline{QD} \arrow[r] & \underline{QT} \arrow[ur] \arrow[dr] & & \underline{Q5} \\
 & & & \underline{QB} \arrow[ur] &
\end{tikzcd}$$

If the system $\underline{QJ}_1$ precedes $\underline{QJ}_2$ in this diagram, then:

if a set α of sentences is $\underline{QJ}_2$ satisfiable, then α is $\underline{QJ}_1$ satisfiable, and

if the sentence f is a $\underline{QJ}_1$-consequence of α, then f is a $\underline{QJ}_2$-consequence of α (and if f is $\underline{QJ}_1$-irrefutable, then f is $\underline{QJ}_2$-irrefutable).

Let us note that $\underline{QK}$-irrefutability implies $\underline{QJ}$-irrefutability, for all $\underline{J}$.

2.5. **Some laws of the Q systems**

(We shall use the variable 'J' exclusively as refering to our system codes $\underline{K,D,T,B,4,5}$.)

2.5.1. General laws.

(i) If the sentence f is QJ-irrefutable, then so are $\underline{N}f$, $\forall x.f(x/c)$, and $\sim \exists x{\sim}f(x/c)$. (The law of generalization.)

(ii) Assume that $f \supset g$ is $\underline{QJ}$-irrefutable. Then: (a) If c is an in-parameter not occurring in f, then $f \supset \forall x.g(x/c)$ is $\underline{QJ}$-irrefutable. – (b) If g does not involve c, then $\exists x.f(x/c) \supset g$ is $\underline{QJ}$-irrefutable.

(iii) If a $\underline{Q}$-sentence is a substitution instance of a tautology of the classical propositional calculus (PC), then it is $\underline{QK}$-irrefutable.

(iv) If f is a valid sentence of first-order logic, then f is $\underline{QK}$-irrefutable. However, Q-instantiations of valid first-order schemata may be $\underline{QK}$-refutable. E.g., if f involves modal connectives, $\underline{a}$ occurs in the scope of a modal connective in f, and t is a (closed) description, then "$\forall x.f(x/a) \supset f(t/a)$" may be $\underline{QK}$-refutable. We formulate a condition on which the cited schema is $\underline{QK}$-irrefutable.

Definition. A closed term t is said to be substitutable for a in f iff either t is an in-parameter, or else no occurrence of a in f stands in the scope of a modal connective $\underline{M}$ or $\underline{S}$.

(v) If f and t are closed, and t is substitutable for a in f, then

$$\forall x.f(x/a) \supset f(t/a) \quad \text{and} \quad f(t/a) \supset \exists x.f(x/a)$$

are QK-irrefutable.

2.5.2. Some irrefutable sentences of $\underline{QK}$. In the following schemata, f and g are sentences, and t is any closed term. All instances of these schemata are irrefutable in $\underline{QK}$ (and, hence, in all $\underline{Q}$ systems).

(1) $\forall x.f(x/c) \equiv \sim \exists x \sim f(x/c)$

(2) $\forall x.f(x/c) \supset \exists x.f(x/c)$

(3) $t = t$

(4) $\underline{N}.a = a$

(5) $\underline{M}.a = b \supset a = b$

(6) $a = b \supset \underline{N}.a = b$

(7) $a = b \supset \underline{M}.a = a \supset \underline{M}.a = b$

(8) $\underline{M}(f \,\&\, g) \supset \underline{M}f$

(9) $\underline{M}(f \vee g) \supset (\underline{M}f \vee \underline{M}g)$

(10) $\underline{N}f \supset \underline{N}g \supset \underline{N}(f \,\&\, g)$

(11) $\underline{L}(f \supset g) \supset \underline{L}f \supset \underline{L}g$

(12) $\underline{L}(f \,\&\, g) \equiv (\underline{L}f \,\&\, \underline{L}g)$

(13) $\underline{L}f \supset \underline{S}g \supset \underline{L}(f \vee g)$

(14) $\underline{S}(f \,\&\, g) \supset (\underline{L}f \vee \underline{L}g) \supset \underline{L}(f \vee g)$

(15) $\sim \underline{S}.a = a \supset \underline{L}f \supset \underline{M}f$

(16) $\underline{S}f$ provided f involves neither parameters nor descriptions

(17) $\underline{S}\forall x.x = x \supset \underline{NE}^0$

(18) $\exists x.\underline{M}f(x/c) \supset \underline{M}\,\exists x.f(x/c)$

(19) $\underline{N}\forall x.f(x/c) \supset \forall x.\underline{N}f(x/c)$

2.5.3. Sentences irrefutable in $\underline{QD}$ (but refutable in $\underline{QK}$):

(1) $\underline{L}.a = b \supset a = b$

(2) $a = b \supset \underline{S}.a = a \supset \underline{M}.a = b$

(3) $\underline{L}.a = b \supset \underline{M}.a = b$

(4) $\underline{L}f \supset \underline{M}f$

(5) $\underline{S}f \supset \underline{M}(f \supset f)$

2.5.4. Sentences irrefutable in QT (but refutable in $\underline{QD}$):

(1) $\underline{M}.a = a$

(2) $\underline{N}.a = b \supset a = b$

(3) $a = b \supset \underline{M}.a = b$

(4) $\underline{L}f \supset f$

(5) $\underline{N}f \supset f$

(6) $f \supset \underline{M}f$

(7) $\underline{N}f \supset \underline{M}f$

(8) $\underline{M}(f \supset f)$

2.5.5. Sentences irrefutable in QB (but refutable both in $\underline{QT}$ and $\underline{Q4}$).

(1) $f \supset \underline{NM}f$

(2) $\underline{MN}f \supset f$

(3) $\underline{MN}f \supset \underline{NM}f$

(4) $\underline{M}\ \exists\, x(f(x/a)\ \&\ \underline{S}.x = x) \equiv \exists\, x.\underline{M}(f(x/a)\ \&\ \underline{S}.x = x)$

(5) $\underline{M}\ \exists x.\underline{S}.x = x \equiv \exists x.\underline{MS}.x = x$

2.5.6. Sentences irrefutable in $Q4$ (but refutable both in $\underline{QT}$ and $\underline{QB}$).

(1) $\underline{N}f \equiv \underline{NN}f$

(2) $\underline{S}f \equiv \underline{NS}f$

(3) $\underline{M}f \equiv \underline{MM}f$

(4) $\sim \underline{S}f \equiv \underline{M} \sim \underline{S}f$

(5) $\underline{L}f \equiv \underline{LL}f$

2.5.7. Sentences irrefutable in $Q5$ (but refutable in the other $\underline{Q}$ systems).

(1) $\underline{MN}f \equiv \underline{N}f$

(2) $\underline{MS}f \equiv \underline{S}f$

(3) $\underline{NM}f \equiv \underline{M}f$

(4) $\underline{ML}f \equiv \underline{L}f$

(5) $\underline{LM}f \equiv (\underline{M}f\ \&\ \underline{S}f)$

(6) $\underline{M}\ \exists x.\underline{S}.x = x \equiv \exists x.\underline{S}.x = x$

(7) $\underline{M}\ \exists x.\underline{M}f(x/c) = \underline{M}\ \exists x.f(x/c)$

2.5.8. Sentences refutable in $Q5$ (and, hence, in all $\underline{Q}$-systems).

(1) $\exists x.x = x$ (i.e. $\underline{E}^0$)

(2) $(\underline{M}f \vee \underline{M}g) \supset \underline{M}(f \vee g)$

(3) $\underline{N}(f \,\&\, g) \supset \underline{N}f$

(4) $\underline{N}(f \supset g) \supset \underline{N}f \supset \underline{N}g$

(5) $\underline{N}(f \supset g) \supset \underline{M}f \supset \underline{M}g$

(6) $\underline{L}f \supset \underline{L}(f \vee g)$

(7) $(\underline{L}f \vee \underline{L}g) \supset \underline{L}(f \vee g)$

(8) $\underline{M}\ \exists x.f(x/c) \supset \exists x.\underline{M}f(x/c)$

(9) $\forall x.\underline{N}f(x/c) \supset \underline{N}\forall x.f(x/c)$

(10) $\forall x.\underline{L}f(x/c) \supset \underline{L}\forall x.f(x/c)$

(11) $\underline{L}\forall x.f(x/c) \supset \forall x.\underline{L}f(x/c)$

Remark. (18) and (19) in 2.5.2, and (8) . . .(11) in 2.5.8, show the logical status of the so-called Barcan formulas in the $\underline{Q}$ systems. – Other laws of the $\underline{Q}$ systems will be mentioned later.

§3. **Descriptions**

(Throughout this section, we assume that $I = \langle W,R,U,\varphi\rangle$ is an interpretation of a $\underline{Q}$-language, and w is any member of W.)

3.1. **Basic laws of the descriptions**

3.1.1. The laws of the descriptions in the $\underline{Q}$ systems are based on the following:

Lemma. For all assignments V, $V(\underline{I}x.f)_w = U$ iff

$$V(\exists x(f \,\&\, \forall y(f(y/\mathbf{x}) \supset y = x)))_w \neq 1.$$

(Here x occurs free in f, and y does not occur in f.)

The formula " $\exists x(f \,\&\, \forall y(f(y/x) \supset y = x))$" may be called the existence formula of the description "$\underline{I}x.f$"; in what follows we shall abbreviate it to "$(\underline{I}x.f)^0$". Note that it cannot be replaced generally by " $\exists x.\forall y(f(y/x) \equiv y = x)$, for the latter may be true and the former false in cases when f itself involves some descriptions.

3.1.2. Definitions. An occurrence of a description in a formula f is said to be outermost if it does not belong to the scope of a descriptor $\underline{I}$ in f. – By the descriptions of a formula f we mean those descriptions occurring in f which have some outermost occurrences in f.

For all formulae f, we define the first description derivative of f – denoted by "$f^{(1)}$" – by induction as follows:

(i) If f is atomic, and no descriptions occur, in f, then $f^{(1)} = \underline{SE}^0$. (Note that $\underline{SE}^0$ is $\underline{QK}$-valid.)

(ii) If f is atomic, and $t_1, \ldots, t_n$ are all the descriptions of $f (n \geqslant 1)$, then $f^{(1)} = t_1^0 \,\&\, \ldots \,\&\, t_n^0$(cf. 3.1.1).

(iii) $(\sim f)^{(1)} = (\underline{S}f)^{(1)} = (\underline{M}f)^{(1)} = f^{(1)}$.

(iv) $(f \supset g)^{(1)} = f^{(1)}$ if $g^{(1)} = \underline{SE}^0$. In other cases:

$$(f \supset g)^{(1)} = \begin{cases} g^{(1)} & \text{if } f^{(1)} = \underline{SE}^0, \\ f^{(1)} \,\&\, g^{(1)} & \text{otherwise.} \end{cases}$$

(v) $(\forall x.f)^{(1)} = (\exists x.f)^{(1)} = f^{(1)}$ if $f^{(1)}$ involves no free occurrences of x. In the contrary case, $(\forall x.f)^{(1)} = (\exists x.f)^{(1)} = \exists x.f^{(1)}$

Obviously, if f involves no descriptions, then $f^{(1)} = \underline{SE}^0$.

For all $n \epsilon \omega$, we define $f^{(n)}$ as follows:

$$f^{(0)} = f, \; f^{(n+1)} = (f^{(n)})^{(1)}.$$

Note that if for some $n, f^{(n)} = \underline{SE}^0$, then for all $m \geqslant n, f^{(m)} = \underline{SE}^0$.

By the description degree of a formula f – denoted by "$dgr(f)$" – we mean the smallest $n \epsilon \omega$ such that $f^{(n+1)} = \underline{SE}^0$. Noting that if f involves descriptions then the number of occurrences of $\underline{I}$ in $f^{(1)}$ is smaller than in f, we conclude that $dgr(f)$ is finite, and uniquely defined. If f involves no descriptions, then, obviously, $dgr(f) = 0$.

3.1.3. Theorems.

(i) If $V(f^{(1)})w \neq 1$, then $V(f)_w = 2$.

(ii) (Corollary of (i).) For all sentences f, "$f \supset f^{(1)}$" is $\underline{QK}$-irrefutable.

(iii) If $\exists x.f$ is closed but f is open, then

$$\forall x((f \,\&\, \forall y(f(y/x) \supset y = x)) \underline{\equiv} . \, x = \underline{I}x.f)$$

is $\underline{QK}$-irrefutable.

3.2 **Statability atoms**

For all parameters p, we define a formula denoted by "p^s" as follows:

$$\forall^s = \forall x.x = x,$$

if a is an in-parameter, then $a^s =$ "$a = a$",

if p is a pred$_n$-parameter, then:

$$p^s = \text{"} \exists x.px^n \supset \exists x.px^{n}\text{"} \quad \text{if} \quad n > 0$$

where

$$x^1 = x, \text{ and } x^{n+1} = x^n x, \text{ and}$$

$$p^s = \text{"}p \supset p\text{"} \quad \text{if} \quad n = 0.$$

These formulae will be called statability atoms, briefly: S-atoms.

Lemmas. (i) For all parameters p: $\varphi(p^s)_w \neq 0$; and $\varphi(p^s)_w = 2$ iff p is degenerate at w. – (ii) $V(f)_w = 2$ iff either $V(f^{(1)})_w \neq 1$, or V pre-degenerates f at w. – (iii) For all closed formulae f: $\varphi(f)_w = 2$ iff either $\varphi(f^{(1)})_w \neq 1$, or for some parameter p of f, $\varphi(p^s)_w = 2$.

The last result is the motivation for the expression 'statability atom'. The statability of a sentence at w depends on the truth of its first description derivative, and on the truth of the statability atoms associated with its parameters. If a sentence involves no descriptions, then its statability depends exclusively on its statability atoms.

Theorem. For all sentences f, the following sentences are $\underline{Q}K$-irrefutable:

$\underline{S}f \supset \underline{S}p^s$ provided p is a parameter of f;

$$\left.\begin{array}{l}\underline{S}f \supset \underline{N}f^{(k)} \\ \underline{S}f \supset \underline{L}f^{(k)}\end{array}\right\} \quad \text{provided } k > 0.$$

4. § **Relevance**

(For the remainder part of this paper, f, g, and h stand for sentences.)

4.0 **Preliminaris**

The sentences $\forall^s$ and "$\forall^s \supset \underline{E}^0$" are $\underline{Q}K$-irrefutable, but $\underline{E}^0$ is $\underline{Q}K$-refutable. Similarly, "$t = t$" and "$t = t \supset (t = t)^{(1)}$" are $\underline{Q}K$-irrefutable, but if $t = \text{"}\underline{I}y. \sim .y = y\text{"}$, then

$$(t = t)^{(1)} = \text{"}\exists y(\sim .y = y \;\&\; \forall x(\sim .x = x \supset x = y))\text{"}$$

is $\underline{Q}K$-refutable (and $\underline{Q}K$-unstatisfiable). – These examples show that datachment is not irrefutability-preserving in the $\underline{Q}$ systems (although it is truth-preserving). In this section we shall show (among others) that there is a restricted form of detachment which is irrefutability-preserving.

We saw in 2.5.8, (4) that "$\underline{N}(f \supset g) \supset \underline{N}f \supset \underline{N}g$" is not irrefutable in the $\underline{Q}$ systems E.g., the sentence "$\underline{N}(\forall^s \supset \underline{E}^0) \;\&\; \underline{M} \sim \underline{E}^0$" is $\underline{Q}J$-satisfiable (for all J). But if f and g satisfy certain "relevance condition", then the quoted schema is $\underline{Q}K$-irrefutable. To formulate this condition is another aim of this section.

4.1. **Primary relevance**

Definition. The relation "f and g are trivially equivalent sentences" is the smallest reflexive, symmetric, and transitive relation satisfying the conditions (i) and (ii):

(i) The members of the following pairs are trivially equivalent: f and "$\sim\sim f$"; f and "$\sim f \supset f$"; "$f \supset \sim g$" and "$g \supset \sim f$"; "$f \supset g \supset h$" and "$g \supset f \supset h$"; f and "$\exists x.f$"; f and "$\forall x.f$"; "$\exists x(f \,\&\, g(x/c))$" and "$f \,\&\, \exists x.g(x/c)$".

(ii) If f and g are trivially equivalent, and h_1 is the same as h except at most that an occurrence of f in h is replaced by g, then h and h_1 are trivially equivalent.

Definition. The relation "f trivially implies g" is the smallest transitive relation including the relation of the trivial equivalence and satisfying the conditions (i) to (iii):

(i) "$f \,\&\, g$" trivially implies f.

(ii) f trivially implies $f^{(1)}$ and "$\exists x.f(x/.c)$".

(iii) If f trivially implies g, then "$\exists x.f(x/c)$" trivially implies "$\exists x.g(x/c)$".

Definition. We say that f is primarily relevant to g iff $g^{(1)}$ trivially implies $f^{(1)}$.

Lemma. Assume that f is primarily relevant to g, and $I = \langle W,R,U,\varphi\rangle$ is an interpretation of "$f \,\&\, g$". Then: For all $w \epsilon W$, if φ does not pre-degenerate f at w, and $\varphi(f)_w = 2$, then $\varphi(g)_w = 2$.

4.2. **Secondary relevance**

We define the relation "f is secondarily relevant to g" by the stipulations (i) to (vii) as follows.

(i) f is secondarily relevant to f, $f^{(1)}$, "$a = b \,\&\, f(b/a)$", and "$c = \underline{I}x.f(x/c)$" (where c occurs in f).

(ii) If f is secondarily relevant to g, t is a closed description, and t is substitutable for c in f, then $f(t/c)$ is secondarily relevant to "$t^0 \,\&\, g$".

(iii) If f and g are secondarily relevant to h, then so are "$\sim f$" and "$f \,\&\, g$".

(iv) If f is secondarily relevant to g, then, for all h, f is secondarily relevant to "$g \,\&\, h$".

(v) If f is secondarily relevant to g, f_1 is one of $\forall x.f(x/c)$, $\exists x.f(x/c)$, and g_1 is one of $\forall x.g(x/c)$, $\sim \forall x \sim g(x/c)$, $\exists x.g(x/c)$, $\sim \exists x \sim g(x/c)$, then f_1 is secondarily relevant to g_1.

(vi) If f is secondarily relevant to g, and h is secondarily relevant to "$f \,\&\, g$", then $\exists x.h(x/c)$ is secondarily relevant to "$\exists x.g(x/c) \,\&\, f_1$" where f_1 is one of $\forall x.f(x/c)$, $\sim \exists x \sim f(x/c)$.

(vii) If f is primarily or secondarily relevant to g, f and f_1 are trivially equivalent, and g and g_1 are trivially equivalent, then f_1 is secondarily relevant to g_1.

Lemma. Assume that f is secondarily relevant to g. For all interpretations $<W,R,U,\varphi>$, and for $w \epsilon W$: If φ does not pre-degenerate f at w, and $\varphi(f)_w = 2$, then $\varphi(g)_w \neq 1$.

4.3. **Weak and strong relevance**

Definitions.

(i) A sentence f is said to be universe dependent (briefly: V-dependent) iff either it involves some descriptions and/or some in-parameters, or else V is a parameter of f.

It follows that if f is V-dependent, and $\varphi(V)_w = 0$, then $\varphi(f)_w = 2$. As an example let us mention that if $\exists x.f(x/c)$ is V-dependent, then "$\sim \exists x.f(x/c) \supset \exists x \sim f(x/c)$" is $\underline{QK}$-irrefutable (in the contrary case, it may be $\underline{QK}$-refutable).

(ii) The sentence f is said to be weakly relevant to the sentence g iff f is primarily or secondarily relevant to g, and, in addition, if f is V-dependent, then so is g.

An important corollary of the lemmas in 4.1. and 4.2. and the present definition: If f is weakly relevant to g, and φ does not pre-degenerate f at w, then $\varphi(f)_w = 2$ implies $\varphi(g)_w \neq 1$.

(iii) f is said to be parameter relevant (briefly: p-relevant) to g iff all parameters of f except perhaps V are parameters of g, and, in addition, if f is V-dependent, so is g.

(iv) f is said to be strongly relevant to g iff f is both weakly relevant and p-relevant to g.

(v) A sentence f is said to be weakly (strongly) relevant to a set α of sentences iff f is weakly (strongly) relevant to a conjunction of some members of α.

(vi) We say that $f(x/c)$ is weakly relevant to $g(x/c)$ iff one of the following two conditions is fulfilled:

(a) c occurs in g, and f is weakly relevant to g.

(b) c does not occur in g (i.e., $g(x/c) = g$), and $Vx.f(x/c)$ is weakly relevant to g.

Lemma. Assume that $I = <W,R,U,\varphi>$ is an interpretation satisfying a set α of sentences, i.e., for an $w \epsilon W$, $\varphi(h)_w = 1$ holds for all $h \epsilon \alpha$. Then: If f is weakly relevant to α, then there is an $I' = <W,R,U,\varphi'>$ such that I' is an interpretation of $\{\alpha, f\}$, for all $h \epsilon \alpha$, and for all $v \epsilon W$, $\varphi'(h)_v = \varphi(h)_v$, and $\varphi'(f)_w \neq 2$. – (The proof exploits the fact that if f is V-dependent, then so are some members of α.)

Theorems. (i) If f and "$f \supset g$" are $\underline{QJ}$-irrefutable sentences, and f is weakly relevant to "$\sim g$", then g is $\underline{QJ}$-irrefutable. – (ii) If f is strongly relevant to "$\sim g$", then $\underline{N}(f \supset g) \supset \underline{N}f \supset \underline{N}g$

is $\underline{QK}$-irrefutable. – (iii) If g is strongly relevant to f,

then $\underline{N}(f \supset g) \supset \underline{M}f \supset \underline{M}g$

is $\underline{QK}$-irrefutable.

§5 The $\underline{Q}$ calculi

Using the definitions of §3 and §4, we are able to formulate the syntactical ("axiomatized") versions of the semantical systems $\underline{QJ}(\underline{J} = \underline{K},\underline{D},\underline{T},\underline{B},\underline{4},\underline{5})$ introduced in §2. We shall define the notions "f is a $\underline{QJ}$-axiom", "f is $\underline{QJ}$-deducible", and "f is $\underline{QJ}$-deducible from the set α of sentences".

5.1. Axioms

The set of $\underline{QJ}$-axioms will be defined by means of $\underline{QJ}$-axiom schemata and the notion of generalization.

5.1.1. By the set of the generalizations of the sentence f we mean the smallest set $G(f)$ such that (i) $f \epsilon G(f)$, and (ii) if $g \epsilon G(f)$, then $\forall x.g(x/c) \ \epsilon G(f)$, $\sim \exists x \sim g(x/c) \epsilon G(f)$, and $\underline{N}g \epsilon G(f)$.

5.1.2. Axiom schemata for all $\underline{Q}$ calculi:

(A1) $f \supset g \supset f$

(A2) $(f \supset g \supset h) \supset (f \supset g) \supset f \supset h$

(A3) $(\sim f \supset \sim g) \supset g \supset f$

(A4.1) $\forall x.f(x/c) \supset f(t/c)$

(A4.2) $f(t/c) \supset \exists x.f(x/c)$

Proviso for (A4.1) and (A4.2): t is a closed term substitutable for c in f.

(A5.1) $\forall x(f(x/c) \supset g(x/c)) \supset \forall x.f(x/c) \supset \forall x.g(x/c)$

(A5.2) $\sim \exists x \sim (f(x/c) \supset g(x/c)) \supset \exists x \sim g(x/c) \supset \exists x \sim f(x/c)$

Proviso for (A5.1) and (A5.2): $f(x/c)$ is weakly relevant to $\sim g(x/c)$.

(A6.1) $\forall x(f \supset g(x/c)) \supset f \supset \forall x.g(x/c)$

(A6.2) $\sim \exists x \sim (g(x/c) \supset f) \supset \exists x.g(x/c) \supset f$

(A7) $c = c$

(A8) $a = b \supset f(a/c) \supset f(b/c)$

(A9) $(f \ \& \ \forall x(f(x/c) \supset x = c)) \underline{\equiv} . \ c = \underline{I}x.f(x/c)$

(provided c occurs in f)

(A10) $\underline{N}f \supset \underline{M} \sim g \supset \underline{S}h \supset \underline{M} \sim (f \supset g)$

provided f is strongly relevant to "$\sim g \;\&\; (h \supset h)$".

(A11) $\underline{N}(f \supset \underline{E}^0) \supset \underline{M}f \supset \underline{M}(f \;\&\; V^s)$

(A12) $\underline{SE}^0$

(A13) $\sim \underline{S}V^s \supset (\sim \underline{S}.c = c \;\&\; \underline{M} \sim \underline{E}^0)$

(A14) $\underline{S}f \supset \underline{L}f^{(1)}$

(A15) $\underline{S}f \supset \underline{S}p^s$ provided p is a parameter of f

(A16) $\underline{L}f^{(1)} \supset \underline{S}p_1^s \supset \ldots \supset \underline{S}p_n^s \supset \underline{S}f$ where $p_1, \ldots p_n$

are all the parameters of f (if any); $n \geqslant 0$

5.1.3. Additional axiom schemata for the single $\underline{Q}$ calculi.

For $\underline{QK}$:

(A17) $\underline{M}.a = b \supset a = b$

(A18.K) $\sim \underline{S}p^s \supset \underline{MSE}^0$

For $\underline{QD}$: (A17) above, and

(A18.D) $\underline{MSE}^0$

For $\underline{QT}$: (A17) above, and

(A18) $f \supset \underline{M}f$

For $\underline{QB}$: (A18) above, and

(A19.B) $f \supset \underline{NM}f$

For $\underline{Q4}$: (A17) and (A18) above, and

(A19.4) $\underline{Z}f \supset \underline{NZ}f$ where $\underline{Z}$ is $\underline{N}$ or $\underline{S}$

For $\underline{Q5}$: either (A18), (A19.B), and (A19.4), or else (A18), and (A19.5) below.

(A19.5) $\underline{MZ}f \supset \underline{Z}f$ where $\underline{Z}$ is $\underline{N}$ or $\underline{S}$

5.1.4. A sentence f is said to be a $\underline{QJ}$-axiom, iff f is a generalization of an instance of a $\underline{QJ}$-axiom schema.

5.2. **Deductions**

5.2.1. A sequence $<f_i>_{i<n}$ $(0 < n\epsilon\omega)$ of sentences is said to be a $\underline{QJ}$-deduction iff for all $i < n$, either f_i is a $\underline{QJ}$-axiom, or f_i is obtained from a preceding member by $\underline{N}$-generalization in the sense that for some $j < i$, $f_i = \underline{N}f_j$, or else f_i is obtained from preceding members by relevant datachment in the sense that for some $j,k < i$, $f_k =$ "$f_j \supset f_i$" where f_j is weakly relevant to "$\sim f_i$".

A sentence f is said to be $\underline{QJ}$-deducible iff there exists a $\underline{QJ}$-deduction terminated by f. (By this, any $\underline{QJ}$-axiom is $\underline{QJ}$-deducible.)

5.2.2. Let α be a (possible empty) set of sentences. A sequence $<f_i>_{i<n}$ $(0 < n\epsilon\omega)$is said to be a QJ-deduction from α iff for all $i < n$, either $f_i \epsilon \alpha$, or f_i is QJ-deducible (in the sense of 5.2.1), or else f_i is obtained from preceding members by α-relevant detachment in the sense that for some $j,k < i$, f_k = "$f_j \supset f_i$" where f_i is weakly relevant to $\{\alpha, \sim f_i\}$. – Let us note that any QJ-deduction (in the sense of 5.2.1) is a QJ-deduction from the empty set.

A sentence f is said to be QJ-deducible from α iff there is a QJ-deduction from α terminated by f.

5.3. **Soundness and completeness**

The following theorems are provable (for all J):

If f is QJ-deducible from α, then f is a QJ-consequence of α. (The case $\alpha = 0$ is included.)

If f is a QJ-consequence of α, then f is QJ-deducible from α (including the case $\alpha = 0$).

The sentence h is QJ-deducible from the set $\{\alpha, f\}$ iff "$f \supset h$" is QJ-deducible from α. (Deduction theorem and its converse.)

REFERENCES

[1] Bull, R.A., An axiomatization of Prior's modal calculus Q. Notre Dame Journal of Formal Logic 5 (1964), 211-214.

[2] Frege, G., On sense and reference. In: Geach, P., and Black, M. (eds.), Translatios from the Philosophical Writings of Gottlob Frege. Oxford, (1960). 56-78.

[3] Kripke, S.A., Semantical analysis of modal logic, I. Normal modal propositional calculi. Zeitschrift für math. Logik und Grundlagen der Math. 9 (1963), 67-96.

[4] Kripke, S.A., Semantical considerations on modal logic. Acta Philosophica Fennica 16 (1963), 83-94.

[5] Prior, A.N., Time and Modality. Oxford, 1957.

[6] Prior, A.N., Papers on Time and Tense. Oxford, (1968).

[7] Prior, A.N., Axiomatizations of the modal calculus Q. Notre Dame Journal of Formal Logic 5 (1964), 215-217.

[8] Ruzsa, I., Prior-type modal logic, I-II. Periodica Mathematica Hungarica 4 (1973), 51-69, 183-201.

[9] Ruzsa, I., Modal logic with descriptions. (To appear.)

INTENSIONAL LOGIC OF ACTIONS

P. Ecsedi – Tóth

Research Group on Mathematical Logic and Automata Theory,
The Hungarian Academy of Sciences, 6720 Szeged, Somogyi u. 7.

0. Introduction

The problem of defining the exact semantics for a given programming language L_p (called the main problem in this paper) can be formalized in the following way: find a suitable language L, the semantics of which are completely understood, and translate each of the formulae (i.e. instructions of programs) of L_p into L in a "meaning preserving" way, that is give a function τ with some nice properties from L_p into L. The known semantics of L will define the semantics of L_p along the line of the translation function τ. We observe that this formalization does not mean any loss of generality; i.e. every known method for defining semantics can be compiled in this form.

The problem of finding a language L, appropriate for the purposes mentioned above, plays a central role in any trial to cope with the main problem. Usually three types of languages are chosen: spoken languages, computer languages and mathematical ones. The use of natural languages for L was entailed by the lack of adequate tools in the heroic age of computer science. Also, concrete programing languages, recognized as well-known ones, cannot be suitable for our aims. Even the more advanced computer languages over "abstract machines" (such as VDL) give rise to an almost insurmountable difficulty by defining the semantics of L_p in a completely implicit way. Both these types of languages have no well-understood semantics so that, on the one hand spoken languages and computer languages are inadequate from a semantical point of view.

The reason for solving the main problem is not in merely knowing the meaning of the programs, but it is also in knowing their properties. To this end, we want to derive certain properties of programs from other ones; in one word we want to develop a proof-theory for L. It is obvious that there is no hope of giving a precise definition of deduction in a non--mathematical language, so on the other hand these languages are not adequate from a proof--theoretical point of view.

Consequently, we can conclude that the only adequate way is to choose a mathematical language for L. This approach has a further advantage, namely that it can utilise existing results from mathematics (e.g. from mathematical logic, universal algebra, category theory, graph-theory, topology, lattice theory, model theory, topos-theory etc.).

Some particular cases of this approach have already been investigated ([12, 13, 14, 15]), while others have not been considered at all.

In this paper we shall present a mathematical logical system called the Intensional Logic of Actions (ILA, for short) which seems to be an appropriate frame within which to attack the main problem.

The use of ILA was motivated by several things, amongst them the desire for a simple translation function from L_p into L. The basic idea of the logic of actions (due to J. McCarthy and P.J. Hayes [8]) is as follows: consider a set of classical algebraic structures each of which is a variant of the "world" to be described (for example, one can imagine that this world is the memory of a computer). These structures are called "states" or "situations" (cf. [5, 17]). Programs (i.e. plans for actions) are treated as partial (in the original work of P.J. Hayes [5] as total) functions among states. Then, construct the language L in such a way that programs would be <u>terms</u> in it and models of this language are the same as the ones described above. Therefore, programs can be named in the logic obtained, and one can answer such questions as:

– Is the program a equivalent to program b?

– Does a program a exist such that the following conditions hold . . . ?

– For every program a such that the following conditions are satisfied . . . we can conclude that . . . .

However, as experiences of applications to concrete programming languages show in "classical" logic of actions we need an additional auxiliary function VALUE (cf. [2]). (The use of the function VALUE was suggested to me in 1974 by B. Dömölki in personal communications).

After a detailed investigation it turns out that the function VALUE plays a vital role in any cases when the logic of actions is applied to attack the main problem. Then, I try to find an answer to the question of what the function VALUE means from a mathematical logical (model theoretic) point of view. After reading the significant papers of R. Montague [9, 10, 11] dealing with the semantics of logic of intensions and its applications in linguistics, I made the discovery that the function VALUE is nothing but a special case of the so called intension function, i.e. it defines intensions of expressions.

The fact that programming languages are essentially intensional in character can be justified by other observations, as well. For example I only mention two of them:

i.) In almost every programming language the use of different names (identifiers) for the same expression is allowed. For example let us consider the following BASIC program:

```
 10   LET     X = 5
 20   LET     Y = 2
 30   LET     Z = X * Y + X
 40   LET     U = X * Y + X
        .
        .
        .
100   LET     X = Y + 1
110   GOTO    40
```

It is not clear from a mathematical logical point of view whether $Z = U$ or not. They are literally equal, but their values are different in different states of computation!

ii) Also, we can write in almost every programming language that

```
LET     X = X + 1
```

If one wants to understand properly the meaning of the "equation" $X = X + 1$, then one needs to refer to the value of x both in the previous and in the actual states (of the "world").

The main difference between the classical logical approach and the intensional one is in the way of interpreting constants. In the classical approach constants are associated with elements of an appropriate domain as indicated by Fig 1;

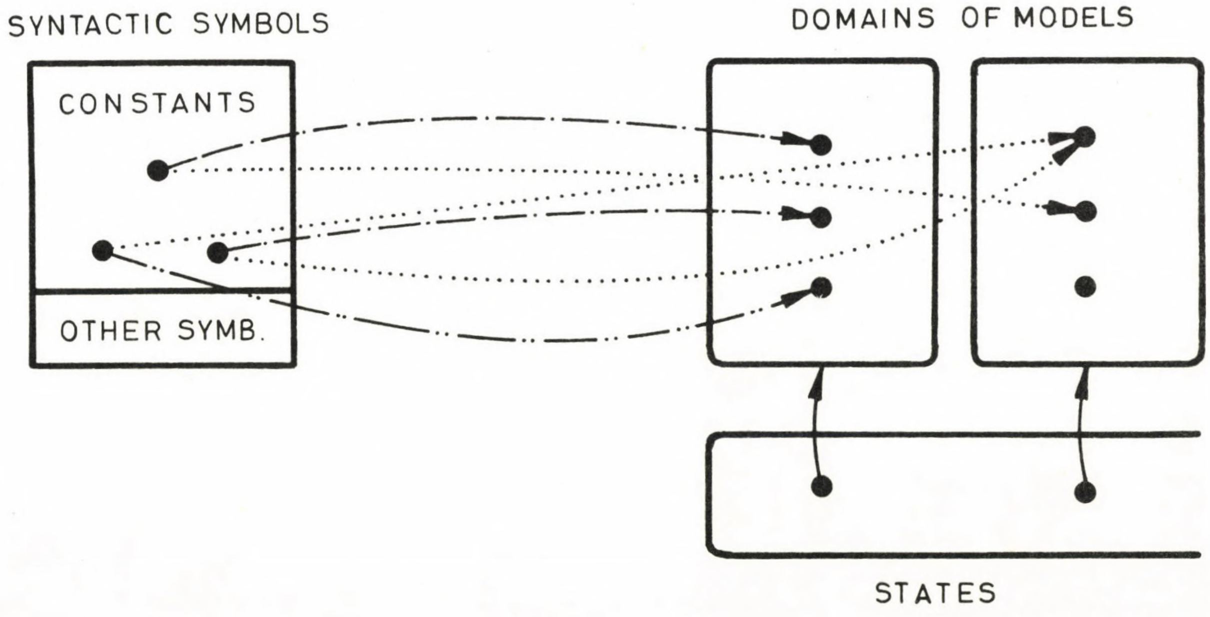


Fig. 1.

In intersional logic constants are interpreted as functions from states into domains (Fig. 2.).

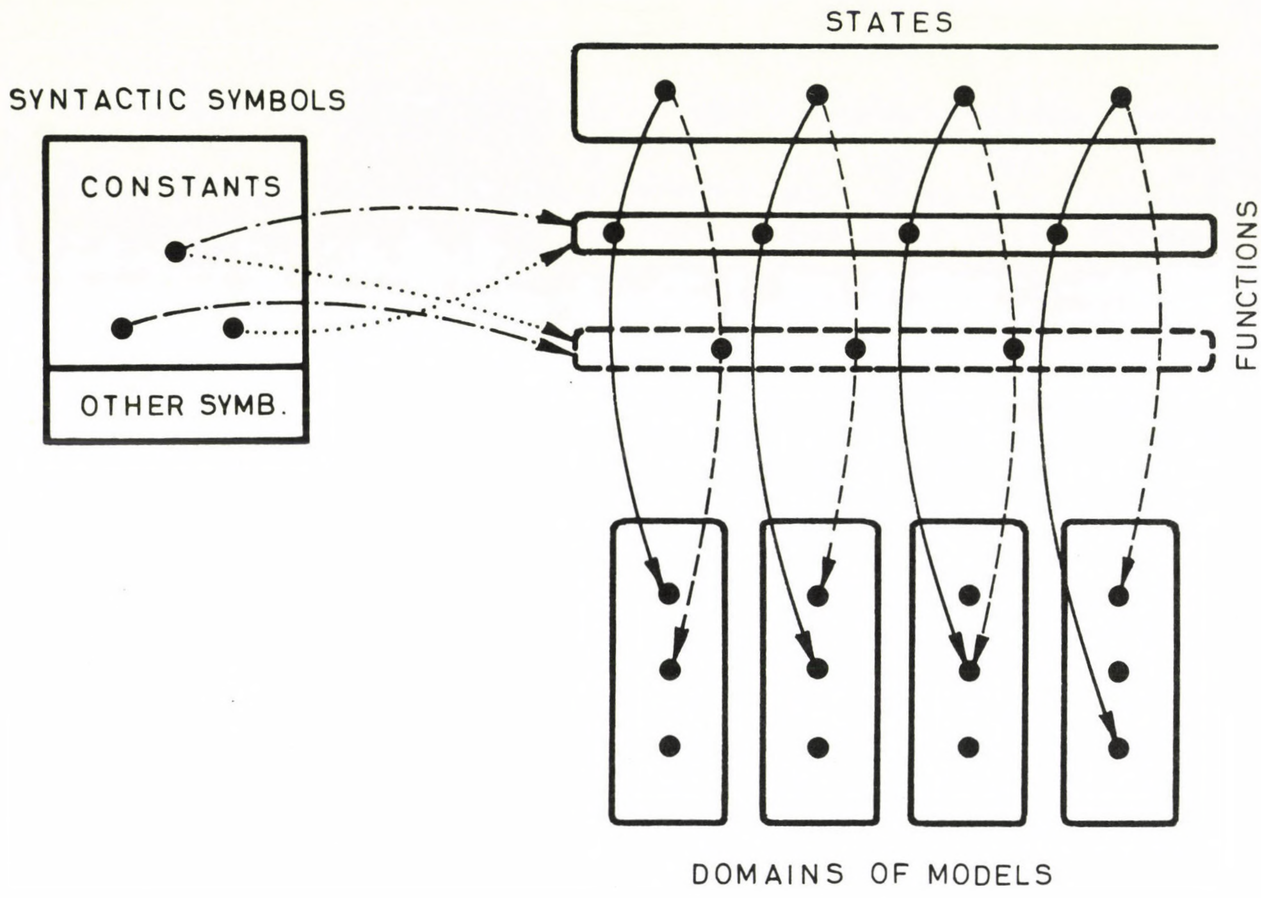


Fig. 2.

In Sections 1 and 2 we develop the syntax of ILA. Semantics are given in Section 3. Some aspects of model-theoretical investigation of ILA and a proof for the Generalized Completeness Theorem will appear in a subsequent paper [3].

Time and space does not allow me to insert a detailed explanation of concrete applications; however, a future paper is planned to deal with these problems.

Acknowledgement: I am indebted to H. Andréka, B. Dömölki, T. Gergely, I. Németi and many other collegues for their encouragement in my work on any form of logic of actions.

Remark

After finishing this paper I discovered that Janssen and van Emde Boas developed a theory [6,7] very closely related to ours. However, their system differs in some relevant aspects from the material presented in this article:

i.) One of the advantages of ILA is that states, and therefore programs, can have names. Janssen and van Emde Boas do not insert such names into their system.

ii.) Technical presentation of ILA is much more simpler than that of the system of Janssen and van Emde Boas, thanks partially to the presence of names for states, and partially to the formalization used.

1. Types

Keeping in view that our main problem is to define semantics of programming languages, we introduce the notion of types as syntactical counterparts of data-types. Types are intended to show the "sort" of things: i.e. if they are digits, integers, lists, trees, sets, classes, variables, operators, expressions, Boolean-expressions, etc. Note that data-types are partially ordered [15]: e.g. digits are integers, sets are classes, lists are trees, etc. .

Definition 1.

Let $<s_0, \leqslant_0>$ be a finite, non-void, partially ordered set, and let s be an arbitrary symbol which is not an ordered n-tuple for any $n\epsilon\omega$. The set of types $<T, \leqslant>$ is the least set defined by:

i.) $S_0 \subset T$, $s\epsilon T$;

ii.) If $t_0, t_1, \dots, t_n \epsilon T$, then $<t_0, \dots, t_n> \epsilon T$;

iii.) If $t_0, t_1 \epsilon S_0$, then $t_0 \leqslant t_1$ iff $t_0 \leqslant_0 t_1$;

iv.) If $t_0 = <t_{00}, t_{01}, \dots, t_{0n_0}> \epsilon T$ and

$t_1 = <t_{10}, t_{11}, \dots, t_{1n_1}> \epsilon T$, then

$t_0 \leqslant t_1$ iff $t_{00} \leqslant t_{10}, t_{01} \leqslant t_{11}, \dots, t_{0n_0} \leqslant t_{1n_1}$.

Objects of type in S_0 will be different sorts of individuals, objects of type s will be states, objects of type $<t_0, \dots, t_n>$ will be predicates, the first argument of which is of type $t_0, \dots,$ the $(n+1)$-th argument of which is of type t_n, respectively.

Note

Predicates and relations are distinguished in this context: relations are among things, predicates (relations-in-intension) are on the names of things (or more precisely on functions from names into things that is, predicates are among "intensions" of things). Note also that functions are treated as special binary relations.

2. **Syntax**

First we have to fix the set of primitive symbols. We shall use the logical symbols: equality (≡), negation (¬), conjunction (∧) disjunction (V), implication (→), equivalence (↔), universal- (∀) and existential quantifications (∃). We use ≡, ¬ , ∧ , ∀ as primitives, while V, →, ↔ and ∃ will be introduced as abbreviations. As an additional symbol we use a symbol for abstraction ({ | }). For motivation see the point 5.

Definition 2

The set of primitive symbols PS is defined by

i.) For every $t \epsilon T$,

$$\{ x_0^t, \dots, x_n^t, \dots, y_0^t, \dots, y_n^t, \dots \} = V^t \subset PS$$

We refer to x_i^t as a function-variable of type t, and to y_i^t as a predicate-variable of type t.

ii.) For every $t \epsilon T$,

$$\{ c_0^t, \dots, c_n^t, \dots, d_0^t, \dots, d_n^t, \dots \} = {}^t \subset PS.$$

We refer to c_i^t as a function-constant (symbol) and to d_i^t as a predicate-constant symbol of type t.

iii.) $\{ \equiv, \neg, \wedge, \forall, \{ | \}, [\ ,\ ] \} \subset PS.$

Definition 3 Formulae F, terms TR, free and bound variables

are defined in a parallel way by:

i.) If α is a function-variable or a function-constant of type $\ll t_0, t_1, \dots, t_n >, t>$

and a_i is a term of type $\leqslant t_i$ for $i \leqslant n$, then $[\alpha a_0 a_1 \dots a_n]$ is a term of type t;

and each occurence of a variable is free or bound as it was in $a_0, a_1, \dots, a_n$.

ii.) If both a_0, a_1 are terms of type $\leqslant t$ (for an arbitrary $t \epsilon T$), then $[a_0 \equiv a_1]$ is a formula, and each occurence of a variable is free or bound as it was in a_0, a_1.

iii.) If α is a predicate-variable or a predicate constant of type $<t_0,t_1, \ldots, t_n>$, and a_i is a term of type $\leqslant t_i$ for $i \leqslant n$, then $[\alpha a_0 a_1 \ldots a_n]$ is a formula, and each occurence of a variable is free or bound just as in $a_0, a_1, \ldots, a_n$. If α is a predicate variable, then its occurence is free in the formula $[\alpha a_0 \ldots a_n]$.

iv.) If φ and ψ are formulae, then $[\neg \varphi]$ and $[\varphi \wedge \psi]$ are also formulae, and each occurence of a variable is free or bound as it was free or boound in φ or ψ.

v.) If ν is a variable and φ is a formula, then $[\forall \nu\varphi]$ is also a formula in which each occurence of ν is bound, other variables are free or bound as they were in φ.

vi.) If ν is a variable of type t and φ is a formula then $\{\nu|\varphi\}$ is a term of type t and every occurence of ν is bound, other variables are tree or bound just as in φ.

As ILA is intended to serve as a frame for definitions of semantics of programming languages, we shall need some distinguishable constant symbols as well. These are a symbol for dependence-relation of objects ($\rightrightarrows$) and a symbol for naming states (δ) [5,2]. The intended meaning of $[\rightrightarrows a,b,c]$ is that object a depends on b in the state $\subset$ (c.f. Definitions 4,5,13), and by $[\delta\ b\ a\ ]$ we mean the state obtained from the state b if action a is executed (c.f. Definitions 6,7,13).

Definition 4

Let $t = <t_0,t_1,s>$ where $t_0 \in S_0$ $t_1 \in S_0 \cup \{<s,s>\}$. Then $\rightrightarrows$ is a new predicate-constant ($\rightrightarrows \notin \bigcup_{t \in T} \mathfrak{S}^t$) and has the type t where t is an element of the set

$$\{<t_0,t_1,s> : t_0 \in S_0, t_1 \in S_0 \cup \{<s,s>\}\} \quad .$$

Definition 5

If a_i is of type $\leqslant t_i$ $i = 0.1$ and a is of type s, then $[\rightrightarrows a_0 a_1 a]$ is a formula, and each occurence of a variable is just as it was in a_0,a_1,a.

Definition 6

Let $t = \ll s,<s,s\gg,s>$. Then δ is a function-constant of type t, other then

$$c_0^t, c_1^t, \ldots, c_n^t, \ldots \quad .$$

Definition 7

If a_0 is a term of type s, and a_1 is a term of type $<s,s>$, then $[\delta a_0 a_1]$ is a term of type s. Every occurence of a variable is just as it was in a_0,a_1.

Definition 8

φ is a <u>sentence</u> if no free variables occur in φ.

3. Semantics

Definition 9

For every $t \epsilon S_0$, let D^t be a non-empty set, and let D^s be non-void as well. By a <u>T-frame</u> we shall mean an indexed family of sets $<M_t>_{t \epsilon T}$ such that:

i.) $M_t = D^t$ if $t \epsilon S_0$ or $t = s$.

ii.) For each $t = <t_0, \ldots, t_n> \epsilon T$,

$M_t \subset D^s(\mathcal{P}(M_{t_0} x \ldots x M_{t_n}))$ and $M_t \neq \emptyset$. Where A_B is the set of <u>partial</u> functions from A into B, $\mathcal{P}$ is the symbol of powerset-construction.

Definition 10

A <u>T-model</u> is an indexed family of sets

$\mathfrak{M}^T = <M_t, m_t>_{t \epsilon T}$ where

i.) $<M_t>_{t \epsilon T}$ is a T-frame.

ii.) $m_t \epsilon\ {}^t\mathfrak{G}\, M_t$

An example of a T-model is shown in Fig.3.

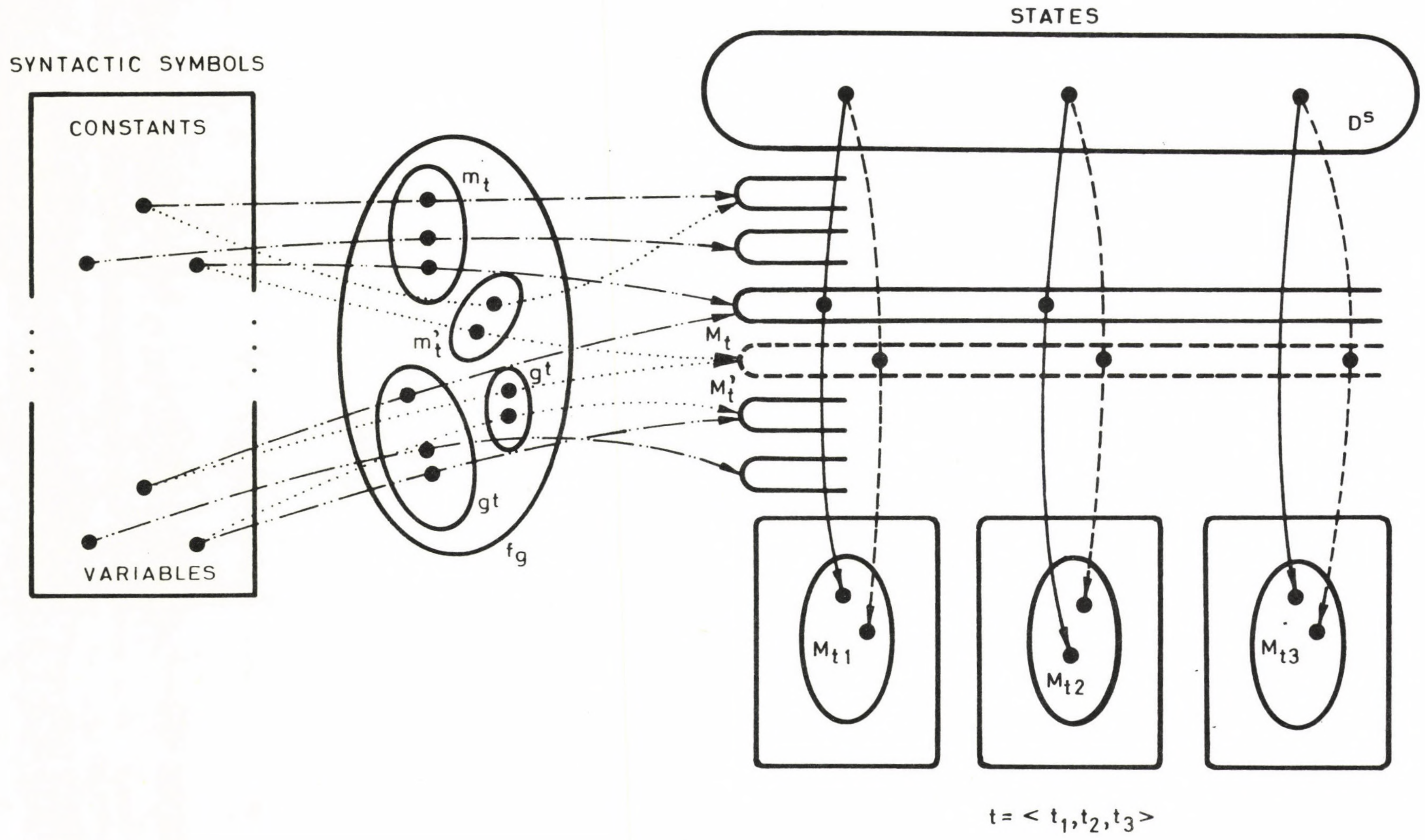


Fig. 3.
A LITTLE PIECE OF A T-MODEL.

Definition 11

The set of assignments on a fixed T-model $\mathfrak{M}^T$ is:

As $(\mathfrak{M}^T) = \{ <g^t>_{t\in T} : g^t \in V^t_{M_t}$ for each $t \in T \}$

Definition 12

Let a T-model $\mathfrak{M}^t$ and on assignement $g = <g^t>_{t\in T}$ be given. The interpretation on $\mathfrak{M}$ defined by g is a partial function f_g such that

$$f_g : TR \to \bigcup_{t\in T} M_t \quad \text{and}$$

$$f_g \supseteq m_t, \; f_g \supseteq g^t \text{ for every } t\in T.$$

(We call that the set theoretic meaning of $f_g \supset g^t$ is that f_g coincides with g^t on the domain of g^t.) On the rest of the elements of TR f_g is the standard extension * of the function defined so for (i.e. of the function $\bigcup_{t\in T} (g^t \cup m_t)$). Note that f_g is partial on TR.

Definition 13

Let a T-model $\mathfrak{M}^T$ and an assignment g be given. We define the forcing relation

$\Vdash_{f_g} \subset D^s \times F$ by:

i.) $d \Vdash_{f_g} [r\ a_0 \ldots a_n]$ iff

$<f_g(a_0)(d), \ldots, f_g(a_n)(d)> \in f_g(r)(d)$

ii.) $d \Vdash_{f_g} [a_1 \equiv a_2]$ iff $f_g(a_1)(d) = f_g(a_2)(d)$.

iii.) $d \Vdash_{f_g} \neg \varphi]$ iff $d \Vdash_{f_g} \varphi$

iv.) $d \Vdash_{f_g} [\varphi \wedge \psi]$ iff $d \Vdash_{f_g} \varphi$ and $d \Vdash_{f_g} \psi$.

v.) $d \Vdash_{f_g} [\forall v \varphi]$ iff for every g' which are defined by $g'(z) = g(z)$ if $z \neq v$ we have $d \Vdash_{f_g'} \varphi$.

vi.) If a is the term $\{v_t \mid \varphi\}$ and there exists a unique member m of M_t such that $d \Vdash_{f_g} \varphi$ iff $f_g(v_t) = g(v_t) = m$, then a is called interpretable in d and

* The $\{\mid\}$ step of this extension is defined in Def. 13 vi. The $[\alpha\, a_0 \ldots a_n]$ part, where α is a function constant or variable is really standard. The $[\delta\, a_0 a_1]$ step is not defined since $\delta \in \mathfrak{S}^{t^n}$.

$f_g(a) = m$. If there are more such elements $m \epsilon M_t$ or there is no one, then f_g is undefined on the argument a; i.e. $f_g(a)$ is undefined. (Recall that f_g is partial.)

As this definition shows, it may happen that some terms have no interpretation in an element of D^s.

In an analogous way, one can define the meaning of $\rightrightarrows$ and δ. We leave this to the reader.

Definition 14.

Let a T-model $\mathfrak{M}^T$ and an interpretation f_g be fixed. We define for $\varphi \epsilon F$

$$\underline{\mathfrak{M}^T \models_{f_g} \varphi} \quad \text{iff for every} \quad d \epsilon D^s, \; d \Vdash_{f_g} \varphi.$$

We say that φ holds in $\mathfrak{M}^T$ under interpretation f_g iff $\mathfrak{M}^T \models_{f_g} \varphi$. $\mathfrak{M} \models \varphi$ iff for every $g \epsilon Ass(\mathfrak{M})$ we have $\mathfrak{M} \models_{f_g} g$. Note that if φ has no free variables then φ holds in $\mathfrak{M}^T$ under any interpretation iff it holds under every interpretations.

Definition 15.

Let φ be a sentence. φ is <u>valid</u> iff for each T-model $\mathfrak{M}^T$, we have $\mathfrak{M}^T \models \varphi$.

Definition 16.

Let $\mathfrak{M}^T$ be a T-model and let an interpretation f_g be fixed. For $\Sigma \subset F$, we define

$$\mathfrak{M}^T \models_{f_g} \Sigma \quad \text{iff} \quad \mathfrak{M}^T \models_{f_g} \varphi \; \text{for every} \; \varphi \epsilon \Sigma.$$

Definition 17

Let $\varphi \epsilon F$ $(\Sigma \subset F)$. $\varphi(\Sigma)$ is <u>T-satisfiable</u> if there exists a T-model $\mathfrak{M}^T$ such that $\mathfrak{M}^T \models \varphi$ ($\mathfrak{M}^T \models \Sigma$).

Definition 18.

We can use the following abbreviations:

i.) $[\varphi \vee \psi]$ for $[\neg[[\neg \varphi] \wedge [\neg \psi]]]$

ii.) $[\varphi \rightarrow \psi]$ for $[\neg[\varphi \wedge [\neg \psi]]]$

iii.) $[\varphi \leftrightarrow \psi]$ for $[[\varphi \rightarrow \psi] \wedge [\psi \rightarrow \varphi]]$

iv.) $[\exists \; v\varphi]$ for $[\neg[\forall v[\neg\varphi]]]$.

To increase readability one can omit brackets if no confusion occurs.

References

[1] Chang, C.C., H.J. Keisler. Model Theory, North-Holland Publishing Company, Amsterdam (1973).

[2] Ecsedi-Tóth, P., A mathematical logical approach to definition of semantics for programming languages, Master Thesis, Rolando Eötvös University, Budapest, (1974) (in Hungarian).

[3] Ecsedi–Tóth, P., Model theory for intensional logic of actions (in preparation).

[4] Gallin, D., Intersional and higher-order model logic, North-Holland Mathematics Studies, Vol. 19., North-Holland Publishing Co., Amsterdam (1975).

[5] Hayes, P.J., A logic of actions, Machine Intelligence 6 (eds. B. Meltzer and D. Michie), Edingburgh University Press (1971) pp. 495-520.

[6] Janssen, T.M.V., van Emda Boas, On the proper treatment of referencing, dereferencing and assignment, Proc. 4-th ICALP Conf. Lécture Notes in Comp. Sci. Vol. 52 (eds. A. Salomaa and M. Steinby), Springer Verlag, Berlin (1977) pp. 282-300.

[7] Janssen, T.M.V., van Emde Boas, The expressive power of intensional logic in the semantics of programming languages, Proc. 6-th MFCS Symp. Lecture Notes in Comp. Sci. Vol. 53. (ed. J. Gruska), Springer Verlag, Berlin (1977) pp. 303-311.

[8] McCarthy, J., Hayes, P. J., Some Philosophical Problems from the Standpoint of Artificial Intelligence, Machine Intelligence 4. (eds. B. Meltzer and D. Michie), Edinburgh Univ. Press. (1969) pp. 463-502.

[9] Montague, R., Pragmatics and intensional logic. Synthese, 22 (1970) pp. 68-94.

[10] Montague, R., Universal grammar, Theoria. 36 (1970) pp. 373- 398.

[11] Montague, R., The proper treatment of quantification in ordinary English. Approaches to natural language, (eds. K.J.J. Hintikka et al.), Dordrecht, (1973) pp. 221-242.

[12] Plotkin, G., A prowerdomain construction, D.A.I Research Report No. 3., Univ. of Edinburgh (1975).

[13] Rasiowa, H., ω^+-valued algorithmic logic as a tool to investigate procedures, Proc. 3-d MFCS Symp. Lecture Notes in Comp. Sci. Vol. 28. (ed. A. Blikle), Springer Verlag, Berlin (1975).

[14] Salwicki, A. Procedures, formal computations and models, ibid. pp. 464-484.

[15] Scott, D., Strachey, C. Towards a mathematical semantics for computer languages, Proc. Symp. on Computers and Automata, (ed. J. Fox), Polytechnic Institute of Brooklyn (1971), pp. 19-46.

[16] Scott, D., Data types as lattices, Proc. Logic Conf. Kiel 1974, (eds. G.H. Müller, A. Oberschelp, K. Potthof), Lecture Notes in Math. Vol. 499. Springer Verlag, Berlin (1975) pp. 579-651.

[17] Štepánková, O., Havel, I. M., A logical theory of robot problem solving. Artifical Intelligence 7(1976) pp. 129-161.

ASPECTS TO CLASSIFY CONTENT OF NATURAL LANGUAGES

P. Kümmel

Institut für Informatik, Universität Stuttgart,
7000 Stuttgart 1, Azenbergstr. 12, Budesrepublik Deutschland

Key words and phrases:

Agglutinated morphologies, auditive/phonographic, content-related, content thesaurus: human- , HCT, individual- , ICT, partial content thesaurus, PCT; deuter-criterion, deuter-disc, deuter-sphere, ideography, isolated morphologies, pictography, self-explanatory.

CR categories: 3.70, 3.71, 3.74, 3.75

Abstract

To bring the function of computer languages closer to that of Natural Languages, more should be known about the phenomena of Natural Language content. One recent approach is to include isolated expression morphologies in the scope of the investigation. Conventional studies have only considered agglutinated morphologies. In fully isolated morphologies syntax is easier to separate from morphology. The whole content treasure of human knowledge to be expressed by Natural Languages can be divided into meanings and syntagms. While syntagms are subdivided into syntax rules and syntax particles, a content analysis primarily centers on the array of single meanings within a thesaurus. Their treedimensional positioning and attachment to vertices of a directed graph substantializes the need to evaluate six deuter--criteria of each meaning and vice-versa. The formalization of a meaning's six deuter-criteria leads the way to classifying content.

Introduction

An increasing number of reasons shows that content of Natural Language deserves more attention than if has been paid the past. (Natural Language is, in the following text, briefly expressed by L_n). Primarily, theoretical evaluations and software aspects in computer science reveal information to be equal with L_n content. They also indicate that bit-evaluations of information are insufficient, if the related content is carried by improper expressions. Consequently, there is a growing interest in the intensification of corresponding research. In common linguistics all evaluations concerning semantics are based on coded expressions in the form of auditive/phonographic utterances. This fact invites a separation between *semantics* and *language content.* Newer and more rewarding ways of enlightening content phenomena of L_n are done by taking self-explanatory and content-related expressions into account. These expressions are characterized by *analog features.* Content-related utterances are often expressed in an *isolated way,* free from syntagms, while autitive/phonographic expressions are mostly agglutinated. Some ideographic, many pictographic, and several gesture expression units help to offer insight into deeper layers of content phenomena. Fully content-related expression units can almost be considered equal to their content, as far as structure and function are concerned. From this angle more solid classification aspects of content phenomena were seen, which helped to classify L_n content. Analyses for formalization purposes achieved results by which meanings in the form of content units are attached to vertices of a graph. The size of such a graph represents simultaneously the dimension of a content thesaurus, which acts as an array in which associations take place. Such a dimension is related to the measurement of the human- , HCT, an individual- , ICT, or a particular content thesaurus, PCT. Operational models deal usually with PCTs, which in a way can be considered as *restricted* ICTs or HCTs. An HCT represent the sum of all ICTs minus redundancy. Association procedures and *thinking operations* are to be considered as movements or navigations along the edges from node to node or meaning to meaning. These dynamic phenomena or movements, four–dimensional in character, are based on three–dimensional conditions. In order to classify these three–dimensional conditions in more subtle ways, six *deuter–criteria* were substantialized: *identity, age, association, frequency, significance, and truth.* Each of them is to be formalized according to procedures, in which one deuter–criterion corresponds to three– and four–dimensional facts.

Information is identical with natural language content

A recent survey on activities in computer science which deal with *database abstractions* and *speech recognition systems* shows a growing necessity to understand more about the field of L_n content. While former so called *information retrieval systems* gave way to *database generalizations,* speech recognition systems were neglected for longer periods, due to difficulties involved. These difficulties turned out to be deeply related to misunderstandings of L_n content structures.

Some research on database concepts and generalized databases deals with two kinds of abstractions: 1. aggregations and 2. generalizations. Aggregation is an abstraction which turns a relationship between objects into an aggregate object. Generalization is an abstraction which turns a class of objects into a generic object. Both abstractions depend heavily on directed graph systems with meanings attached to their vertices ((14,15) , p. 109 f) and consequently point into the direction of L_n content phenomena.

Another way to understand the significance of L_n content for information arrays in information systems is represented by research on *automatic speech recognition.* To many of the researchers involved, the final outcome of their developments turned out to be the learning that so called knowledge sources and their mutual associations are represented by fundamental central databases in a system ((4,12,) , p. 2ff). Also experiences with the Hearsay I and the Hearsay II projects at the Computer Science Department of the Carnegie-Mellon University in Pittsburgh, Pa., showed that central attention has to be paid to a so called conceptual level and a global database, both closely related to L_n content phenomena ((1,2,9,13) , p. 14–M2 and 15–M2). Within these systems a *focus of attention* movement corresponds to association navigations ((3,11), p. 3 ff).

A four year old project by IBM to conceive an *automatic dictation system* is still on an experimental level and unfinished. Charecterized as a *speech recognition system* with continu–ously spoken utterances, the vocabulary is restricted to *patent filing matters* and among them to the field of *laser inventions.* This enterprise is aimed at the substitution of a secretary by a form of typewriter which perceives input in auditive shape and delivers it in the form of typed phonographical output. Another IBM project is concerned with the input of discretely uttered words. A device is to serve traveling customers with luggage in their hands. The uttered and separately spoken command words to the system are automatically interpreted to give corresponding advice in L_n to the user, concerning the handling of his luggage. This project is only two years old. Earlier attempts by *Bolt Beranek & Newman* in Boston Mass. as well as all these projects of automatic speech recognition by the CMU and IBM have one thing in common: The recognition of physical/phonetic data exclusively does not encounter too many difficulties, due to technical improvements of corresponding hardware devices. But, real and grave obstacles appear, if the meaning of those spoken expressions is to recognized automatically.

All these concepts, to recognize the carried content of L_n auditive expressions contain a second serious problem: the automatic search of suitable answers to a recognized content of the input question. This search procedure has to cope with a realtime speed of 4 to 6 syllables/sec–ond *speech velocity* and a 0.5 to 1.5 second *response time.* Both, the content recognition as well as the search procedure for a suitable and adequate answer require a basic dependence from L_n content phenomena, organized in graph shape. *Content recognition* and *answer search procedure* are also mutually dependent.

Highly optimized expression systems and cleverly coded information is always reducible to content of L_n, no matter whetter coded by auditive/phonographic, numerical or digital expression sequences.

Finally, in addition, information carried by artificial language expression *(artificial lan–guage* is abbreviated to L_a), including all varieties of lowest to highest level programming languages, must be explainable by L_n utterances. Consequently information and content of L_a is reducible to L_n content. L_a systems can be regarded as subset systems of L_n.

Unreliability of coded expression

Though auditive/phonographic and especially auditive expression represents the most inexpensively and most easily produced utterance by human beings, it is characterized by no, or almost no, self–explanatory features and no content–relation. This led to unsurmountable difficulties in separating syntax from morphology. Consequently all linguistic research into the field of semantics centered on sequences of utterances considering phrases and even whole sentences in connection. Agglutinations in form of syntax particles gave no chance to look through the expression layer into deeper regions of content. The investigation of content structures of a single meaning has been widely neglected. Computer scientists working on the semantics of programming languages and semantics of databases were eager to adopt these practices all too rapidly. Their results were correspondingly modest. In order to achieve powerful and highly effective expression analyses for the evaluation of content, the tool of self–explanatory and content–related utterances becomes *obligatory*. In a way the expression function *self–expla–batory* or *content–related* can also be considered *analog* in character. This means expression functions of Chinese characters, Hieroglyphs, and pictographic signs or sign sequences for road– and pedestrian traffic must be taken into consideration ((5), p. 31 ff). If *acute visual* utterances of some deaf–mute gestures or American Indian gestures are included in the investigations, but not *permanent visual* and consequently graphic expressions, identical self–explanatory expression functions can be recognized ((5), p. 84 ff).

Following these findings, extensive analyses of ideo– and pictogenetic expression functions led to results which deliver handier tools for classifying content of L_n on the one hand and give a clear distinction between syntax and morphology of L_n on the other.

Analyses of content-related utterances

Expectations of extracting more knowledge of L_n content from self–explanatory expression morphologies, lead the way to the analysis of the structure and function of ideo– and pictographic carrier systems. Among Chinese characters, Hieroglyphs and traffic signs particular carrier properties were recognized, which appear clearer if fully content-related units are observed. A significant proof of these phenomena could be substantiated, when permanent expression morphologies for the visual perception were compared with acute ones ((5), p. 103). In other words, highly self–explanatory ideographic signs, in the shape of Chinese characters, Hieroglyphs, and traffic signs turned out to be structure–identical with those of content–related expression units among deaf–mute and American Indian gestures ((5) , p. 103 f). The next step was to realize that the morphology of fully content–related and self–explanatory expression units is equal to the structure phenomena of their transported content. This means expression is identical to its carried content if self–explanatory morphologies and functions of that expression are guaranteed. A recognition of this fact led to a development by which meanings could be carried and arranged into proper hierarchies. The structure of these hierarchies became particularly clear after *root–signs,* also designated *radicals* or deuter–signs, were taken into account. The phenomenon of *deuter–signs* among Chinese characters could also be proved among Hieroglyphs and traffic signs. With the help of deuter–signs the *expression quotient* of sign content has been established ((6), p. 19 ff).

Besides better understanding of one meaning's particular properties, sequences of fully content–related expression units also indicated that conventional morpho–syntactic hurdles were easy to overcome when analyzing isolated morphologies. Isolated morphologies, usually present in sequences of ideographic expression units like ancient Chinese texts, sequences of Hieroglyphs, or gesture units, are seldomly found in expressions for the auditive perception. Expections are *air traffic control language, children's language* within a restricted development phase and adult expression habits among foreign tourists and guest workers. Among these particular language systems the first one, air traffic control language, serves experimental purposes best. Better material for morpho–syntactic analyses in connection with content analyses is rendered by genuine ideographic expressions. Morphology, or the structure of lan– guage expression, can be classified to a high level in sequences of isolated ideographic and fully content–related expression units. This concatenation of morphological phenomena per– mits a distinction between solid morphological components and syntax particles. E. g. in ancient Chinese texts one character or *expression unit* carrying the meaning *past tense* puts a whole sentence into the past tense. Syntactic constituents of L_n expressions, the syntagms, are divided into syntax rules, e.g. word order rules like *subject/predicate/object,* and syntax particles, like the *past tense particle,* expressed by one unit. All syntagms are now easy to locate and to analyze.This clear destinction between morphology and syntax within L_n expressions additionally cleared the way to more thorough investigations of one meaning's particular phenomena.

The appearance of the deuter–sign among Chinese characters supplied results which made the following classifications of a meaning possible:

1. Sets of meanings can vary in size, due to the configuration of a thesaurus as a PCT, ICT, or HCT.
2. With an elapse of time the number of meanings in a thesaurus usually increases and only seldom decreases. The thesaurus grows by adding new and younger meanings to the stockpile of older ones. It grows like a plant or sponge ((6) , p. 64).
3. A meaning is to be incorporated into a hierarchy in which different meanings posses differentiated vertical and horizontal *concatenation values* or *grades* like condisions in a directed graph.

The preceding three classification aspects, derived from the function of the Chinese deuter–sign, resulted in the deduction of the first three sub–criteria of a meaning: *1. Identity 2. Age, and 3. Association.* The remaining three criteria: *4. Frequency, 5. Significance, and 6. Truth,* are logically based on the first three. One meaning, split into six sub/criteria, has been designated *deuter* and the six subcriteria *deuter–ciretiria.*

The deuter–sphere, a three-dimensional array of content

Concerning the third criterion, *association,* it very soon became clear that associations in a thesaurus do not only take place along vertical concatenations or edges in a hierarchy, but also may occur along horizontal edges. As well as all edges in a directed graph, designated *vertical edges,* the remainder of potential concatenations, which fill up a directed graph to produe a *complete graph,* are designated *horizontal edges.* Graph theory shows, however, that a directed graph with more than five nodes loses its planarity if all nodes are concatenated to all nodes. A fully functioning thesaurus of meanings, in which all potential horizontal associa– tions could be established consequently gains three–dimensional structural features if its volume exceeds five meanings. Trees as used in conventional linguistics do not serve the purpose as they are two–dimensional in structure and function. To improve conditions originally a three– -dimensional

deuter–sponge was introduced ((6), p. 64 ff). However very soon formalization requirements indicated the development from a deuter–sponge to a *deuter–sphere* by the following method. In order not to harm the vertical hierarchy, a branch with leaves is observed. As nature provides suitable models, a branch with leaves can easily be compared with the structure of an *agave tree.* In a similar way to an agave tree, the whole body of branch ramifications is adapted appropriately to form a structure in which all leaves are positioned flatly behind one another. If each leaf is considered as a disc, several discs or deuter–discs can shape a sphere. Naturally the diameters of the discs have to be different, which is normally possible as all discs possess different implementations. The whole purpose of this arrangement is to achieve an optimized formalization effect for horizontal associations. These horizontal associations take place along horizontal edges spanning between the leaves in disc–shapes. Thus the model of a deuter–
–sphere provides valuable possibilities for deriving algorithms for associative functions as in the memory of a human brain ((8), p. 619 f). The deuter–sphere concept also helped to offer new possibilities for updating procedures in database systems ((7) , p. 558).

The six sub–criteria of one meaning

1. Identity Value:

One content unit of L_n or one meaning is identified by Arabic numerals according to the internationally known decimal classification system, DC. This permits unlimited hierarc–hical growth of vertical associations by time. Newly generated meanings to be added to a thesaurus are equipped with a decimal fraction.

2. Age Value:

The age value is separated into two different data:
2a) the absolute age value and 2b) the relative age value. The absolute age value of a meaning is measured by a continuous count and implementation count into a thesaurus: 1 to n from old to new. The absolute age value of the latest or youngest meaning added indicates simulta–neously the size of a thesaurus. The relative age value stands in relation to existing calendar and time measurement systems. The newest meaning in a thesaurus usually touches the periph–ery or the sphere–skin of a thesaurus.

3. Association Value:

The association value is separated into two different data: 3a) the vertical association value A_v and 3b) the horizontal association value A_h. The A_v value features logic concatenations between meanings naturally grown in a time continuum like the hierarchical structure of a plant. Vertical associations compare with edges in a directed graph. The vertical association value of a node measured by the number of edges leading from this node, or, more exactly, the number of strings in the node-ring ((8), p. 615). Horizontal associations between meanings and meaning nodes represent edges in a complete graph minus the edged belonging to vertical hierarcies in that same graph. Potential connections between all nodes in a graph or thesaurus are possible. Horizontal associations compare with *free thinking* and undogmatic association procedures, while vertical associations take place in strict hierarchies.

4. Frequency Value:

Frequency values of a deuter are separated into 4a) absolute frequency and 4b) relative frequency values. Both frequency values refer to active expression procedures or passive per– ceiving processes. This way of explaining content with expression sequences is related to expression habits concerning active and passive vocabulary treasures. The absolute frequency count of a meaning measures the actual event in which that meaning is used in an HCT, ICT, or a PCT. A relative frequency count considers all absolute frequency data of all investigated meanings and assigns the value 1 to the meaning with the highest absolute frequency value. The meaning with the second largest absolute frequency value receives the relative frequency value 1/2 and so on. In an ICT frequencies are measures by the number of attention proce– dures concerning a meaning. This attention procedure stands in relation to a) thinking, b) moving, c) working, d) expressing, and e) perceiving processes of an individual. High frequency meanings with a high usage frequency are characterized by closer proximities to the center of a thesaurus or the focus of attention. Their connection facilities are characterized by better conductivity, while meanings of lower usage frequency are demond to die out, if no minimum frequency occurences are guaranteed.

Frequency values, separated into absolute and relative frequencies, refer to occurences of attention processes in ICTs, HCTs, or PCTs.

5. Significance Value:

The significance of a content unit or meaning is closely related to the meanings of the expressions *priority* and *relevance.* This kind of relevance or urgency has been substantialized three–dimensionally by priority zones, realized by a deuter-sponge, the forerunner of a deuter– –sphere ((6), p. 68 f). Unnaturally retarded areas in the homogeneous growth of the deuter- –sponge reveal higher *distance–values* to the common periphery zones. With the deuter– –sphere another method to indicate the significance value three–dimensionally has been introduced ((7), p. 558). By this new method meanings attached to their nodes and their vertical association values A_v are compared. Besides the vertical associations A_v horizontal association values A_h play a role in determining significance values of a meaning. Content units or research topics of low significance value may be new and 100% true, but are lacking extreme urgency and relevance, which also can be the case with inventions. Content units or meanings feature high significance values, if they affect and modify the lives of many/people.The significance value of a meaning affects human life in general. For example the meaning *gravitation* represents a higher significance value than the meaning *weight.* Or the meaning *bread* a higher one than the meaning *spice.*

Significance values of a meaning or a content unit are subject to the vertical association values A_v of neighbouring nodes in a deuter–sphere. In particular the A_v values of the next upper to the root nodes play a role in determining the significance value of a content unit. In addition A_h values of the same node and its neighbouring vertices have to be taken into consideration when determining the significance value.

6. Truth Value:

Particularly relevant in individual content thesauri, ICTs, the truth value can range from O % to 100 %. Only a 100 % truth value represents a fact. In particular content thesauri, PCTs or the human content thesaurus, the HCT, only facts should be stored. Consequently all questions about the percentage of a truth value are related to association procedures and thinking operations in ICTs. That means that highly valuable thinking, *conceiving* and *original creating functions,* are accompanied by differentiated degrees of truth values. In other words, fiction is closely related to original thinking, the conception of new ideas and the bearing of new information. A theory of truth represents a theory of knowledge. Ouestions about the truth value are always related to new information. New information or content units are positioned directly at the periphery of a deuter–disc, and thus of a deuter-sphere if the model of a deuter-sphere is conoerned.

The truth value of a newly born content unit or meaning is substantialized particularly by, and rests on, a logic geometry of horizontal as well as vertical associations. The new content unit may not exist without proper and sufficient connections to the exprerienced substance of knowledge. All fictional association processes in the human mind, which can accomplish highly valuable association results and masterpieces of art, carr the risk of *untrue* or *partially true* results. Thus an association procedure and projection into the transcendental and areas outside the skin of a deuter-sphere will lead to true results, only if a sucfficient number of horizontal associations A_h to the infrastructure of knowledge is guaranteed. Geometric infrastructures, especially of the A_h values, are necessary to substantialize the truth of a newly originated content unit. Only if all possible association angles and connections or a *sufficient* number of associations is substantialized does the truth value amount to 100%. This means, that if only one among the necessary potential associations is not realized, then no 100% truth value is obtained. If conceptions of new meanings or content units in the from of inventions or scientific discoveries are concerned a sufficient number of infra associations of all related areas has to exist. In other words, related areas which are positioned in border regions must be penetrated very thoroughly in order to achieve an expansion of the outer sphere-skin and enlarge the volume of the content thesaurus in a deuter-sphere. This means that in an ICT the individual is only capable of penetrating the peripheral outer skin of a deuter-sphere if fundamental knowledge and an exhaustive comprehension of all regions underneath that portion of the peripheral skin exists. Otherwise empty assumptions may lead to results which lack the necessary 100% truth value. Even an increasing frequency value along horizontal associations may not yet assure the truth of the new content unit aimed at.

Horizontal associations can point in the direction of a newly born vertex of the ICT in the form of a deuter-sphere, but the vertex cannot yet be vertically concatenated to an already existing meaning. Only after a proper connection by means of a vertical association in an ICT is a new meaning achieved and a 100% truth value estàblished. A corresponding lower percentage of truth is in proportion to existing association within the amount of sufficient associations. Sufficient associations are equal to 100% of a truth value. The logic procedure of a final vertical concatenation runs parallel with an empirical proof by experiments in the case of an invertion.

Conclusion

Computer Science as a scientific discipline is not restricted to the application of conventional and *all purpose* digital hardware. In order to cultivate conventional digital equipment from calculating machines to a more user-friedly *language- robot,* all potential hardware means should be used to adapt machinery to Natural Language requirements. How Natural Language requirements really are conditioned only a thorough analysis of Natural Language content phenomena will show. Such an enterprise postulates more investigations of isolated morphologies for restricted thesauri as well as increased efforts to evaluate self-explanatory and content-related expression structures. A sub-division of one meaning into six deuter-criteria and their three--dimensional array indicates the way for better navigation procedures to simulate natural asociations.

References

[1] Erman L., Organization of the HEARSAY II Speech Understanding System, in: Working Papers in Speech Recognition III, Carnegie-Mellon University, Computer Science Speech Group, Pittsburgh, Pa, (1974)

[2] Fennel R., co-author, see No. 1

[3] Hayes–Roth F., Focus of Attention in a Distributed Logic Speech Understanding System, Working Papers of the Department of Computer Science, Carnegie-Mellon University (1976).

[4] Hayes-Roth F., Syntax and Semantics in a Distributed Speech Understanding System, Working Papers of the Department of Computer Sciences, CMU, (1976).

[5] Kümmel P., Struktur und Funktion sichtbarer Zeichen, 223 pp. Verlag-Schnelle GmbH, Quikborn bei Hamburg, (1969)

[6] Kümmel P., Wertbestimmung von Information, (zur qualitativen Analyse des Inhalts natülicher Sprachen), 90 pp., Max Niemayer-Vlg., Tübingen, (1972).

[7] Kümmel P., Improved Updating in Relational Data Base Systems by Deuter-Sphere Algorithms, in: Proc. of the 2nd Int. Conf. on Software Engineering, IEEE and ACM, pp. 556-561, Oktober 1976. San Francisco, (1976).

[8] Kümmel P., Deuter-Criteria's Impact on the Computation of Natural Languages, in: Proc. of the 2nd Hungarian Computer Science Conference, pp. 609-623, Budapest, (1977).

[9] Lesser V., co-author, see No.1, (1974).

[10] Lesser V., co-author, see No. 4, (1976).

[11] Lesser V., co-author, see No. 3, (1976).

[12] Mostov D., co-author, see No. 4, (1976).

[13] Reddy D., co-author, see No. 1, (1974).

[14] Smith J., Data Base Abstractions; Aggregation and Generalization, in: ACM Transactions on Database Systems, Vol. 2, No. 2, June 77, pp. 105-133, (1977).

[15] Smith D., co-author, see No. 14, (1977).

TRANSCELL – A CELLULAR AUTOMATA TRANSITION FUNCTION DEFINITION AND MINIMIZATION LANGUAGE FOR CELLULAR MICROPROGRAMMING

T. Legendi

Research Group on Mathematical Logic and Automata Theory of,
Hungarian Academy of Sciences

1 gives proposals for the construction of cellular automata emulators with variable transition function. This idea has been developed and concretized in 2 for the construction of cellular processors. In this paper the proposal of 2 for transition function minimization is treated in details. A semi – automatic solution is proposed – the user may (and should) control (to program in a very simple way) the minimization process.

The basic idea of 1 is the following: variable transition function makes cellular spaces more flexible. Variable transition functions may be implemented by RAM-s in each cell or by one RAM in a CCPU (cellular CPU) driving quasi cells by sending terms sequentially to them. The later solution looses the totally parallel operation. However, it drastically reduces the size of cells and (as explained in detail in 2) in this way by increassing the number of cells price/performance is improving. It is reasonable for the loss of speed does not depend on the number of cells connected to a CCPU – it will be proportional to the number of terms of the transition function. However this number may be yet large, slowing down the operation to an unacceptable level. Therefore the loss of speed should be (at least partially) balanced by transition function minimization, this is of vital inportance for this construction.

It would be fine to find a general automatic minimization algorithm applicable to any transition function.

As such an algorithm has not yet been found, a set of minimizing operators had been worked out. They can be called through directives and the user (knowing the specialties of his actual transition function) should determine the sequence of minimizing operators.

Usually (traditionally) a transition function is defined by a set of terms. A term consists from the old state of a cell, the states of its neighbours and the corresponding new state. For example: 0 1234 5

During execution a cell in state 0 having neighbours in states 1, 2, 3, 4 will assume state 5.

In 10 some transition function definition and representation forms were examined. It is very important to choose an appropriate form of transition function evaluation. Examining among others software 5, 6, 7 and hardware 8, 9 approaches and having some experience in simulation 10, 11 and model building 12 the following way was chosen:

As a starting point the definition form of 3 has been accepted:

Here a term is given in the following form:

0 +1 – 2 – 5 * 4

old state condition part new state

condition + i means: there exists in the neighbourhood minimum one cell in state i

condition – i means: there exists no cell in state i in the neighbourhood

and a term will be executed by cells in "old state" that meet all the conditions (e.g. they will assume the "new state").

A transition function is an ordered set of terms where later terms override the earlier ones if there is a contradiction. (for example 0 +1 3
0 +2 4

might contradict if in the actual neighborhood of a cell there are state(s) 1 and state(s) 2)

This definition is well-suited for a relatively simple hardware implementation; it is easy to find very fast and memory saving computing methods for simulation purposes and finally it is naturally applicable for sequential transition function execution (for quasi cells), too.

However transition functions defined in the above way are not direction sensitive, (e.g. for example they cannot distinct between neighborhoods 1121 and 1222 – both are classified as +1 +2) so this type of definition is weaker than the usual one.

For many purposes this fact may be not disturbing. In 13 it was shown that cellular spaces consisiting of apolar cells may show a polar behaviour (e.g. symmetric local transition functions may induce asymmetric global transitions), the main result in 3 was the construction of a concrete transition function and a set of building blocks allowing universal computation – using the above definition form exclusively. As the TRANSCELL run $N^{0}1$ in the Appendix shows in the well-known Codd function 14 the direction sensitivity is hardly utilized (although generally supposed that it is; e.g. the function is defined in a stronger definition form than it would be necessary as it will be shown in the following).

For proving this the Codd function had been coded in the following way:
the condition part $a_1a_2a_3a_4$ (4 neighbours) had been changed for containing all the neighbours, with plus sign if they had appeared in the original condition part (one or more times) and with minus sign if they had not appeared at all.

For example:

0 1221 2 ⟶ 0-0+1+2–3–4–5–6–7 * 2

It is clear that a new rule contains the old one from which it was generated, but there is no one to one correspondence. However this extension of the function consisting of the generated rules has a very low number of contradictions showing that direction sensitivity is not a very important property for the Codd function. Moreover it can be detected that it is not needed at all – the contradictory rules might be classified by the number (or partity) of some distinguished states.

However for effective cellular programming in style of 4 direction sensitivity is very important. Therefore a natural generalization is applied:
±i in the condition part will refer to the i-th property of the neighbourhood rather than to the occurence of state i in the neighbourhood. Typical examples of the properties of the neighbourhood might be: the existence of some state(s), the partity of some state(s), the number of some state(s) and combined versions of these and similar elementary properties. The property definition and minimization part of the language is yet under development, it will not be treated in detail in this paper; only one characteristic result will be shown in the Appendix: TRANSCELL run N^0 2 uses 12 properties for an obvious one to one transformation of the Codd function – the bits of neighbours are chosen for properties. For example:

0 1234 5 ↔ 0–0–1+2–3+4–5–6+7+8+9–10 –11 * 5.

This is a mechanical transformation, showing not very much from the characteristics of a transition function. However a simple, short TRANSCELL program was able to find a solution, equivalent to the result in 15. (For the subfunction from terms for old state 6, 7; that says: the new state will be 0 or 1 depending on the only property: does there exist odd state in the neighborhood or does not.)

The language TRANSCELL is very simple. It consists of the above term definition rules (transition function definition) and directives (property definition not included here) for minimization and housekeeping. They may form a program in a sequential order.
The directives may be grouped:

1.applicable at any time
2. applicable only before the rules
3. applicable only after the rules

For the easier formulation of the functions of the directives the following definitions are needed. They are valid for pairs of rules where the old states are identical.

Two rules are independent if at least one property is prescribed in a contradictory way in them.

for example:
```
3  -1  +4       * 4
3  +1       -6  * 5
```

or:
```
1  +0+1  -2       * 3
1     +1  +2  -4  * 3
```

Two rules are contradictory if they are not independent and the prescribed new states are different.

for example:
```
0 + 0  + 3        * 4
0     +1      -4  * 5
```

Two rules are intersecting if they are not independent and the prescribed new states are identical.

for example:
```
0 + 0  +3       * 4
0     +1    -4  * 4
```

The rule A contains rule B if

either 1. each property required in A is required in the same way (with the same sign) as in B and the new states are identical

for example:
```
A  2  -0  +2            * 2
B  2  -0  +2  +3   -4 * 2
```

or 2. each property required in A is required in the same way (with the same sign) as in B and the states are different but rule A is stronger (see directives FIRST, LAST)

for example:
```
A  2  -0  +2            * 3
B  2  -0  +2  +3   -4 * 2
```

Two rules are contractable if the new state is the same, the required properties are identical except one.

for example:
```
3  -1  +4  +5 * 3
3  -1  -4  +5 * 3
```

Two properties are 2-equivalent if in any rule they occur together with the same sign for example: (properties 0 and 2)

3	+0	–1	+2	–3		* 0
3	–0	–1	–2	+3		* 1
3		+1			–4	* 0
3	+0		+2	+3	+4	* 3
3	+0	+1	+2	–3	+4	* 4

Only the most important directives will be shortly characterized, a lot of utilities and directives of secondary importance will be omitted here.

The group directives that are valid in any part of a TRANSCELL program is used for the control of printing trace information. There are pairs of directives prescribing or prohibiting to print information on contradictory, intersecting, containing and contractable rules.

Only before finishing the group of rules (by an END directive) can be used the FIRST and LAST directives ordering the rules. FIRST means that earlier rules are stronger, LAST means that later rules override earlier ones. (As all the directives, they may be issued more times in any order).

The minimization directives may be used only after having closed (by an END directive) the group of rules.

It is possible to CONTRACT rules:

for example:	1.	1	+2	–4	+6	–7	* 7
	2.	1	+2	+4	+6	–7	* 7
the result is		1	+2		+6	–7	* 7

(contractable rules may be changed for one rule)

CONTAIN ed rules may be cleared

for example:	A			+3	–5	4
	B	0	–2	+3	–5	4

B may be cleared here

or

A	0	–2	+3	–5	6
B	0	–2	+3	–5	7

B may be cleared only if A was stronger

2-EQUIVALENCE clears one of the equivalent properties.

The MINIMUM directive clears all such (unnecessary) property definitions in the condition part that might be cleared without causing contradiction with weaker rules (i.e. with stronger rules a contradiction would not change the transition function).

However a weaker form, MINIMUM * is applicable, too, where no contradiction may arise during the clearing process. The need for it is based on the following considerations: there are operations that work (at all or effectively) only on groups of independent rules, so in some cases to preserve independence is important.

For this purpose serve the directives SMOOTHC and SMOOTHI: generating equivalent independent rules from any group of contradicting or intersecting rules.

The use of these rules is especially important in connection with property minimization. Here is some virtual contradiction: minimizing the number of rules and minimizing the number of properties are contradictory conditions in general. However in practice taking into account a given hardware construction the task is modified: give the minimum number of rules with a limited number of properties.

This means that both types of minimizations should be built in and used in a flexible way.

Examples: LAST

```
0   +1   +2   +3   * 1
0   +1   +2   –2   * 2
0   –1   –2   –3   * 3
```

MINIMUM gives the result:

```
0              * 1
0   +2    –3   * 2
0   –2         * 3
```

MINIMUM * gives the result:

```
0   +2   +3  * 1
0   +2   –3  * 2
0   –2       * 3
```

For the simulation system, from transition functions tables should be generated. As minimized functions may differ each from other, differently minimized tables may be generated and for achieving fastest execution specialized FORTRAN rutinus are also generated by TRANSCELL. (for the search from the minimized (tables).

It is possible to print the rules, etc.
All minimization directives are limited only for some specified old state groups.

Using the above simple directives very effective optimizations may take place as the examples in the following Appendix show.

R e f e r e n c e s

[1] T. Toffoli, On the Large-Scale Implementation of Cellular Spaces by means of Integrated Circuit Arrays. Consiglio Nazionale Delle Ricerche Roma, (1972)

[2] T. Legendi, Cellprocessors in Computer architecture. Computational Linguistics and Computer Languages XI (1976) pp. 147-168.

[3] A. Domán, A three-dimensional cellular space (A challenge to Codd-Icra) Acta Cybernetica, Tom. 2, Fasc. 4, Szeged, (1975) pp. 345-360.

[4] T. Legendi, Programming of cellular processors. Procedings of the Braunschweig Cellular Meeting, June 2-3, (1977).

[5] W.H. Liu CELIA (A Cellular Linear Interative Array Simulator) State University of New New York at Buffalo (1972).

[6] R.F. Brender, A Programming System for the Simulation of Cellular Spaces. Ph. D. Thesis University of Michigan. Computer and Communications Sciences Dep. (1970).

[7] R. Vollmar Über einen Interpretierer zur Simulation zellularen Automaten Angewandte Informatik (6/1973).

[8] K. Preston Jr., Use of the Golay Logic Processor in Pattern-Recognition Studies Using Hexagonal Neigborhood Logic. Symposium on Computers and Automata. Polytechnic Institute of Brooklyn, (1971).

[9] M. J. B. Duff, The cellular logic array image processor D.M. Watson, Computer Journal Vol. 20 N. 1. (1977).

[10] T. Legendi, INTERCELLAS-an interactive cellular space simulation language. Acta Cybernetica Tom. 3. Fasc. 3 (1978)

[11] F. Sümeghy, A group of instructions for transition function definition and minimization using treeform representation. Diploma-work (in Hungarian) JATE, university, Szeged, (1977).

[12] G. Balázs – I. Orovecz, Design and Implementation of microprogrammable cellular processors. Diploma work(s) (in Hungarian) Institute GAMF, Kecskemét, (1977).

[13] G.T. Herman, Polar organisms with apolar individual cells. Proc. of the IV. International Congress for Logic, Methodology and Philosophy of Sciences, (1971) pp. 665-675.

[14] E.F. Codd, Cellular automata, ACM Monograph Series, Academic Press (1968).

[15] M. Szőke Codd-Icra: Regularization of Codd's transition function (in Hungarian) KGM ISZSZI Reports. Budapest, (1974).

SEMANTIC ANALYSIS OF SYNTACTIC METHODS

Heikki Heiskanen
University of Helsinki Department of Social Policy

Two branches of general linguistics are relevant to this paper: syntactics and semantics.

Syntactics has two different meanings. According to its narrower, linguistic meaning it is that branch of linguistics that deals with the syntax of natural languages, i.e. with the grammatical rules describing how sentences and clauses are made up of words and phrases. Syntactics in its wider sense deals with the formal relations of signs and expressions, not only in the linguistic meaning. Here syntactics is used in this broader sense.

Thus syntactic theories give recognition to the constituent structure, i.e. the way in which words and phrases form sentences and clauses. Conversely, a syntactic theory gives a method by which verbal expressions can be divided into parts and by which these parts can be connected to each other. Thus a syntactic theory has opposite functions: synthetic and analytic. Here syntactics will be examined from the viewpoint of its analytic function.

A good number of syntactic theories have been introduced not only into general linguistics, but also other branches of the behavioral sciences. Syntactic theories have been introduced especially in areas where content analysis is relevant in one form or another. Here brief reference will be made to the syntactic methods with either a linguistic or other background and also to a new method called linguistic-mathematical systems theory. These methods represent general linguistics, political science, psychology, jurisprudence and management science.

The selection of syntactic theories is not meant to be exhaustive, far from it. It is intended simply as an example of the different approaches on which such theories are based.

The methods introduced here have been chosen on the following grounds:

1.) The methods introduces an idea according to which verbal expression is divided into linguistic, logical or other kinds of units and is then expressed in terms of these.

2.) It gives an idea of how to compile the analysis results of individual sentences into a summary in the form of a network presentation.

The methods introduced here are subjected to a semantic analysis, i.e. the meanings of the concepts used in these methods are analyzed and compared using the new linguistic-mathematical systems theory as a frame of reference.

The linguistic-mathematical systems theory, or in short, the LM-theory (Heiskanen 1972 and 1975), used here as a frame of reference, will first be introduced. It was developed for the analysis of wage decisions, i.e. for the purpose of content analysis. Its principal function is to transform verbal text into a form in which the content is expressed by means of logico-mathematical concepts.

The logico-mathematical concepts have been chosen for use as basic concepts in order to work up the analyzed text into a form in which it can be processed further by mathematical methods or data machines: both of these operate by means of a set logico-mathematical concepts. Therefore, if the content analysis also operates using these concepts, the content is ready after analysis for mathematical or data machine processing.

There are two kinds of basic concepts in the LM-theory: terms and relations. The terms are either

1.) values of variables such as "dirty" or "high", or

2.) variables defined to be combinations of mutually alternative and exclusive values such as "dirtiness" or "height", or

3.) entities defined to be things to which values and variables belong such as "job conditions" or "salary".

The relations are different kinds of connections such as influence, measurement, definition or membership connections between the terms. However, in this article interest is confined to influence relations of which there are three kinds:

4.) relations between values such as "if" or "because",

5.) relations between variables such as "depends on", and

6.) relations between entities such as "is influenced by".

The analytical idea

In the analysis the individual sentences are divided into parts corresponding to the concept categories mentioned above. Then the content of each sentence is expressed by means of these concepts. In the last phase the contents of individual sentences are combined into compilations called models.

There are three kinds of models depending on the category to which the terms by which the model is expressed belong. Thus there are

a.) value models expressing values and relations between them as illustrated in example A,

b.) variable models expressing variables and relations between them (example B), and

c.) entity models expressing entities and relations between them (example C).

Examples

The examination of syntactic methods and their comparison with LM-theory later in this article will be done using the following sentences as examples. These will be analyzed, at least in part, by the methods to be referred to.

One of them is a value sentence expressing two values of which one is cause and the other is effect. It is analyzed and expressed according to the LM-theory in the following way:

Example A

"Smith's wage is high, because his job is dirty".

Concepts

high = value of the variable "level of wage"
dirty = value of variable "dirtiness of job conditions"
because = cause-effect-relation = = ➤

Content

"dirty = = ➤ high" or
" 'job conditions are dirty' is the cause and 'high wage' is the effect."

The corresponding value model could be as follows:

Bonus for dirtiness

Description of job conditions		Bonus
clean	= = ➤	no bonus
dirty	= = ➤	small bonus
very dirty	= = ➤	high bonus
dirtiness	——➤	level of wage

Thus the value model is a combination of two or more value sentences.

The other sentence is a variable sentence which comprises two variables and the influençe relation between them. It contains information that there is an influence relation between these two variables.

Example B

"The level of wage depends on the dirtiness of job conditions".

Concepts

level of wage = variable
dirtiness = variable
depends on = influence-relation $\longrightarrow$

Content

"dirtiness $\longrightarrow$ level of wage".

The variable model is a combination of two or more variable sentences. Thus a variable model corresponding to the above example could be the following:

$$\begin{tikzcd}
\text{dirtiness} \arrow[dr] & \\
\text{level of skill} \arrow[r] & \text{level of wage} \\
\text{sex} \arrow[ur] &
\end{tikzcd}$$

The content of this model reads:

"The level of wage depends on

1.) the dirtiness of job conditions,

2.) the level of skill required in the job and

3.) the sex of wage earner."

The third sentence is an entity sentence describing the influence relation between two entities.

Example C

"Wage is influenced by job conditions".

Concepts

wage = entity
job conditions = entity
is influenced by = influence relation $\longrightarrow$

Content

$$\begin{tikzcd}\text{"Job conditions} \arrow[r] & \text{Wage"}\end{tikzcd}$$

The corresponding entity model could be the following one, which is based on the variable model in example B:

$$\begin{tikzcd}\text{"Job conditions} \arrow[dr] & \\ \text{Job} \arrow[r] & \text{Wage"} \\ \text{Wage earner} \arrow[ur] & \end{tikzcd}$$

This model reads:

"Wage depends on the

a.) job conditions in which the wage earner works,

b.) job performed by the wage earner and

c.) wage earner."

Connection between different levels of terms

The presence of three parallel "languages", each of them representing of the three categories of terms is essential to the LM way of thinking. Hence, for each sentence, disregarding the concept level on which it is expressed, one can imagine one parallel sentence onthe other two levels. The three examples are just such parallel sentences corresponding to one and the same piece of information. This means that from each value sentence one variable and one entity sentence can be derived and that all data should be specified by the value, the variable and the entity to which value belongs.

Introduction of other syntactic methods

Other syntactic methods will now be introduced and compared with the LM-theory. Here attention will be paid only to the following characteristics in the methods:

1.) What categories of terms are represented in the method at hand: does it operate by values or entities?

2.) What kind of compilations can be expressed by the method:
are they value, variable or entity models?

3.) What connection exists between different term categories: does the method explicitly give the connection between values, variables and entities?

Conventional linguistic theories

The conventional syntactic theories in linguistics usually assume that any sentence of a natural language may be presented as a particular arrangement of elements which are words or parts of words and which are called by name expressing their grammatical function. According to this (Lyons 1970)

1.) the text to be analyzed is first divided into sentences,

2.) these sentences are divided into main and subordinate clauses,

3.) these clauses are divided into noun and predicate phrases,

4.) the noun phrases are divided into nouns and determiners, and

5.) the predicate phrases are divided into verbs and auxiliaries.

After sentences are divided into elements in this way, they are expressed by means of tree diagrams as illustrated in figure 1.

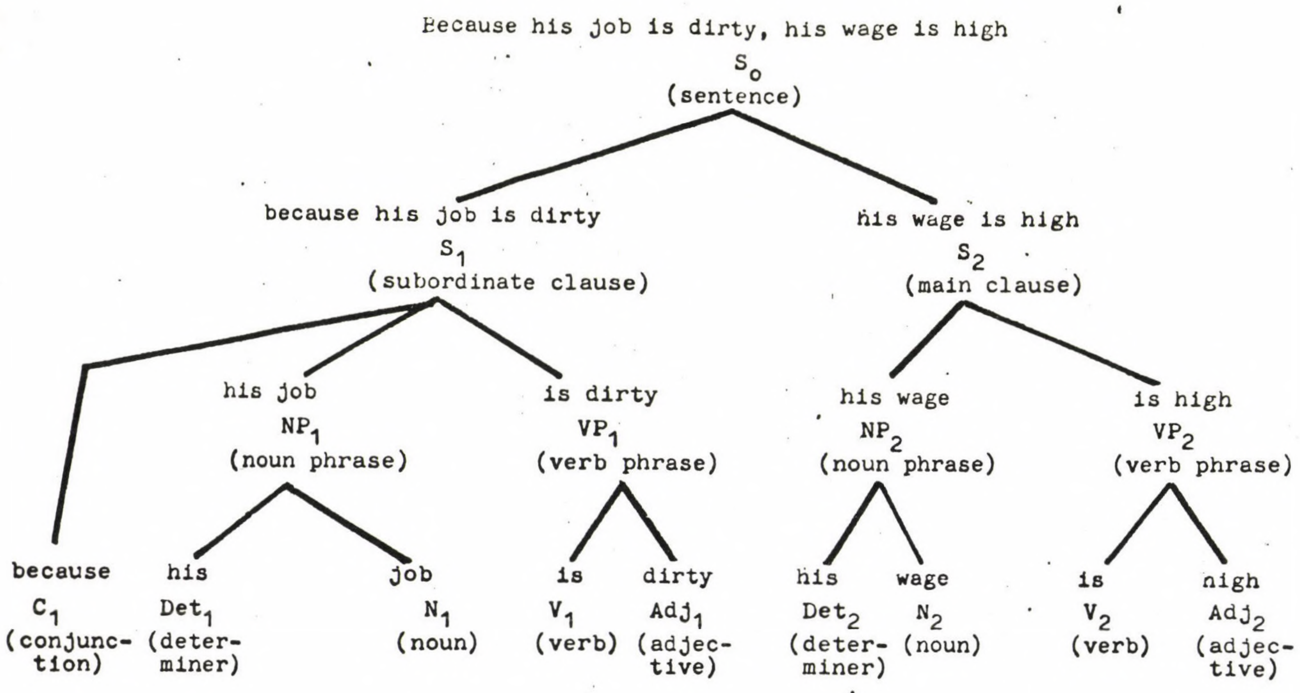


Figure 1

A. Comparison with the logico-mathematical concepts. In the method it is essential that the sentences be divided into parts corresponding to different grammatical categories. The correspondence to the logico-mathematical concepts is as follows:

1.) values of variables (dirty, high) = adjectives,

2.) variables (dirtiness, level) = nouns

3.) entities (wage, job conditions) = nouns

4.) relations between values (because) = conjunctions

5.) relations between variables (depends on) = verbs

6.) relations between enties (is influenced by) = verbs.

B. The method of compilation is not given.

C. The connection between different term categories is not given.

Syntactic graphs

Another less known linguistic syntactic theory is the syntactic graphs technique, which was developed in Czechoslovakia and which has been introduced e.g. in Klir (1967).

According to this theory a sentence is conceived to be made up of so called sentence elements or phrases. These elements are interpreted to form syntactic pairs. In every pair one element is always the main and the other is the dependent element. The syntactic pairs are indicated by graphs, i.e. by arrows. The relations between the elements are expressed by interrogative words such as "what kind of", "how is it" or "where".

The first example sentence takes the from illustrated in figure 2.

A. Comparison with the logico-mathematical concepts. All the logico-mathematical concepts are simply conceived as elements

$$\begin{tikzcd}
\text{his} & & & & \text{his} & & \\
\text{wage} \arrow[u] \arrow[r] & \text{is} \arrow[r] & \text{high} \arrow[r] & \text{because} \arrow[r] & \text{job} \arrow[u] \arrow[r] & \text{is} \arrow[r] & \text{dirty}
\end{tikzcd}$$

Figure 2

B. The method of compilation is given. The compilation is a kind of combination of value, variable and entity models.

C. The connection between different term categories is not given simply because there is only one concept category.

Discoursive content analysis

There are different approaches in content analysis as understood in political science. One of them is the discoursive content analysis method or discourse model which considers contents as linguistic referents and expresses them in denotations and connotations (Harris 1952, Pietilä 1975).

The text is divided into elements, which usually represent different kinds of linguistic concepts which might be syntactic categories such as sentences, clauses, phrases and the like or lexical categories such as nouns,adjectives, verbs, conjunctions and the like. Then the content is presented by means of these elements by expressing what kind of combinations of these elements have been found in the text and the way, in which they are linked to each other. The concepts presented by a language are conceived as 1) things or entities, 2) properties of things, 3) relations of things to other things and 4) propositions about things, properties and relations. The propositions are built up by means of concepts and the sentences by means of propositions. The sentences are either

a.) propositions of simple classification, i.e. $P(a)$ meaning "the entity 'a' has a property P"

b.) propositions of relations, i.e. $R(a,b)$ which is read "between entities 'a' and 'b' there is a relation R"

c.) combinations of a and b, e.g. $R(P_1(a), P_j(b))$ and

d.) inferences such as implications, i.e. $a \Longrightarrow b$ meaning "if a then b".

The first of our examples takes the following form according to the discoursive analisys method:

"His wage is high, because his job is dirty".

Concepts

wage = entity a
high = property P_1
job = entity b
dirty = property P_2
because = $\Longrightarrow$

Content

"$P_1(a) \Leftarrow P_2(b)$" or
"dirty (job) $\Rightarrow$ high (wage)

The other examples take the following form:

"Level of wage depends on the dirtiness of job conditions"

Concepts

level of wage = entity a
dirtiness of job conditions = entity b
depends on = relation R

Content

"R(a,b)" or "dependence (level, dirtiness)"

"Wage is influenced by job conditions".

Concepts

wage = entity a
job conditions = entity b
influenced by = relation R

Content

"R(a,b)" or "influence (job conditions, wage)".

The summaries in the content analysis are mostly in the form of tables. They describe the content either qualitatively by enumerating what kinds of combinations have been found or the position taken towards different topics, or quantitatively by describing how many compound or simple sentences or how many different sorts of verbs have been found.

A Comparison with the logic-mathematical concepts. Even if the discoursive content analysis method employs logical concepts, its concepts differ from the LM-concepts:

1.) values of variables (dirty, high) = properties,

2.) variables (dirtiness, level) = entities

3.) entities (job conditions, wage) = entities

4.) relations between values (because) = relation ()

5.) relations between variables (depends on) = relations R

6.) relations between entities (influenced by) = relations R.

B The method for compiling results is given, but it is in tabular form, which does not give correspondence of terms in the sense the LM-theory means. Some of the tables resemble the value models, but most of them do not.

C The connection between different term categories is not given and the same terms can be in different categories.

Signed digraphs

Roberts (1971) among others has used a method based on theory of graphs for the analysis of complex structures of interconnections. The basic elements in this method, called the signed digraph technique, are variables and the cause-effect-relations between them. The sentences to be analyzed are thus divided into variables and cause-effect-relations. The content is then compiled into a network where points stand for variables and the arrows connecting these points stand for the cause-effect-relations. This network is called a signed digraph. The presentation is exactly the same as in the LM-theory for the variable sentences in example B.

A Comparison with the logico-mathematical concepts

1.) variables (dirtiness, level) = variables

2.) values of variables (dirty, high) = there is no special concept for values, but they are also interpreted as standing for variables,

3.) relations between variables (depends on)
= cause effect relations
4.) relations between values (because)

Entities and relations between entities are not included in the concepts.

B The method for compiling results is given and it is identical to the variable model of the LM-theory.

C The connection between different term categories is not given.

Cognitive mapping

Cognitive mapping is a method for the content analysis developed by political scientists. It has been applied by many scientists in different variations, but here it will be introduced mainly in the form in which Axelrod (1975) has developed and used it, because he has greatly improved and systematized this method and also successfully used it to research.

The basic elements are the concepts and the causal links between these concepts. The former are something like "the amount of security in Persia", i.e. concepts that can take different values such as "a great amount of security", or " a small amount of security".

The causal links or beliefs are regenerated as expressions relating concepts to each other.

The content of the text to be analyzed is compiled into a network where the concepts are represented by points and the causal links by arrows between these points. The network is called a cognitive map.

Both the value and variable sentence of our examples are interpreted in the same way according to the cognitive mapping method:

"His wage is high, because his job is dirty."

"The level of wage depends on the dirtiness of job conditions."

Concepts

level of wage = concept variable A
dirtiness = concept variable B
depends on, because = causal belief $\longrightarrow$

Content

"A $\xleftarrow{+}$ B"

A. Comparison with the logico-mathematical concepts

$$\left.\begin{array}{l}\text{1.) values of variables}\\ \text{2.) variables}\end{array}\right\} = \text{concept variables}$$

$$\left.\begin{array}{l}\text{3.) relations between values}\\ \text{4.) relations between variables}\end{array}\right\} = \text{causal beliefs}$$

Entities and entity relations are ignored.

B. The method for compiling results is introduced and is identical with the variables model of the LM-theory.

C. The connection between different concept categories is not given explicitly, even if the fact that both values and variables are interpreted to be concept variables implicitly involves the idea that one variable can be derived from every value.

Normalized sentence-index matrix

Allen (1965) has developed a content analysis method for jurisprudence called the normalized sentence-index matrix (N-SIM). Its purpose is to draft complicated statutes in an accurate way to be used both manually and in an automatic data processing system. According to this system the sentences to be analyzed will be regarded as "if and only if"-sentences. The text to be analyzed is divided into sentences such as "his wage is high" and "his job is dirty" and into syntax words such as "if . . . then", "or" and "if and only if . . . then". Thus the first of our example sentences is analyzed and expressed as follows:

Concepts:

"his wage is high" = sentence S1
"his job is dirty" = sentence S2
"because" = syntax word $\longrightarrow$

Content

"S1 $\longleftarrow$ S2"

When the sentences analyzed in this way are compiled into a network, the normalized sentence-index matrix is formed.

A. Comparison with the logico-mathematical concepts.

1.) values of variables = sentences
2.) relations between values = syntax words

The variables and entities are ignored in the method.

B. The method for compiling results is involved in the method, i.e. the normalized sentence-index matrix. It is to some extent analogical with the value model of the LM-theory.

C. The connection between the different concept categories is not given, obviously because the system operates only by means of the values and their relations.

Psycho-logic

Abelson and Rosenberg (1958) have developed a method for the description of human cognitive processes. The method is called psycho-logic. The essential concepts are on the one hand "thing-like" concepts that for a given person are relevant to a given object and on the other hand the desirableness-relations between these concepts. Thus the verbal presentation is divided into these concepts and these relations. The compilation of the analysis results is presented in the form of a network where the points represent concepts and the arrows the relations: a positive arrow refers to usefulness or desirableness and a negative one to harmfulness or undesirableness.

This method can not be applied directly to our examples because they do not express such desirableness-relations as required by psyho-logic. However, indirectly the description could be as follows:

"Wage $\xrightarrow{+}$ wage earner", i.e.
"wage is desirable for the wage earner".

A. Comparison with logico-mathematical concepts. The "thinglike" concepts are obviously the same as entities in the LM-theory, i.e. things to which the values and variables belong. The relation concept of psycho-logic is narrower than the influence-relation of the LM-theory. It corresponds to those influence-relations that can be interpreted as desirableness or usefulness. Thus:

1.) entities (wage, job conditions) = concepts,
2.) relations between entities (is useful for) = relation

Other categories of terms are ignored.

B. The method for compiling results is involved in the system, i.e. the network. It is analogical with the entity-model of the LM-theory.

C. The connection between the different concept categories is not given, again for the obvious reason that the method operates only with one kind of concepts, i.e. with entities.

Discussion

The syntactic methods introduced above can be classified into five categories:

1.) linguistic syntactic methods, which include the conventional and graph methods,
2.) discoursive content analysis method,
3.) cognitive mapping which also includes signed digraphs,
4.) normalized sentence-index matrix,
5.) psycho-logic.

These are compared in table 1 with the LM-concepts. This comparison gives the following results:

a.) the linguistic method operates with the same concepts as the LM-theory but includes variables and entities in the same category; there is no suggestion of compilations.

b.) the discoursive analysis method also operates with all concepts and includes variables and entities the same category. The concept names have been taken from logics, whereas the linguistic methods use grammatical concept names. Summaries are given in tabular forms.

c.) cognitive mapping operates with variables and compilations are analogical with the variable-models.

d.) normalized sentence-index matrix operates with values and the compilation is analogical to the value-model.

e.) psycho-logic operates with entities and the compilation is analogical to the entity--models.

	LINGUISTIC SYNTACTIC METHODS	DISCOURSIVE ANALYSIS	COGNITIVE MAPPING	N-SIM	PSYCHO-LOGIC
VALUE	adjective	property	concept variable	sentence	–
VALUE-RELATION	conjuction	implication	causal belief	syntax words	–
VARIABLE	noun	entity	concept variable	–	–
VARIABLE-RELATION	verb	relation	causal belief	–	–
ENTITY	noun	entity	–	–	concept
ENTITY-RELATIONS	verb	relation	–	–	desirableness-relation
VALUE-MODEL	–	tables	–	normalized sentence-index matrix	–
VARIABLE-MODEL	–	–	cognitive map	–	–
ENTITY-MODEL	–	–	–	–	network
CONNECTION BETWEEN DIFFERENT CATEGORIES OF TERMS	–	–	–	–	–

TABLE 1

f.) no methods give any description of the connection between various term categories.

The LM-method can be conceived as a summary of the methods introduced here: it operates on all three levels of terms, has three kinds of model; one for each level, and the relations between various terms are thoroughly defined. The methods referred to are more one-sided: either they operate by means of one language (N-SIM by values, cognitive mapping by variables and psychologic by entities) or else variable and entity languages are identical as in linguistic and discoursive methods and in each of them there is at most only one type of model.

Further the terminology of the LM-theory means one step toward a standardization of terms. It operates with three logico-mathematical terms instead of the 20 or so in the methods introduced in this paper.

References

[1] Axelord, R. (ed.) Stricture of Decisions. Princeton University Press, Princeton, (1976)

[2] Axelrod, R. Decision for Neoimperialism: The Deliberations of the British Eastern Committee in 1918. In Axelrod (1976) pp. 77-95

[3] Bach, E. and Harms, R.T.: Universals in Language. Holt, Rinehart and Winston, New York (1968)

[4] Bach, E. Nous and Phrases. In Bach and Hammon (1968)

[5] Harris, Z. Discourse Analysis. Language 1952, pp. 1-30 and 474-494

[6] Heiskanen, H. Palkan muodostumisen teoriaa ja käytäntöä. Helsinki University of Technology, Research Papers No. 38, Otaniemi, (1972)

[7] Heiskanen, H, A Systems Theoretical Approach to the Analysis of Social Action and Decisions. Annales Academiae Scientiarum Fennicae, Ser. B 191, Helsinki 1975.

[8] Klir, J and Valach, M Cybernetic Modelling. Iliffe Books, London (1967)

[9] Lyons, J. (ed.) New Horizons in Linguistics. Penguin, Harmondsworth, (1970)

[10] Lyons, J. Generative Syntax. In Lyons (ed.) (1970), pp. 115-139

[11] Pietilä, K. A Study in the Methodology of Consciousness Research. Research Institute for Social Sciences, University of Tampere B Report No 24/1975

[12] Roberts, F.S. Signed Digraphs and the Growing Demand for Energy. Environment and Planning 3, 1971, 395-410

[13] Roberts, S.F. Strategy for Energy Crisis: the Case of Commuter Transportation Policy. In Axelrod (ed.) (1976) pp. 142-179.

THE STRUCTURE OF THE SNEL SYSTEM FOR AUTOMATIC DOCUMENTATION

C. Anzaldi – L. Mirri Passerini

Summary

In this paper the Authors present the SNEL system for automatic documentation.

Sinces this a system is modularly structured, it has been described by pointing out the function carried out by each module and showing the logic and information flows which link the different modules.

This system, planned to be run on a minicomputer, was implemented according to criteria chosen to ease its portability.

The paper includes a complete example.

Introduction

The SNEL system has been planned for the management, on minicomputers, of collections of papers concerning a very specialized field: such an implementation is meaningful because, generally, the number of papers is relatively small.

To improve such a system the use of indexing and retrieval techiques based on a thesaurus is required. On the other hand we do not know of any techniques for the automatic construction of thesauri to implement on minicomputers.

The SNEL system constructs a semiautomatic thesaurus, i.e. an automatic base thesaurus to be manually modified using prefixed rules (in such a way it is possible to avoid some traps of the automatic ones.) Moreover, SNEL indexes documents automatically, manages the file of indexed documents and is supplied with a retrieval module.

This system has been tested at IAC to construct a thesaurus for a collection of papers on Computational Mathematics.

For this purpose about 300 papers have been selected from various journals and manually indexed in the simplest way i.e. either by keeping the key words when they are already present or by choosing meaningful sentences from the summary.

About 1500 descriptors have been obtained (not all different) and used as input data for the construction of the thesaurus. These descriptors have been divided according to the date of issue of the paper, and then processed in order to see how the thesaurus develops.

In this way we have obtained a thesaurus of about 300 words; it has been used to test the system from the point of view of its performance in indexing and retrieval.

For indexing purposes it is enough to supply the system with the summaries of the chosen papers and, eventually, bibliographical information.

For the construction of the thesaurus, on the contrary, the presence of a qualified person is required for choosing papers and descriptions in the text, and for judging when the thesaurus is ready for use.

From the journals

Journal of Computer and System Science
International Journal of Computer Mathematic
Mathematic of Computation
Journal of the ACM
Communications of the ACM

of the years from 1972 to 1975, all the relevant papers have been chosen; their summaries have been punched for processing by the indexing module and most of the papers have been significantly indexed.

By considering the incompleteness of the collection used for the thesaurus construction, this result may be considered satisfying.

The following example shows how the system operates. Let us suppose we want to add the following paper to the collection:

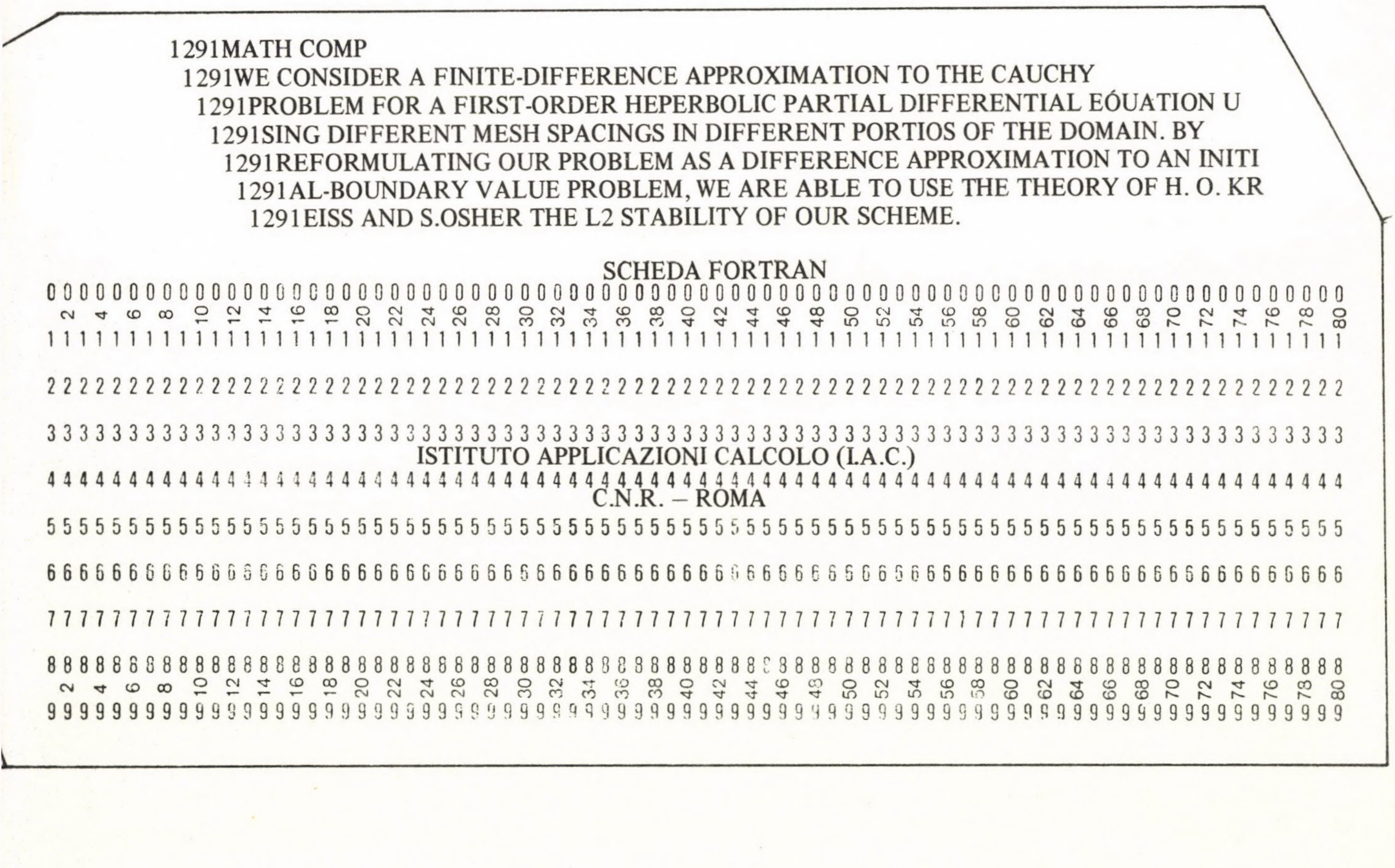
MATHEMATICS OF COMPUTATION, VOLUME 25, NUMBER 114, APRIL, 1971.

Stable Difference Schemes with Uneven Mesh Spacings

By Melvyn Ciment

We consider a finite-difference approximation to the Cauchy problem for a first-order hyperbolic differential equation using different mesh spacings in different portions of the domain. By reformulating our problem as a difference approximation to an initial-boundary value problem, we are able use the theory of H.O. Kreiss and S. Osher to analyze the L_2 stability of our scheme.

Its summary punched as follows is input without any other information to the indexing module which will index and store it.



```
1291MATH COMP
 1291WE CONSIDER A FINITE-DIFFERENCE APPROXIMATION TO THE CAUCHY
  1291PROBLEM FOR A FIRST-ORDER HEPERBOLIC PARTIAL DIFFERENTIAL EÓUATION U
   1291SING DIFFERENT MESH SPACINGS IN DIFFERENT PORTIOS OF THE DOMAIN. BY
    1291REFORMULATING OUR PROBLEM AS A DIFFERENCE APPROXIMATION TO AN INITI
     1291AL-BOUNDARY VALUE PROBLEM, WE ARE ABLE TO USE THE THEORY OF H. O. KR
      1291EISS AND S.OSHER THE L2 STABILITY OF OUR SCHEME.
```

SCHEDA FORTRAN

```
00000000000000000000000000000000000000000000000000000000000000000000000000000000
 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74 76 78 80
11111111111111111111111111111111111111111111111111111111111111111111111111111111
22222222222222222222222222222222222222222222222222222222222222222222222222222222
33333333333333333333333333333333333333333333333333333333333333333333333333333333
44444444444444444444444444444444444444444444444444444444444444444444444444444444
55555555555555555555555555555555555555555555555555555555555555555555555555555555
66666666666666666666666666666666666666666666666666666666666666666666666666666666
77777777777777777777777777777777777777777777777777777777777777777777777777777777
88888888888888888888888888888888888888888888888888888888888888888888888888888888
 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74 76 78 80
99999999999999999999999999999999999999999999999999999999999999999999999999999999
```

ISTITUTO APPLICAZIONI CALCOLO (I.A.C.)
C.N.R. – ROMA

When we wish to know which are the papers of the collection which deal with the solution of finete diference methods, we use the retrieval module.

```
1 IAC-RESEARCH REQUEST   IAC   1   A

SEEK ALL ARTIC MEETING FOLLOWING REQ.

* CONTAINING ALL KEYWORDS

BOUNDARY VALUE PROBLEM
DIFFERENTIAL EOUATIONS

                         RETRIEVED ARTICLES

        5028    CACM                    05    3821973
        5167    CACM                    05    3471972
        5227    JACM                    01     631971
        5065    CACM                    04    6081970
        4580    MATH COMP               25    6751971
   ⟹   1291    MATH COMP               25    2191971
        4100    MATH COMP               26    8591972
        4160    MATH COMP               26    6051972
         980    INTERN J COMPUT MATH     3     751971
```

As can be seen, the paper is indicated as relevant to the query.

The SNEL system has been implemented on a PDP 11/40 32K with 2 disk and 1 tape units and an IBM 1130, 16K with 1 disk unit.

The latter is the smallest configuration that allows the storage of a useful number of documents.

1. **The System Structure**

The system consist of portable independent elements, whose only connection is, as we shall see, the information pattern.

The elements can be grouped according to their functions in there modules: one for construction of the thesaurus, one to obtain the file of indexed papers and another for retrieval.

The first module constructs the thesaurus using descriptors freely allocated to the papers selected from the set of collected papers.

The second module indexes the papers by comparing each paper, or its summary, with a thesaurus which may be either the one obtained from the first module or another supplied by the user. It also manages the updating of the indexed papers file.

The retrieval module extracts those papers concerning topics described by terms from the thesaurus.

Such a structure implies a great flexibility i.e. it makes possible man-system interaction and permits the substitution of each system function.

Each module is divided into submodules, each one subdivided into blocks consisting of one or more units.

Each unit performs a complete function less and less complex with the decreasing of the level of the units, and which is independent from the others as the only vehicle of conection between the different units is given by the data structures which are external to the units. Each block can take out of such structures all the information it needs.

Thus it is easy to direct the system flow according to the results obtained.

Such an organization is also important from the point of view of portability since it allows the reformulation of the system according to the configutation requirements of different users.

For portability the choice of language is important; it must as widespread as possible and standardized.

We chose Basic Fortran Language with some facilities of Fortran IV which are generally implemented on minicomputers.

2. **The thesaurus construction module**

The module selects from the descriptors used for the chosen papers a set of words named terms. Such a set is sufficient to index each paper of the considered subset. If the subset is representative of the field, the set of terms is sufficient to index any paper of the field.

The representativity of the chosen subset can be evaluated by means of the following heuristic method: we choose a narrow subset and construct the appropriate thesaurus and then we increase the subset and update the thesaurus.

We repeat this operation until the difference between two successively obtained thesauri can be judged to be irrelevant.

This method can be easily applied thanks to the system structure which, as we have said is composed of independent and interactive bloks and gives the possibility of iterating the main blocks.

The module is divided in three submodules; the first obtains a provisional set whose elements will be named simple terms, the second allows their modification and the third forms the final set or thesaurus.

The connections between the submodules are shown in Diagram 1.

Note that to evaluate the representativity of the chosen subset it is sufficient to iterate the execution of the first submodule because the submodules and modules are suppiled only with information deducible from those present in the system.

2.1. The submodule for construction of simple terms

The submodele codifies the descriptors and subdivides them into simple words arranging them in a normalized list. Finally, it processes this list to obtain the thesaurus of simple terms.

The submodule is divided in four blocks.

1.) Acquiring data

2.) Codifying the descriptors and updating the data base

3.) Sentence scanning

4.) Formation of simple terms.

Diagram 2 shows the flow of information and the possibility of iteration between the various blocks.

2.1.1 Data acquisition

The input to this block consists of the descriptors and any other information concerning the paper (for instance data of issue) which is to be stored.

The user gives a code to each type of information and to a numeric identifier to each paper; this must always appear together with the relevant information.

We assume that each completed set of information is puched on a card. The block reads the cards and subdivides the information according to their code, updating the relevant files.

Descriptors are checked for formal correctness and the file of correct descriptors is the input for the next block.

2.1.2. Codifying the descriptors and updating the data base

The input is the file which associates the descriptors with the identifier of the paper in wich they appear. The block codifies each descriptor that does not yet appear in the data base and updates the data base. Moreover it updates a file named file of occurences, in which each descriptor's code is associated with all the identifiers of the papers in wich the descriptor appeard. The codifying of the descriptors is obtained as a function of the input order of the descriptor in the system. See fig.1.

2.1.3 **Descriptor scanning**

The input to this block consists of that part of the data base not yet processed and of a negative thesaurus, i.e. a list of words which must not belong to the thesaurus.

The block decomposes descriptors into simple normalized words, rejects all the words which appear in the negative thesaurus and associates the code of the descriptor to which the word belonged to each of those remaining.

The list thus obtained is alphabetically ordered.

We suppose that the user, according to his requirements, fixes the negative thesaurus taking into account above all the importance of rejecting those words which could bring noise or hide information.

An example is the preposition ”for”, which if not included in the negative thesaurus could form a term including words like ”formal” or ”formula”.

The negative thesaurus would be omitted in the case where the user a priori establishes which words to rejct. On the other hand, as we have already seen and will illustrate in the following, the system to modify the thesaurus includes an interactive module which, among its functions, can show the different words grouped under the same term.

2.1.4 **Formation of simple terms**

The block removes from the list obtained in the previous block a sublist by use of which it updates the thesaurus of simple terms and, with the remaining words, updates a file called the remainder file.

For a better understanding of how the block operates, let us suppose that it only works out a list consisting of the input list, of the exisitng thesaurus and of the remainder file.

From this list we take off the simple terms i.e.: words either with a prefixed occurrency in the list or stems of others in the list, or words having both qualifications(fig. 2.).The remaining words of the list form the remainder list (fig.2.1.).

The simple terms thesaurus file and the remainder file are obviously input and output files for the block and are the output of the whole submodule.

We must note that the information contained in the thesaurus simple terms file allows it to trace back to the descriptors of the term itself. The file is ordered alpabetically and terms are codified according to the order of their appearance in the file.

2.2. **Correction submodule**

The set of simple terms obtained from the previous process may be considered a sufficient thesaurus even if the operations performed for its construction are exclusively morphological ones with no other linguistic or semantic considerations.

We have already explained why we did not attempt a fully automatic procedure, choosing only the formation of a basic thesaurus and leaving to the user the possibility of improving it. For such a purpose the submodule examines all the information of the system, supplying easy lists from which the user deduces all the changes to bring in the thesaurus of simple terms.

The submodule is divided into two blocks:

1.) Proposals of possible changes

2.) Modifying the simple terms.

2.2.1. **Proposals of possible changes**

This block has to show the logical consistence of the produced terms and to suggest how to produce new terms or modify those already existing.

This information is contained in three lists.

The first list consists for each term of all its derivations in the data base and it is obtained from the examination of the data base and of the thesaurus (fig.3a).

The second list, obtained by examining the thesaurus of simple terms and the remainder file, shows how to match the components of the two files. It is formed by groups of words having a common stem with an user defined length (fig.3b).

For the third list the user must supply an a occurrences cut off value.

The following sublists are produced:

a.) All the descriptors not occurring in any term and whose occurrences in the data base exceed the cut off value are listed.

b.) The remainder, whose occurences exceed the cut off value and do not appear in the preceding sublist.

2.2.2. **Modifying the simple terms**

The foreseen modifications are the following

1.) To reduce the meanings of a term

2.) To create a new term for each meaning of a term

3.) To erase a term

4.) To add meanings to a term

5.) To gather two or more terms into one term

6.) To add a new term to the thesaurus

7.) To substitute a term for another one.

The block is asked for each one of these modifications through a code and a detailed indication of the terms and words concerred (fig.4.).

These requests are checked in the thesaurus and remainder files and in the data base and accepted if correct. On the contrary, the user has the possibility of correcting his data.

Original files are left unalterated to allow the user to test different variation of the thesaurus and to resume if necessary, the automatic procedure from where it left off.

The module , besides creating terms and remainder files modified according the user's choices stores some inferred information which is of interest to the next indexing module, such as the synonyms file.

2.3. **The submodule for thesaurus construction**

The thesaurus thus obtained consists of a list of simple words. By combining them it is possible to represent any concept, which is already expressed by the descriptors introduced in the system.

The aim of this submodule is to show those concepts in the theasurus which have appeared more frequently in the input descriptors and which cannot be described by only a term. The sequences of words thus obtained are called compound terms.

The process of formation of compound terms affects the set of simple terms in that if there are simple terms which are meaningful only when combined with other terms, they are erased after the formation of the relevant compound terms.

The input to the submodule consists of the simple terms file, the data base and the synonyms file; the output consists of the final thesaurus and the re-elaborated synonyms file.

Moreover, the submodules prepares, according to the indexing module , the thesaurus, synonyms (see 3.2) and negative thesaurus files.

3. **The indexing module**

This module indexes the collected documents, according to the system's thesaurus. It assigns to each document the terms that appear in the document text, either just they are or as stems. The relevant file is updated with the indexed documents.

Moreover, the module produces lists of words that may suggest modifications for the thesaurus.

This module, unlike the previous one, is fully automatic and the indexing is carried out strictly with the same criteria used in the thesaurus construction, taking into account all variations manually made.

The strong connection with the thesaurus construction module eases indexing and ret–rieval but does not affect their mutual independence; so it is possible to use the indexing module in a different environment.

Moreover, a procedure has been implemented that makes it possible to use whatever satisfying thesaurus the user may have acquired.

The module is divided into blocks which are linked by a directory file.

The first block codifies and normalises documents; the second one indexes each document and updates the file of document references; the third is optional and provides suggestions for thesaurus modifications.

3.1 **Codifying and normalisiting texts**

The document texts or their summaries, together with a negative thesaurus, are the input.

The negative thesaurus may either result from the correction module or it my be chosen by the user, or both.

The user assigns to each document a numerical identifier. The document texts are punched on cards, each card bearing the identifier of the relevant document.

Each text is supplied with a bibliographic reference card. The block codifies and stores each text; moreover it prepares for the updating of the file of indexed documents.

Then it takes every sentence from each text, i.e. every sequence of words between two punctuation marks, and decomposes each sentence into simple normalized words.

It associates with each word a code that is a function of

a) the document code
b) the relative position of the sentence
c) the relative position of the word in the sentence.

The words are then arranged in alphabetic order; those which appear in the negative thesaurus are rejected and the remainder are stored.

The block, by using the directory file, iterates the whole procedure in order to permit the processing of any number of documents.

Document indexing

The input is the file produced by processing the previous block and the following files:

1) Vocabulary – all the words that constitute the thesaurus are arranged in alphabetical order each word is associated with a numerical code.

2) Specifications – it is indicated whether a word of the vocabulary is a term or a component of a term, or both.

3) Phrases – The codes of the compound terms of the thesaurus are recorded.

4) Synonyms – The synonyms of the vocabulary words are alphabetically recorded.

This is not a one to one relation because a word may have more than one synonym or none.

A term of the thesaurus is considered to be a descriptor of a document when it satisfies at least one of the following qualitifications.

1) It is a simple term which appears in the document
2) It is a simple term which has a synonym in the document
3) It is a stem of a word which appears in the document
4) It is a compound term and its components appear, themselves or their synonyms, in the same phrase of the document. Moreover, the sequence of component words must be the same in the document as in the term.

Fig. 7. shows an abstract and the assigned terms.

The block updates the file of indexed documents using those documents having a number of descriptors greater than a prefixed one and rejects the others.

The rejected documents are shown to the user, who decides whether they are relevant and, in this case, he may add them within the file.

3.3 Candidate terms

The block may be optionally processed: its purpose is to show those words of documents which are not terms but have the qualifications to be examined by the user who decides on their inclusion in the thesaurus.

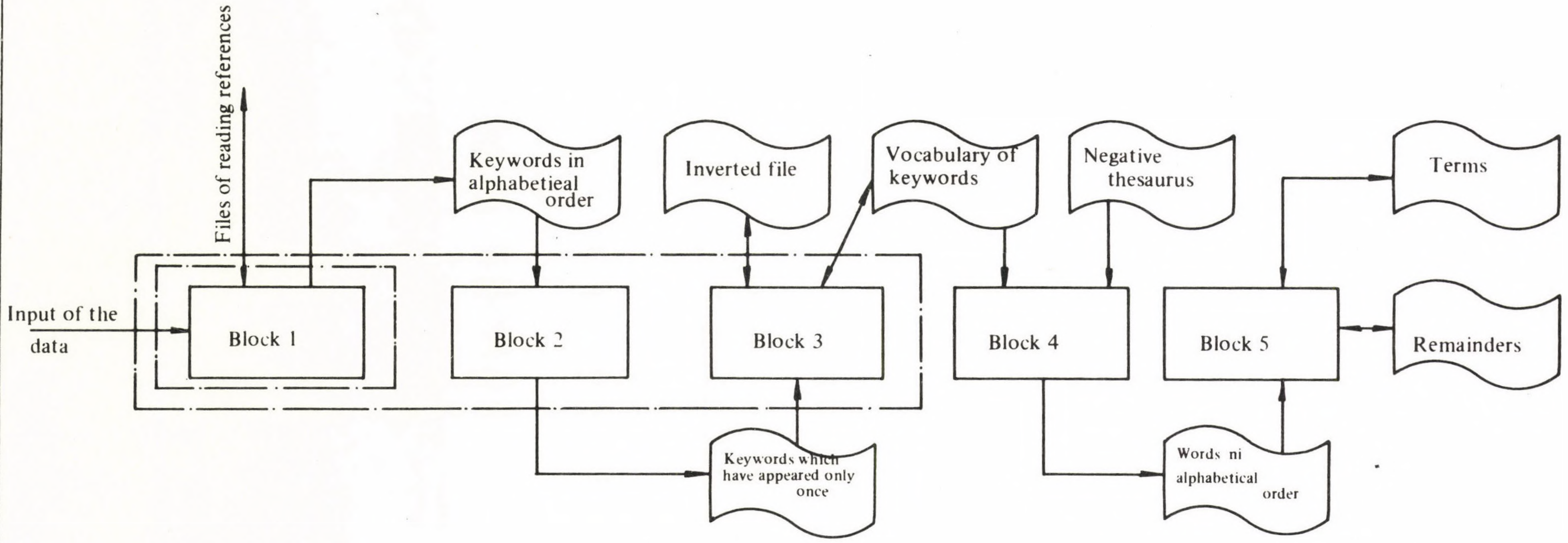
The displayed words are those whose occurences are in a suitable range, and those which are stems of some thesaurus term(fig.8.).

If these words are either real stems or, according to the user's opinion, relevant for the considered field, they may be inserted to modify or to increase the previous thesaurus (fig.9.).

Moreover, the negative thesaurus may be updated for decreasing the list in *fig. 9.*

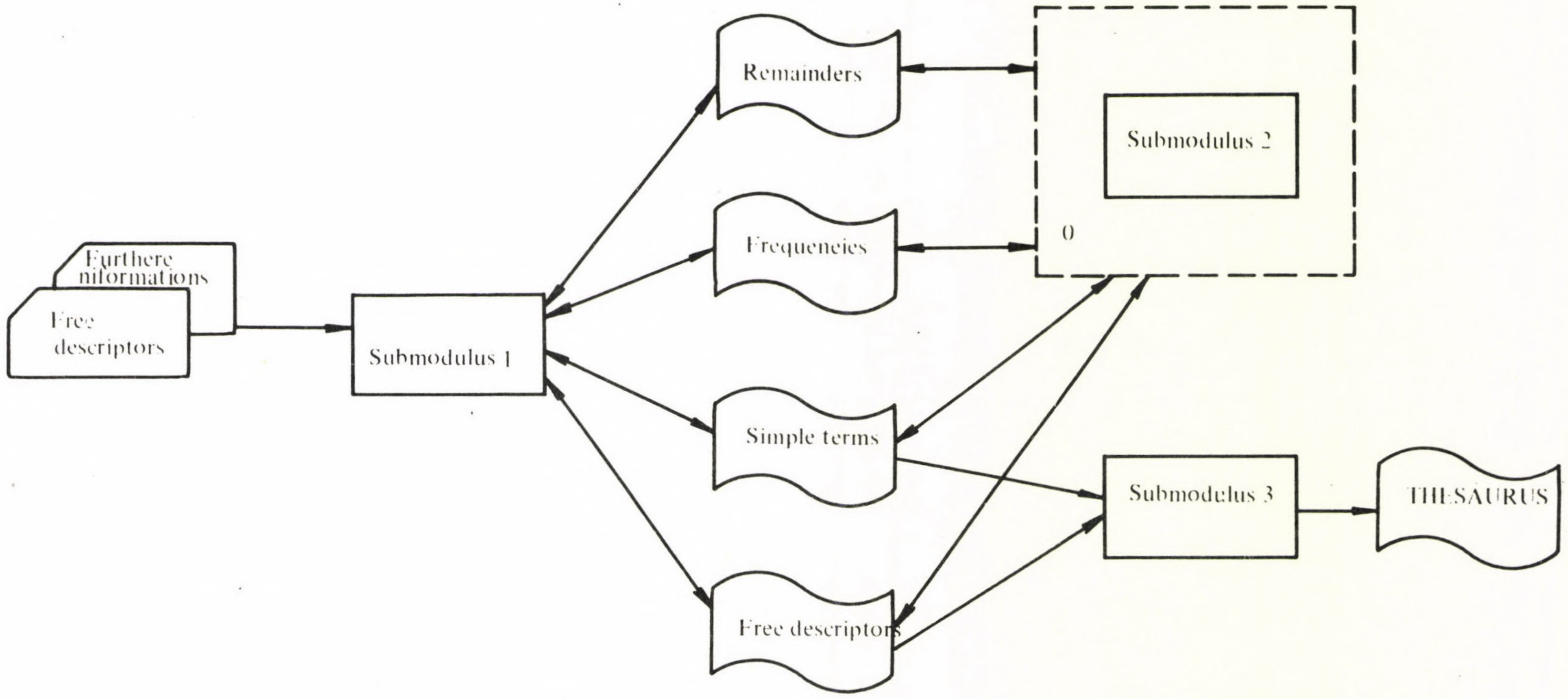
Acknowledgements

We are grateful to Mrs. Robusti and Mrs. Di Sarra for their kind and useful assistance.


Files of reading references
Input of the data
Block 1
Keywords in alphabetieal order
Block 2
Inverted file
Block 3
Vocabulary of keywords
Keywords which have appeared only once
Negative thesaurus
Block 4
Words ni alphabetical order
Terms
Block 5
Remainders

SCHEMA 2

"E" indicates that the block is repretitive



SCHEMA 1

0 Submodulus 2 is optional

FREE DESCRIPTORS

ALGORITHMIC SOLUTION
ALGORITHMS
ALL INTEGER ALGORITHM
APPROXIMATION
APPROXIMATIONS
BOOLEAN EQUATIONS
BRANC AND BOUND
CHANGE MAKING
CHANGE MAKING PROBLEM
COMPLEMENTARY PROBLEM
COMPUTATIONAL RESULTS
CONCURRENT PROBLEM
CONSTRAINTS
DATA FITTING
DISCRETE OPTIMIZATION PROBLEM
DISCRETE PROBLEMS
DISTRIBUTION
DYNAMIC PROGRAMMING
FEASIBLE SOLUTION
FUNCTIONAL APPROXIMATION
GOMORY CUT
HEURISTIC SEARCH
IMPLICIT ENUMERATION
INEOALITY
INEOUALITY CONSTRAINTS
INTEGER NONLINEAR PROGRAMMING
INTEGER PROGRAMMING
INTERACTIVE GRAPHICAL DISPLAY
ITERATION
LEAST SOARES
LEMKE S ALGORITHM
LINEAR PROGRAM
LINEAR PROGRAMMING
MANAGEMENT SCIENCE
MATHEMATICAL PROGRAMMING
MINIMAX APPROXIMATION
MOMENTS
MONTECARLO
MULTIVARIATE NORMAL DISTRIBUTION
MULTIVARIATE NORMAL GENERATOR

FREE DESCRIPTORS

NONLINEAR EQATIONS
NONLINEAR PROGRAMMING
NONLINEAR REGRESSION
NORMAL DISTRIBUTION
OPERATIONS RESEARCH
OPTIMAL SOLUTION
OPTIMIZATION
PARTIAL SEARCH
PROBABILITY
PSEUDU BOOLEAN FUNCTION S
QUADRATIC PROGRAM
QUADRATIC PROGRAMMING
QUADRATIC PROGRAMMING METHODS
RANDOM NUMBER GENERATOR
RANDOM NUMBERS
RANDOM SEARCH
RANDOM VARIABLES
REVISED SIMPLEX ALGORITHM
SCHDULING MODELS
SET COVERING PROBLEM
SIMPLEX METHOD
SIMULATION
SPLINE FUNCTIONS
STATISTICS
ZERO ONE PROGRAMMING
ZERO ONE VARIABLES

Fig.1.

SIMPLE TERMS

ALGORITHM
APPROXIMATION
BOOLEAN
CHANGE
CONSTRAINTS
DISCRETE
DISTRIBUTION
EQUATIONS
FUNCTIONS
GENERATOR
INEQUALITY
INTEGER
LINEAR
MAKING
METHOD
MULTIVARIATE
NONLINEAR
NORMAL
NUMBER
ONE
OPTIMIZATION
PROBLEM
PROGRAM
QUADRATIC
RANDOM
SEARCH
SIMPLEX
SOLUTION
VARIABLES
ZERO

Fig. 2.

REMAINDERS

AND
BOUND
BRANCH
COMPLEMENTARY
COMPUTATIONAL
CONCURRENT
COVERING
CUT
DATA
DISPLAY
DYNAMIC
ENUMERATION
FEASIBLE
FITTING
FUNCTIONAL
GOMORY
GRAPHICAL
HEURISTIC
IMPLICIT
INTERACTIVE
ITERATION
LEAST
LEMKE
MANAGEMENT
MATHEMATICAL
MINIMAX
MODELS
MOMENTS
MONTECARLO
OPERATIONS
OPTIMAL
PARTIAL
PROBABILITY
PSEUDO
REGRESSION
RESEARCH
RESULTS
REVISED
SCHEDULING
SCIENCE

SET
SIMULATION
SPLINE
SQUARES
STATISTICS

Fig.2.1.

a) POSSIBLE FUSIONS

FUNCT	FUNCTIONAL
	FUNCITONS
OPTIM	OPTIMAL
	OPTIMIZATION

b) DERIVATIONS

ALGORITHM	ALGORITHMIC
	ALGORITHMS
APPROXIMATION	APPROXIMATIONS
METHOD	METHODS
NUMBER	NUMBERS
PROBLEM	PROBLEMS
PROGRAM	PROGRAMMING

c) KEY-WORDS WHOSE FREGUENCY ALLOWS THEM TO BE INSERTED IN THE THESAURUS

13 BRANC AND BOUND
57 INTERATION
59 LEAST SQUARES
89 OPERATIONS RESEARCH
85 REGRESSION

Fig. 3.

```
OP4
    42OPTIMIZATION
    91OPTIMAL SOLUTION
    91 OPTIMAL
OP6
    100BRANCH
      13BRANCH AND BOUND
OP6
    100AND
      13BRANCH AND BOUND
OP6
    100BOUND
      13BRANCH AND BOUND
OP6
    100LEAST
      59LEAST SQUARES
OP6
    100SQUARES
      59LEAST SQUARES
OP6
    100OPERATIONS
      89OPERATIONS RESEARCH
OP1
    46PROGRAM
    63LINEAR PROGRAM
   101QUADRATIC PROGRAM
    63LINEAR
OP8
```

Fig. 4.

```
     NEGATIV THEASURUS
 AUTOMATICALLY PRODUCED

 PROGRAM
```

Fig. 5.

THESAURUS

ALGORITHM
APPROXIMATION
BOOLEAN
BRANCH AND BOUND
CHANGE MAKING
CONSTRAINTS
DISCRETE PROBLEM
DISTRIBUTION
EQUATIONS
FUNCTIONS
GENERATOR
INEQUALITY
INTEGER
INTEGER PROGRAMMING
LEAST SQUARES
LINEAR
METHOD
MULTIVARIATE NORMAL
NONLINEAR
NONLINEAR PROGRAMMING
NORMAL
NORMAL DISTRIBUTION
OPERATIONS RESEARCH
OPTIMIZATION

OPTIMAL

PROBLEM
PROGRAMMING
QUADRATIC
QUADRATIC PROGRAMMING
RANDOM
RANDOM NUMBER
SEARCH
SIMPLEX
SOLUTION
VARIABLES
ZERO ONE

Fig. 6. – Theasurus and synonyms

```
ART N.1002                                      READING REFERENCES
                                  A QUADRATIC PROGRAMMING ALGORITHM IS DESCRIBED FOR USE WITH THE MA
G DIAGONAL METHOD OF NONLINEAR REGRESSION WITH LINEAR CONSTRAINTS:

ALGORITHM,CONSTRAINTS,LINEAR,METHOD,NONLINEAR,PROGRAMMING;QUADRATIC PROGRAMMING.

ART N.200                                       IINDICAZIONI BIBILIOGRAFICHE
                                  CERTAIN NONLINEAR MINIMAX APPROXIMATION PROBLEM ARE CHARACTERIZED
BY PROPERTIES WHICH PERMIT THE APPLICATION OF SPECIAL ALGORITHMSí MAINLY BASED ON THE EXCH
ANGE ALGORITHMS OF REMES(1934,1935), FOR THEIR SOLUTION. IN THIS PAPER THE APPLICATION TO
PROBLEMS OF THIS TYPE OF A GENERAL NONLINEAR ALGORITHM DUE TO OSBORNE AND WATSON (1969) IS
 CONSIDERED. EXAMPLES ARE GIVEN TO ILLUSTRATE THAT THIS ALGORITH CAN GIVE SATISFACTORY RE
SULTS AND, IN PARTICULAR, CAN SUCCESSFULLY SOLVE PROBLEMS WHICH LEAD TO DIFFICULTIES WITH THE
MORE CONVETIONAL SPECIALIST METHOD.

ALGORITHM,APPROXIMATION,METHOD,NONLINEAR,PROBLEM,SOLUTION.
```

Fig. 7.

```
WORDS FROM DOCUMENTS, KEY-WORD  STEMS
        1 FUNCTION
        1 NORM
WORDS WHICH APPEAR MORE OF THAN  "n" TIMES IN THE DOCUMENTS
        2 APPLICATION
        2 BASED
        2 DESCRIBED
        2 RESULTS
        2 SATISFACTORY
        2 SOLVE
        2 USE
```

Fig. 8.

THESAURUS

LAGORITHM
APPROXIMATION
BOOLEAN
BRANCH AND BOUND
CHANGE MAKING
CONSTRAINTS
DISCRETE PROBLEM
DISTRIBUTION
EQUATIONS
FUNCTION
GENERATOR
INEQUALITY
INTEGER
INTEGER PROGRAMMING
INTERACTIVE
LEAST SOUARES
LINEAR
METHOD
MULTIVARIATE NORMAL
NONLINEAR
NONLINEAR PROGRAMMING
NORMAL
NORMAL DISTRIBUTION
OPERATIONS RESEARCH
OPTIMIZATION
OPTIMAL
PROBLEM
PROGRAMMING
QUADRATIC
QUADRATIC PROGRAMMING
RANDOM
RANDOM NUMBER
SEARCH
SIMPLEX
SOLUTION
VARIABLES
ZERO ONE

Fig. 9. – Modified thesaurus

BIBLIOGRAPHY

[1] Anzaldi, C. – Anzelotti, P. – Mirri Passerini, L. – Testi V.:
Un package di subroutines Fortran per standardizzare l'uso degli accessi random all'unita disco.
Rapp. IAC (1976).

[2] Anzaldi, C. – Mirri Passerini, L.: Un sistema di documentaione per minicalcolatori.
Applicazioni del Calcolo – Galligani, I. Ed. (1975).

[3] Anzaldi, C. –Mirri Passerini, L.: Panoramica de linguaggi indice.
Rapp. IAC. (1092).

[4] Fangmeyer, H. – Lustig, G.: Experiments with the Cetis Automatic Indexing System.
Intern. Atomic Energy Agency,
Vienna (1970).

[5] Gilchrist, A.: The Thesaurus in Retrieval. Aslib, Londra (1971).

[6] Koymen, K. – Tos: A text organizing System. Information Processing & Management (1975).

[7] Palazzi, A.: I1 sistema RDS per il reperimento automatico della documentazione in siderurgia.
Rivista AIRO (1968).

[8] Salton, G.: Automatic Information Organization and Retrieval,
McGraw–Hill, New York (1968).

[9] Samuelson, K. Ed.: Mechanized Information Storage, Retrieval and Dissemination.
North Holland, Amsterdam (1968).

[10] Unesco: Guidelines for the Establishment and Development of Monolingual Thesaurus.
Parigi (1973).

A BNF SYNTAX FOR THE QUERY LANGUAGE FORAL WITH A DISCUSSION OF POSSIBLE MODIFICATIONS

Ilona Fidrich*

Theoretical Laboratory, Institute for Co-ordination of Computer Techniques

Budapest Hungary

FORAL was introduced as a representation independent accessing language for the Data Independent Accessing Model (DIAM) first published in [5].

DIAM has multiple self-sufficient levels of abstraction for describing and solving problems concerned with information systems. At the interface between real-world information and the overall descriptive model the entity set model is specified as a structured model of the real-world information. The entity set model together with an accompanying representation independent accessing language for describing the accessing of the entity set descriptions are tailored to be a selt-sufficient level where the end-user can discuss, specify and solve the problems of information structuring independently of extraneous details of implementation.

A new version of DIAM, DIAM II was introduced in [6]. It utilizes a modified binary relationship between names of things as its basic user-system interface form, and a new binary relationship language, FORAL for both data description and data access.

In DIAM II a new level, the end-user level was added on top of the four original levels of DIAM. It provides for many user views of data including declaration of different synonyms, reports and other structures for use by different users. The entity set model was replaced by an infological level which is the consistent, relatively non-redundant central model of the user's real world.

The DIAM II infological level is constructed using two basic types of building blocks:

– entity sets which are sets of entities of the real world and
– fact sets which provide a means for describing entities.

Since most entities cannot be stored in an information system, storable entities are used to stand for them. They are called identifier values and their collections are called identifiers.

Fact sets are sets of associations that allow us to state desciptive facts about entities. As with entities, actual facts cannot be stored in the system; we can only store things that can stand for them. They are called basic facts. A basic fact is represented by an association of names of two entities. They show how to drive attribute values from facts. If our point of attention is on one entity of a basic fact then we can get an attribute value of this entity by following the

* This work was mainly carried out while the author was visiting the University of Stockholm Research Group CADIS.

association to the value at its other end. This is expressed by the concept "context" used in FORAL.

The original FORAL was reviewed and a new version, FORAL II was developed for DIAM II. Its output statement is described in [1].

In the first part of this report the output statement defined in [1] is summarized and its syntax is specified using BNF. Some remarks concerning the output statement are given in the second part. The third part contains some modifications to the output statement described in the first part.

The figures referred to in any part of this report are given at the end of it.

I. The FORAL output statement according to [1]

In this part of the report the FORAL output statement is briefly described essentially as it defined in [1]. The features which increase the similarity between FORAL and English are not taken into account. Similarly, some convenience features, given in section IV. of [1], are left out of consideration.

The syntax is given in Backus-Naur form. A syntactical entity identical with one of [1] is usually referred to by the same name as that of [1]. Each syntactic rule is provided with a number for reference to the rules from the textual parts of the report. A group of logically connected rules is followed by some explanations, mostly of a semantic nature.

The main point of the semantics is given by a rough description of an algorithm performing the transfer of the information specified by the output statement.

1. <output statement>: : = <file specification > : <format specification>.

An output statement describes the transfer of one "dose" of information. The first part of the output statement specifies the destination and the sources of the transfer. The information to be transferred is specified by the second part of the statement.

```
2. <file specification> : : = <destination specification> |
                              <file specification><source specification>

3. <destination specification> : : = <destination indicator> <output file name>

4. <destination indicator> : : = output | temporary

5. <source specification> : : = from <internal file name>
```

The destination of the transfer should be specified explicitly in the output statement. In an unsophisticated interpretation the output destination indicator indicates an actual output under the title specified by the output file name, while the temporary one indicates a transfer to the file that can be used as an internal one and has the name shown in the destination specification.

If no source specification is given in the file specification, then implicitly the central data base is assumed as the source.

6. <format specification> : : = <forprint specificaton> | <forprint specification>
   <where overlay specification>

The mandatory part of the format specification specifies the headline (in some columns possibly specifying a secondary one) in the case of an actual output, and the information to be transferred in both cases.

7. <forprint specification> : : = <entity set print clause> | <forprint specification>
   <for attribute print clause>

8. <entity set print clause> : : = <initial context indicator> | <initial context indicator>
   <print clause>

9. <initial context indicator> : : = <initial entity set specification> |
   <initial entity set specification>(<where qualifier>)

10. <initial entity set specification> : : = <entity set name> |system

11. <where qualifier> : : = where <condition>

12. <print clause> : : = print < field specification> |
    <print clause>,<field specification>

13. <field specification> : : = <attribute expression> | <attribute expression><renamer>

14. <renemer> : : = called <field name>

15. <for attribute print clause> : : = <removed context indicator> |
    <removed context indicator><print clause>

16. <removed context indicator> : : =
    <additional entity set specification> |
    <additional entity set specification>(<where qualifier>)

17. <additional entity set specification> : : = for <context remover>

18. <context remover> : : = <attribute name> | <attribute name><called phrase>

19. <called phrase> : : = called <new name>

The forprint specification should contain one and only one entity set print clause, the only mandatory part of which is the initial entity set specification. Considering the system as the name of the entity set, the only entity of which is an identifier value – the name of the system – it can be stated that an initial entity set specification specifies an entity set, which serves as the basis of the transfer.

If an initial, or additional, entity set specification is not followed by a where qualifier in the output statement, then the whole entity set is specified for the transfer. Otherwise, the subset having the same name as the whole entity set is asumed to be specified for the transfer. This subset consists of those and only those entities of the originally specified set which satisfy the condition given by the where qualifier.

Once an entity set is specified for the transfer, in the case of an actual output, its name becomes part of the headline and (if the entity set is not an identifier) the names of its identifiers become parts of the secondary one over the columns corresponding to the specified entity set.

A specified entity set can obviously be considered as the indicator of its context, which consists of those and only those entity sets which are connected to it by fact sets. Having a context indicator and the name of a fact set about it, one of its (direct) attributes is specified: it is the entity set in the context, which is connected to the context indicator by the named fact set and has the same name as the fact set. An indirect attribute of a context indicator is an entity set, which is connected to it by a sequence of more than one fact sets.

In Figure 1 some entity sets are presented showing the relations between them. If the specification of the entity set E1 is followed by a where qualifier in the output statement then its subset called E1 becomes the context indicator. The entity sets called A1, E2 (which is in fact a subset of the entity set called by the same name) and A2 are in the context of the entity set E1 – they are (direct) attributes of it. If the context indicator is the entity set called A1, then the entity subset E1 (called in this case AA1) is in the context, but neither the complementer set of AA1 nor the entity set A3 is in it. The entity set A3 is one of the indirect attributes both of the entity set A1 and the subset E2 of the entity set E2.

All field specifications of a print clause are related to the initial, or removed, context indicator preceding the given print clause. A field specification specifies a (possibly empty) set of values for the transfer by an attribute expression over the values of the direct and/or indirect attributes of the given context indicator. The field specification also specifies the names of the specified set of values, explicitly by a renamer or implicitly by the given attribute expression, and possibly a set of its secondary names. In the case of an actual output, the names of the set of values become parts of the headline and the secondary names become those of the secondary one over the columns corresponding to the specified set of values.

Having a context indicator on hand, an entity set, in the context of which it is contained, can be specified for a context indicator by an additional entity set specification.

If the context remover of an additional entity set specification appoints an attribute, the name of which differs from that of the context indicator preceding the one on hand, or, though it shows the name of the preceding context indicator, renames it by a called phrase, then the additional entity set specification specifies the attribute of the context indicator on hand which has the specified name. A context remover of this type is called a context remover forward throughout this report. In this case the explanation of the output statement can be continued at the second paragraph of the text following rule 19.

If the context remover appoints an attribute, the name of which is the same as that of the context indicator preceding the one on hand, and no called phrase is contained in it, then the context indicator preceding the one on hand is the entity set, specified by the given additional entity set specification. In this case the explanation of the output statement can be continued at the fourth paragraph of the text following rule 19.

In Figure 2 the entity set CI1 is the first context indicator. The additional entity set specification

for A1

specifies the entity set A1, which is restricted by a where qualifier to the entity subset called A1. Having the subset A1 as the context indicator on hand, the additional entity set specification

for CI1 called A2

specifies the entity set A2 (or perhaps a subset of it restricted by a where qualifier) that is specified for the transfer. In the case of having the additional entity set specification

for CI1

instead of the previous one, the entity set CI1 is the specified one – but not for the transfer. This means that, though the entity set CI1 becomes the context indicator, no information will be transferred about its entities (but information can be transferred, of course, about its attributes as the effect of a print clause following the given additional entity set specification).

The information specified by a forprint specification for the transfer consists, in fact, of the identifier values of some entities of the entity sets shown in the given forprint specification. The entities, the identifier values of which should be transferred, and the sequence of taking them, are specified by the forprint specification as well and can be defined in the following way.

It is quite clear that the above explanations describe a method for creating a directed ordered graph on the basis of a given forprint specification. This grahp, called the schema of the forprint specification throughout this report, is a tree, nodes of which are of two types. The basic nodes of the tree represent the context indicators specified in the given forprint specification, representing a context indicator as many times as context removers forward are shown in the forprint specification concerning it. The branches of the schema of a forprint specification represent the fact sets, specified by the context removers forward, in the order specified by the given forprint specification. Some additional nodes can be connected to a basic node in the following way. If the entity set represented by a basic node is not an identifier, then the additional nodes representing the identifiers of the entity set are connected to the basic node. If a specification of the entity set represented by a basic node is followed by a print clause in the forprint specification, then the additional nodes representing the sets of values specified by the field specifications of the print clause are connected to the basic node. These additional nodes are connected to the basic node by the branches representing the fact sets also specified by the field specifications.

In Figure 3.1 the schema of the forprint specification

A (<u>where</u> . . .) <u>print</u> D,E <u>for</u> F <u>for</u> G (<u>where</u> . . .)
<u>print</u> H <u>for</u> F <u>called</u> I <u>print</u> J <u>for</u> G <u>for</u> F
<u>print</u> K <u>for</u> L <u>for</u> G <u>print</u> M

is shown supposing that all context indicators are identifiers except the initial one, which is identified by the entity sets B and C. (The basic nodes of the graph are marked by double circles in Figure 3.1.)

The schema of a forprint specification is represented in the headlines of the output file in the case of an actual output in a linearized form. This linearized form of the schema determines the sequence in which the entity sets should be taken into account during transfer, so that it is defined by the semantics of the forprint specification.

The parenthesized expression

(A (B,C,D,E,F(G(H,I(J)),K,L(G(M))))

is the form of the tree shown in Figure 3.1, linearized in the way prescribed by the semantics of the forprint specification.

Now the situation is as follows:

– on the syntactic basis of the forprint specification its schema is defined;

– on the semantic basis of it a linearized form of the schema is known.

Having a tree and a linearized form of it, a traversal algorithm is defined which takes the nodes of the tree in the sequence, prescribed by the linearized form of the tree. The nodes of the schema of a forprint specification are taken in the sequence prescribed by the semantics of the forprint specification by one of the well-known algorithms of [2] for the preorder traversal of trees.

The algorithm for the traversal of trees defines the "macro structure" of the transfer algorithm only, since it is defined on the level of the entity sets. The information to be transferred is in fact the one concerning the entities, so the transfer algorithm should be defined on their level. The information specified by a forprint specification can be transferred by applying the same traversal algorithm to a set of trees of entities determined on the basis of the schema of the given forprint specification.

For each entity of the entity set, represented by the root of the schema of the forprint specification, a tree should be traversed, the root of which is the given entity. If an entity is taken as a root of a traversal tree, then its traversal forest is specified in the following way.

It consists of those and only those traversal trees the roots of which are the entities connected to the given root by the fact sets represented by the branches of the schema of the forprint specification.

In Figure 3.2 two trees are shown that can be specified as traversal ones on the basis of the schema shown in Figure 3.1. It should be mentioned that entities represented by nodes having the same name are identical.

The information specified by a given forprint specification is transferred by traversing the entity trees specified above, and performing the operation

> "if the entity on hand is an
> identifier value, then transfer it"

for each entity traversed.

Datails about the attribute expressions are given by the syntactical rules and in the explanations below.

20. <attribute expression>: : = <one-value expression> |
<multivalue expression> | <set-value expression>

An attribute expression generally contains some entity sets as its variables that are direct or indirect attributes of the context indicator for the moment. An attribute expression specifies a (perhaps empty) set of values for each entity of the context indicator.

If an attribute expression specifies not more than one value for one given entity of the context indicator, then it is called a one-value expression. If for one entity of the context indicator the attribute expression may specify more than one value of the same type that can be considered as the entities of one entity set, then it is called a multivalue expression. A set-value expression specifies a set of values of more than one subset for one entity of the context indicator. The subsets may differ regarding the number and the type of elements, but the elements of one subset are of the same type.

In the case of the sets shown in Figure 4, A and 2+B are one-value expressions, C is a multivalue one, and the attribute expression having the value of the set consisting of A, B and C is a set-value one.

21. <one-value expression> : : = <one-value numeric expression> |
<one-value string expression> |
<one-value expression of abstract type>

The type of a one-value expression (as that of a multivalue one) is determined by the type of the values specified by it for the entities of the context indicator. The values of a numeric or a string expression can be considered as the entities of an identifier, the identifier values of which are of numeric or string type respectively. The values of an expression of abstract type are the entities of an entity set which is not an identifier.

```
22. <one-value numeric expression> : : = <one-value numeric term> |
              <sign symbol><one-value numeric term> |
              <one-value numeric expression>
              <additive operation symbol>
              <one-value numeric term>

23. <one-value numeric term> : : =
              <unsigned numeric constant> |
              <one-value numeric attribute specification> |
              <one-value numeric term>
              <multiplicative operation symbol>
              <one-value numeric term> |
              <one-value numeric function specification> |
              (<one-value numeric expression>)

24. <unsigned numeric constant> : : =
              <unsigned integer number> |
              <unsigned real number>

25. <one-value numeric attribute specification> : : =
              <(direct) one-value numeric attribute specification> |
              <indirect one-value numeric attribute specification>

26. <(direct) one-value numeric attribute specification> : : =
              <one-value numeric attribute name>

27. <indirect one-value numeric attribute specification> : : =
              <numeric end attribute specification>
              <one-value attribute chain specification>

28. <numeric end attribute specification> : : =
              <numeric entity set name> |
              (<one-value numeric expression>)

29. <one-value attribute chain specification> : : =
              <one-value attribute connector> |
              <one-value of phrase>
              <one-value attribute chain specification>

30. <one-value attribute connector> : : =
              <one-value of phrase> of
              <inverse of a one-value attribute>

31. <one-value of phrase> : : =
              of <inverse of a one-value attribute> |
              of <inverse of a one-value attribute>
              (<where qualifier>)
```

32. <inverse of a one-value attribute> : : = <attribute name>

33. <one-value numeric function specification> : : =
<name of a one-value numeric function on a multivalue term><multivalue term> |
<name of a one-value numeric function on a multivalue numeric term>
<multivalue numeric term>

34. <name of a one-value numeric function on a multivalue term> : : = count

35. <multivalue term> : : =
<multivalue numeric term> |
<multivalue string term> |
<multivalue expression of abstract type>

36. <name of a one-value numeric function on a multivalue
numeric term> : : = sum | maximum | minimum | average

37. <multiplicative operation symbol> : : = x | /

38. <sign symbol> : : = + | –

39. <additive operation symbol> : : = + | –

The structure of a one-value numeric expression determined by rules (22)–(39) seems to be similar to that of the arithmetic expression determined in the usual way. However, a great difference is hidden in the variables of the expressions of the two types. While the values of the variables are considered as given in an arithmetic expression, the method of retrieving them is specified in a one-value numeric attribute expression itself.

The values of the variables of a one-value numeric expression (as those of the expressions of other types) are specified by the attribute specifications. An attribute specification specifies a direct or an indirect attribute of the context indicator. A direct attribute is specified by its name. An indirect attribute specification consists of the specification of the end attribute and that of an attribute chain leading from the end attribute to the context indicator.

The value, specified by a (direct) one-value numeric attribute specification for an entity of the context indicator, is the numeric identifier value connected to the given entity by the fact showing a 1-1 or a many-1 relation, the name of which is specified by the given attribute specification.

The value specified by an indirect one-value numeric attribute specification for an entity of the context indicator can be determined in the following way.

If an indirect one-value numeric attribute specification contains a numeric end attribute specification that shows more than one numeric entity set, then it can be reduced to an expression over the indirect one-value numeric attributes, specifications of which contain the numeric end attribute specifications that show one numeric entity set only. This reduction can be achieved by the successive application of the following rule.

If a given

<indirect one-value numeric attribute specification> : : =
    (<one-value numeric expression>)
    <one-value attribute chain specification>

and the

<one-value numeric expression> : : =
    <one-value numeric expression 1>
    <operation symbol>
    <one-value numeric expression 2>

where the

<one-value numeric expression 2>
shows one entity set only, then the given
<indirect one-value numeric attribute specification> : : =
    <indirect one-value numeric attribute specification 1>
    <operation symbol>
    <indirect one-value numeric attribute specification 2>

where the

<indirect one-value numeric attribute specification 1> : : =
    (<one-value numeric expression 1>)
    <one-value attribute chain specification>

and the

<indirect one-value numeric attribute specification 2> : : =
    (<one-value numeric expression 2>)
    <one-value attribute chain specification>

Consequently, it is sufficient to determine the value specified by an indirect one-value numeric attribute specification, the numeric end attribute specification of which shows one entity set only. This value can be determined by the help of the same traversal algorithm as used in the case of the interpretation of the whole forprint specification.

The shema of an indirect one-value numeric attribute specification is a linear path leading from the node representing the entity set specified by the numeric end attribute specification to the node representing the context indicator. The branches of the path represent the fact sets, the names of which are specified by the given indirect one-value attribute specification. The branches of the shema of an indirect one-value numeric attribute specification represent the fact sets that show the inverses of the 1-1 or many-1 relations.

Though the schema of an indirect one-value numeric attribute specification is a linear path only, a traversal tree of entities is specified on its basis for each entity of the entity set specified by the numeric end attribute specification. The values of an indirect one-value numeric attribute for an entity of the context indicator are the roots of those and only those traversal trees of entities that have a path leading to the given entity of the context indicator.

Taking into account the type of the fact sets represented by the branches of the schema of an indirect one-value numeric attribute specification it can be stated that not more than one traversal tree of entities exists, the root of which is the value of the specified indirect one-value numeric attribute.

In Figure 5.1 the schema of the indirect one-value attribute specification

E <u>of</u> D <u>of</u> C <u>of</u> B <u>of</u> A

is shown while Figure 5.2 shows the traversing trees of entities specified on its basis in the case shown in Figure 5.3. The value of the specified indirect one-value attribute for the entity a3 is marked with a double circle in Figure 5.2.

The value of an indirect one-value numeric attribute defined in the above way is a numeric one, since the entities of the entity set specified by the numeric end attribute specification are numeric identifier values.

The value of a one-value numeric expression can be determined also in case some of its variables have no value. For this it is sufficient to take 1 as the default value of the variables for the multiplicative operations and 0 as that for the additive ones.

40. <one-value string expression> : : = <string constant> |
    <one-value string attribute specification>

41. <string constant> : : = '<sequence of characters>'

42. <one-value string attribute specification> : : =
    <(direct) one-value string attribute specification> |
    <indirect one-value string attribute specification>

43. <(direct) one-value string attribute specification> : : =
    <one-value string attribute name>

44. <indirect one-value string attribute specification> : : =
    <string end attribute specification>
    <one-value attribute chain specification>

45. <string end attribute specification> : : =
    <string entity set name>

A one-value string expression deffined by rules (40)–(45) consists of either a string constant or a single string variable.The value of the variable of a one-value string expression is specified by a one-value string attribute specification in a way analogous to that of the one--value numeric expression. However, while the end attribute specification of an indirect numeric attribute specification may be a complex expression, that of the string one is a string entity set name only because of the primitive stucture of the one-value string expression.

46. <one-value expression of abstract type> : : =
    <specification of a one-value attribute of abstract type>

47. <specification of a one-value attribute of abstract type> : : =
    <specification of a (direct) one-value attribute of abstract type>|<specification of an indirect one-value attribute of abstract type>

48. <specification of a (direct) one-value attribute of abstract type> : : = <name of a one-value attribute of abstract type>

49. <specification of an indirect one-value attribute of abstract type> : : =<specification of an end attribute of abstract type><one-value attribute chain specification>

50. <specification of an end attribute of abstract type> : : = <name of an entity set of abstract type>

A one-value expression of abstract type defined by rules (45)–(50) consists of a single variable of abstract type. The value of the variable of a one-value expression of abstract type is specified by a specification of a one-value attribute of abstract type in a way analogous to that of the one-value string expression.

Since the value of a one-value expression of abstract type for an entity of the context indicator is the entity of an entity set that is not an identifier, its identifier values are transferred instead of it.

51. <multivalue expression> : : =
    <multivalue numeric expression> |
    <multivalue string expression> |
    <multivalue expression of abstract type>

The type of a multivalue expression is determined by the type of the values specified by it for the entities of the context indicator in the same manner as that of a one-value expression.

52. <multivalue numeric expression> : : =
    <multivalue numeric term> |
    <sign symbol><multivalue numeric term> |
    <one-value numeric expression>
    <additive operation symbol><multivalue numeric term> |
    <multivalue numeric term><additive operation symbol>
    <one-value numeric expression>

53. <multivalue numeric term>: : =
<multivalue numeric attribute specification> |
<one-value numeric term>
<multiplicative operation symbol> <multivalue numeric term> |
<multivalue numeric term><multiplicative operation symbol>
<one-value numeric term> |
(<multivalue numeric expression>)

54. <multivalue numeric attribute specification>: : =
<(direct) multivalue numeric attribute specification> |
<indirect multivalue numeric attribute specification>

55. <(direct) multivalue numeric attribute specification>: : =
<multivalue numeric attribute name>

56. <indirect multivalue numeric attribute specification>: : =
<numeric end attribute specification>
<multivalue attribute chain specification>

57. <multivalue attribute chain specification>: : =
<multivalue attribute connector> |
<of phrase><multivalue attribute chain specification> |
<multivalue of phrase><attribute chain specification>

58. <multivalue attribute connector>: : =
<multivalue of phrase>of <inverse of an attribute> |
<of phrase><inverse of a multivalue attribute>

59. <multivalue of phrase>: : = of <inverse of a multivalue attribute> |
of<inverse of a multivalue attribute>(<where qualifier>)

60. <inverse of a multivalue attribute>: : = <attribute name>

61. <of phrase>: : = <one-value of phrae > |< multivalue of phrase>

62. <attribute chain specification>: : =
<one-value attribute chain specification> |
<multivalue attribute chain specification>

The structure of a multivalue numeric expression determined by rules (51)–(62) seems to be similar to that of the expressions constructed from one linear vector and some scalars by the additive and/or multiplicative operations. However, the difference between the two types of expressions is the same as that between the arithmetic and the one-value numeric expressions. Moreover, the method of retrieval of the values of a variable specified by a multivalue attribute specification (of any type) is more complicated than that of those specified by a one-value one.

The values specified by a (direct) multivalue numeric attribute specification for an entity of the context indicator are the numeric identifier values, connected to the given entity by the fact set showing a 1-many or a many-many relation, the name of which is specified by the given attribute specification.

The values specified by an indirect multivalue numeric attribute specification for an entity of the context indicator can be determined with the help of the schema of the indirect multivalue attribute specification.

This schema can be defined in a way analogous to that of an indirect one-value attribute specification. The only difference is that the branches of the schema of an indirect multivalue attribute specification represent the fact sets, among which there is at least one showing the inverse of a 1-many or a many-many relation. Therefore more than one traversal tree of entities may exist, the root of which is the value of the specified indirect multivalue attribute for one entity of the context indicator.

In Figure 6.1 some traversal trees of entities are shown specified by the indirect multivalue attribute specification, the schema of which is shown in Figure 5.1. These trees are specified for the case shown in Figure 6.2. The values of the specified indirect multivalue attribute for the entity a3 of the context indicator are marked with the double circles in Figure 6.1.

63. <multivalue string expression>: : = <multivalue string term>

64. <multivalue string term>: : = <multivalue string attribute specification>

65. <multivalue string attribute specification>: : = <(direct) multivalue string attribute
specification> |
<indirect multivalue string attribute specification>

66. <(indirect) multivalue string attribute specificaton> : : =
<multivalue string attribute name>

67. <indirect multivalue string attribute specification>: : =
<string end attribute specification>
<multivalue attribute chain specification>

The difference between the multivalue string expressions and the numeric ones is similar to that in the one-value case.

68. <multivalue expression of abstract type> : : =
<specification of a multivalue attribute of abstract type>

69. <specification of a multivalue attribute of abstract type> : : =
<specification of a (direct) multivalue attribute of abstract type> |
<specification of an indirect multivalue attribute of abstract type>

70. <specification of a (direct) multivalue attribute of abstract type >: : =<name of a multivalue attribute of abstract type>

71. <specification of an indirect multivalue attribute of abstract type>: : = <specification of an end attribute of abstract type><multivalue attribute chain specification>

The difference between the multivalue expressions of abstract type and the numeric and string is similar to that in the one-value case.

72. <set-value expression>: : = all |
all-except <sequence of attribute names>

73. <sequence of attribute names>: : = <attribute name> |
<sequence of attribute names>,<attribute name>

74. <attribute name>: : = <one-value attribute name> |
<multivalue attribute name>

75. <one-value attribute name>: : = <one-value numeric attribute name> |
<one-value string attribute name> |
<name of a one-value attribute of abstract type>

76. <multivalue attribute name>: : =
<multivalue numeric attribute name> |
<multivalue string attribute name> |
<name of a multivalue attribute of abstract type>

The value of the set-value expression all for an entity of the context indicator is a set consisting of the subsets of the (direct) attributes of the context indicator that are connected to the given entity. The value of the set-value expression all-except for an entity of the context indicator is the same, excluding those subsets which are contained in the entity sets specified by the sequence of attribute names of the expression.

In the case shown in Figure 7 the value of the set-expression all for the entity e is the set of the marked subsets of A1, A3 and A4, while that of the set-expression all-except A3 is the set of the marked subsets of A1 and A4.

Details about the conditions are given by the syntactical rules and in the explanations below.

77. <condition>: : = <condition on the context indicator> |
<condition on the attributes> |
<condition><logical operation symbol>
<logical term>

78. <logical term>: : = <condition on the attributes> |
<logical term><logical operation symbol>
<condition on the attributes> |
(<logical term>)

Considering the conditions on the context indicator and those on the attributes as logical variables, a condition can be considered as a logical expression constructed from some logical variables by the logical operation symbols in the general case. This logical expression specifies a logical value for each entity of the entity set, the specification of which is followed by the where qualifier containing the given condition. A condition is considered satisfied by an entity if and only if the value specified by the logical expression for it is true.

79. <condition on the context indicator> : : =
 <incomplete numeric relation> |
 <incomplete string relation> |
 <incomplete element relation> | <incomplete set relation>

80. <incomplete numeric relation> : : =
 <numeric relation symbol><numeric expression>

81. <numeric relation symbol> : : = eq | ne | lt | le | gt | ge

82. <numeric expression> : : = <one-value numeric expression> |
 <multivalue numeric expression>

83. <incomplete string relation> : : =<string relation symbol>
 <string expression>

84. <string relation symbol> : : eq | ne

85. <string expression> : : =<one-value string expression> |
 <multivalue string expression>

86. <incopmlete element relation> : : =<element relation symbol>
 <set specification>·

87. <element relation symbol> : : = blt | nbt

88. <set specification> : : <set specification by name> |
 <set specification by set affix>

89. <set specification by name> : : = <multivalue attribute name>

90. <set specification by set affix> : : =
 set (<specification of elements>)

91. <specification of elements> : : =
 <indirect multivalue attribute specification> |
 <sequence of elements>

92. <indirect multivalue attribute specification> : : =
 <indirect multivalue numeric attribute specification> |
 <indirect multivalue string attribute specification> |
 <specification of an indirect multivalue attribute of abstract type>

93. <sequence of elements>: : =<element specification> |
<sequence of elements>,<element specification>

94. <element specification>: : = <one-value attribute name> | <constant>

95. <constant>: : = <numeric constant> | <string constant>

96. <incomplete set relation>: : =<set relation symbol> <set specification>

97. <set relation symbol>: : = inc | nic | sus | sas | nas

A condition has its own scope indicated by an entity set, entities and attributes of which are to be tested for the condition. Initially the scope indicator is the entity set, the specification of which is followed by the where qualifier containing the given condition.

A condition on the context indicator is a condition on the entities of the initial scope indicator. Conseqently, the initial scope indicator of the condition is implicitly assumed as the left side of an incomplete relation.

The logical value specified by an incomplete relation for an entity of the initial scope indicator is true if and only if the entity is in the given relation to the values specified by the right side of the incomplete relation.

The relation "is equal ", "is not equal", "is less than", "is less or equal", "is greater than" and "is greater or equal" are meant in their usual sense for the numeric values under the symbols given in rule (81) respectively.

The string relations "is equal" and "is not equal" are meant under the symbols given in rule (84).

If an incomplete numeric or string relation has a multivalue expression as its right side, then it is assumed to be the conjuction of the relations having the single values of the multivalue expression as their right sides.

The relations "belongs to" and "doesn't belong to" between an element and a set are denoted by the symbols given in rule (87).

The relations "includes","doesn't include", "is a subset of", "is same as" and "is not same as" between two sets are meant under the symbols given in rule (97). A set having a single element is assumed for an entity of the initial scope indicator in an incomplete set relation.

98. <condition on the attributes>: : =
<condition on the attributes in scope> |
<removed scope indicator>,
<condition on the attributes in scope>

99. <condition on the attributes in scope>: : =
<numeric relation> | <string relation> |
<element relation> | <set relation> |
<set quantification>

100. <numeric relation> : : = <numeric expression>
<incomplete numeric relation>

101. <string relation> : : = <string expression>
<incomplete string relation>

102. <element relation> : : =
<one-value expression><incomplete element relation> |
<multivalue expression><incomplete element relation>

103. <set relation> : : =<set specification><incomplete set relation>

104. <set quantification> : : = <set specification> exists |
all <set specification>

105. <removed scope indicator> : : = <removed context indicator>

The scope of a condition can be removed in the same way as removing the context in the forprint specification.

A condition on the attributes in scope is a condition on the direct or indirect attributes of the scope indicator for the moment. Since a removed scope indicator is connected, perhaps through some other removed scope indicators, to the initial one, all conditions on the attributes are in fact related to the entities of the initial scope indicator.

The logical value specified by the relations defined by rules (100) – (103) for an entity of the initial scope indicator is true if and only if the values specified by the left side are in the given relation to the values specified by the right one.

If a numeric or a string relation has a multivalue expression either as its left side or its right one or both, then it is assumed to be the conjunction of the relations having the single values of the multivalue expressions on the corresponding sides. If an element relation has a multivalue expression as its left side, then it is assumed to be the conjunction of the ralations having the single values of the multivalue expression as their left sides.

In rule (104) the symbol exists means the existential quantifier, while the symbol all means the universal one in the sense of mathematical logic.

106. <logical operation symbol> : : = , |and|or

The , as a logical operation symbol has the same meaning as and. The logical operation symbols and, and or mean the conjunction and the disjunction in the sense of mathematical logic.

The syntactic entities left undefined throughout this description should be defined on the level of implementation.

II. Remarks concerning the FORAL output statement

This section contains some remarks concerning the output statement described in the first part of the report. The remarks are given in the sequence of the syntactic rules and the explanations of the first part that they refer to.

Remarks contained in this report are of the following types. Remarks of type (a) concern only the terminology used in [1] (consequently used in the first part of this report). Some redundancies and/or inconsistencies in the syntax are pointed out by remarks of type (b). The semantics are treated in remarks of type (c): they either show an interpretation different from that of the first part of the report for some syntactic rules or they suggest an interpretation where none is given in the first part. Rules of type (d) show some possibilities of restrictions without decreasing the power of the output statement in its essential respects, and some of extensions that seem to be necessary for many applications and can be implemented without any additional complications. The type of the remark is shown at its beginning giving the possibility of reading only the remarks of interest.

Concerning rule (1):

(a) – It is not necessary for the user to know anything about the files as they are mostly connected with the internal representation of information. Consequently, it would be better to refer to the first part of the output statement by another term, for example, "transfer specification" or – according to [3] – "record set specification".

The second part of the output statement specifies not only the format of the information, but – and it is not less important – the information itself. So it would be better to refer to the second part of the output statement by another term, for example, "information specification", "data specification", etc.

Concerning rules (2) – (6):

(c,d) – The possibility of an indication of more than one source specification in one file specification seems to be superfluous in an output statement without any where overlay specifications on the basis of the following considerations.

The entity sets, information about the entities of which is to be delivered while transferring on the basis of one output statement without any where overlay specifications, should be connected directly, or indirectly, to that indicated as the basis of the transfer at the beginning of the format specification. If the internal representation allows the storing of some entity sets, connected with each other, on different files then the information necessary for the switching from one file to another is represented interally. In this case the output statement need not specify any source files in addition to that containing the entity set identicated as the basis of the transfer. In the case of a less poweful internal representation it is impossible to switch from one source file to another without showing the switching points in the output statement itself.

The problem of switching from one source file to another having only one output statement can be solved with the help of the where overlay specifications – if they are intended to serve this purpose. If the where overlay specification has no effect in addition to the switching from one source file to another, then it seems to be superfluous since the same effect can be achieved by two output statements without a where overlay specification but having different source specifications. If the where overlay specification is intended to serve other purposes too (for example, for connecting on the temporary file, the entity set specified in it as an initial context indicator to the entity set indicated as the basis of the transfer) then it is very reasonable to allow the use of more than one where overlay specification in one format specification.

On the basis of the considerations above it seems to be reasonable

– either to restrict the output statement replacing rule (2) by the rule:

```
<file specification> : : = <destination specification> |
                            <destination specification>
                            <source specification>;
```

– or to extend it replacing rule (6) by the rule:

```
<format specification> : : = <forprint specification> |
                              <format specification>
                              <where overlay specification>
```

However, even with the above extension it is impossible to begin the transfer from the central data base and to continue it from other internal files. The modification of the output statement described in the third part of this report permits transfer from the central data base and from other internal files in any sequence , supposing that one of the effects of the syntactic entity, called a where overlay specification in the first part of this report, is the switching from one file to another.

Concerning rules (3) – (4):

(c) – It is reasonable to treat the output statements with different destination indicators together since the syntax is common for the output statements of either type. The only difference in the syntax is that the set of the output file names allowed by an implementation in the case of the output destination indicator may be wider than that in the case of the temporary one.

The main difference between the output statements with different destination indicators is on the level of semantics covered by the interpretation of the term "transfer". It is quite clear that in the case of an actual output the information to be transferred has to be placed in different rows and columns of the output file, so that it cannot be transferred from this file to an other one by a similar output statement. However, in the case of an output statement with the temporary destination indicator the term "transfer" has to have a meaning different from that in the case of an actual output. This is left out of consideration in [1] and, consequently, in the first part of this report. It is quite reasonable to suppose that an auxiliary data base is created in the internal form as the effect of an output statement with the temporary destination indicator. The schema of this data base is that of the forprint specification given in the output statement.

Concerning rules (6) – (9) and (18):

(a) – The description of a language is easily understandable if the names of the syntactic entities show their semantic content rather than their form. Therefore it would be better to call the "forprint specification", for example, "information specification".

Analogous remarks can be given to the names of the following syntactic entities.

The meanings of the syntactic entities

– entity set print clause

– for attribute print clause

– print clause

introduced in rules (7) and (8) are not very obvious, particularly in the case of a temporary output statement.

Similarly, it would be better to use the term "condition" instead of the "where qualifier" introduced in rule (9). Perhaps s better names can also be found for the "field specification" introduced in rule (12) and for the "called phrase" introduced in rule (18).

Concerning rules (9) and (11):

(b) – These two rules show a redundancy in syntax, since it is not necessary to signal the beginning of a condition with a pair of "open brackets" in the form

(where . . .

This redundancy can be eliminated in the way of using an appropriate symbol as the "close bracket" for the "open bracket" where, though the end of a condition seems to be syntactically recognisable in all cases of its use. For example, an asterisk* can be chosen as the "close bracket" for the "open bracket" where resulting in the form

where . . . *

* The form

where . . . wend

was proposed by Prof. J. Bubenko (University of Stockholm).

Using only a pair of brackets is another way of eliminating the redundancy – in this case the word <u>where</u> can be used as a readability aid.

Remark on the "schema of the forprint specification":

(c) – The schema of a forprint specification can be defined in another way that seems to be more natural than the one defined in the first part of this report.

The schema of a forprint specification is a directed ordered graph for which a one-to-one correspondence exists between the set of its basic nodes and that of the context indicators specified in the given forprint specification. The branches of the graph represent the fact sets specified by the context removers forward in the same order as they are shown in the given forprint specification. Additional nodes are connected to the basic ones in the same way as it is described in the first part of the report.

The schema of the forprint specification

A (<u>where</u> . . .) <u>print</u> D,E <u>for</u> F <u>for</u> G (<u>where</u> . . .)
<u>print</u> H <u>for</u> F <u>called</u> I <u>print</u> J <u>for</u> G <u>for</u> F
<u>print</u> K <u>for</u> L <u>for</u> G <u>print</u> M

corresponding to the definition above is the one shown in Figure 8 instead of that shown in Figure 3.1.

It is quite obvious that the simple traversal algorithm specifying the traversal trees of entities correspondingly to the schema defined above does not give a correct result. For example, in the case shown in Figure 8 the traversal trees of the entities of the context indicator G contain the subtrees corresponding to the whole subschema marked in Figure 8, regardless of the branch leading to the given entity. The reason is the branches leaving a node of the schema, to which more than one branch leads, are all traversed each time the node is traversed.

This problem can be solved by giving some additional information in the schema of the forprint specification. One possible solution seems to be the labelling of the critical branches by the numbers of traversal of the node they leave, after which they are "valid". In Figure 8 the critical branches are marked in this sense.

In the general case the schema of a forprint specification, defined in the above way, can be a very complicated one. As an example the schema of the forprint specification

A <u>print</u> B <u>for</u> C <u>for</u> D <u>for</u> E <u>print</u> F
<u>for</u> C <u>print</u> G <u>for</u> H <u>print</u> G <u>for</u> E <u>print</u> I

is shown in Figure 9.

The traversal trees of entities corresponding to the schema of a forprint specification having some marked branches can also be specified by the traversal of the schema. However, at a traversed node of the schema a distinction should be made between the branches leaving the node. The proper traversal trees of entities can be specified in when the number of traversals of entities is known. The traversal subtrees are specified for a traversed entity following those and only those branches of the schema that have the label equal to the current number of traversals of the given entity. Naturally, the traversal algorithm specified above is more complicated than that referred to in the first part of this report.

Concerning rules (29) – (32) and (57) – (62):

(b,c) – It seems to be superfluous (not only from a syntactic point of view, but also from a semantic one) to connect an indirect attribute to the context indicator showing the context indicator itself. If it is possible to reach a direct attribute from the context indicator by the name of the attribute (without showing the name of the context indicator) – and it is possible – why should a secondary attribute be specified by 3 names? The reason is of a semantic nature. A direct attribute value is defined from the context indicator to the attribute, while an indirect attribute value is searched from the end attribute to the context indicator. Consequently, the in direct attribute specification can be relieved of the superfluous name by having a uniform semantic approach to the attributes of the two types. This uniformity can be achieved on the basis of the following definitions.

– An attribute of rank 1 of the context indicator is an entity set connected to the context indicator by a fact set.

– An attribute of rank n (>1) of the context indicator is an entity set connected to an attribute of rank $n-1$ of the context indicator by a fact set.

The attributes of rank 1 as defined above correspond to the (direct) attributes in the first part of this report, and those of rank greater than 1 correspond to the indirect ones.

On the basis of the definitions above an attribute of rank n can also be specified by n names in the case of $n > 1$. It should be mentioned that fact sets named in the specification of an attribute of rank $n > 1$ are the opposite ones to those shown in the indirect attribute specification defined in the first part. However, this shortened specification does not seem to be less adequate than the original one from the point of view of the English language.

Corresponding to the definitions above, the schema of the shortened specification of an attribute of rank n (≥ 1) is a linear path consisting of n nodes. The nodes of the schema represent the entity sets determined by the fact sets, the names of which are shown in the shortened specification of the attribute. The branches of the schema represent the named fact sets themselves. The schema of a shortened attribute specification defined in this way is called the opposite schema of the attribute.

The values of an attribute for an entity of the context indicator can be retrieved by the traversal of a directed graph of entities specified on the basis of the opposite schema of the attribute defined above.

The graph of entities to be traversed for one entity of the context indicator is a linear path only in the one-value case. A directed graph of a more complex structure having the given entity of the context indicator as the source node is specified on the basis of the opposite schema of an attribute in a multivalue case.

In Figure 10.1 a situation is shown for the multivalue case shown in Figure 6.2. This situation has to be taken account while determining the values of the attribute specified by the shortened attribute specification

E or D of C of B

for the entity a3 of the context indicator A. In the case shown in Figure 10.1 the traversal tree of entities shown in Figure 10.2 is specified by the simple traversal algorithm on the basis of the opposite schema. This traversal tree of entities shows that an attribute value is retrieved more than once by the simple traversal algorithm.

In the general case an attribute value for an entity of the context indicator is retrieved by the simple traversal algorithm as many times as paths lead to it from the given entity, on the basis of the opposite schema of the attribute. Consequently, on the basis of the opposite schema of an attribute, the simple traversal algorithm results in an interpretation of the indirect multivalue attributes different from that of the first part.

In a discussion of this problem with Prof. J. Bubenko (University of Stockholm) it appeared that from the user's point of view it might also be better (in a "standard interpretation") to transfer a value of an indirect multivalue attribute only once in the case where more than one path leads to it. This purpose can, however, be achieved by applying an algorithm for traversing the graphs that traverses a node of a graph of any structure only once. Such traversal algorithms are used in the data base management sytem LIDI, described in [4].

The traversal tree of entities shown in Figure 10.3 is specified by the algorithm for the traversal of the graphs of any structure mentioned above instead of that shown in Figure 10.2 in the case shown in Figure 10.1.

Concerning rules (40)-(45) and (63)–(67):

(d) – For some applications it seems to be necessary to have a set of operations over strings, such as concatenation, insertion, etc., so the referred set of rules should be extended.

Concerning rule (72):

(d) – Having the relations defined for sets it seems to be reasonable to have some operations over them, such as union, intersection, etc., so the rule should be completed.

Concerning rules (72) – (73) and (90):

(b) – There is no essential difference betweeen the operands of the all-except operator and those of the set type, so it would be more consistent to specify the operands of both operators either in parentheses or without them. The first version is better from the point of view of an implementation, since in the print clauses similar to

print all-except A, B+2

the scope of all-except operator becomes clear after one additional attribute name only.

Another consistent solution might be to find the proper "close brackets" for the "open brackets" all-except and set.

Concerning rule (75):

(d) – Following this rule it is impossible to give more than one condition concerning the entities of the context indicator, though it might be needed in many cases.

Concerning rule (84):

(d) – In consequence of the remark conserning rules (40)–(45) and (63) – (67) it is reasonable to widen the set of relations over strings.

Concerning rules (89) – (91):

(b) – There is no difference between the direct and indirect attributes from the point of view of the interpretation of sets, so it would be more consistent to show both of them either with a set affix or without it. The first version would be nicer from the syntactic point of view.

III. Modifications to the FORAL output statement

This section contains suggestions for some modifications of the FORAL output statement described in the first part of this report, based on the remarks given in the second part. Some remarks are not considered, mainly those of type (d) since they require considerable further, detailed study.

It is mainly the syntax which is described in this part of the report, since the semantics can be regarded as defined by the previous parts of the report and by the names of the syntactic entities used in this part.

Some (groups of) syntactic rules that correspond to those of the first part of this report are provided by a reference to their numbers.

Instead of rules (1) – (3):

```
<output statement>: : = <destination specification>
                        <information specification>.

<destination specification>: : = <destination  indicator>
                                 <destination name>
```

Rule (4) is not modified.

Instead of rules (5) – (9):

```
<information specification>: : =
        <initial information specification> |
        <information specification>
        <additional information specification>

<initial information specification> : : =
        <initial source specification>:
        <information specification from one source>

<initial source specification>: : = |
        <additional source specification>
<additional source specification> : : = from <source name>
<source name>: : = | central|<auxiliary source name>
<additional information specification>: : =
        <additional source specification>:
        <information specification from one source>
<information specification from on source> : : =
        <information specification about the initial entity set> |
        <information specification from one source>
        <information specification about an additional entity set >
```

<information specification about the initial entity set>: : =
    <initial context indicator> |
    <initial context indicator>
    <information specification about the attributes>
<initial context indicator> : : =
    <initial entity set specification> |
    <initial entity set specification><condition>

<condition>: : = <condition open bracket><logical expression>) |
    where <logical expression><condition end>
<condition open bracket>: : = (|(where
<condition end> : : * |wend

Rule (10) is not modified.

The above set of rules specifies an output statement that makes possible the transfer of information from different sources in any sequence. The rules forbid the use of the source name central as an auxiliary one. However, this name can be considered as a reading aid.

The specification and the use of the syntactic entity called the where qualifier in the first part of this report is modified according to the remarks of the second part.

Some remarks concerning the names of the syntactic entities are taken into account in the above rules and in the following ones.

Instead of rules (11) – (16):

<information specification about the attributes>::=
    transfer <field specification> |
    <information specification about the attributes>,
    <field specification>
<field specification>:: = <attribute expression> |
    <attribute expression><field renamer >
<field renamer> : : = called <field name>
<information specification about an additional entity set>: : =
    <removed context indicator> |
    <removed context indicator>
    <information specification about the attributes>
<removed context indicator>: : =
    <additional entity set specification> |
    <additional entity set specification><condition>

Rule (17) is not modified.

Instead of rules (18) – (19) :

```
<context remover> : : = <attribute name>|
          <attribute name><attribute renamer>
<attribute renamer> : : = called <new attribute name>
```

The following rules are concerned with the details of the attribute expressions.

Rules (20) – (22) are not modified.

Instread of rule (23):

```
<one-value numeric term>: : = <unsigned numeric constant>|
          <one-value numeric attribute specification>|
          <one-value numeric term><multiplicative operation symbol>
          <one-value numeric term>|
          <one-value numeric function specification>
```

Rule (24) is not modified.

Instead of rules (25) – (32):

```
<one-value numeric attribute specification>: : =
          <one-value numeric attribute name>|
          (<one-value numeric expression>)|
          <one-value numeric attribute specification>
          <one-value attribute mediator>
<one-value attribute mediator>::=
          of <one-value attribute name>|
          of <one-value attribute name><condition>
<one-value attribute name> : : =
          <one-value numeric attribute name>|
          <one-value string attribute name>|
          <name of a one-value attribute of abstract type>
```

Rules (33) – (39) are not modified.

The above rules determine the one-value numeric expression of the same structure as rules (22) – (39) of the first part. The only difference is in the one-value numeric attribute specification. The rules above specify the direct and the indirect one-value numeric attributes in a uniform way corresponding to the remarks of the second part.

The following rules reflect the analogous modifications cuncerning the one-value string expression, the one-value expression of abstract type and the multivalue expressions of all types.

Rules (40) – (41) are not modified.

Instead of rules (42) – (45):

```
<one-value string attribute specification> : : =
          <one-value string attribute name> |
          <one-value string attribute specification>
          <one-value attribute mediator>
```

Rule (46) is not modified.

Instead of rules (47) – (50):

```
<specification of a one-value attribute of abstract type> : : =
          <name of a one-value attribute of abstract type> |
          <specification of a one-value attribute of abstract type>
          <one-value attribute mediator>
```

Rules (51) – (52) are not modified.

Instead of rules (53) – (62):

```
<multivalue numeric term> : : =
          <multivalue numeric attribute specification> |
          <one-value numeric term><multiplicative operation symbol>
          <multivalue numeric term> |
          <multivalue numeric term><multiplicative operation symbol>
          <one-value numeric term>
<multivalue numeric attribute specification> : : =
          <multivalue numeric attribute name> |
          (<multivalue numeric expression>) |
          <multivalue numeric attribute specification>
          <attribute mediator> |
          <one-value numeric attribute specification>
          <multivalue attribute mediator>
<attribute mediator> : : = <one-value attribute mediator> |
                           <multivalue attribute mediator>
<multivalue attribute mediator> : : =
          of <multivalue attribute name> |
          of <multivalue attribute name><condition>
<multivalue attribute name> : : =
          <multivalue numeric attribute name> |
          <multivalue string attribute name> |
          <name of a multivalue attribute of abstract type>
```

Rules (63) – (64) are not modified.

Instead of rules (65) – (67):

<multivalue string attribute specification> : : =
    <multivalue string attribute name>|
    <multivalue string attribute specification>
    <attribute mediator>|<one-value string attribute specification>
    <multivalue attribute mediator>

Rule (68) is not modified.

Instead of rules (69) – (71):

<specification of a multivalue attribute of abstract type>: : =
    <name of a multivalue attribute of abstract type>|
    <specification of a multivalue attribute of abstract type>
    <attribute mediator>|
    <specification of a one-value attribute of abstract type>
    <multivalue attribute mediator>

The following modification concerning the set-value expression unifies the specification of the operands of the all-except operator and those of set.

Instead of rule (72):

<set-value expression>: : = all | all-except <set of attributes>
<set of attributes> : : = (<sequence of attribute names>)|
    <sequence of attribute names> xend

Rules (73) – (74) are not modified.

Rules (75) – (76) are excluded.

The following rules concern the details of the logical expressions.

Instead of rules (77) – (79):

<logical expression> : : = <logical term>|
    <logical expression><logical operation symbol><logical term>
<logical term> : : = <incomplete relation>|
    <complete relation>|<set quantification>|
    <scope remover> , <logical term> | (<logical expression>)

The above rules define a more complex structure of the logical expression than that defined in the first part of this report.

<incomplete relation> : : = <incomplete numeric relation>|
    <incomplete string relation>|<incomplete element relation>|
    <incomplete set relation>

The scope indicator for the moment is assumed as the left side of an incomplete relation in the case where no complete relations precede it following the specification of the scope indicator for the moment. Otherwise, the left side of the last preceding complete relation is assumed to be that of the incomplete relation.

Rules (80) – (88) are not modified.

Instead of rules (89) – (94):

<set specification by name> : : =
    <multivalue attribute name>|
    <set specification by name><attribute mediator>|
    <element specification by name>
    <multivalue attribute mediator>
<element specificaton by name>: : =
    <one-value attribute name>|
    <element specification by name>
    <one-value attribute mediator>
<set specification by set affix> : : = set <set of elements>
<set of elements> : : = (<sequence of elements>)|
    <sequence of elements> send
<sequence of elements> : : = <element specification>|
    <sequence of elements> , <element specification>
<element specification>: : =
    <element specification by name> | <constant>

Rules (95) – (97) are not modified.

The above rules define sets either by a set affix or without it. A multivalue attribute can also be specified as a set without a set affix in the case of an indirect attribute. A one-value attribute can also specify an element of a set in the case of an indirect attribute.

Instead of rules (98) – (99):

<complete relation> : : = <numeric relation>|
    <string relation> | <element relation> | <set relation>

Rules (100) – (106) are not modified.

Conclusions

Further studies are necessary concerning some parts of the FORAL output statement described in this report. Concerning the attribute expressions, the complex structure of string expressions is an object for further study both in the one-value and in the multivalue cases. The operations over sets can be introduced. Concerning conditions, string relations are objects for further elaboration.

All the discussions in this report suggest an implementation of the output statement by an algorithm consisting of two phases.

In the first phase the schema of the output statement is constructed. From the point of view of the implementation it seems to be better to construct the schema of a forprint specification in the form of a tree as introduced in the first part of this report. The opposite schemas of the indirect attributes, introduced in the second part, can be used. The schemas of the whole attribute expressions and the conditions should be specified. The method of connecting the schemas of the attribute expressions and the conditions to the schema of the whole forprint specification should be determined.

The syntactic analysis of a given output statement can also be performed in the first phase of the algorithm. It should be mentioned that a full syntactic analysis corresponding to the syntax described in this report can be performed only in the case where information not shown explicitly in [1] is represented in the data base. The types of the fact sets and those of the entity sets have to be retrievable for a full syntactic analysis. For each fact set it should be known whether it shows a 1-1, many-1, 1-many or a many-many relation. Similarly, for each entity set it is necessary to recognize whether it consists of numerical or string identifier values or of entities of abstract type.

The second phase of the algorithm for the interpretation of the output statement performs the traversal of the entities of the data base on the basis of the schema of a given output statement. The simple traversal algorithm can be applied for traversal on the basis of the schema of the forprint specification. The algorithm for traversal of the graphs of any structure mentioned in the second part can be used for traversal on the basis of the schemas of indirect attributes.

Acknowledgement

The author wishes to express her gratitude to Janis Bubenko, Professor at the University of Stockholm, for the initiation of this work and for his valuable suggestions.

References

[1] M.E. Senko: FORAL II for DIAM II: Information Structures and Query-Maintenance Language, IBM Research, Yorktown Heights, (1976) (Submitted for publication)

[2] D.E. Knuth: The Art of Computer Programming Vol. 1, Addison Wesley, 1968.

[3] ANSI/X3/ SPARC Study Group on Data Base Management Systems Interim Report, 1975.

[4] Fidrich I.: The Data Base Management System LIDI, Proceedings of the CAD Seminar, Budapest, Hungary, 1976.

[5] M.E. Senko, E.B. Altman, M.M. Astrahan, P.L. Fehder: Data Structures and Accessing in Data Base Systems
IBM System Journal, Vol. 12, N^{0}-1, 1973.

[6] M.E. Senko: Data Description Language in the Context of a Multilevel Structured Description: DIAM II with FORAL, Data Base Description, Processings of the IFIP TC-2 Special Working Conference on Data Description, Ed. B. C. M. Douque, G.M. Nijssen, Nort-Holland, 1975.

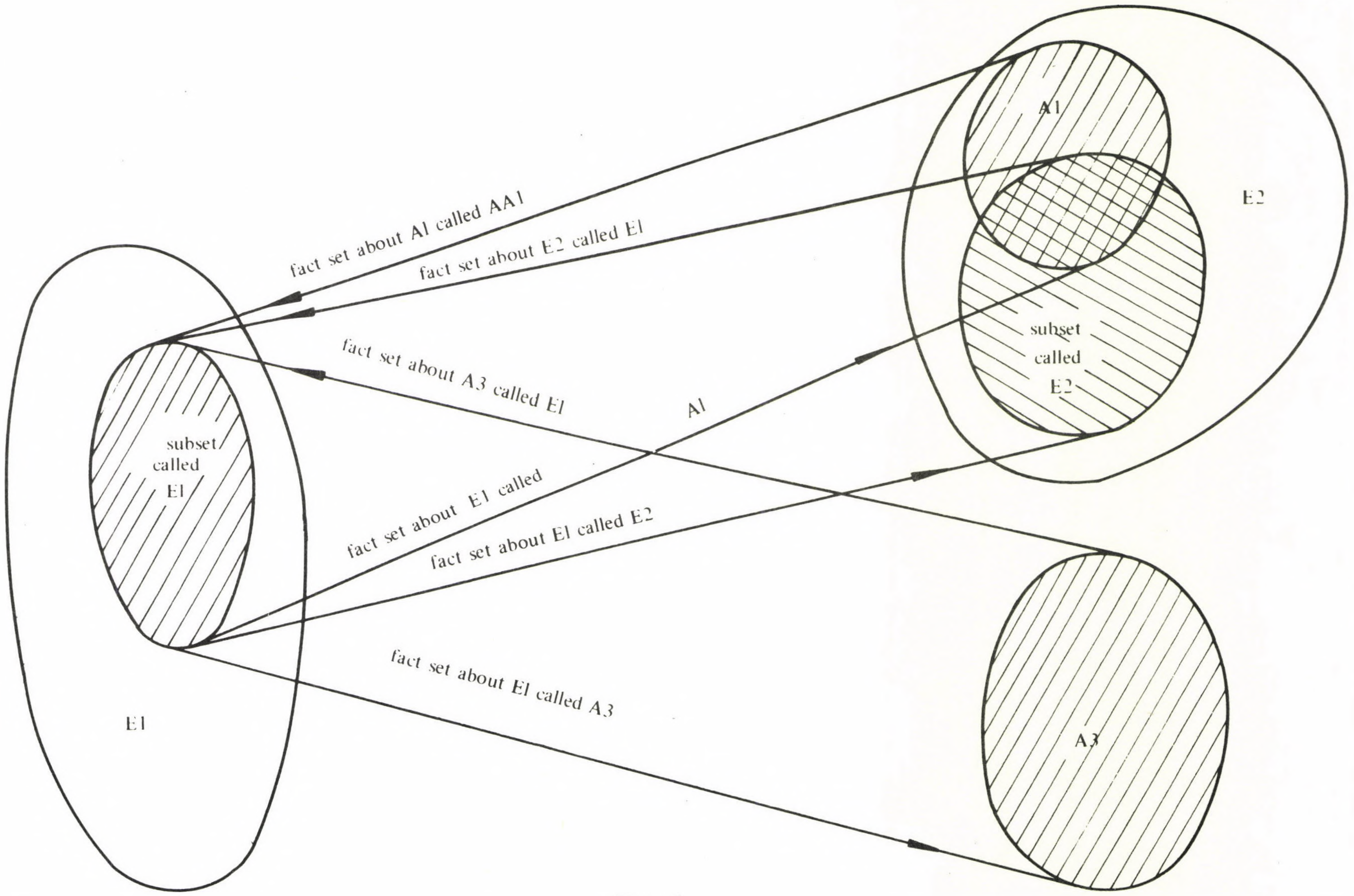

fact set about A1 called AA1
fact set about E2 called E1
fact set about A3 called E1
A1
fact set about E1 called
fact set about E1 called E2
fact set about E1 called A3
A1
E2
subset called E2
subset called E1
E1
A3

Figure 1

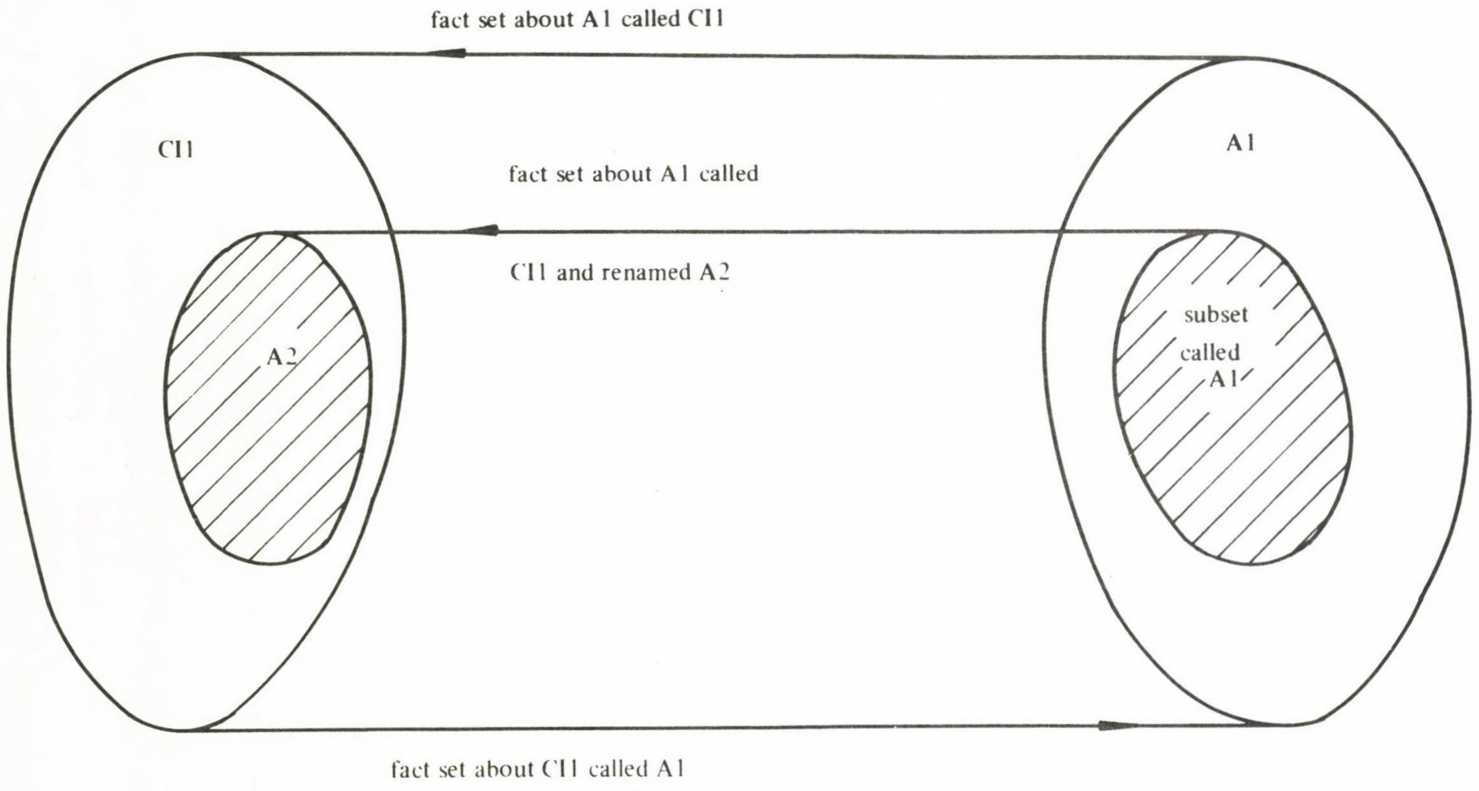

fact set about A1 called CI1
CI1
A1
fact set about A1 called
CI1 and renamed A2
A2
subset
called
A1
fact set about CI1 called A1

Figure 2

$$\begin{tikzcd}
A \arrow[r] \arrow[rd] \arrow[rdd] \arrow[rddd] \arrow[rdddd] & B & & & \\
& C & & & \\
& D & & & \\
& E & & & \\
& F \arrow[r] \arrow[rd] \arrow[rdd] & G \arrow[r] \arrow[rd] & H & \\
& & K & I \arrow[r] & J \\
& & L \arrow[r] & G \arrow[r] & M
\end{tikzcd}$$

Figure 3.1

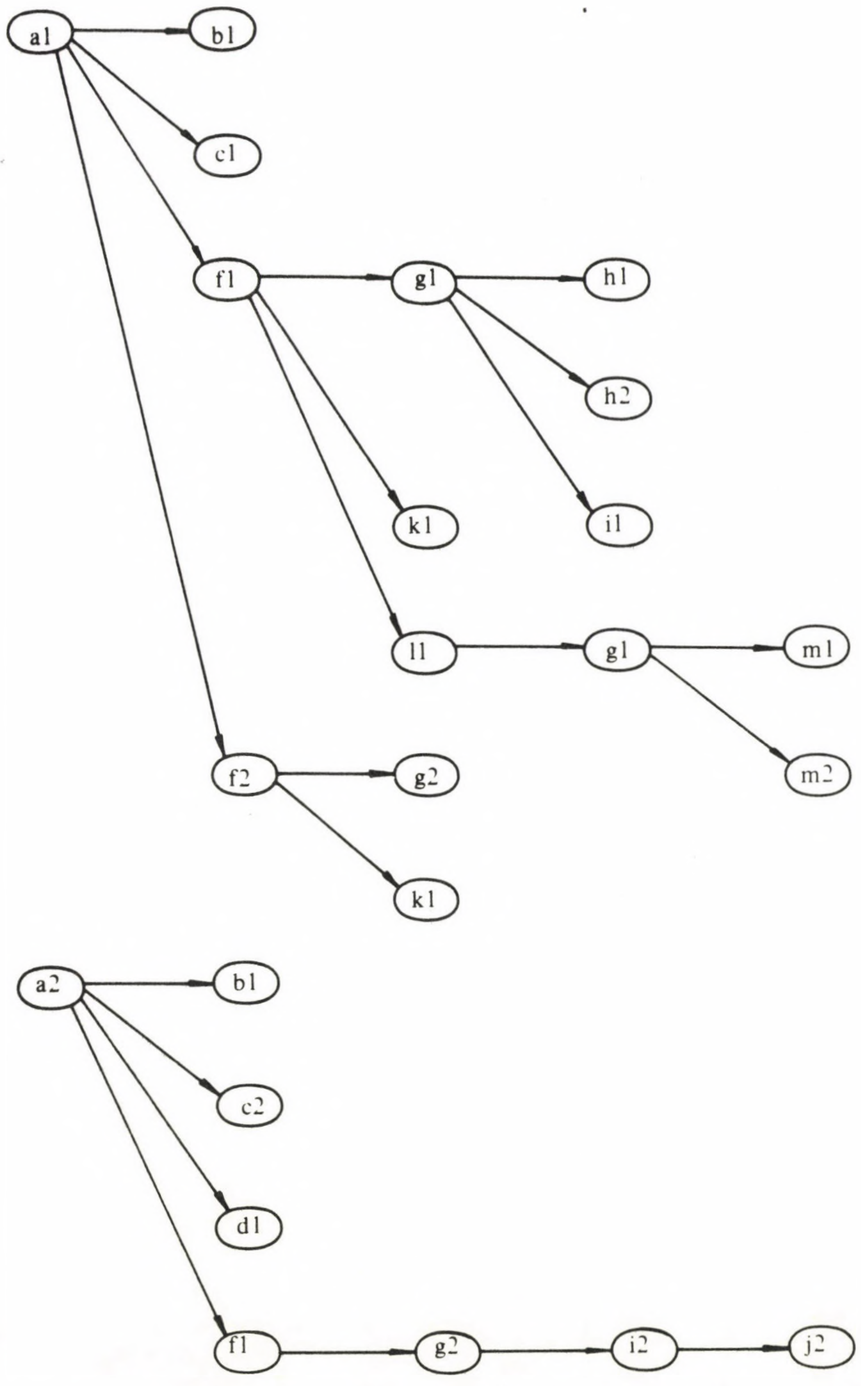


Figure 3.2

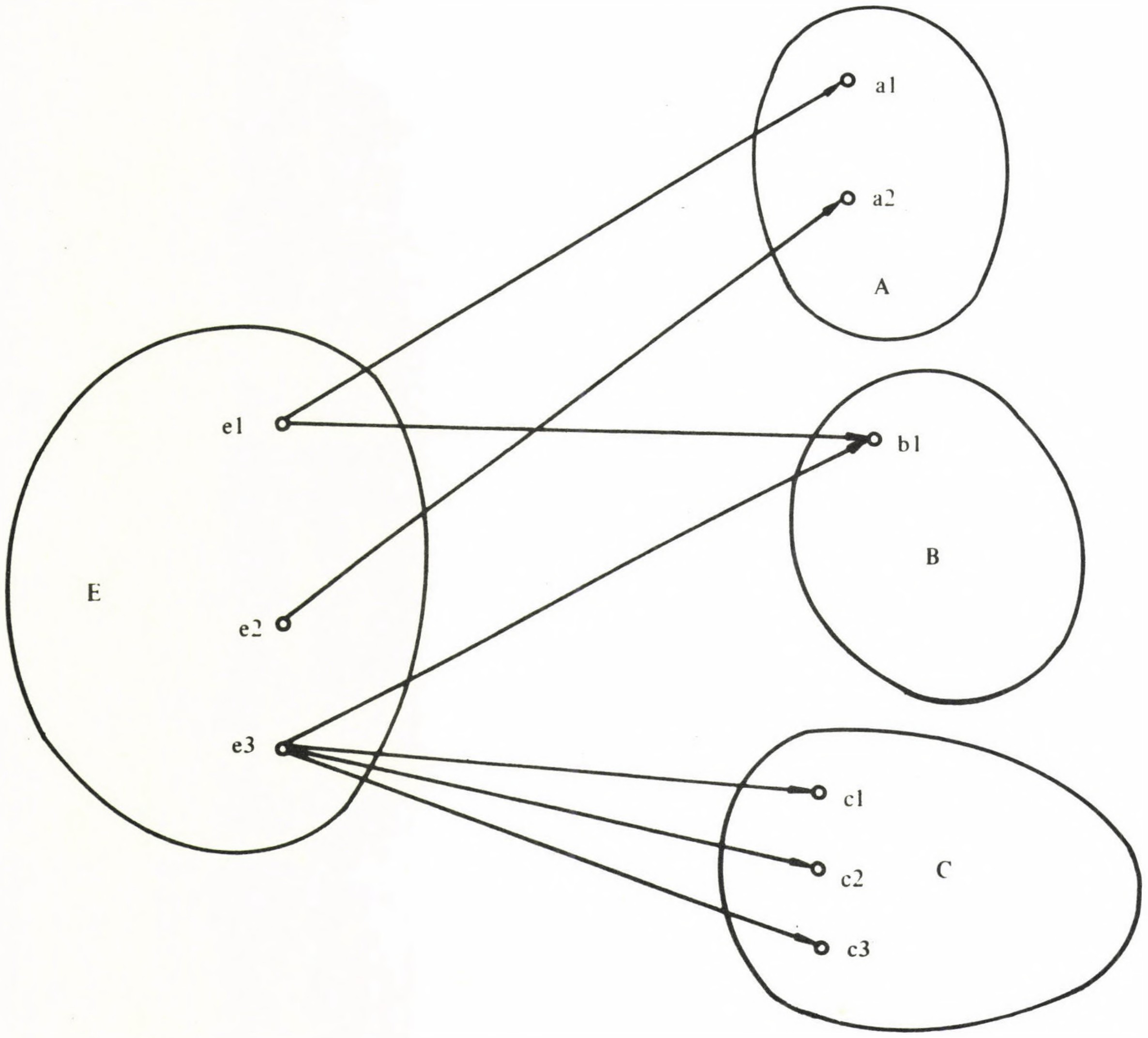


Figure 4

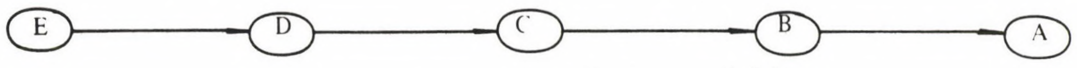


Figure 5.1

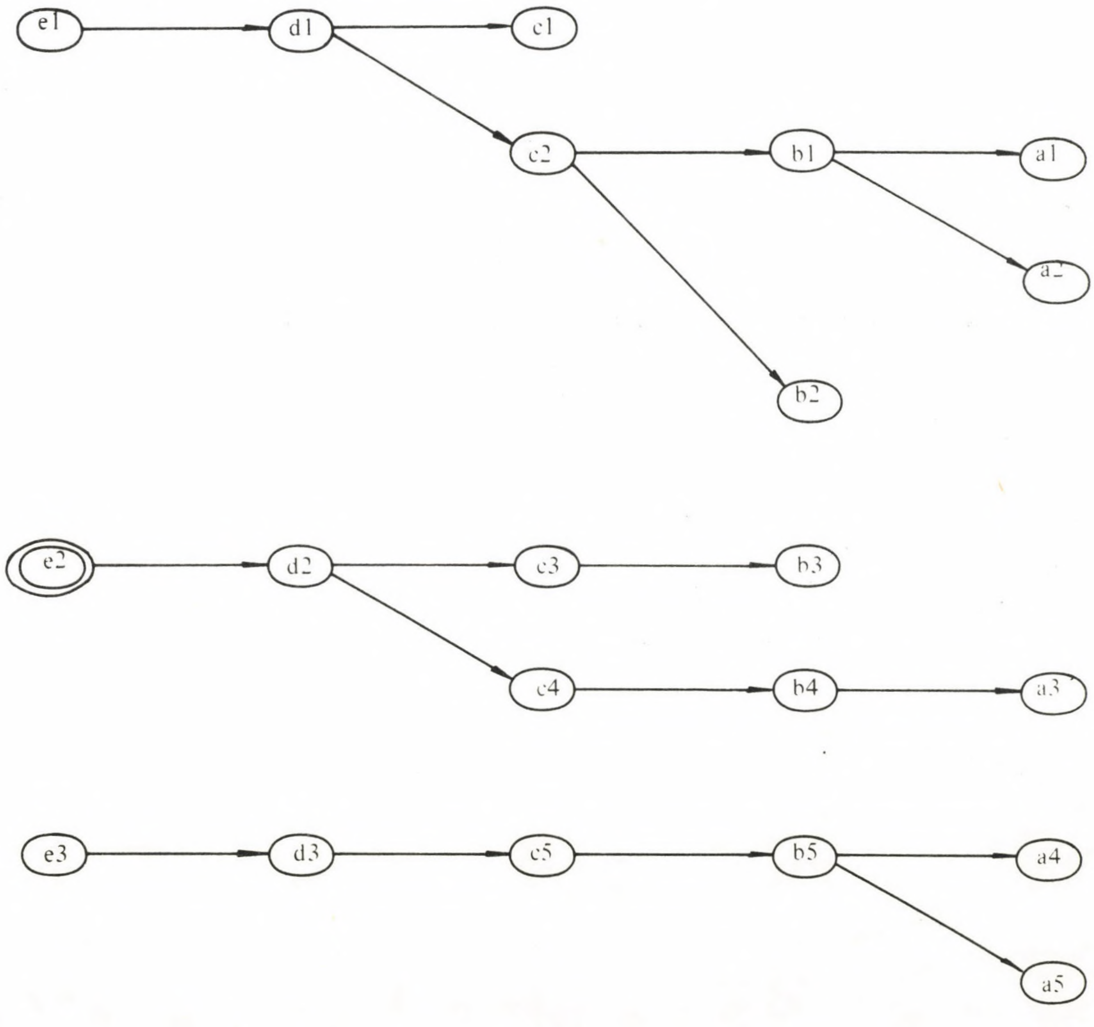


Figure 5.2

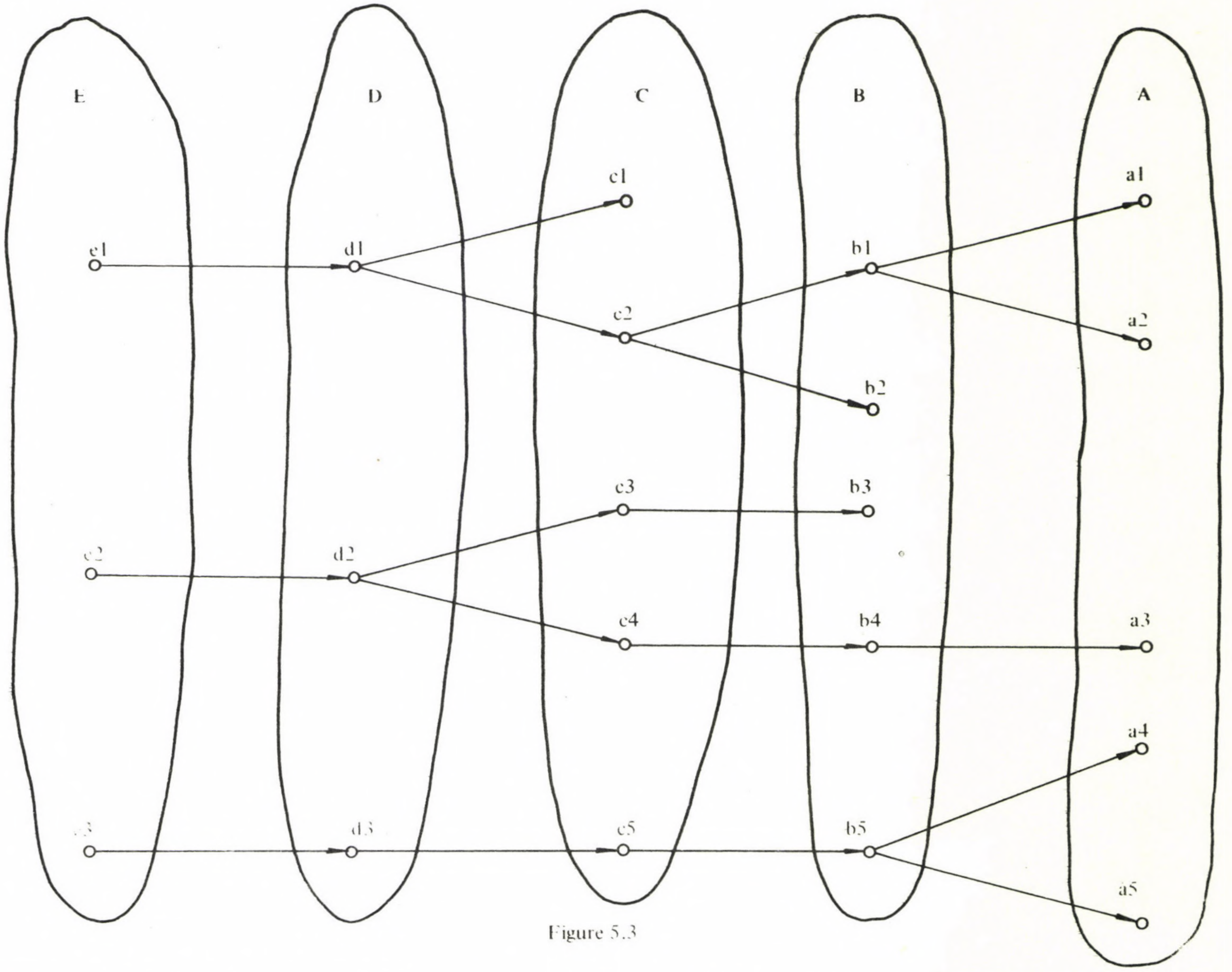


Figure 5.3

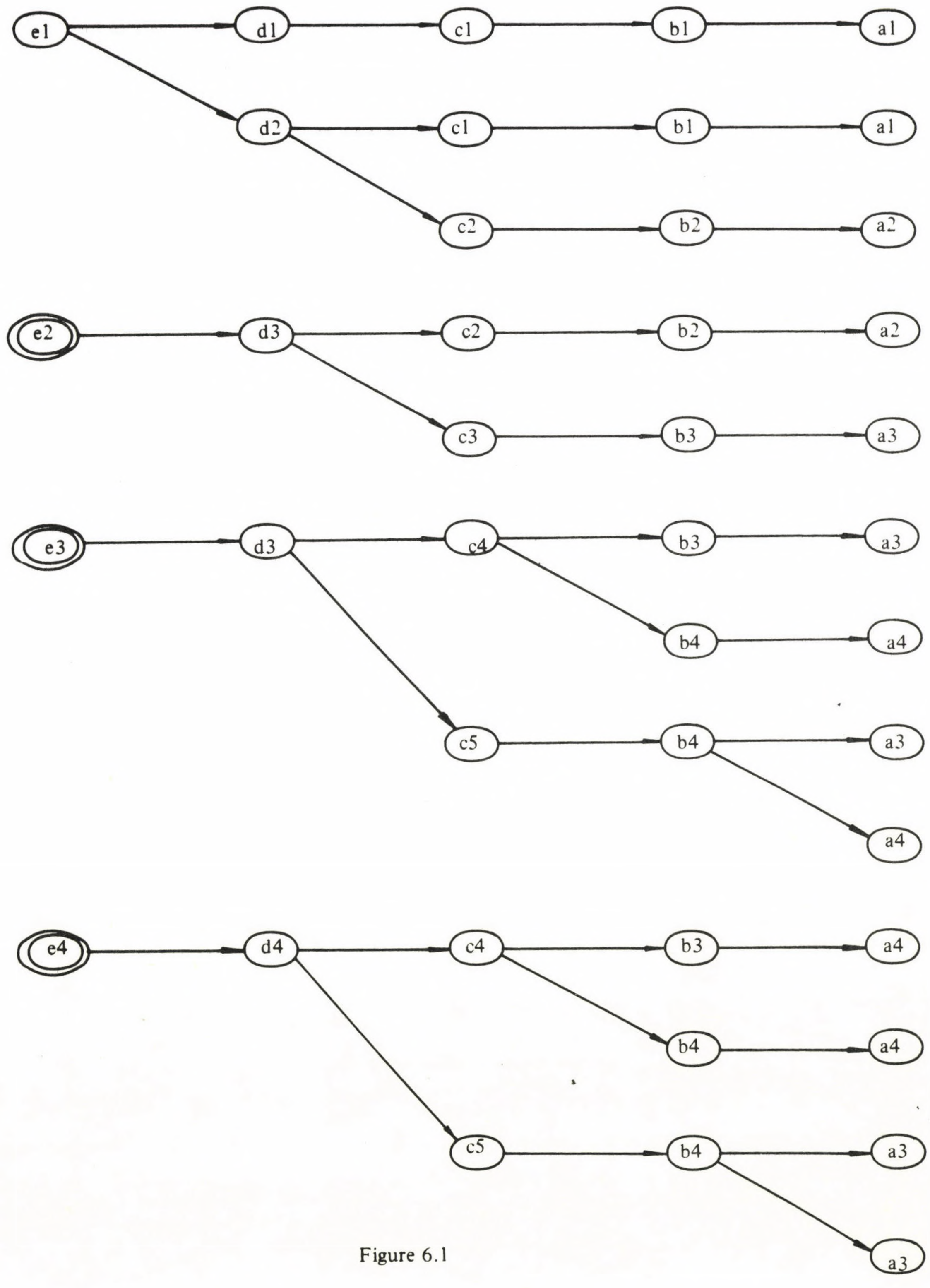

e1
d1
c1
b1
a1
d2
c1
b1
a1
c2
b2
a2
e2
d3
c2
b2
a2
c3
b3
a3
e3
d3
c4
b3
a3
b4
a4
c5
b4
a3
a4
e4
d4
c4
b3
a4
b4
a4
c5
b4
a3
a3

Figure 6.1

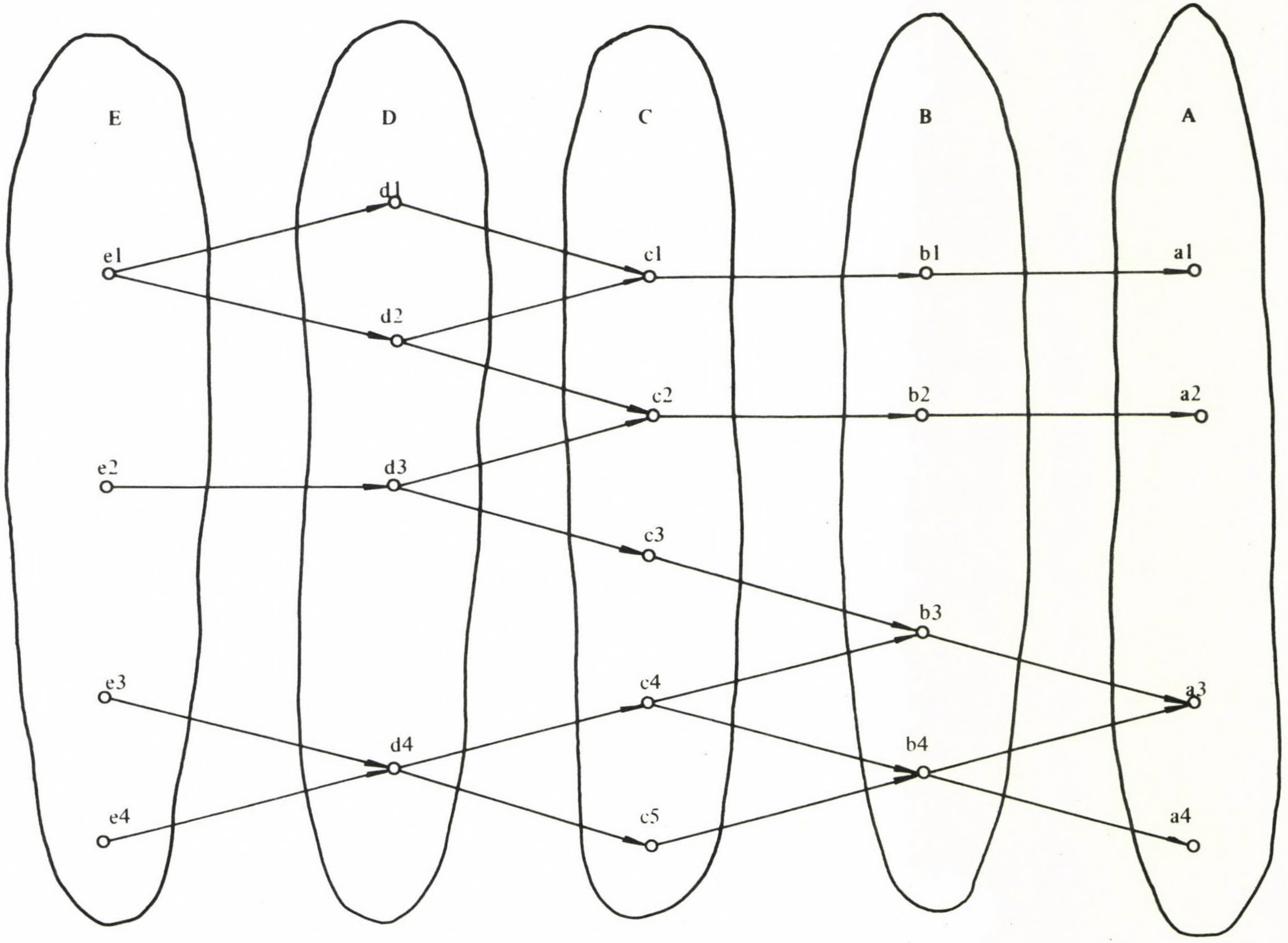


Figure 6.2

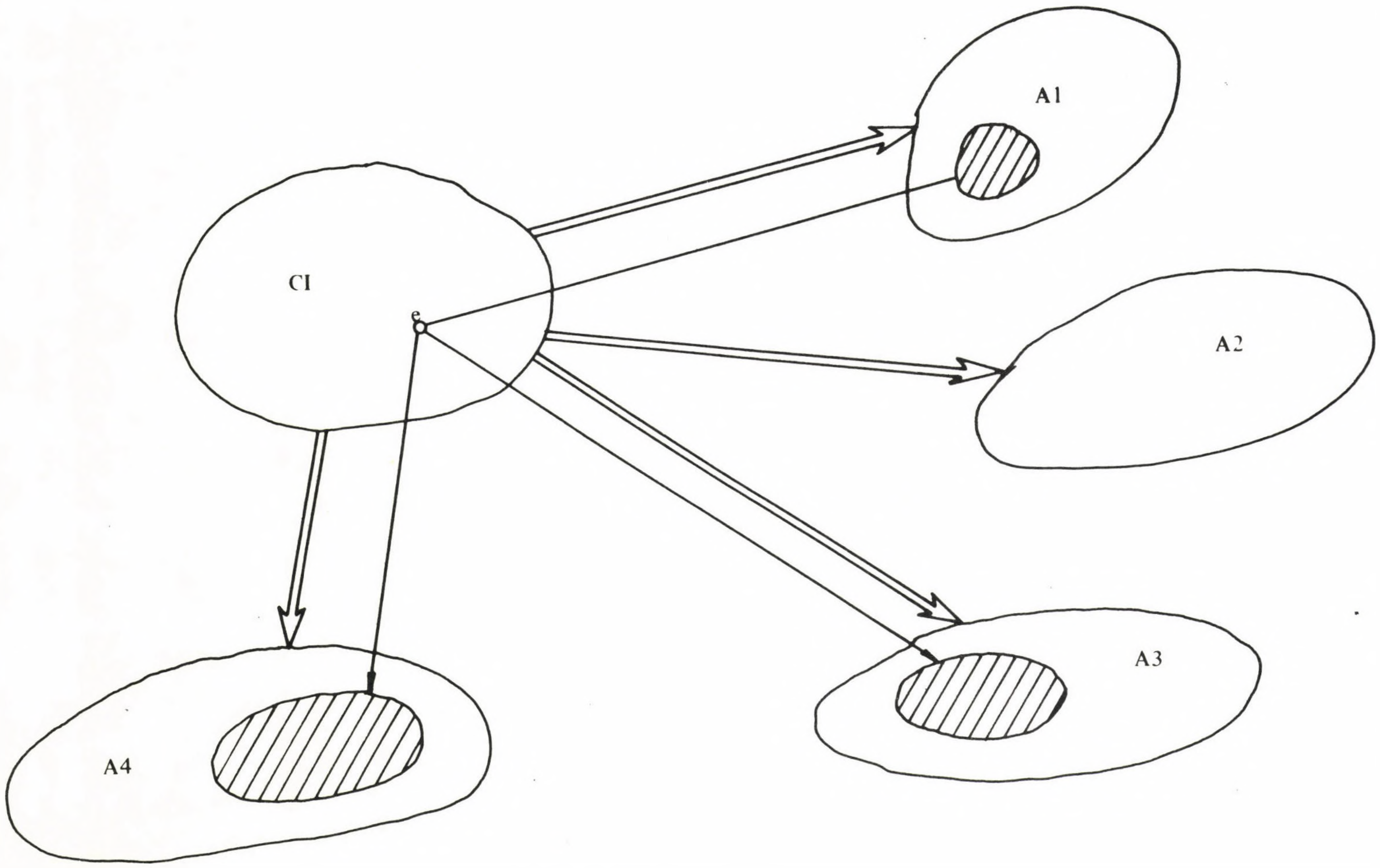


Figure 7

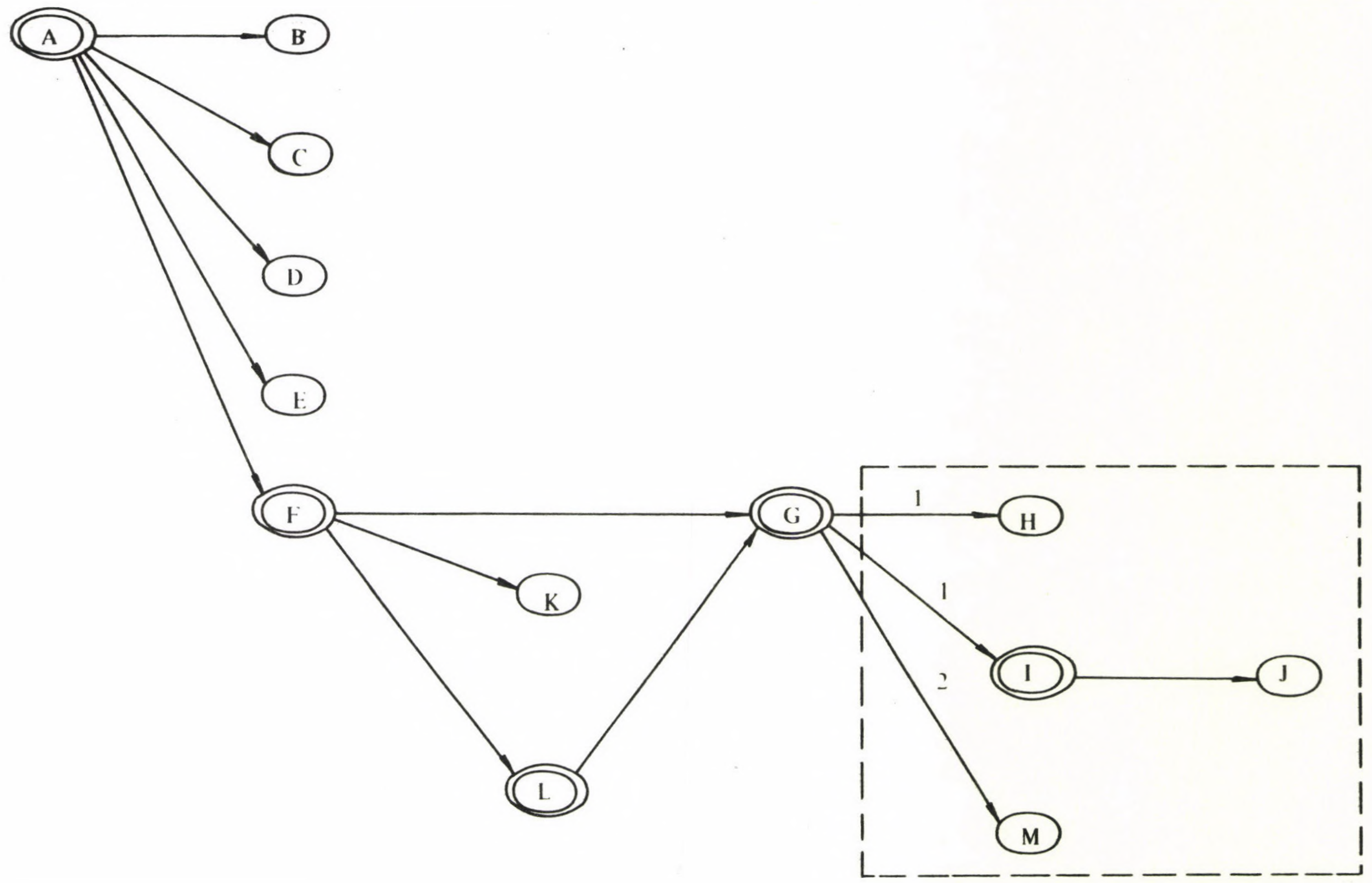


Figure 8

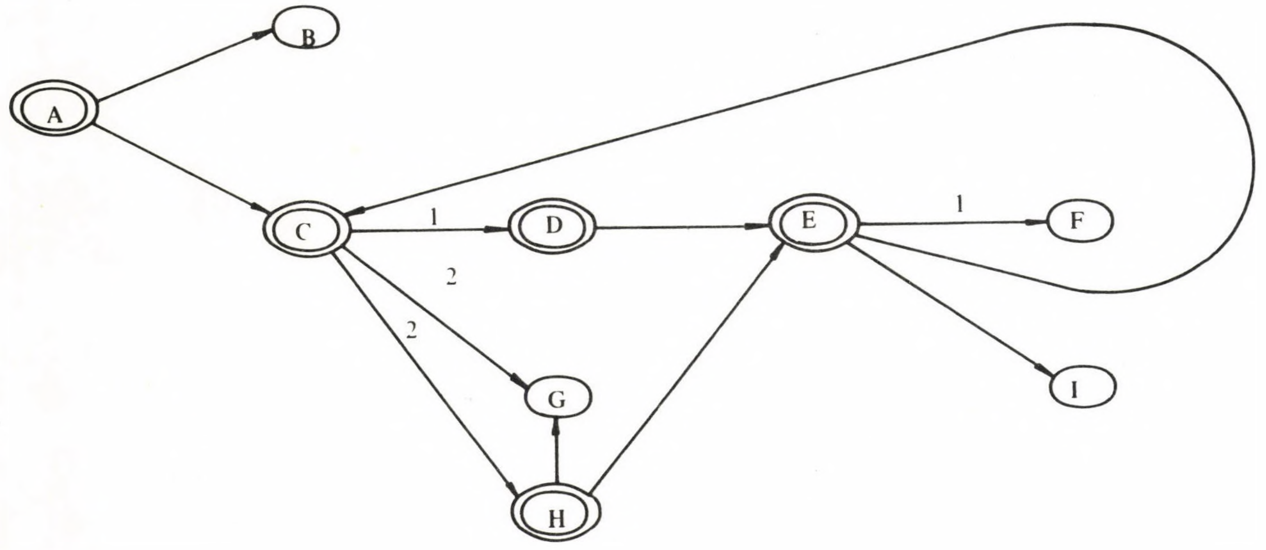


Figure 9

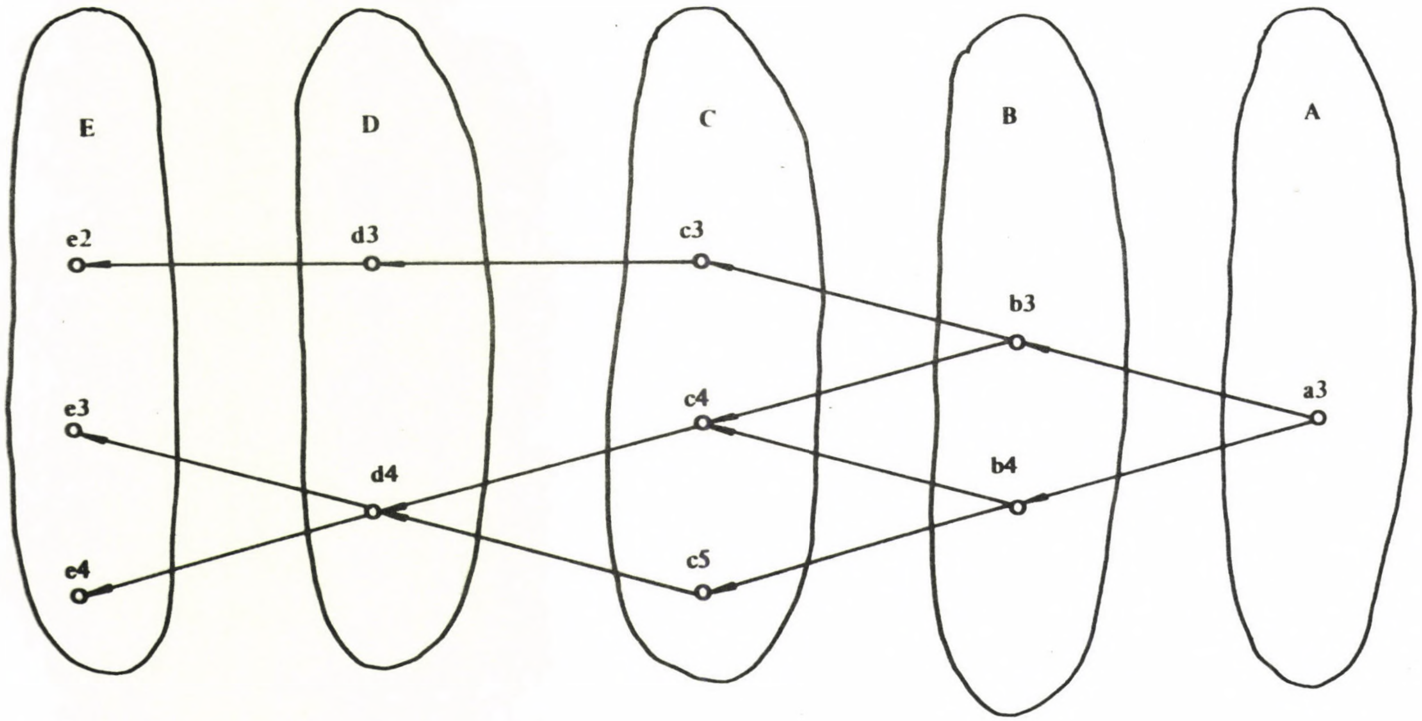


Figure 10.1

$$\begin{tikzcd}
e2 & d3 \arrow[l] & c3 \arrow[l] & b3 \arrow[l] \arrow[dl] & a3 \arrow[l] \arrow[dddl] \\
e3 & d4 \arrow[l] \arrow[dl] & c4 \arrow[l] & & \\
e4 & & & & \\
e3 & d4 \arrow[l] \arrow[dl] & c4 \arrow[l] & b4 \arrow[l] \arrow[ddl] & \\
e4 & & & & \\
e3 & d4 \arrow[l] \arrow[dl] & c5 \arrow[l] & & \\
e4 & & & &
\end{tikzcd}$$

Figure 10.2

$$\begin{tikzcd}
e2 & d3 \arrow[l] & c3 \arrow[l] & b3 \arrow[l] \arrow[dl] & a3 \arrow[l] \arrow[dddl] \\
e3 & d4 \arrow[l] \arrow[dl] & c4 \arrow[l] & & \\
e4 & & & & \\
 & & c5 & b4 \arrow[l] &
\end{tikzcd}$$

Figure 10.3

CELLPROCESSORS IN COMPUTER ARCHITECTURE

T. Legendi

Research Group on Mathematical Logic and Theory of Automata,
Hungarian Academy of Sciences

The real effectivity of digital computers – characterized by the working/waiting ratio of basic elements, gates and bits –*is very low.* There are also speed limits for the basically sequentially organized computers.

Cellular automata organization offers *in principle a solution* to these problems. However the *traditional approach* – stand alone cellular computer consisting of a high number of cells with a fixed relatively large transition function – gives results of only *theoretical importance* at the present state of technology – which demands at the same time the development of a homogeneous basic element.

This paper gives proposals for the structure and programming of *very effective medium speed cellprocessors based on existing technology*:

– *Cellprocessors* are treated as an *organic part of computer architecture* for solving tasks of a wide but limited class of algorithms.

– The *flexibility* of cells is increased by *variable transition function* which *reduces the size of cells* and the number of cells for solving a given task.

– Reorganization of processing in a cellular space – *centralized sequential transition function processing* – *drastically reduces the size of cells* thus making it possible to integrate 10^2-10^3 cells on one chip. With the growing number of cells executing the same transition function *price/performance is improved,* since the loss of speed depends only on the size of the transition function. (This advantage againts supposed parallel spaces is also independent of the development of technology.)

– A 16 state cell for general computations and a 2 state cell for specific applications have been designed and models have been built.

– Cellular automata cross–software – simulation languages and a transition function minimization language – have been implemented.

– General principles of cellular space programming – mapping algorithms on sets of interrelated processing elements connected and working in pipe–line – have been introduced. The conception of a cellular macroassembly language and directions towards higher level cellular languages are shown.

1. The actuality of the topic and the goals of research

The objective is justified by the low level of effectivity of computers in common use; $10^{-4} - 10^{-6}$ part of the hardware components work usefully at the same time – although in principle all of them could work in parallel. Speed limits for sequential processing also involve other organization principles.

Cellular automata could satisfy in principle these requirements.

The implementation of cellular automata for practical purposes first became realistic with the advent of the LSI technology. The *heterogeneity of LSI* circuits (because of their complexity) means at the same time a *demand* for a *homogeneous basic element* suitable for mass production.

According to estimations [13] further development of technology and detailed research will make general the commercial use of homogeneous computers, but not before the mid 80'-s.

This paper reports our research in order to speed up this process. The main *goal* is to design a *smaller new basic element* built on the basis of *existing technology* and to ensure *more effective and/or faster processing.*

2. The role of cellular automata in computer architecture

Theoretical [1,2] and practical [13, 3, 4] works concentrate mainly on the idea of a *stand alone cellular automata-computer based on cells with fixed transition function.*

The main advantage of this approach is the total homogeneity (of neighbourhood and transition functions) which simplifies theory, hardware design and programming of the cellular space.

The relatively small programmable basic element (cell) guarantees deep simulation level; gates and hardware constructions may be embedded into cellular spaces by software means.

However there are arguments against a stand alone cellular automata–computer. The most important of them states that *cellular automata* (especially taking into consideration the existing technology) *cannot be applied in an economical way for the execution of arbitrary tasks.* The main reason for this is that in general it is very hard to utilize the level of paralelism of cellular spaces.

Among other problems the initialization of the space and I/O in the traditional way – only through dummy cells – are quite inconvenient and slow.

The connection of a cellular computer to mass storage is also unsolved.

It is natural to use digital computers to solve the above problems. For example a computer can load the initial configuration (program) into the cellular space and can handle the

I/O as well. In this way there is no need for large and slow configurations (the own software of the cellular automata), since computers may also assemble cellular programs.

Special purpose applications may be satisfied by a system consisting of only one digital processor and one cellular processor, where the cellular automata may be interpreted as a peripheral firmware which is able to execute an extra instruction of the digital computer (see.e.g. [6] where picture preprocessing is executed, speeding up the computation with a factor of 10^4).

Another possible construction would be to apply a computer as a front–end processor to a larger cellular space.

For *general* computations it is reasonable to use *cellular automata as processors in an architecture* where they execute algorithms effectively computable in a parallel way at cell level. Examples are indicated in the chapter on cellular programming. Extending the class of effectively computable (in cellular spaces) algorithms is an important research task. Algorithms outside of this class should be executed by other processors in the architecture.

3. The transition function problem

A fixed transition function is not flexible enough and as a consequence it must be relatively large to be universal. In this way the use of *variable transition functions* also helps considerably to decrease the cell size. A fixed transition function may be interpreted as a union of partly defined functions (subfunctions) [1,2,5], since practically, in different groups of cells different subfunctions are used during longer periods, rather than the whole transition function. In this way a space consisting of *groups of cells with independently variable functions can replace a space of cells with a larger fixed transition function.*

There is another obvious advantage – the possible use of arbitrary functions, i.e. not only a fixed set of subfunctions may be accessed.

The definition of transition functions for different computations is a cell microprogramming task which is carried out now by hand (in the future this work should be automated by cellular programming languages, at first partly and then totally).

Cell microprogramming means a big economy in configuration (program) size and thus in execution speed as well; using special microprograms small groups of cells or even individual cells may replace larger configurations of a space with a fixed transition function.

Cell microprogramming does not exclude production homogeneity.

Assembling and loading of microprograms for cellprocessors is the task of other processors in the task of other processors in the architecture.

4. Cellprocessors

The size and therefore the economy and programmability of cellprocessors is determined mainly by the size of the basic cell. A fixed set of states and neighbours always reflects a compromise: in a space of smaller cells more cells are needed for a given task (space, speed); in the case of larger cells less cells are needed, speed increases, but the cells are less utilized [5].

Theoretical and simulation results, as well as the existing technology, suggest choosing for the basic element a maximum 16 state cell with a (static) neighbourhood of 4 to 8 cells. For special classes of algorithms 2 state cells may offer special advantages.

In the previous chapter the necessity and advantages of the variable transition function were explained. Here we emphasize that its use reduces the size of the basic cell and the number of cells needed for a given task.

The technical solution does not involve any serious problem. Existing design methods and technology determine the use of RAM memory for storing transition functions.

In this case the next state of a cell is defined by mapping the state word composed of the neighbour cells – including the cell itself – onto a contiguous address interval of the RAM where the result points to the value of the next state. This is a totally homogeneous construction from the production point of view.

However a cell containing a RAM represented transition function, even with relatively few states and neighbours, and some limitations for possible transition functions, is quite expensive as compared with the computational power of a single cell.

A relatively obvious method may be suggested to improve the price/performance ratio. It is based on the contradiction between the sizes of the memory and the processor part of a cell. A cell, as a basic element of a special distributed computing system, has a memory component (its own state, 1-4 bits) and an information processing component (finite automata with 4-20 bits of input and 1-4 bits of output). It is natural *to centralize the automata part* of cells having the same transition function. In this case, a cell does not include any processor component. It has only a memory component plus a small additional circuit which interacts with the neighbour cells and with the centralized transition function and stores the resulting new state in the memory component.

This organization results in slower speed and cheaper, modular construction. Such a quasi-space may be interpreted as a cellular space emulator taking into consideration the usual definition. This quasi – space emulator works as an ordinary space – only at a lower speed.

This centralization is naturally modular in the case of RAM stored transition functions and eliminates a specific problem: namely, that loading the transition function into each cell may cause problems.

The main advantage of the method is that *the size of a cell is drastically reduced,* giving chances for implementation of 10^2–10^4 cells on one chip which is the *preliminary design condition of an acceptable cellprocessor,* containing 10^5–10^7 cells.

But how has price/performance been improved? The loss of speed seems to compensate for the decrease in price. Although the loss of speed is necessary, the solution proposed by T. Toffoli [7] for experimental cellular space emulators makes *the loss of speed proportional to the length of the microprogram* (proportional to the number of terms of the transition function) *rather than to the number of cells* connected to a RAM stored microprogram. This means that by increasing the number of cells belonging to the same RAM, price/performance is improved. This argument shows not only that in this way are there chances for implementation based on existing technologies, but (it should also be pointed out), that having better technologies in the future, which will enable the integration of a minimal sufficient number of (non quasi) cells on one chip, will not make the above organization unecessary. Of course, the number of quasi-cells on one chip increases too, and therefore the advantage in price/performance remains. When extra speed is necessary, however, a genuinely parallel space should be used.

Returning to the actual situation, the loss of speed (relative to a completely parallel space) does not effect the priority in speed against sequential or mainly sequential computers (estimated order of 2 magnitudes) and the advantage in more frequent use of components (estimated order of 3 magnitudes). The legitimacy of these estimations depends on the concrete parameters of the cellprocessors to be implemented, and on the cellular programming and microprogramming.

Two types of quasi–cells with fixed 5 neighbours (including the cell itself) have been designed: a 2 state one (about 10 gates and in average 8 substeps e.g. 8 microinstructions) and a 16 state one (about 100 gates, the number of substeps largely depends on the transition function; it may vary from 40 to 1000).

These results ensure minimum precondition for producing cellprocessors. Details of the system design may be found in L7.

According to L7 a *basic logical module* is a *group of cells with a common variable transition function* which may be realized as a *set of physical modules – cell microprocessors (CMP) consisting of an internal array of quasi–cells, driven by a common external RAM stored microprogram.*

L7 contains further optimization techniques to reduce the number and time of substeps while preserving the relative simplicity of the quasi–cells. Minimization is also supported by software tools (L6).

According to L7 a *cellprocessor* consists of a set of CMP-s and RAM-s, and a microprocessor which controls the CMP-s and RAM-s including the I/O among them (internal I/O) and the external I/O with the other processors in the architecture.

The programs of a cellprocessor (the programs for the microprocessor, the cellular programs and microprograms) should usually be computed (in the optimal case by a compiler of a cellular language) and transformed by other processors in the architecture.

5. **Software tools**

In order to maintain design and later use of the described cellprocessor system, software tools (running on digital computers) are required for microprogramming, machine code and higher level cellular programming.

Three implemented subsystems of a future cross-software package and further design principles are explained in the next paragraphs.

5.a. **The CELLAS cellular space simulation language**

In the abundant literature on cellular automata there are relatively few simulation languages [2, 9, 10, 11, 12] and their structures are not suitable for our purposes. For this reason it was necessary to define a relatively simple language with the consideration that a limited inhomogeneity is needed and that simulation should be as effective as possible. The CELLAS language may be used for more general tasks than the explicit goals of the author but is far from being a general cellular space simulation language.

For special purpose simulations we have adapted other simulators: the interpreter of the SICELA language [9] (with some improvement; a higher level input language was implemented) and the CELIA [11, 12] processor.

CELLAS is a command-type language, implemented in form of interpreters.

The basic group of instructions in CELLAS ensures the direct simulation of the space (input vectors to dummy cells on the boundary of the space may be specified if desired). The interpreter computes the transition function only on open cells that may change their states; on closed cells (that certainly will not change their states) it does not perform any operations. A one step look ahead algorithm continously classifies the cells as being open or closed. On permanently (more than 1 step) closed cells the look ahead does not require any computation. This look ahead algorithm ensures the significant increase of speed which is especially needed, since the characteristics of the task demand ineffective simulation (on a relatively large sequential digital computer) of many relatively small elements working in parallel.

The group of instructions for *functional simulation* helps in speeding up simulation as well. These instructions ensure direct simulation of groups of cells (not cell by cell) and the connections among groups and cells. The main advantage of this group of instructions is given mainly by the possibility of *top-down programming* rather than by economy. The whole configuration (program) should be decomposed; the decomposition can be tested by functional simulation and afterwards the parts may be decomposed again or changed to real cell configurations continuosly and independently. At each level the space may be simulated, giving a very strong debugging tool at the same time.

Transition function definition instructions enable a simple function decription including the inhomogeneous case.

From the users' point of view the simple, flexible group of instructions for *control of printing* the space is very important. It is possible to define independently which parts of the space should be printed (or sent to a file) and in what form and at which steps.

The space may be continuosly monitored by ON instructions – when a condition of a previously executed ON instruction is met, the prescribed action (a CELLAS program) of the same ON instruction is executed. There are in the language *assignment*, very simple *arithmetic, I/O* and *library handling* instructions for the basic data types (integers, vectors, configurations, transitions functions and CELLAS programs). Branching is performed by simple skip instructions.

More detailed descriptions of the language may be found in L4, L2, L3. Different versions have been implemented in FORTRAN IV on CDC 3300, IBM 360/40 and Honeywell 6060 computers.

5b **The INTERCELLAS interactive cellular space simulation language**

The language is a subset of CELLAS, except for a few added instructions for handling interrupts and for dialogue [L5, L9].

A typical simplification is shown by the changed ON instruction group: here the conditions cannot be complex and the action is to pass control to the main input peripheral (console, in general), i.e. the interpreter prints the condition that was met and waits for the next instruction from the main peripheral. Different versions of the language have been implemented on CII-10010, PDP-8 and MITRA 15 minicomputers.

5c **The TRANSCELL cellular microprogramming language (for definition and minimization of the transition function)**

The size of a microprogram effects RAM costs, but this is not too serious taking into consideration the relatively great number of quasi-cells belonging to the same RAM.

However, the size of microprograms is critical, since the speed is proportional to the length of the microprogram as it is executed sequentially.

Therefore microprogram minimization is of prime importance.

A semi-automatic solution is given by TRANSCELL. Special transition function definition instructions may describe transition functions, and minimization directives control the minimization process. Detailed descriptions of the language and its practical application may be found in [L6, L7, L8].

The TRANSCELL interpreter may produce a minimized microprogram for the hardware and/or may produce a transformed minimized transition function table and a FORTRAN search program fitting the specialities (caused by minimization) of the generated table. The search program maps any given combination of a cell's state and those of its neighbours onto the contiguous address interval of the table containing the values of the next states.

Thus TRANSCELL is a medium level cell microprogramming language; its processor will be built in the simulator in the near future, and it may serve as a subsystem in a cellular programming language as well.

6. **Cellular programming**

The first *cellular programs* appeared in the form of *proofs in constructive mathematics* (simulation of the universal Turing machine and self-reproductive automata). The traditional way was to embed hardware-like components into the cellular space. This approach to the problem involved construction of *modular, hierarchically structured configurations.*

Here elementary subtasks are realized by small basically static configurations interconnected for solving subtasks; there is an explicit analogy with subfunctions, subroutines, modules. Higher levels may be built up hiererchically by interconnections.

For *special purpose applications* (such as wave- and simulation-spread out, simple self reproduction and picture preprocessing) free structured (non-hierarchical, non-modular) cellular programs may be used that do not directly control the flow of information in the space. Such programs may be characterized as *direct mapping of the problems* (embedding of systems) into the cellular space. Their application is basically limited for modelling discrete systems, or problems equivalent to them where homogeneous local changes dominate the behaviour of the system.

For general purpose computations the embedding and hierarchical interconnections of hardware components give the possibility of emulating digital processors. However the emulation of sequentially working hardware (devices like general purpose computers with CPU, memory, etc.) seems to be justified only for theoretical and simulation purposes. Such emulation needs a high number of cells and for general purpose computations it is very ineffective since the emulation factor effects further decreases in the low price/performance of the sequential machine. Essentially a cellular space type distributed computing system is very different from a sequential machine consisting of relatively large interacting modules with long data paths. A more effective way would be to emulate systems with more parallelism and short data paths.

The distributed computing nature of the cellular spaces justifies our proposal for effective, modular/parallel cellprogramming by means of *embedding into the cellular space processing elements working in parallel on moving data. Different algorithms are to be mapped onto different sets and interconnections of processing elements.*

The principle of modular cellprogramming is preserved. For elementary configurations *software configurations* (processing elements, open coroutines) are selected.

The cellular spece is used in fact as a *distibuted computing system* by mapping the algorithms directly into the cellular space in the form of static, modular, *hierarchical subconfigurations connencted in pipe-line,* e.g. data are moved and transformed along the pipe--line continously. All the subconfigurations work at the same time ensuring high productivity. The subconfigurations should be of medium or small size and may be interpreted as open coroutines (e.g. their function may be A = B + C of A = A + 1, search form a symbol-table; in general the instruction set of a cellprogramming language).

In this way cellular processors are suitable for executing a wide class of parallel algorithms (including array processing). *Direct* (machine code level) *programming* and microprogramming are enabled by the use of cellular *simulation languages* (ensuring many extra utilities but not generating configurations). One type of utility shows possible development to increase the level of programming, namely the library handling routines. It is possible to write subconfigurations by hand, to store them in a library and to call them afterwards.

An assembly type cellprogramming language may consist of an instruction set to call implicitly the members of a set of basic subconfigurations. There exists a difference not only in form but also in that being more than the original utility (in addition to the basic subconfigurations at our disposal) the call may specify parameters which effect the called subconfigurations so that the process seems to be a simple configuration generation rather then making a copy from a library.

In a *cell macro assembly language* the user will have the possibility to add his own subconfigurations to the assembler's basic set.

The trend of development points to more sophisticated semi-automatic and *automatic subconfiguration generations.* In higher level cell-languages the algorithmic description of subfunctions and their relations will be compiled to subconfigurations and their interconnections.

7. Summary

This paper treats cellprocessors (cellular automata type processors) as *organic parts in computer architecture* and gives proposals for the structure of very *effective medium speed cellprocessors based on existing technology.*

The task of cellprocessors in the architecture should be to execute procedures effectively computable in parallel at cell level.

The proposed cellprocessor emulates a *finite cellular space* where different groups of cells (represented by cell microprocessors) may have independently variable transition functions (represented by RAM memories).

A cell microprocessor consists of an array of (quasi) cells that may execute the same microinstruction simultaneously. A microprogram (equivalent to a complete transition function, represented by RAM contents) is executed sequentially. Therefore microprogram minimization is of prime importance.

A cell may interpret instructions of two types: during the first phase of a transition step the execution of *feature extraction instructions* results in storing the characteristic information about the neighbourhood (in each cell), and during the second phase the execution of *state assignment instructions* defines the next state (separately in each cell, using the stored neighbourhood description).

For general computations a *16 state cell* / ~ 100 gates/ and for specific purpose applications a *2 state cell* /~10 gates/ have been constructed.

Programs (initial configurations) and microprograms should be computed and loaded by other processors of the architecture. The information to be processed (input to cellprocessor) and the results (output from cellprocessor) are handled in the same way.

General procedures should be programmed by embedding software configurations (open coroutines) connected in *pipe-line.* Data are moved and transformed parallel along the pipe-line continuosly.

Software support of the above system consists of *simulation languages and a microprogram definition and minimization language.* Higher level cellular languages are under design.

Literature

[1] Neumann, J.: Theory of Automata: Construction, Reproduction, Homogeneity, Part II of "The Theory of Self-Reproducing Automata" ed. A.W. Burks. University of Illinois, 1966.

[2] Codd, E.F.: Cellular Automata Academic Press. Inc. New York, London 1968.

[3] Fáy, Gy.: Circuitry Realization of Codd's Cellular Automaton's Cell. Technical Report KGM ISZSZI, 1973.

[4] Takács, D.V. (Mrs): A Bootstrap for Codd's Cellular Space, I. Hungarian Conference on Computer Science, Székesfehérvár, 1973.

[5] Nourai, Farhad – Sohrab Kasef, R.: A Universal Four-State Cellular Computer IEEE Transactions on Computers Vol. C-24. No. 8. August 1975.

[6] Preston, Jr. K.: Use of the Golay Logic Processor in Pattern-Recognition Studies Using Hexagonal Neighborhood Logic
Symposium on Computer and Automata
Polytechnic Institute of Brooklyn, April, 13-15. 1971.

[7] Toffoli, T.: On the Large-Scale Implementation of Cellular Spaces by Means of Integrated--Circuit Arrays. CNR, Istituto per le Applicazioni del Calcolo, 1972.

[8] Domán, A.: A Model of Parallel Processing. Információ Elektronika 1975.2. -in Hungarian-

[9] Vollmar, R.: Über einen Interpretierer zur Simulation Zelluraren Automaten. Angewandte Informatik 6/1973.

[10] Brender , R.F.: A Programming System for the Simulation of Cellular Spaces The University of Michigan, Ann Arbor 1970. (Ph. D. Thesis)

[11] Baker, R. – B.T. Herman: CELLA – A Cellular Linear Iterative Array Simulator. Proceedings of the Fourth Conference on Applications of Simulation, pp. 64-73 (1970).

[12] Wu-Hung-Liu: CELIA Users Manual
Dept. of Computer Science, State University of New York at Buffalo, October 1972.

[13] Cellular Spaces, Homogeneous Structures -in Russian - Institute of Mathematical Machines, Warsaw, 1973.

[14] Cellular Automata, Bibliography, KGM ISZSZI Budapest, 1973.

[15] Bibliography on Cellular Automata Papers in Internal Series of KGM ISZSZI 1971-1974.

[16] Vollmar, R. (manuscript; the bibliography from A.R. Smith III. Introduction and Survey on Polyautomata Theory with supplement)

[17] Rozenberg, G. – Wood,D.: Generative Models for Parallel Processes. The Computer Journal Volume 17 Number 4 pp. 344-348.

[18] Toffoli, T.: Cellular Spaces – An Extensive Bibliography. Dept. of Computer and Communications Sciences. The University of Michigan. 1976.

Publications and internal reports.

L1 Legendi, T.: Simulation and Synthesis of Cellular Automata. Conference on "Programming Systems'75".
Szeged, (in Hungarian).

L2 Legendi, T.: Simulation of Cellular Automata, the Simulation Language CELLAS. Conference on"Simulation in Medical, Technical and Economy Sciences".
Pécs 1975 (in Hungarian).

L3 Legendi, T.: The CELLAS Cellular Space Simulation Language, (Manuscript in Hungarian 1974.)
Martoni, V. – Legendi, T.: CELLAS 0.9 User's Manual, 1974 (in Hungarian)

L4 Czibik, I. – Legendi, T.: User's Manual CELLAS 1.0, 1976. (in Hungarian)

L5 Merényi, E.: The INTERCELLAS Interactive Cellular Simulation Language.
Diploma work. University JATE, Szeged 1975. (in Hungarian).

L6 Zsiros, P.: The TRANSCELL Cellular Automata Transition Function Definition and Minimization Language.
Diploma work, JATE (in Hungarian). (1975.)

L7 Legendi, T. – Kacsuk, P.: Hardware Design of a 16 state Cellprocessor.
Manuscript in Hungarian, Szeged, 1976.

L8 Legendi, T.: INTERCELLAS an Interactive Cellular Spece Simulation Language.
Acta Cybernetica, Hungarian Academy of Sciences. (1977.) Tom. 3, Fasc. 3
pp. 261-267.

L10 Legendi, T.: Programming of cellular processors.
"Cellular meeting" Braunschweig, 2-3 June 1977. Informatik – Berichte Nr. 7703
Technische Universitat Btaunschweig pp. 53-56.

L11 Legendi. T.: TRANSCELL – a cellular automata transition function and minimizatior language for cellular microprogramming.
Computational Linguistic and Computer Languages XII.

CELLULAR DESIGN PRINCIPLES:
A Case Study of Maximum Selection in CODD-ICRA Cellular Space
PART TWO
DETAILED DESIGN

G. Fay

Chapter 6

DATA, SIGNALS AND STAKING

6.1. Peripherals and capacities

The input data to the cellular maximum selector automaton MAXEL are the numbers given in the form of a system of l numbers of k bits each:

$$n^j = \sum_{i=1}^{k} 2^{k-1} x_i^j \qquad \text{for } j = 1,2, \ldots , l$$

here

$$x_i^j = 0 \quad \text{or} \quad x_i^j = 1 \qquad \text{for} \quad i = 1, \ldots , k; \quad j = 1, \ldots , l,$$

The least significant bit of n^j is x_k^j, and the most significant bit of n^j is x_j^i. Data are supposed to be stored in a peripheral device like a disc or drum having parallel access for all the n^j-s simultaneously.

As the necessary time lag between the consecutive bits is 26 shots, (see Ch. 10), one shot being around 1 microsecond, the throughput of the storage device (dumping speed) will be:

$$l x \frac{1}{26} \frac{\text{bit}}{\text{microsec}} = \frac{l}{8} \frac{1}{26} \text{ Mbyte /sec.} \approx 5l \text{ Kbyte/sec.}$$

Owing to the organization the throughput is independent of k.

The output data $n^j_{\max}$ are available roughly in

k x 200 microseconds

from which the first k x150 microseconds are for setting the MAXEL (first stage), and the remaining k x 50 microseconds are for the actual selection (second stage). In the case where $k = 50$ it is 0,1 msec. (= 10^{-4} sec). With a cellular space of CODD-ICRA type having

2.10^7 cells/space,
1 mm^2/cell,
1 microsec/shot
40 x 50 cells/module
k x l modules/device

one can process (select maximal) data at a speed of l numbers (k x 200 microsec = 5000 x $\frac{l}{k}$ numbers /sec = 5000 rec/sec = $\frac{5000}{8}$ l byte/sec .

In the case of (k = 50), l = 250 this speed amounts to 160 Kbyte/sec = 5000 records/sec = = 25000 numbers/sec being l number = kbites = $\frac{k}{8}$ byte and l bits = 1 record = $\frac{l}{k}$ number.

6.2. Signals

As elementary signals we use; [06], [07] and [– –], the lack of any signal. For [06],[07] see Codd, (1968). The seven basic and elementary signal (types) can be found in table 6.2 – 1. Cellnames in the fourth and fifth columns refer to figure 5.2 – 1 and tables 7.1 – 1,2.

6.3. Staking

The 0-1 configuration (staking) of a MAXEL-module can be seen on figure 6.3–1. One can check that all the warnings have been taken into consideration, viz.:

– there are no jogs (in Codd's sense);

– parallel paths are not sharing sheaths;

– no corners or junctions are placed opposite gates or locks;

– generally different parts are placed distant enough from each other;

– no possibility occurs for "lonely 2" configuration.

Staking can be carried out, theoretically, in a number of different ways.
Firstly the whole automaton MAXEL can be staked step by step by using a subroutine for moving the writing head. (c.f. Fazekas's RETINA in Fazekas, 1975.)

Secondly, using the RETINA, one presents the information about the staking, i.e. the 0-1 configuration row by row, at the edge of the cellspace;

TABLE 6.2-1 Signals

SYMBOL	SIGNALNAME	D E F I N I T I O N	ENTERING U_i^j AT	LEAVING U_i^j AT	SOJOURNING SHOTS
a	Activation signal	$a = [07]$, See fig. 9.1-1 and Ch. 9.1	$/E_1/_i^j$	$/S_1/_i^j$	73
cl_i	Collector signal	$cl_i = [--]$, if there is a $U_i^{j'}$ with $j' > j$ such that $/G_9/_i^j$ is on,	$/E_4/_i^j$	$/S_4/_i^j$	41
		$cl_i = [07]$ otherwise			
cr_i	Collector signal	$cr_i = [--]$ if there is a U_i^j with $j \geq 1$ such that $/G_9/_i^j$ is on	$/E_5/_i^j$	$/S_5/_i^j$	33
		$cr_i = [07]$ otherwise			
x_i^j	Databit signal	$x_i^j = [07]$ if its bit is 0	$/E_2/_i^j$	$/S_2/_i^j$	48
		$x_i^j = [06]$ if its bit is 1			
x_h^j	Heading information signal	$x_h^j = [07]$ if its bit is 0	$/E_1/_i^j$	$/G_5/_h^j$	51
		$x_h^j = [06]$ if its bit is 1	$h = i$		
x_t^j	Trailing information signal	$x_t^j = [07]$ if its bit is 0	$/E_1/_i^j$	$/S_1/_i^j$	73
		$x_t^j = [06]$ if its bit is 1	$i < t$	$i < t$	
r	Reset signal	$r = [06]$ see fig. 10.2-1 and Ch. 10.2	$/E_2/_i^j$	$/S_2/_i^j$	33

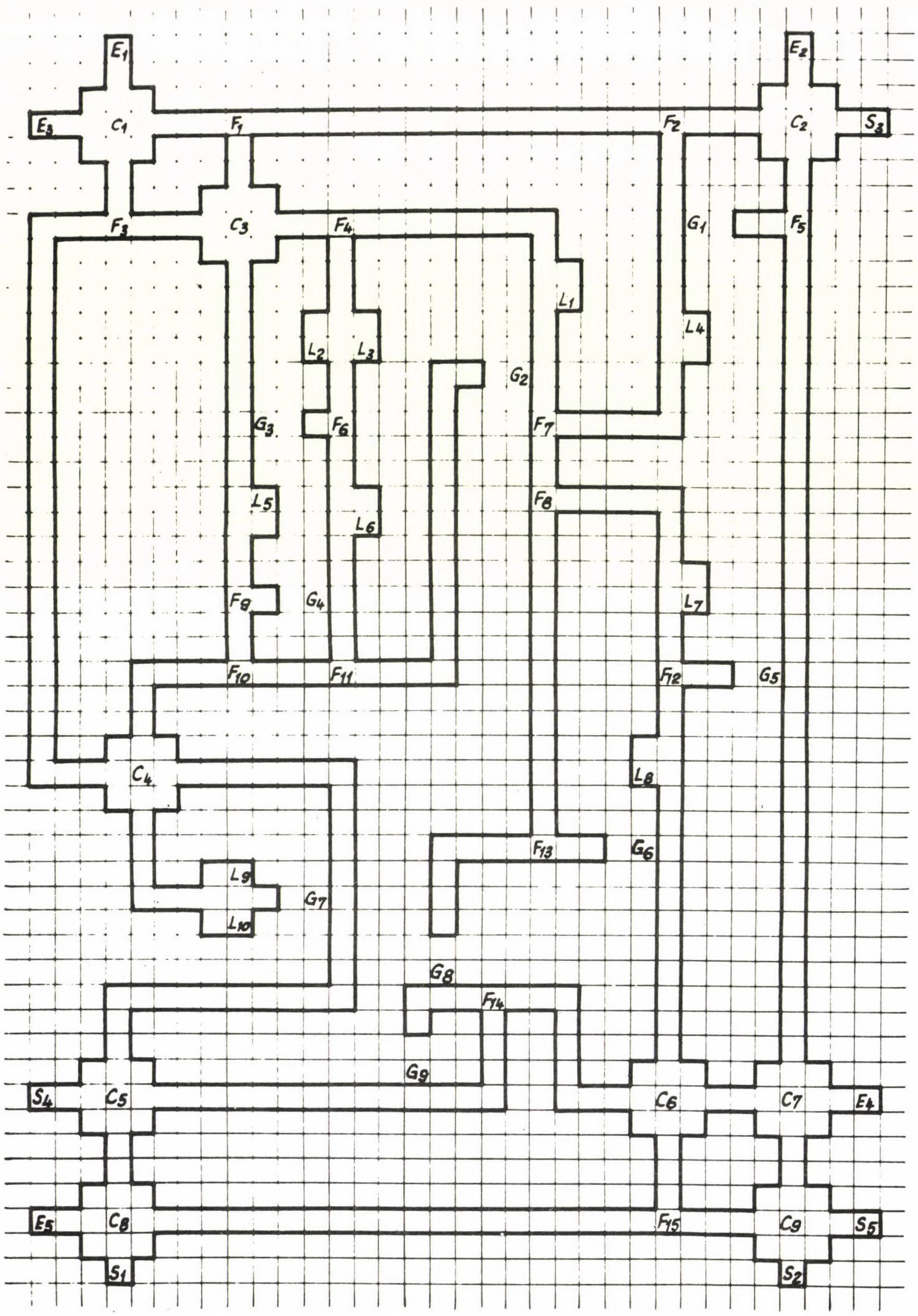


Figure 6.3-1

The staking of the MAXEL module.

Thirdly, using the shift procedure, (again invented by Fazekas, 1975), and

Fourthly, by the modular retina technique when only one module's program is supposed to be stored, $l = 2^8$ modules can be staked simultaneously using a writing arm-tree. Of course the MAXEL's perimeter structures (see figure 3.2-2) are tackled separately.

Roughly, an elementary staking step takes a hundred shots, a module consists of (approximately) 40 x 50 = 2000 cells and so, at 1 microsecond / shot the cycle time to stake k rows of a module (say 256 modules per row) is:

$$l \times (40 \times 50) \frac{\text{cells}}{l \text{ modules}} \times \frac{l \times 100 \text{ shots}}{l \text{ cells}} \times \frac{10^{-6}\text{sec}}{\text{shot}} \times \frac{k\text{modules}}{\text{MAXEL's core}} =$$

$$= k \times 200 \text{ milliseconds/MAXEL's core}$$

$k = 50$, <u>a staking takes 10 seconds.</u>

CHAPTER 7

ELEMENTARY STRUCTURES AND THEIR FUNCTIONS

7.1 Entries and sorties

In the frame of relational data processing (DSL ALPHA invented by Codd in 1970 and 1971) "ENTRY" can be considered as a relation of three attributes

ENTRY: (ENTRYSYMBOL, ENTRYNAME, ENTRYSIGNALS)

similarly:

SORTIE: (SORTIESYMBOL, SORTIENAME, SORTIESIGNALS)

Tables 7.1 – 1 and 7.1 – 2 are set up in this fashion. For signalnames see table 6.2 –1.

TABLE 7.1-1 Entries

ENTRY symbol	ENTRYNAME	Signal entering U_i^j
E_1	Information entry	$a = [07]$, $x_h^j = [07, 06]$, $h \geq i$
E_2	Data entry	$x_h^j = [07, 06]$ $h \geq 1$
E_3	Reset entry	$r = [06]$
E_4	Collector entry	$cl_i = [07, --]$
E_5	Corrector entry	$cr_i = [07, --]$

TABLE 7.1-2 Sorties

SORTIE symbol	SORTIENAME	Signal/s/ leaving U_i^j
S_1	Information sortie	$a = [07]$, $x_h^j = [07, 06]$, $h \geq i+1$
S_2	Data sortie	$x_h^j = [07, 06]$, $h \geq 1$
S_3	Reset sortie	$r = [06]$
S_4	Collector sortie	$cl_i = [07, --]$
S_5	Corrector sortie	$cr_i = [07, --]$

7.2 Crossovers and forks

Crossovers are characterized, from the designer's point of view, by the paths crossing (stuctural characteristic) and by the signals involved (functional characteristic). It is natural, therefore, to handle the set of all crossovers (of a MAXEL module) as a relation with three attributes (in Codd's sense of DSL ALPHA relational data calculus, Codd 1970 and 1971).

CROSSOVER: (CROSSYMBOL, CROSSPATHS, CROSSIGNALS)

On the other hand, forks serve to separate an inferior path from a superior path. With regard to forks, structural characterization seems to be enough (in our case), so for the sake of simplicity we omit the fourth attribute: involved signals (functional characterisitc). Thus, viewed as a relation, the set of all the forks of a MAXEL module is,

FORK: (FORKSYMBOL, SUPPATH, SUBPATH)

Tables 7.2 – 1 and 7.2 – 2 show the details. (for signal names see 6.2 –1.)

TABLE 7.2-1 Crossovers /for pathterms see table 8.1-2/

SYMBOL	CROSSING PATHS	AFFECTED SIGNALS IN U_i^j
C_1	RST+INF	$r = [06]$; $a = [07]$, $x_h^j = [07, 06]$, $h \geq i$
C_2	DTP+RST	$x_h^j = [07, 06]$; $r = [06]$, $h \geq 1$
C_3	RST+SLR	$r = [06]$; $x_h^j = [07, 06]$, $h \geq i$, $a = [07]$
C_4	FBC+INF	$a = [07]$; $x_h^j = [07, 06]$, $h \geq i$
C_5	CLR+INF	$a = [07]$, $cl_i = [07, --]$
C_6	CLR+CRR	$cl_i = [07, --]$; $cr_i = [07, --]$
C_7	CLR+DTP	$cl_i = [07, --]$; $x_h^j = [07, 06]$, $h \geq 1$
C_8	CRR+INF	$cr_i = [07, --]$; $x_t^j = [07, 06]$, $t > i$
C_9	CRR+DTP	$cr_i = [07, --]$; $x_h^j = [07, 06]$, $h \geq 1$

TABLE 7.2-2 Forks and their main functions.

FORK SYMBOL	SUPERIOR PATH	INFERIOR PATH
F_1	RST	RST
F_2	RST	RST
F_3	INF	SLR
F_4	SLR	FBC
F_5	DTP	PG_1
F_6	FBC	PG_3
F_7	SLR	RST
F_8	SLR	CCL
F_9	FBC	PG_4
F_{1o}	FBC	FBC
F_{11}	FBC	SCL
F_{12}	SLR	PG_5
F_{13}	CCL	PG_6
F_{14}	CLR	PG_9
F_{15}	CRR	CRR

7.3. Gates and locks

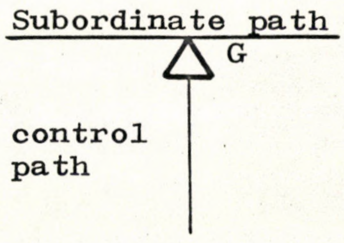
Structurally, a gate is characterised by its two related paths; the control path and the subordinate path. Functionally, on the other hand, the effected signals are relevant. In terms of the relational data processing technique (Codd, 1970, 1971) we define the set of all the gates (of a MAXEL module) as a relation of four attributes:

GATE: (GATESYMBOL, CONTROLPATH, SUBPATH, SIGNALS)

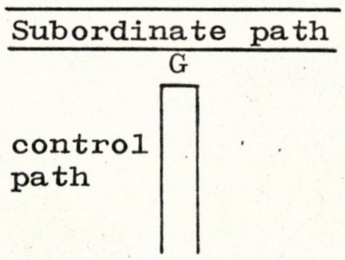
Locks are permanent gates in the on state without control paths. Their function is to direct paths and they are structurally determined by the paths they direct. Thus, relationally, we define the set of all the locks (including lockpairs, where signal transformation replaces signal annihilation) as a relation of three attributes:

LOCK: (LOCKSYMBOL, DIRECTED PATH, AFFECTED SIGNALS)

Gates are listed in table 7.3 – 1 and in 7.3 –2, while locks are in 7.3 – 3 (Pathnames can be found in table 8.1 – 2 and signalnames in table 6.2 – 1)
Symbols (in drawings) for gates are as usual:



drawing



staking

TABLE 7.3-1 Gates /for pathterms see table 8.1-2/

GATE SYMBOL	CONTROL PATH	SUB-ORDINATE PATH	Affected signal/s/ in U_i^j
G_1	DTP	RST	$r = [06]$
G_2	SCL	SLR	$x_t^j = [07, 06], \quad t > i$
G_3	FBC	RST	$a = [07]$
G_4	FBC	FBC	$x_t^j = [07, 06], \quad t > i$
G_5	SLR	DTP	$x_i^j = [07, 06]$
G_6	CCL	CCR	$a = [07]$
G_7	FBC	INF	$x_i^j = [07, 06]$
G_8	CCL	CLR	$x_i^j = [07, 06]$
G_9	CLR	CLR	$cl_i = [07, --]$

TABLE 7.3-2 Path dependencies by gates

Subord / Control	CCR	CLR	DTP	FBC	INF	RST	SLR
CCL	G_6	G_8	.	.	.	.	.
CLR	.	G_9	.	.	.	.	.
DTP	.	.	.	.	.	G_1	.
FBC	.	.	.	G_4	G_7	G_3	.
SCL	.	.	.	.	.	.	G_2
SLR	.	.	G_5	.	.	.	.

TABLE 7.3-3 Locks /for pathterms see table 8.1-2/

LOCK SYMBOL	DIRECTED PATH	AFFECTED SIGNAL/S/ in U_i^j	
L_1	SLR	$r = [06]$	
$L_2 L_3$	FBC	$x_h^j = [07,\ 06]$	$h \geqq i$
L_4	RST	$x_i^j = [07,\ 06]$	
L_5	RST	$x_i^j = [07,\ 06]$	
L_6	FBC	$r = [06]$	
L_7	SLR	$cr_i = [07,\ --]$	
L_8	CRR	$x_i^j = [07,\ 06]$	
$L_9 L_{1o}$	FBC	$r = [06],\ x_i^j = [07,\ 06]$	

Chapter 8

PATH AND EVENTS

8.1 Pathnames, functions and graphs

Structurally, a cellular automaton is usually a system of related paths. Functionally, only the events taking place along the paths are characteristic. A path is structurally determined by the elementary structures contained by it. This is actually the "meaning" (or content) of a path (name). As for the pathfunctions, they are too numerous to be described other than verbally. Thus, relationally, the set of all the paths (of a MAXEL module) is defined as a four-attribute relation:

PATH: (PATHTERM, PATHNAME, MEANING, FUNCTION)

Table 8.1 – 2 containes the paths. A structural dependency occurs between the paths by the crossovers (table 8.1 – 1). Graphical characterization of the (more complex) pathsystems can be found in figures 8.1 – 1,2,3.

TABLE 8.1 – 1 Pat dependencies by crossovers

	CLR	CRR	DTP	FBC	INF	RST	SLR
CLR	.	c_6	c_7	.	c_5	.	.
CRR	c_6	.	c_9	.	c_8	.	.
DTP	c_7	c_9	.	.	.	c_2	.
FBC	.	.	.	.	c_4	.	.
INF	c_5	c_8	.	c_4	.	c_1	.
RST	.	.	c_2	.	c_1	.	c_3
SLR	.	.	.	.	.	c_3	.

TABLE 8.1-2 Paths

PATH TERM	PATHNAME	MEANING	FUNCTION
ACT	Activation path	See figure 8.1-1	See Chapter 9.
CCL	Collector control path	F_7-F_8-F_{13}-G_8	Enabling signal x to reach G_8
CLR	Collector path	E_4-C_7-C_6-F_{14}-C_5-S_4	Collecting the effects of the zero indicating gate G_9
CRR	Corrector path	E_5-C_8-F_{15}-C_6-F_{12}-C_9-S_5	Providing signal for gate G_5 in case all the i-th bits of the n^j-s /j-1, ...,1/ are zero. In this case G_5 will be turned off. /Allnought case/
DTP	Data path	E_2-C_2-F_5-C_7-C_9-S_2	Along this path numbers n^1, n^2,..., n^j,..., n^ℓ propagate in adequate signal representation.
FBC	First bit chopper path	F_4-F_{11}-F_{10}-C_4-L_9L_{10}-G_7	Operating gate G_7 to turn it off permanently, once the first bit has been annihilated.
INF	Information path	E_1-C_1-F_3-C_4-C_5-C_8-S_1	Along this path bit series n_i^1, n_i^2,..., n_i^j,..., n_i^ℓ propagate representing the information about the data n^j.

TABLE 8.1-2 /cond't/ Paths

PATH TERM	PATHNAME	MEANING	FUNCTION
OPR	Operation path	See figure 8.1-2	See Chapter 10.1
PG_i	Path of G_i	$F/G_i/\rightarrow G_i$	See table 8.2-1
RST	Reset path	See figure 8.1-3	See Chapter 10.2
SCL	Selector control path	$F_{11}\rightarrow G_2$	To control selector gate G_2
SLR	Selector path	$F_3-C_3-F_4-X_4/L_1/-X_2/G_2/-$ $-F_7-F_8-X_4/L_7/-$ $-F_{12}\rightarrow G_5$	To select the first bit out of the chopped string n^j to provide signal for the Data Control Gate G_5

$$\begin{tikzcd}
 & & E_1 \arrow[d, "a"'] & & & & & \\
 & & C_1 \arrow[d, "a"'] & & & & & \\
 & & F_3 \arrow[dl, "a'"'] \arrow[d, "a"'] & & & & & \\
 & C_4 \arrow[d, "a'"'] & C_3 \arrow[d, "a"'] & & & & & \\
 & X_3(\overline{G_7}) & F_4 \arrow[d, "a"'] \arrow[drr, "a''"] & & & & & \\
 & & L_2 L_3 \arrow[d, "a"'] & & X_4(L_1) \arrow[d, "a''"'] & & & \\
 & & F_6 \arrow[dl, "a'''"'] \arrow[d, "a"'] & & X_4(G_2) \arrow[d, "a''"'] & & & \\
 & G_3 & X_4(L_6) \arrow[d, "a"'] & & F_7 \arrow[d, "a''"'] \arrow[drrr, "a^{IV}"] & & & \\
 & & F_{11} \arrow[d, "a"'] \arrow[dr, "a^{V}"] & & F_8 \arrow[d, "a''"'] \arrow[drr, "a^{VII}"] & & & X_4(L_4) \arrow[d, "a^{IV}"'] \\
 & & F_{10} \arrow[dl, "a^{VI}"'] \arrow[d, "a"'] & G_2 & F_{13} \arrow[d, "a''"'] \arrow[dr, "a^{X}"] & & X_4(L_7) \arrow[d, "a^{VII}"'] & X_4(\overline{G_1}) \\
 & F_9 \arrow[dl, "a^{VIII}"'] \arrow[d, "a^{VI}"'] & C_4 \arrow[d, "a"'] & & G_8 & G_6 & F_{12} \arrow[d, "a^{VII}"'] \arrow[dr, "a^{IX}"] & \\
G_4 & X_4(L_5) \arrow[d, "a^{VI}"'] & L_9 L_{10} \arrow[d, "a"'] & & & & X_2(L_8) \arrow[d, "a^{VII}"'] & G_5 \\
 & X_4(\overline{G_3}) & G_7 & & & & X_2(\overline{G_6}) &
\end{tikzcd}$$

Figure 8.1-1
Activation Path graph

$$\begin{tikzcd}
E_1 \arrow[d, "\substack{y\\ x}"'] \\
C_1 \arrow[d, "\substack{y\\ x}"'] \\
F_3 \arrow[d, "\substack{y\\ x}"'] \arrow[dr, "y'x'"] \\
C_4 \arrow[d, "\substack{y\\ x}"'] & C_3 \arrow[d, "\substack{y'\\ x'}"'] \\
X_3(G_7) \arrow[d, "y"'] & F_4 \arrow[d, "\substack{y'\\ x'}"'] \arrow[drrrrrr, "y''x''"] \\
C_5 \arrow[d, "y"'] & X_4(L_4) \arrow[d, "\substack{y'\\ x'}"'] & & & & & & L_2 L_3 \arrow[d, "\substack{y''\\ x''}"'] \\
C_8 \arrow[d, "y"'] & X_4(\check{G}_2) \arrow[d, "x'"'] & & & & & & F_6 \arrow[d, "\substack{y''\\ x''}"'] \arrow[dl, "x'''y'''"'] \\
S_1 & F_7 \arrow[d, "x'"'] \arrow[drrr, "x^{IV}"] & & & & & \check{G}_3 & X_4(L_6) \arrow[d, "\substack{y''\\ x''}"'] \\
& F_8 \arrow[d, "x'"'] \arrow[drr, "x^{VI}"] & & & X_4(L_4) & & & X_4(\check{G}_4) \arrow[d, "x''"'] \\
& F_{13} \arrow[d, "x'"'] \arrow[dr, "X^{X}"] & & X_4(L_7) \arrow[d, "x^{VI}"'] & & & & F_{11} \arrow[d, "x''"'] \arrow[dr, "X^{V}"] \\
& \check{G}_8 & \check{G}_6 & F_{12} \arrow[d, "x^{VI}"'] \arrow[dr, "X^{IX}"] & & & & F_{10} \arrow[d, "x''"'] \arrow[dl, "x^{VII}"'] & \check{G}_2 \\
& & & X_2(L_8) & \check{G}_5 & & F_9 \arrow[d, "x^{VII}"'] \arrow[dl, "x^{VIII}"'] & C_4 \arrow[d, "x''"'] \\
& & & & & \check{G}_4 & X_4(L_5) & L_9 L_{10} \arrow[d, "x''"'] \\
& & & & & & & \bar{G}_7
\end{tikzcd}$$

Figure 8.1-2
Operation Path graph

$$\begin{tikzcd}
E_2 \arrow[d, "r"'] & & & & & & & & \\
C_1 \arrow[d, "r"'] & & & & & & & & \\
F_1 \arrow[d, "r"'] & & & & & & & & \\
C_3 \arrow[d, "r"'] \arrow[rrrrrrd, "r'"] & & & & & & & & \\
X_4(G_3) \arrow[d, "r"'] & & & & & & F_2 \arrow[ld, "r''"', "r'"] \arrow[d, "r'"'] & & \\
X_4(L_5) \arrow[d, "r"'] & & & & & X_4(L_4) \arrow[d, dash, "r''"'] & C_2 \arrow[d, "r'"'] & & \\
F_9 \arrow[d, "r"'] \arrow[rrrd, "r'''"] & & & & & F_7 \arrow[ld, "r^{VI}"'] \arrow[d, "r''"'] & S_3 & & \\
F_{10} \arrow[d, "r"'] \arrow[rrd, "r^{IV}"] & & & G_4 & X_2(G_2) \arrow[d, "r^{VI}"'] & F_8 \arrow[d, "r''"'] \arrow[rrd, "r^{VII}"] & & & \\
C_4 \arrow[d, "r"'] & & F_{11} \arrow[ld, "r^{V}"'] \arrow[d, "r^{IV}"'] & & X_4(\overline{L_1}) & F_{13} \arrow[d] \arrow[rd, "r^{IX}"] & & X_4(\overline{L_7}) \arrow[d, "r^{VII}"'] & \\
L_9\,L_{10} \arrow[d, "r \overset{1}{=} {[07]}"'] & \overline{G_2} & X_4(L_6) & & & \widetilde{G_8} & \widetilde{G_6} & F_{12} \arrow[d] \arrow[rd, "r^{VIII}"] & \\
G_7 & & & & & & & X_2(\overline{L_8}) & \widetilde{G_5}
\end{tikzcd}$$

Figure 8.1-3
Reset Path graph

8.2 Events and event types

Events are given in two forms depending on the context in which they are used. First is the quadruple form:

$$e = (t, C, h, s) \equiv (e_t)$$

where

t stands for the time when the event e occurs

C stands for the name of the cell where the event e takes place. For the possible elementary cellnames see table 8.2 – 1

h is an element of the set **H** of all the possible elementary event types (”happenings”)

$h \epsilon (x, T, p, e, s, L_j, L_i L_k)$
where table 8.2 – 2 gives the meaning of these elementary event type symbols.

s is a signalname (for details see 6.2)

Starting from the elementary cellnames one can construct combined cellnames which are mainly gatenames, see table 8.2 – 3.

If one wants to stress that a lock L_i is closed (as it always is) then one writes $\overline{L}_i$.

Of course, state symbols for gates (ν, –, ~) can be combined with the neighbour symbols X_2, X_3, X_4, X_5. For instance, $X_2(\overline{\overline{\overline{G}}}_7)$ means the right (eastern) neighbour of gatecell G_7 which has been previously on but is turned on again (by a signal 07) so it is, by the event in question, turned off. See table 8.2 – 3.

8.2

TABLE 8.2-1 ELEMENTARY CELLNAMES

SYMBOL	MEANING
C_i	Crossover centre
E_i	Entry cell of a unit
F_i	Forkcell
$F/G_i/$	The nearest forkcell to gate G_i
G_i	Gatecell
L_i	Lockcell
L_iL_k	The cell between locks L_i and L_k
S_i	Sortiecell /output/ of a unit
$X_1/C/$	Cell C itself
$X_2/C/$	Right /eastern/ neighbourof cell C.
$X_3/C/$	Bottom /southern/ neighbourof cell C.
$X_4/C/$	Left /western/ neighbourof cell C.
$X_5/C/$	Upper /northern/ neighbourof cell C.

TABLE 8.2-2 ELEMENTARY EVENT TYPES

SYMBOL	MEANING
✱	Signal birth /duplication/
+	Signal death /annihilation/
c	Signal collision
e	Entering the unit in question
L_j	Activating the lock L_j.
L_iL_k	Activating the lockpair L_iL_k
p	Passing a cell
s	Signal s is born
t	Signal transformation

TABLE 8.2-3 Combined gatenames

GATE SYMBOL	MEANING
$\overset{v}{G}_i$:	Gate G_i in state off
$\bar{G}_i$:	Gate G_i in state on
$\bar{G}_i = \overset{\bar{v}}{G}_i$:	Gate G_i which has been turned off is on
$G_i = \bar{\bar{G}}_i$:	Gate G_i which has been turned on is turned on again so it is in off state
$\tilde{G}_i$:	Gate G_i is in an arbitrary state on or off /"Gate in changed state"/
$\overset{v}{\tilde{G}}_i$:	Gate G_i previously in one of the states on or off but it is now certainly in off state: $\overset{v}{\tilde{G}}_i = \overset{v}{G}_i$

Some obvious relations:

$$\bar{\bar{G}}_i = \overset{v}{G}_i ; \quad \overset{v}{\overset{v}{G}}_i = \overset{v}{G} ; \quad \overset{v}{\bar{G}}_i = \overset{v}{G}_i ; \quad \overset{\bar{v}}{G}_i = \bar{G}_i$$

$$\overset{v}{\tilde{G}}_i = \overset{v}{G} \; ; \quad \overset{\sim}{\overset{v}{G}}_i = \tilde{G}_i ; \quad \overset{\approx}{G}_i = \tilde{G}_i ; \quad \overset{\bar{\sim}}{G}_i = \tilde{G}_i$$

If we introduce $\varepsilon, \mu \epsilon \{ -, \nu, \sim \}$ and define algebraic operation "0" between the symbols $-, \nu, \sim$ by

$$\overset{\overset{\mu}{\varepsilon}}{G_i} = \overset{\varepsilon o \mu}{G_i}$$

then we get the following operation table (table 8.2 – 4)

It is very important that

$$\overset{\overset{-}{\sim}}{G_i} = \overset{\sim}{G_i} \quad \text{whereas} \quad \overset{\overset{\nu}{\sim}}{G_i} = \overset{\nu}{G_i}$$

This is because we have a "gate opener signal" (06) but do not have a "gate closer signal", since signal (07) is only a "gate changer signal".

The second form is the bracketed from of the events:

$$e = (\text{time, cellname})^{\text{event type}}$$

For instance $(31, X_2(G_4))^p$ means that a signal (it does not matter which) passes at $t = 31$ an open gate G_4, placed on the left side of propagation.

This way of coding events is useful in the ~~event~~ consecutions ("event fates") when everything is described about a certain signal from birth to death or from entry to sortie. etc. As in the case of the cellnames elementary events can also be combined. The combined event types playing essential roles in our design are shown in table 8.2 – 5.

TABLE 8.2 – 4 The algebra of gatestate symbols

$\varepsilon o \mu$	$-$	ν	$\sim$	
$-$	ν	ν	$\sim$	
ν	$-$	ν	$\sim$	
$\sim$	$\sim$	ν	$\sim$	

TABLE 8.2-5 Combined event types

EVENT TYPE TERM	MEANING	EXAMPLE	EXPLANATION
ALL	Activating a left lock	$/44, L_7, L_7, a^{vii}/$ or $/44, L_7/^{L_7}$	a signal $/a^{vii}/$ is activating lock L_7 at t = 44/and the lock is placed on the leftside of signal propagation/
ALP	Activating a lockpair	$/55, L_9L_{1o}, L_9L_{1o}, a/$ or $/55, L_9L_{1o}/^{L_9L_{1o}}$	a signal /a/ is activating lockpair L_9L_{1o} at /time/ t = 55
ARL	Activating a right lock	$/46, L_5, L_5, a^{vi}/$ or $/46, L_5/^{L_5}$	a signal $/a^{vi}/$ is activating lock L_5 at t = 46/and the lock is placed on the leftside of signal propagation/
BIF	Birth in a fork	$/16, F_4, *, x''/$ or $/16, F_4/^{*}$	a signal /x"/ was born in /fork/ cell F_4 at /time/ t = 16

TABLE 8.2-5 /cont'd/ Combined event types

EVENT TYPE TERM	MEANING	EXAMPLE	EXPLANATION
DAG	Death by altering a gate state	$/51,\ G_5\ +,\ x^{ix}/$ or $/51,\ G_5/^{+}$	a signal $/x^{ix}/$ changes the state of gate G_5 /depending on the signal content and the gate state
DBC	Death by collision	$/39,\ C_3,\ +c,\ x/$ or $/39,\ C_3/^{+c}$	a signal /x/ collides /with an other/ in /the center of/ crossover C_3 at /time/ $t = 39$ and dies
DCG	Death by closing a gate	$x_{54}=/54,\ \bar{\bar{G}}_2,\ +,\ x^{vi}/$ or $/54,\ \bar{\bar{G}}_2/^{+}$	a signal $/x^{vi}/$ arrives at the gatecell $\bar{\bar{G}}_2$ at $t = 54$, turns it on and is annihilated
DOG	Death by opening a gate	$/58,\ G_7,\ +,\ x''/$ or $/58,\ G_7/^{+}$	a signal $/x''/$ turns gate G_7 off and dies at $t = 58$

TABLE 8.2-5 /cont'd/ Combined event types

EVENT TYPE TERM	MEANING	EXAMPLE	EXPLANATION
DUG	Death under a gate e.g.:	$/57,\ X_2/G_4/,\ +,\ y''/$ or $/57,\ X_2/G_4//^+$	a signal $/y''/$ is annihilated at $t = 57$ when passing the closed gate G_4 placed on its left hand side; /it reached the right neighbour of G_4 and died at the same time /
DUL	Death under a lock	$/40,\ X_4/L_6/,\ +,\ r^{iv}/$ or $/40,\ X_4/L_6//^+$	a signal $/r^{iv}/$ is annihilated at $t = 40$ by the /already activated/ lock L_6 which is placed on the right hand side of the signal's propagation
ENT	Entering a unit	$/0,\ E_1/^e$ or $/0,\ E_1,\ e,\ a/$	at $t = 0$ in cell E_1, a signal /a/ enters a unit

TABLE 8.2-5 /cont'd/ Combined even types

EVENT TYPE TERM	MEANING	EXAMPLE	EXPLANATION
LEA	Leaving a unit	/73, S_1, s, a'/ or /0, $S_1/^{s}$	at t = 73 in cell S_1 a signal /a'/ leaves a unit
PAC	Passing a crossover	/36, C_4, p, x/ or /36, $C_4/^{p}$	a signal /x/ is passing the crossover C_4 at t = 36
PLG	Passing a left gate	/29, $X_4/G_1/$, p, r"/ or /29, $X_4/G_1//^{p}$	a signal /r"/ is passing the gate G_1 placed on the left side of propagation
PLL	Passing a left lock	/54, $X_4/L_7/$, p, r^{vii}/ or /54, $X_4/L_7//^{p}$	a signal /r"/ is passing the lock L_7 placed on the left side of propagation
PRG	Passing a right gate	/75, $X_2/G_7/$, p, y/ or /75, $X_2/G_7//^{p}$	a signal /y/ is passing the /open/ gate G_7 placed on the right side of propagation at t = 75

TABLE 8.2-5 /cont'd/ Combined event types

EVENT TYPE TERM	MEANING	EXAMPLE	EXPLANATION
PRL	"Passing a right /activated/ lock"		is this is impossible; if it occurs in an event table then it is an error
RIF	Replication in a fork	/16, F_4, x", x'/ or /16, $F_4/^{x''}$	a signal /x'/ is given birth to signal x" at t = 16 in forkcell F_4
SCC	Signal collision in a corner	/21, C, c, x'/ or /21, $C/^{c}$	a signal /x'/ collides /with an other/ at a /corner/ cell C at /time/ t = 21
SCP	Signal collision on a path	/35, C, c, y/ or /35, $C/^{c}$	a signal /y/ collides /with an other/ in a path in a cell C at /time/ t = 35

TABLE 8.2-5 /cont'd/ Combined event types

EVENT TYPE TERM	MEANING	EXAMPLE	EXPLANATION
TBC	Transformation by collision	/41, C, tc, s/ or $/41, C/^{tc}$	a signal /s/ at time t = 41, collides /with an other with the same content/ and is transformed according to $\boxed{04} \times \boxed{04} = \boxed{05}$; $\boxed{05} \times \boxed{05} = \boxed{06}$; $\boxed{06} \times \boxed{06} = \boxed{06}$; $\boxed{07} \times \boxed{07} = \boxed{04}$
TBL	Transformation between locks	/21, L_2L_3, t, x"/ or $/21, L_2L_3/^{t}$	a signal /x"/ is transformed /into $\boxed{07}$/ between locks L_2 and L_3.

Needless to say, a great number of other event types can occur within a cellular automaton, (eg. "transformation between gates" or "collision with an echo signal" etc.). Their relations to the above and to each other is to be elaborated by later studies.

As for the exact formulation of the event types, it can be done almost automatically. E.g.: passing a right gate (PRG) means that at the previous shot the signal is in the position of figure 8.2 – 1. It is easy to see that PRG occurs at t if at time t – 1 the signal is in the cell $X_4/G_i/$

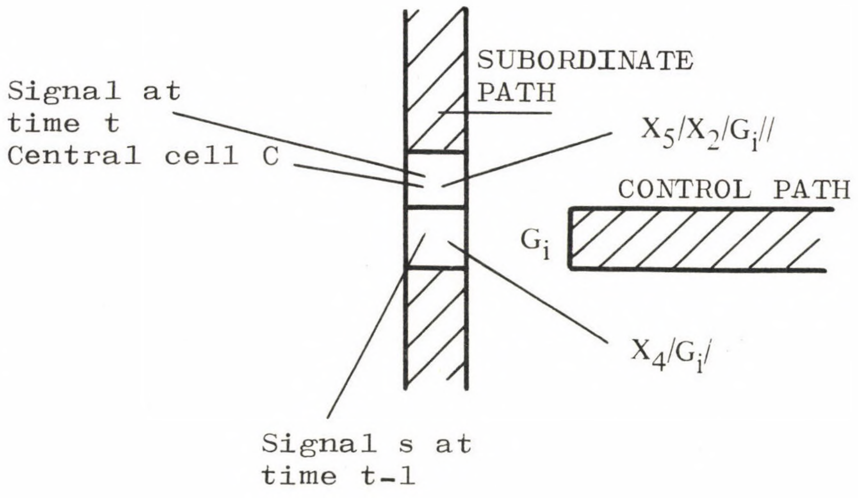


Figure 8.2-1

Approaching a right side gate southern neighbour of the cell under the gate, in the case of the upright position of figure 8.2 – 1. The three other positios can be obtained by rotation. See table 8.2 – 6.

TABLE 8.2-6 Cases for approaching a right side gate

Direction of the subordinate path	Which neighbour is G_i of the central cell C	Signal is in time t-1 is in cell
UPRIGHT /fig. 8.2-1/ (k=1)	EASTERN: $C=X_4/G_i/$ $G_i=X_2/C/.$	$X_3/C/=$ $X_3/X_4/G_i//$
LEFT TO RIGHT (k=2)	SOUTHERN: $C=X_5/G_i/$ $G_i=X_3/C/$	$X_4/C/=$ $X_4/X_5/G_i//$
DOWNWARD (k=3)	WESTERN: $C=/X_2/G_i/$ $G_i=X_4/C/$	$X_5/C/=$ $X_5/X_2/G_i//$
RIGHT TO LEFT (k=4)	NORTHERN: $C=X_3/G_i/$ $G_i=X_5/C/$	$X_2/C/=$ $X_2/X_3/G_i//$

So, generally, the definition of the PRG at t is the event

$(t-1, X_{k\oplus 2}/X_{k\oplus 3}/G_i//, p, s).$

where the addition "$\oplus$" is meant in the cyclic fashion i.e.:

$4 \oplus 2 = 3, \quad 4 \oplus 3 = 2.$

The use of the event types can be experienced along with the discussion of activation, operation and resetting in Chapters 9 and 10.

There remain some additional concepts and notations to be introduced.

The time of an event $e = (t, C, h, s)$ is denoted by

$t(t, C, h, s) = t(e)$

for example:

$t(46, C_4, p, a) = 46.$

If

$$t(e_1) = t(e_2)$$

then e_1 and e_2 are called simultaneous events. Similarly, by definition, the place of an event e is:

$$P(e) = P(t, C, h, s,) \equiv C.$$

If

$$P(e_1) = P(e_2)$$

then e_1 and e_2 are called collocal events.

Events which are both simultaneous and collocal are called coincident events (or coincidences).

Chapter 9

ACTIVATION

9.1 Activation signal

The aim of activation is to set all the locks $L_1, \ldots, L_{10}$. See figures 8.1 – 1 and 9.1 – 1. Beginning at cell E_1 with a signal $a = [07]$, after a number of replications all the locks will be set and all the gates (except G_1 and G_9 which have been handled by outside sources) will be in the on state. By means of two signals $p_1 = [06], p_2 = [06]$ (presetting) applied at E_3 all the important gates will be set i.e.:

G_1	immaterial
G_2	in of state (open)
G_3	immaterial
G_4	in off state (open)
G_5	in off state (open)
G_6	in off state (open, although immaterial)
G_7	in off state (closed)
G_8	in off state (open)
G_9	by outside source through CLR.

The exact history of the signals $a', \ldots, a^x$ can be followed on table 9.1 – 1. As for the single signal fates these are as follows:

9.1

a: /0, $E_1/^{e}$; /3, $C_1/^{p}$;/7, $F_3/^{a}$; /12, $C_3/^{p}$; /16, $F_4/^{a''}$;

/22, $L_2L_3/^{L_2L_3}$; /24, $F_6/^{a'''}$/29, $L_6/^{L_6}$; /31, $X_2/G_2//^{p}$

/34, $F_{11}/^{a^{v}}$; /38, $F_{1o}/^{a^{vi}}$; /46, $C_4/^{p}$;

/55, $L_9L_{1o}/^{L_9L_{1o}}$; /58, $G_7/^{+}$.

a': /7, $F_3/^{*}$; /36, $C_4/^{p}$; /49, $X_2/G_7//^{p}$; /66, $C_5/^{p}$;

/71, $C_8/^{p}$; /73, $S_1/^{s}$

a'': /16, $F_4/^{*}$; /28, $L_1/^{L_1}$;/30, $X_2/G_2//^{p}$; /32, $F_7/^{a^{iv}}$;

/36, $F_8/^{a^{vii}}$; /49, $F_{13}/^{a^{x}}$; /58, $G_8/^{+}$.

a''': /24, $F_6/^{*}$; /27, $G_3/^{+}$.

a^{iv}: /32, $F_7/^{*}$; /42, $L_4/^{L_4}$; /45, $X_4/G_1//^{+}$.

a^{v}: /34, $F_{11}/^{*}$; /53, $G_2/^{+}$.

a^{vi}: /38, $F_{1o}/^{*}$; /41, $F_9/^{a^{viii}}$; /46, $L_5/^{L_5}$;

/48, $X_4/G_3//^{+}$.

a^{vii}: /36, $F_8/^{*}$; /44, $L_7/^{L_7}$; /47, $F_{12}/^{a^{ix}}$; /51, $L_8/^{L_8}$;

/54, $X_2/G_6//^{+}$.

a^{viii}: /41, $F_9/^{*}$; /44, $G_4/^{+}$.

a^{ix}: /47, F_{12}/*; /51, G_5/$^{+}$.

a^{x}: /49, F_{13}/*; /53, G_6/$^{+}$.

TABLE 9.1-1 Activation by signal /1 of 3/

time	place	happening	SIGNAL	event type
0	E_1	e	a	ENT
3	C_1	p	a	PAC
7	F_3	a'	a	RIF
12	C_3	p	a	PAC
16	F_4	a''	a	RIF
22	L_2L_3	L_2L_3	a	ALP
24	F_6	a'''	a	RIF
29	L_6	L_6	a	ALL
31	$X_2/G_4/$	p	a	PRG
34	F_{11}	a^{v}	a	RIF
38	F_{1o}	a^{iv}	a	RIF
46	C_4	p	a	PAC
55	$\bar{L}_9\bar{L}_{1o}$	L_9L_{1o}	a	ALP
58	$\bar{G}_7$	+	a	DCG
7	F_3	*	a'	BIF
36	C_4	p	a'	PAC
49	$X_2/\bar{G}_7/$	p	a'	PRG
66	C_5	p	a'	PAC
71	C_8	p	a'	PAC

TABLE 9.1-1 /cont'd/ Activation by signal /2 of 3/

time	place	happening	SIGNAL	event type
73	S_1	s	a'	LEA
16	F_4	*	a''	BIF
28	L_1	L_1	a''	ALL
30	$X_2/G_2/$	p	a''	PRG
32	F_7	a^{iv}	a''	RIF
36	F_8	a^{vii}	a''	RIF
49	F_{13}	a^{x}	a''	RIF
58	G_8	+	a''	DCG
24	F_6	*	a'''	BIF
27	G_3	+	a'''	DCG
32	F_7	*	a^{iv}	BIF
42	L_4	L_4	a^{iv}	ARL
45	$X_4/G_1/$	+	a^{iv}	DUG
34	F_{11}	*	a^{v}	BIF
53	G_2	+	a^{v}	DCG
38	F_{1o}	*	a^{vi}	BIF
41	F_9	a^{viii}	a^{vi}	RIF
46	L_5	L_5	a^{vi}	ARL
48	$X_4/G_3/$	+	a^{vi}	DUG
36	F_8	*	a^{vii}	BIF
44	L_7	L_7	a^{vii}	ALL
47	F_{12}	a^{ix}	a^{vii}	RIF

TABLE 9.1-1 /cont'd/ Activation by signal /3 of 3/

time	place	happening	SIGNAL	event type
51	L_8	L_8	a^{vii}	ARL
54	$X_2/G_6/$	+	a^{vii}	DUG
41	F_9	✱	a^{viii}	BIF
44	G_4	+	a^{viii}	DCG
47	F_{12}	✱	a^{ix}	BIF
51	G_5	+	a^{ix}	DCG
49	F_{13}	✱	a^{x}	BIF
53	G_6	+	a^{x}	DCG

Using the signal table (Table 9.1 – 1) one can check the following statements. It should be noted that all the statements can be proved mechanically using the relational data processing technique invented by Codd (1970)'

STATEMENT 9.1 – 1

All signals begin with begin with one of the event types ENT or BIF, and are terminated by one of the event types

LEA, DCG, DUG

It follows that

STATEMENT 9.1 – 2

a.) All the gates are closed
(for there is no event of type DOG or DAG)

b.) No signal remains within the MAXEL module.

Then, again, it is easy to see that,

STATEMENT 9.1 –3

After $t = 73$ all the locks are set.

However, the last two statements are more directly obtained from the PLACE-table . (Table 9.3 – 1)

9.2 Activation by time

Rearranging the signal table (Table 9.1 – 1) one gets the activation timetable, table 9.2 - 1. One can mechanically check the following

STATEMENT 9.2 – 1

If two or more events occur at the same time (i.e. they are simultaneous events) then either two of them take place at the same place (i.e. collocal events) in form of a RIF-BIF event type pairs or the events are mutually distant (i.e. they do not disturb each other)

TABLE 9.2-1 Activation by time /1 of 3/

TIME	place	happening	signal	event type
0	E_1	e	a	ENT
3	C_1	p	a	PAC
7	F_3	a'	a	RIF
7	F_3	*	a'	BIF
12	C_3	p	a	PAC
16	F_4	a''	a	RIF
16	F_4	*	a''	BIF
22	$\bar{L}_2\bar{L}_3$	L_2L_3	a	ALP

TABLE 9.2-1 /cont'd/ Activation by time /2 of 3/

TIME	place	happening	signal	event type
24	F_6	a'''	a	RIF
24	F_6	$\ast$	a'''	BIF
27	$\bar{G}_3$	+	a'''	DCG
28	$\bar{L}_1$	L_1	a''	ALL
29	$\bar{L}_6$	L_6	a	ALL
30	$X_2/G_2/$	p	a''	PRG
31	$X_2/G_4/$	p	a	PRG
32	F_7	a^{iv}	a''	RIF
32	F_7	$\ast$	a^{iv}	BIF
34	F_{11}	a^{v}	a	RIF
34	F_{11}	$\ast$	a^{v}	BIF
36	C_4	p	a'	PAC
36	F_8	a^{vii}	a''	RIF
36	F_8	$\ast$	a^{vii}	BIF
38	F_{1o}	a^{vi}	a	RIF
38	F_{1o}	$\ast$	a^{vi}	BIF
41	F_9	a^{viii}	a^{vi}	RIF
41	F_9	$\ast$	a^{viii}	BIF
42	$\bar{L}_4$	L_4	a^{iv}	ARL
44	$\bar{L}_7$	L_7	a^{vii}	ALL
44	$\bar{G}_4$	+	a^{viii}	DCG
45	$X_4/\bar{G}_1/$	+	a^{iv}	DUG
46	C_4	p	a	PAC

TABLE 9.2-1 /cont'd/ Activation by time /3 of 3 /

TIME	place	happening	signal	event type
46	$\bar{L}_5$	L_5	a^{vi}	ARL
47	F_{12}	a^{ix}	a^{vi}	RIF
47	F_{12}	$\ast$	a^{ix}	BIF
48	$X_4/\bar{G}_3/$	+	a^{vi}	DUG
49	$X_2/G_7/$	p	a'	PRG
49	F_{13}	a^{x}	a''	RIF
49	F_{13}	$\ast$	a^{x}	BIF
51	$\bar{L}_8$	L_8	a^{vii}	ARL
51	G_5	+	a^{ix}	DCG
53	$\bar{G}_2$	+	a^{v}	DCG
53	$\bar{G}_6$	+	a^{x}	DCG
54	$X_2/\bar{G}_6/$	+	a^{vii}	DUG
55	$\bar{L}_9\bar{L}_{1o}$	L_9L_{1o}	a	ALP
58	$\bar{G}_7$	+	a	DCG
58	$\bar{G}_8$	+	a''	DCG
66	C_5	p	a'	PAC
71	C_8	p	a'	PAC
73	S_1	s	a'	LEA

9.3 Activation by place

A newer rearrangement of the signal table (Table 9.1 – 1 9.2 – 1) produces the place-table, i.e. the list of events ordered by place. By this we get the following (See Table 9.3 –1)

STATEMENT 9.3 – 1

a.) Collocal events are either of birth-replacation type or distant enough (in time)

b.) Simultaneous events are either of brithreplication type or distant enough (in space)

c.) Coincident events and only the coincident events are of brith-replication type.

Example:

$$t(46, C_4, p, a) - t(36, C_4, p, a') = 46 - 36 = 10 > 6$$

(6 is the smallest possible distance between consecutive signals passing a crossover)
Briefly: Both collocal and simulataneous events are legitimate.

Now it is very easy to see by the place-table that

STATEMENT 9.3 – 2

By the end of activation (t = 73)

a.) All the gates are turned on

b.) All the locks are set in.

TABLE 9.3-1 Activation by place /1 of 3/

Time	PLACE	happening	signal	event type
3	C_1	p	a	PAC
12	C_3	p	a	PAC
46	C_4	p	a	PAC

TABLE 9.3-1 /cont'd/ Activation by place /2 of 3 /

Time	PLACE	happening	signal	event type
36	C_4	p	a'	PAC
66	C_5	p	a'	PAC
71	C_8	p	a'	PAC
0	E_1	e	a	ENT
7	F_3	a'	a	RIF
7	F_3	x	a'	BIF
16	F_4	a''	a	RIF
16	F_4	x	a'	BIF
24	F_6	a'''	a	RIF
24	F_6	x	a'''	BIF
32	F_7	a^{iv}	a''	RIF
32	F_7	x	a^{iv}	BIF
36	F_8	a^{vii}	a''	RIF
36	F_8	x	a^{vii}	BIF
41	F_9	a^{vii}	a^{vi}	RIF
41	F_9	x	a^{viii}	BIF
38	F_{1o}	a^{vi}	a	RIF
38	F_{1o}	x	a^{vi}	BIF
34	F_{11}	a^{v}	a	RIF
34	F_{11}	x	a^{v}	BIF
47	F_{12}	a^{ix}	a^{vii}	RIF
47	F_{12}	x	a^{ix}	BIF
49	F_{13}	a^{x}	a''	RIF

TABLE 9.3-1 /cont'd/ Activation by place /3 of 3/

time	PLACE	happening	signal	event type
49	F_{13}	x	a^{x}	BIF
45	$X_4/\bar{G}_1/$	+	a^{iv}	DUG
53	$\bar{G}_2$	+	a^{v}	DCG
30	$X_2/G_2/$	p	$a^{''}$	PRG
27	$\bar{G}_3$	+	$a^{'''}$	DCG
48	$X_4/\bar{G}_3/$	+	a^{iv}	DUG
44	$\bar{G}_4$	+	a^{viii}	DCG
31	$X_2/G_4/$	p	a	PRG
51	$\bar{G}_5$	+	a^{ix}	DCG
53	$\bar{G}_6$	+	a^{x}	DCG
54	$X_2/\bar{G}_6/$	+	a^{vii}	DUG
58	$\bar{G}_7$	+	a	DCG
49	$X_2/G_7/$	p	$a^{'}$	PRG
58	$\bar{G}_8$	+	$a^{''}$	DCG
28	$\bar{L}_1$	L_1	$a^{''}$	ALL
22	$\bar{L}_2\bar{L}_3$	L_2L_3	a	ALP
42	$\bar{L}_4$	L_4	a^{iv}	ARL
46	$\bar{L}_5$	L_5	a^{vi}	ARL
29	$\bar{L}_6$	L_6	a	ALL
44	$\bar{L}_7$	L_7	a^{vii}	ALL
51	$\bar{L}_8$	L_8	a^{vii}	ARL
55	$\bar{L}_9\bar{L}_{1o}$	L_9L_{1o}	a	ALP
73	S_1	s	$a^{'}$	LEA

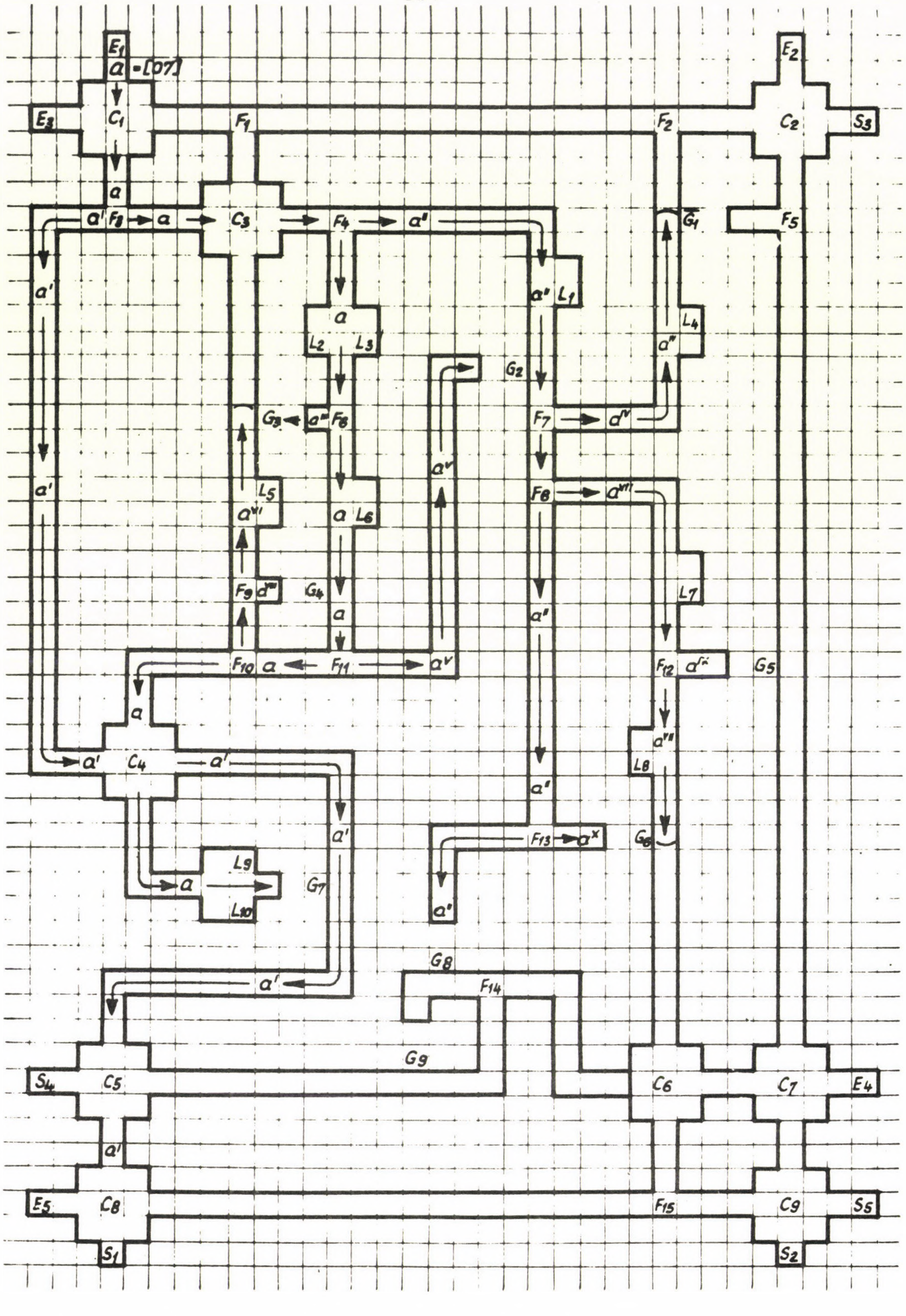


Figure 9.1-1

The activation of the MAXEL module

Chapter 10

OPERATION

10.1 Operation signal by time and place

During the operation the first member of the signal pair x, y where y follows x, with a time lag $T = 26$, is separated. x goes to control gate G_5; y (and, in fact all the rest) leaves the MAXEL module U_i^j to enter its southern neighbour, U_{i+1}^j. From figure 10.1 – 1 one can easily describe the signal fates as follows:

x: $/0,\ E_1/^{e};\ /3,\ C_1/^{p};/7,\ F_3/^{x'};\ /36,\ C_4/^{p};$
$/49,\ X_2/\bar{G}_7//^{+}.$

x': $/7,\ F_3/^{\divideontimes};\ /12,\ C_3/^{p};/16,\ F_4/^{x''};\ /26,\ X_4/L_1//^{p};$
$/30,\ X_2/\overset{v}{G}_2//^{p};\ /32,\ F_7/^{x^{iv}};\ /35,\ F_8/^{x^{vi}};$
$/49,\ F_{13}/^{x^{x}};\ /58,\ G_8/^{+}.$

x'': $/16,\ F_4/^{\divideontimes};\ /21,\ L_2L_3/^{t};\ /24,\ F_6/^{x'''};$
$/28,\ X_4/L_6//^{p};\ /31,\ X_2/\overset{v}{G}_4//^{p};\ /34,\ F_{11}/^{x^{v}};$
$/38,\ F_{10}/^{x^{vii}};\ /46,\ C_4/^{p};\ /55,\ L_9L_{10}/^{t};$
$/58,\ \bar{G}_7/^{+}.$

x''': $/24,\ F_6/^{\divideontimes};\ /27,\ \bar{G}_3/^{+}.$

x^{iv}: $/32,\ F_7/^{\divideontimes};\ /41,\ X_4/\bar{L}_4//^{+}.$

x^{v}: $/34,\ F_{11}/^{\divideontimes};\ /54,\ \bar{G}_2/^{+}.$

x^{vi}: $/35,\ F_8/^{\text{ж}};\ /44,\ X_4/L_7//^{P};\ /47,\ F_{12}/^{x^{ix}};$

$/51,\ X_2/L_8//^{+}.$

x^{vii}: $/38,\ F_{10}/^{\text{ж}};\ /41,\ F_9/^{x^{viii}};\ /45,\ X_4/L_5//^{+}.$

x^{viii}: $/41,\ F_9/^{\text{ж}};\ /44,\ \bar{G}_4/^{+}.$

x^{ix}: $/47,\ F_{12}/^{\text{ж}};\ /51,\ \bar{G}_5/^{+}.$

x^{x}: $/49,\ F_{13}/^{\text{ж}};\ /53,\ G_6/^{+}.$

y: $/26,\ E_1/^{e};\ /29,\ C_1/^{P};\ /33,\ F_3/^{y'};\ /62,\ C_4/^{P};$

$/75,\ X_3/\bar{\bar{G}}_7//^{P};\ /92,\ C_5/^{P};\ /97,\ C_8/;\ /99,\ S_1/^{s}.$

y': $/33,\ F_3/^{\text{ж}};\ /38,\ C_3/^{P};\ /42,\ F_4/^{y''};\ /52,\ X_4/L_1//^{P};$

$/56,\ X_2/\bar{G}_2//^{+}.$

y'': $/42,\ F_4/^{\text{ж}};\ /47,\ L_2L_3/^{t};\ /50,\ F_6/^{y'''};$

$/54,\ X_4/L_6//^{P};\ /57,\ X_2/\bar{G}_4//^{+}.$

y''': $/50,\ F_6/^{\text{ж}};\ /53,\ \bar{\bar{G}}_3/^{+}$

From this we can construct the signal table (Table 10.1 – 1) of operation and check the validity of the following

STATEMENT 10.1 – 1

All the signals begin with one of the event types and are terminated by one the event types

DAG, DCG, DOG, DUG, or LEA.

Signal *y*, and only signal *y* leaves the unit having begun with ENT and been terminated by LEA.

TABLE 10.1-1 Operation by signal /1 of 3/

time	place	happening	SIGNAL	event type
0	E_1	e	x	ENT
3	C_1	p	x	PAC
7	F_3	x'	x	RIF
36	C_4	p	x	PAC
49	$X_2/G_7/$	+	x	DUG
7	F_3	x	x'	BIF
12	C_3	p	x'	PAC
16	F_4	x''	x'	RIF
26	$X_4/L_1/$	p	x'	PLG
30	$X_4/G_2/$	p	x'	PRG
32	F_7	x^{iv}	x'	RIF
35	F_8	x^{iv}	x'	RIF
49	F_{13}	x^{x}	x'	RIF
58	G_8	+	x'	DAG
16	F_4	x	x''	BIF
21	L_2L_3	t	x''	TBL
24	F_6	x'''	x''	RIF
28	$X_4/L_6/$	p	x''	PLL

TABLE 10.1-1 /cont'd/ Operation by signal /2 of 3/

time	place	happening	SIGNAL	event type
31	$X_2/G_4/$	p	x''	PRG
34	F_{11}	x^{v}	x''	RIF
38	F_{1o}	x^{vii}	x''	RIF
46	C_4	p	x''	PAC
55	L_9L_{1o}	t	x''	TBL
58	$\bar{G}_7$	+	x''	DOG
24	F_6	ж	x'''	BIF
27	$\bar{G}_3$	+	x'''	DCG
32	F_7	ж	x^{iv}	BIF
41	$X_4/\bar{L}_4/$	+	x^{iv}	DUL
34	F_{11}	ж	x^{v}	BIF
54	$\bar{G}_2$	+	x^{v}	DCG
35	F_8	ж	x^{vi}	BIF
44	$X_4/L_7/$	p	x^{vi}	PLL
47	F_{12}	x^{ix}	x^{vi}	RIF
51	$X_2/\bar{L}_8/$	+	x^{vi}	DUL
38	F_{1o}	ж	x^{vii}	BIF
41	F_9	x^{viii}	x^{vii}	RIF
45	$X_4/\bar{L}_5/$	+	x^{vii}	DUG
41	F_9	ж	x^{viii}	BIF
44	$\bar{G}_4$	+	x^{viii}	DCG
47	F_{12}	ж	x^{ix}	BIF
51	G_5	+	x^{ix}	DAG

TABLE 10.1-1 /cont'd/ Operation by signal /3 of 3/

time	place	happening	SIGNAL	event type
49	F_{13}	$\ast$	x^x	BIF
53	G_6	+	x^x	DAG
26	E_1	e	y	ENT
29	C_1	p	y	PAC
33	F_3	y'	y	PAC
62	C_4	p	y	PAC
75	$X_2/\bar{\bar{G}}_7/$	p	y	PRG
92	C_5	p	y	PAC
97	C_8	p	y	PAC
99	S_1	s	y	LEA
33	F_3	$\ast$	y'	BIF
38	C_3	p	y'	PAC
42	F_4	y''	y'	RIF
52	$X_4/L_1/$	p	y'	PLL
56	$X_2/\bar{\bar{G}}_2/$	+	y'	DUG
42	F_4	$\ast$	y''	BIF
47	L_2L_3	t	y''	TBL
50	F_6	y''	y''	RIF
54	$X_4/L_6/$	p	y''	PLL
57	$X_2/\bar{\bar{G}}_4/$	+	y''	DUG
50	F_6	$\ast$	y'''	BIF
53	$\bar{\bar{G}}_3$	+	y'''	DOG

A rearreangement of the signal table (Table 12.1 – 1) produces the timetable (Table 10.1 – 2), from which one can easily deduce the following

<u>STATEMENT 10.1 – 2</u>

Simultaneous events are either collocal or distant enough.

<u>STATEMENT 10.1 – 4</u>

Coincident events are always of birth-replacation (BIF-RIF) type pairs.

<u>STATEMENT 10.1 – 5</u>

The duration of the whole operation in the unit U_i^j is

$$t(U_i^j) = 73 + (k - i)26$$

TABLE 10.1-2 Operation by time /1 of 4/

TIME	place	happening	signal	event type
0	E_1	e	x	ENT
3	C_1	p	x	PAC
7	F_3	x'	x	RIF
7	F_3	$\cancel{x}$	x'	BIF
12	C_3	p	x'	PAC
16	F_4	x''	x'	RIF
16	F_4	$\cancel{x}$	x''	BIF
21	L_2L_3	t	x''	TBL
24	F_6	x'''	x''	RIF

TABLE 10.1-2 /cont'd/ Operation by time /2 of 4/

TIME	place	happening	signal	event type
24	F_6	~~x~~	x'''	BIF
26	$X_4/L_1/$	p	x'	PLL
26	E_1	e	y	ENT
27	$\bar{G}_3$	+	x'''	DCG
28	$X_4/L_6/$	p	x''	PLL
29	C_1	p	y	PAC
30	$X_2/\check{G}_2/$	p	x'	PRG
31	$X_2/\check{G}_4/$	p	x''	PRG
32	F_7	x^{iv}	x'	RIF
32	F_7	~~x~~	x^{iv}	BIF
33	F_3	y'	y	RIF
33	F_3	~~x~~	y'	BIF
34	F_{11}	x^{v}	x''	RIF
34	F_{11}	~~x~~	x^{v}	BIF
35	F_8	x^{vi}	x'	RIF
35	F_8	~~x~~	x^{vi}	BIF
36	C_4	p	x	PAC
38	F_{10}	x^{vii}	x''	RIF
38	F_{10}	~~x~~	x^{vii}	BIF
38	C_3	p	y'	PAC
41	$X_4/\bar{L}_4/$	+	x^{iv}	DUL
41	F_9	x^{viii}	x^{vii}	RIF
41	F_9	~~x~~	x^{viii}	BIF

TABLE 10.1-2 /cont'd/ Operation by time /3 of 4/

TIME	place	happening	signal	event type
42	F_4	y''	y'	RIF
42	F_4	~~x~~	y''	BIF
44	$X_4/L_7/$	p	x^{vi}	PLL
44	$\bar{G}_4$	+	x^{viii}	DCG
45	$X_4/L_5/$	+	x^{vii}	DUG
46	C_6	p	x''	PAC
47	L_9L_{1o}	t	x''	TBL
47	F_{12}	x^{ix}	x^{vi}	RIF
47	F_{12}	~~x~~	x^{ix}	BIF
49	$X_2/\bar{G}_7/$	+	x	DUG
49	F_{13}	x^{x}	x'	RIF
49	F_{13}	~~x~~	x^{x}	BIF
50	F_6	y'''	y''	RIF
50	F_6	~~x~~	y'''	BIF
51	$X_2/L_8/$	+	x^{vi}	DUL
51	$\bar{G}_5$	+	x^{ix}	DCG
52	$X_4/L_4/$	p	y'	DUL
53	$\tilde{G}_6$	+	x^{x}	DAG
53	$\bar{\bar{G}}_3$	+	y'''	DOG
54	$\bar{G}_2$	+	x^{v}	DCG
54	$X_4/L_6/$	p	y''	PLL
55	L_9L_{1o}	t	x''	TBL
56	$X_2/\bar{G}_2/$	+	y'	DUG

TABLE 10.1-2 /cont'd/ Operation by time /4 of 4/

TIME	place	happening	signal	event type
57	$X_2/\bar{G}_4/$	+	y''	DUG
58	$\tilde{G}_8$	+	x'	DAG
58	$\bar{\bar{G}}_7$	+	x''	DOG
62	C_4	p	y	PAC
75	$X_2/\bar{\bar{G}}_7/$	p	y	PRG
92	C_5	p	y	PAC
97	C_8	p	y	PAC
99	S_1	s	y	LEA

Consider table 10.1 – 3 which is a reordering of the signal table (Table 10.1 – 1). One can discern, quite easily, the following facts.

STATEMENT 10.1 – 6

a.) Collocal events are either simultaneous or distant enough;

b.) all the coincident events are of brith-replication type.

STATEMENT 10.1 – 7

There is a "beheading" process by gate G_7:

At $t = 0$ gate G_7 is on, then

$(49, X_2(\bar{G}_7), +, x)$	DUG
$(58, \bar{\bar{G}}_7, +, x'')$	DOG
$(75, X_2(\bar{G}_7), p, y)$	PRG

occurs. In other words

at $t = 49$ x dies by gate G_7, then

at $t = 58$ x'' turns off gate G_7, and then

at $t = 75$ y passes gate G_7.

STATEMENT 10.1 – 8

There is a " headletting" process by gate G_2.

Proof:

At the beginning G_2 is off. Then,

at $t = 30$ x passes it, x^{ν} closes it at $t = 54$, finally y dies by it at $t = 56$ in the form of y'.

TABLE 10.1-3 Operation by place /1 of 4 /

time	PLACE	happening	signal	event type
3	C_1	p	x	PAC
29	C_1	p	y	PAC
12	C_3	p	x'	PAC
38	C_3	p	y'	PAC
36	C_4	p	x	PAC
46	C_4	p	x''	PAC
62	C_4	p	y''	PAC
92	C_5	p	y	PAC
97	C_8	p	y	PAC
0	E_1	e	x	ENT
26	E_1	e	y	ENT
7	F_3	x'	x	RIF

TABLE 10.1-3 /cont'd/ Operation by place /2 of 4/

time	PLACE	happening	signal	event type
7	F_3	*	x'	BIF
33	F_3	y'	y	RIF
33	F_3	*	y'	BIF
16	F_4	x''	x'	RIF
16	F_4	*	x''	BIF
42	F_4	y''	y'	RIF
42	F_4	*	y''	BIF
24	F_6	x'''	x''	RIF
24	F_6	*	x'''	BIF
50	F_6	y'''	y''	RIF
50	F_6	*	y'''	BIF
32	F_7	x^{iv}	x'	RIF
32	F_7	*	x^{iv}	BIF
35	F_8	x^{vi}	x'	RIF
35	F_8	*	x^{vi}	BIF
41	F_9	x^{viii}	x^{vii}	RIF
41	F_9	*	x^{viii}	BIF
38	F_{1o}	x^{vii}	x''	RIF
38	F_{1o}	*	x^{vii}	BIF
34	F_{11}	x^{v}	x''	RIF
34	F_{11}	*	x^{v}	BIF
47	F_{12}	x^{ix}	x^{vi}	RIF
47	F_{12}	*	x^{ix}	BIF

TABLE 10.1-3 /cont'd/ Operation by place /3 of 4/

time	PLACE	happening	signal	event type
49	F_{13}	x^{x}	x'	RIF
49	F_{13}	x	x^{x}	BIF
54	$\bar{G}_2$	+	x^{v}	DCG
30	$X_2/\check{G}_2/$	p	x'	PRG
56	$X_2/\bar{G}_2/$	+	y'	DUG
27	$\bar{\check{G}}_3$	+	x'''	DCG
53	$\bar{\bar{G}}_3$	+	y'''	DOG
44	$\bar{G}_4$	+	x^{viii}	DCG
31	$X_2/\check{G}_4/$	p	x''	PRG
57	$X_2/\bar{G}_4/$	+	y''	PRG
51	$\tilde{G}_5$	+	x^{ix}	DAG
53	G_6	+	x^{x}	DAG
58	$\bar{\bar{G}}_7$	+	x''	DOG
49	$X_2/\bar{G}_7/$	+	x	DUG
75	$X_2/\bar{\bar{G}}_7/$	p	y	PRG
58	$\tilde{G}_8$	+	x'	DAG
21	L_2L_3	t	x''	TBL
47	L_2L_3	t	y''	TBL
26	$X_4/L_1/$	p	x'	PLL
52	$X_4/L_1/$	p	y'	PLL
41	$X_4/L_4/$	+	x^{iv}	DUL
45	$X_4/L_5/$	+	x^{vii}	DUL

TABLE 10.1-3 /cont'd/ Operation by place /4 of 4/

time	PLACE	happening	signal	event type
28	$X_4/L_6/$	p	x''	PLL
54	$X_4/L_6/$	p	y''	PLL
44	$X_4/L_7/$	p	x^{vi}	PLL
51	$X_2/L_8/$	+	x^{vi}	DUL
55	L_9L_{1o}	t	x''	TBL
99	S_1	s	y	LEA

More formally, the proof issimplythe following:

/30, $X_2/\overset{v}{G}_2/$, p, x/ PRG

/54, $\overset{\bar{v}}{G}_2$, +, x^v/ DCG

/56, $X_2/\bar{G}_2/$, +, y'/ DUG

And of course, it is true that

<u>STATEMENT 10.1-9</u>

x^v is a descendant of x .

<u>Proof</u>:

/34, F_{11}, ж, x^v/ BIF

/34, F_{11}, x^v, x''/ RIF

/16, F_4, ж, x''/ BIF

/16, F_4, x'', x'/ RIF

/7, F_3, ж, x'/ BIF

/7, F_3, x', x/ RIF.

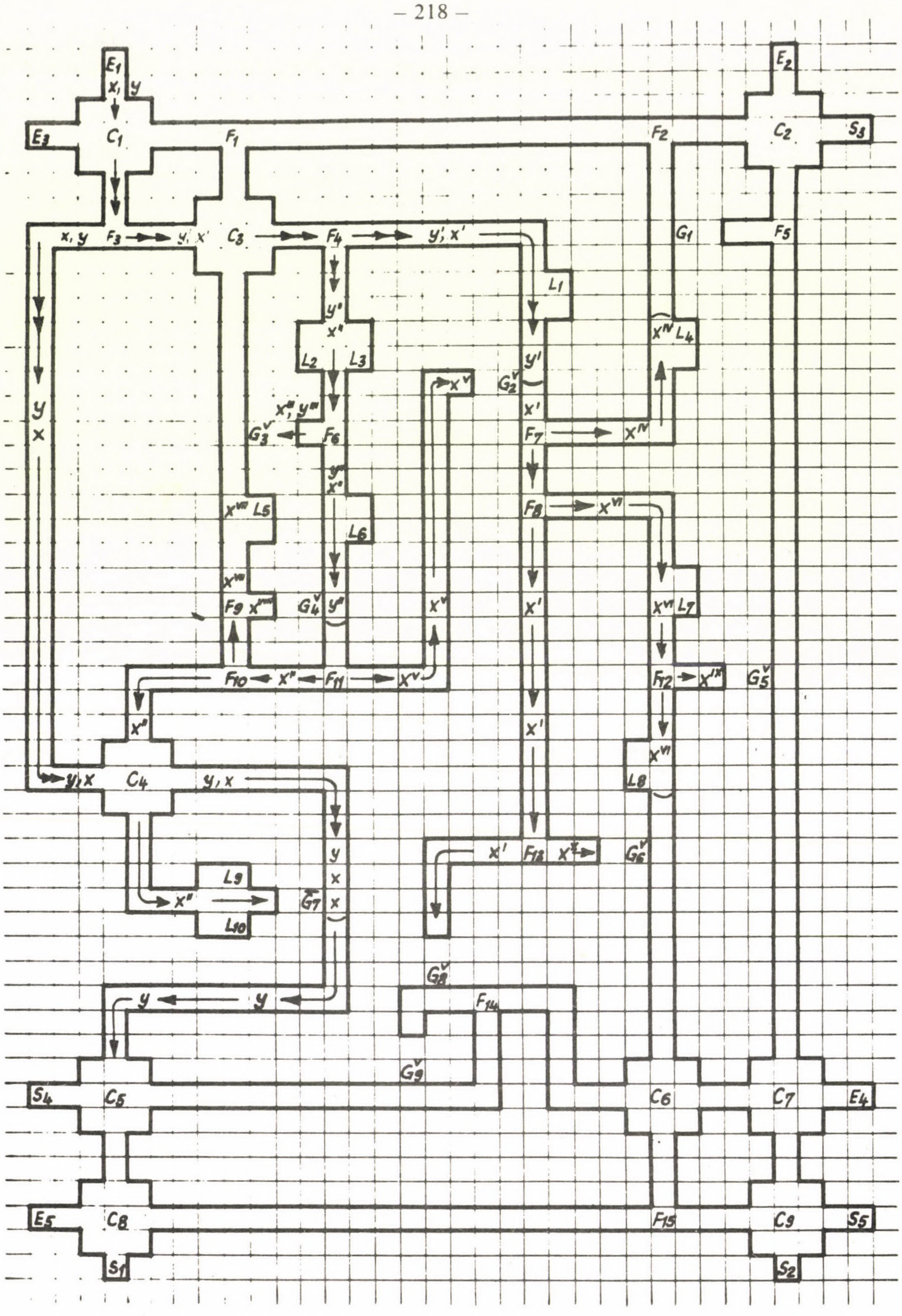


Figure 10.1-1

The operation of the MAXEL module

These kinds of proofs, which can be produced very easily by the relational data processing technique invented by Codd (1970), can as well be utilized in constructing the paths graphs (c.f. figures 8.1 – 1,2,3,) and carried out quite mechanically once the tables (signaltime and placetables) are given.
In a similar fashion to the above, we can show that

<u>STATEMENT 10.1 – 10</u>

y' is a descendant of y.

In this fashion each detail of MAXEL's operation can be understood, cleared up and eventually debugged.

10.2 <u>Resetting</u>

The purpose of the reset operation is to produce all gates in the off state, except G_7 which is to be closed, and, of course, except G_1, G_9 which are outside gates. In effect, gates $G_1 G_3$ and G_6 are used only during activation so their states are immaterial. Still, G_6 can also be turned off during the reset procedure.

The reset signal $r = [06]$ enters the MAXEL module at E_3, duplicates throughout the unit to yield altogether eight descendants and leaves at S_3 to enter the eastern neighbour U_i^{j+1} of U_i^j 33 shots later.

The signal-, time-, and place-tables can be constructed, in a similar fashion to that of described for the activation and operation stages see tables 10.2 – 1.2 and 3. Based on figure 10.2 – 1 one can automatically set up the fates of signals $r, r, \ldots, r^{ix}$ as follows.

10.2

r: /0, $E_3/^{e}$; /3, $C_1/^{p}$;/8, $F_1/^{r}$; /12, $C_3/^{p}$;
/20, $X_4/G_3//^{p}$; /23, $X_4/\bar{L}_5//^{p}$; /27, $F_9/^{r'''}$;
/30, $F_{1o}/^{r^{iv}}$; /38, $C_4/^{p}$; /47, $L_9L_{1o}/^{t}$; /50, $\bar{G}_7/^{+}$.

r': /8, $F_1/^{\ast}$; /25, $F_2/^{r''}$; /30, $C_2/^{p}$;/33, $S_3/^{s}$.

r'': /25, $F_2/^{\ast}$; /29, $X_4/G_1//^{p}$; /33, $X_3/\bar{L}_4//^{p}$
/42, $F_7/^{r^{vi}}$; /45, $F_8/^{r^{vii}}$; /59, $F_{13}/^{r^{ix}}$;
/68, $\overset{v}{G}_8/^{+}$.

r''': /27, $F_9/^{\ast}$; /30, $\bar{\bar{G}}_4/^{+}$.

r^{iv}: /30, $F_{1o}/^{\ast}$; /34, $F_{11}/^{r^{v}}$; /37, $X_2/\bar{\bar{G}}_4//^{p}$;
/40, $X_4/\bar{L}_6//^{+}$.

r^{v}: /34, $F_{11}/^{\ast}$; /53, $\bar{\bar{G}}_2/^{+}$.

r^{vi}: /42, $F_7/^{\ast}$; /47, $X_4/\bar{L}_1/,/^{+}$.

r^{vii}: /45, $F_8/^{\ast}$; /54, $X_4/L_7//^{p}$; /57, $F_{12}/^{r^{viii}}$;
/61, $X_2/L_8//^{+}$.

r^{viii}:/57, $F_{12}/^{\ast}$; /61, $\overset{v}{G}_5/^{+}$.

r^{ix}: /59, $F_{13}/^{\ast}$; /63, $\overset{v}{G}_6/^{+}$.

With these signals table 10.2 – 1 is constructed, containing the events in order of signal names.

TABLE 10.2-1 Resetting by signal /1 of 2/

time	place	happening	SIGNAL	event type
0	E_3	e	r	ENT
3	C_1	p	r	PAC
8	F_1	r'	r	RIF
12	C_3	p	r	PAC
20	$X_4/\bar{G}_3/$	p	r	PLG
23	$X_4/\bar{L}_5/$	p	r	PLL
28	F_9	r'''	r	RIF
30	F_{10}	r^{iv}	r	RIF
38	C_4	p	r	PAC
47	L_9L_{10}	t	r	TBL
50	$\bar{G}_7$	+	r	DCG
8	F_1	*	r'	BIF
25	F_2	r''	r'	RIF
30	C_2	p	r'	PAC
33	S_3	s	r'	LEA
25	F_2	*	r''	BIF
29	$X_4/G_1/$	p	r''	PLG
33	$X_4/\bar{L}_4/$	p	r''	PLL
42	F_7	r^{vi}	r''	RIF
45	F_8	r^{vii}	r''	RIF

TABLE 10.2-1 /cont'd/ Resetting by signal /2 of 2/

time	place	happening	SIGNAL	event type
59	F_{13}	r^{ix}	r''	RIF
58	$\overset{v}{G}_8$	+	r''	DOG
27	F_9	$\ast$	r'''	BIF
30	$\overset{v}{\bar{G}}_4$	+	r'''	DOG
30	F_{1o}	$\ast$	r^{iv}	BIF
34	F_{11}	r^{v}	r^{iv}	RIF
37	$X_2/\bar{\bar{G}}_4/$	p	r^{iv}	PLG
40	$X_4/\bar{L}_6/$	+	r^{iv}	DUL
34	F_{11}	$\ast$	r^{v}	BIF
53	$\bar{\bar{G}}_2$	+	r^{v}	DOG
42	F_7	$\ast$	r^{vi}	BIF
47	$X_4/\bar{L}_1/$	+	r^{vi}	DUL
45	F_8	$\ast$	r^{vii}	BIF
54	$X_4/L_7/$	p	r^{vii}	PLL
57	F_{12}	r^{viii}	r^{vii}	RIF
61	$X_2/\bar{L}_8/$	+	r^{vii}	DUL
57	F_{12}	$\ast$	r^{viii}	BIF
61	$\overset{v}{G}_5$	+	r^{viii}	DOG
59	F_{13}	$\ast$	r^{ix}	BIF
63	$\overset{v}{G}_6$	+	r^{ix}	DOG

From the signal table one can get the following

STATEMENT 10.2 – 1

signal r	resets gate G_7	by turning it on
signal r''	resets gate G_8	by turning it off
signal r'''	resets gate G_4	by turning it off
signal r^{v}	resets gate G_2	by turning it off
signal r^{viii}	resets gate G_5	by turning it off
signal r^{ix}	resets gate G_6	by turning it off

As for resetting by time we have:

STATEMENT 10.2 – 2

All the simultaneous events are either coincident or distant enough.

STATEMENT 10.2 – 3

All the coincident events are of brith-replication type pairs.

STATEMENT 10.2 – 4

Reset process takes 68 shots.

Proof: from the timetable (Table 10.2 – 2)

TABLE 10.2-2 Resetting by time /1 of 3/

TIME	place	happening	signal	event type
0	E_3	e	r	ENT
3	C_1	p	r	PAC
8	F_1	r'	r	RIF

TABLE 10.2-2 /cont'd/ Resetting by time /2 of 3/

TIME	place	happening	signal	event type
8	F_1	*	r'	BIF
12	C_3	p	r	PAC
20	$X_4/\bar{G}_3/$	p	r	PLG
23	$X_4/\bar{L}_5/$	p	r	PLL
25	F_2	r''	r	RIF
25	F_2	*	r''	BIF
27	F_9	r'''	r	RIF
27	F_9	*	r'''	BIF
29	$X_4/G_1/$	p	r''	PLG
30	F_{1o}	r^{iv}	r	RIF
30	F_{1o}	*	r^{iv}	BIF
30	C_2	p	r'	PAC
30	$\overset{v}{\bar{G}}_4$	+	r'''	DOG
33	S_3	s	r'	LEA
33	$X_4/\bar{L}_4/$	p	r''	PLL
34	F_{11}	r^{v}	r^{iv}	RIF
34	F_{11}	*	r^{v}	BIF
37	$X_2/\bar{\bar{G}}_4/$	p	r^{iv}	PLG
38	C_4	p	r	PAC
40	$X_4/\bar{L}_6/$	+	r^{iv}	PRL
42	$\overset{*}{\bar{G}}_7$	+	r	DOG
42	F_7	r^{vi}	r''	RIF

TABLE 10.2-2 /cont'd/ Resetting by time /3 of 3/

TIME	place	happening	signal	event type
45	F_8	r^{vii}	r''	RIF
45	F_8	$\cancel{x}$	r^{vii}	BIF
47	L_9L_{1o}	t	r	TBL
47	$X_4/L_1/$	+	r^{vi}	DUL
50	G_7	+	r	DCG
53	$\overset{=}{G}_2$	+	r^{v}	DOG
54	$X_4/L_7/$	p	r^{vii}	PLL
57	F_{12}	r^{viii}	r^{vii}	RIF
57	F_{12}	$\cancel{x}$	r^{viii}	BIF
59	F_{13}	r^{viii}	r''	RIF
59	F_{13}	$\cancel{x}$	r^{ix}	BIF
61	$X_2/L_8/$	+	r^{vii}	DUL
61	$\overset{v}{G}_5$	+	r^{viii}	DOG
63	$\overset{v}{G}_6$	+	r^{ix}	DOG
68	$\overset{v}{G}_8$	+	r''	DOG

From the place-table (Table 10.2 – 3) we get

STATEMENT 10.2 –5

All the collocal events are coincident.

STATEMENT 10.2 – 6

All the coincident events are birth-replication type pairs.

TABLE 10.2-3 Resetting by place /1 of 2/

time	PLACE	happening	signal	event type
3	C_1	p	r	PAC
30	C_2	p	r'	PAC
12	C_3	p	r	PAC
38	C_4	p	r	PAC
0	E_3	e	r	ENT
8	F_1	r'	r	RIF
8	F_1	x̶	r'	BIF
25	F_2	r''	r'	RIF
25	F_2	x̶	r''	BIF
42	F_7	r^{vi}	r''	RIF
42	F_7	x̶	r^{vi}	BIF
45	F_8	r^{vii}	r''	RIF
45	F_8	x̶	r^{vii}	BIF
27	F_9	r'''	r	RIF
27	F_9	x̶	r'''	BIF
30	F_{1o}	r^{iv}	r	RIF
30	F_{1o}	x̶	r^{iv}	BIF
34	F_{11}	r^{v}	r^{iv}	RIF
34	F_{11}	x̶	r^{v}	BIF
57	F_{12}	r^{viii}	r^{vii}	RIF
57	F_{12}	x̶	r^{viii}	BIF
59	F_{13}	r^{ix}	r''	RIF

TABLE 10.2-3 Resetting by place /cont'd/ /2 of 2/

time	PLACE	happening	signal	event type
59	F_{13}	x	r^{ix}	BIF
29	$X_4/G_1/$	p	r''	PLG
53	$\bar{\bar{G}}_2$	+	r^{v}	DOG
20	$X_4/\bar{G}_3/$	p	r	PLG
30	$\overset{v}{\bar{G}}_4$	+	r'''	DOG
37	$X_2/\bar{\bar{G}}_4/$	p	r^{iv}	PLG
61	$\overset{v}{G}_5$	+	r^{viii}	DOG
63	$\overset{v}{G}_6$	+	r^{ix}	DOG
50	$\bar{G}_7$	+	r	DCG
68	$\overset{v}{\bar{G}}_8$	+	r''	DOG
47	$X_4/\bar{L}_1/$	+	r^{vi}	DUL
33	$X_4/L_4/$	p	r''	PLL
23	$X_4/\bar{L}_5/$	p	r	PLL
40	$X_4/\bar{L}_6/$	+	r^{iv}	DUL
54	$X_4/L_7/$	p	r^{vii}	PLL
61	$X_2/L_8/$	+	r^{vii}	DUL
47	L_9L_{1o}	t	r	TBL
33	S_3	s	r'	LEA

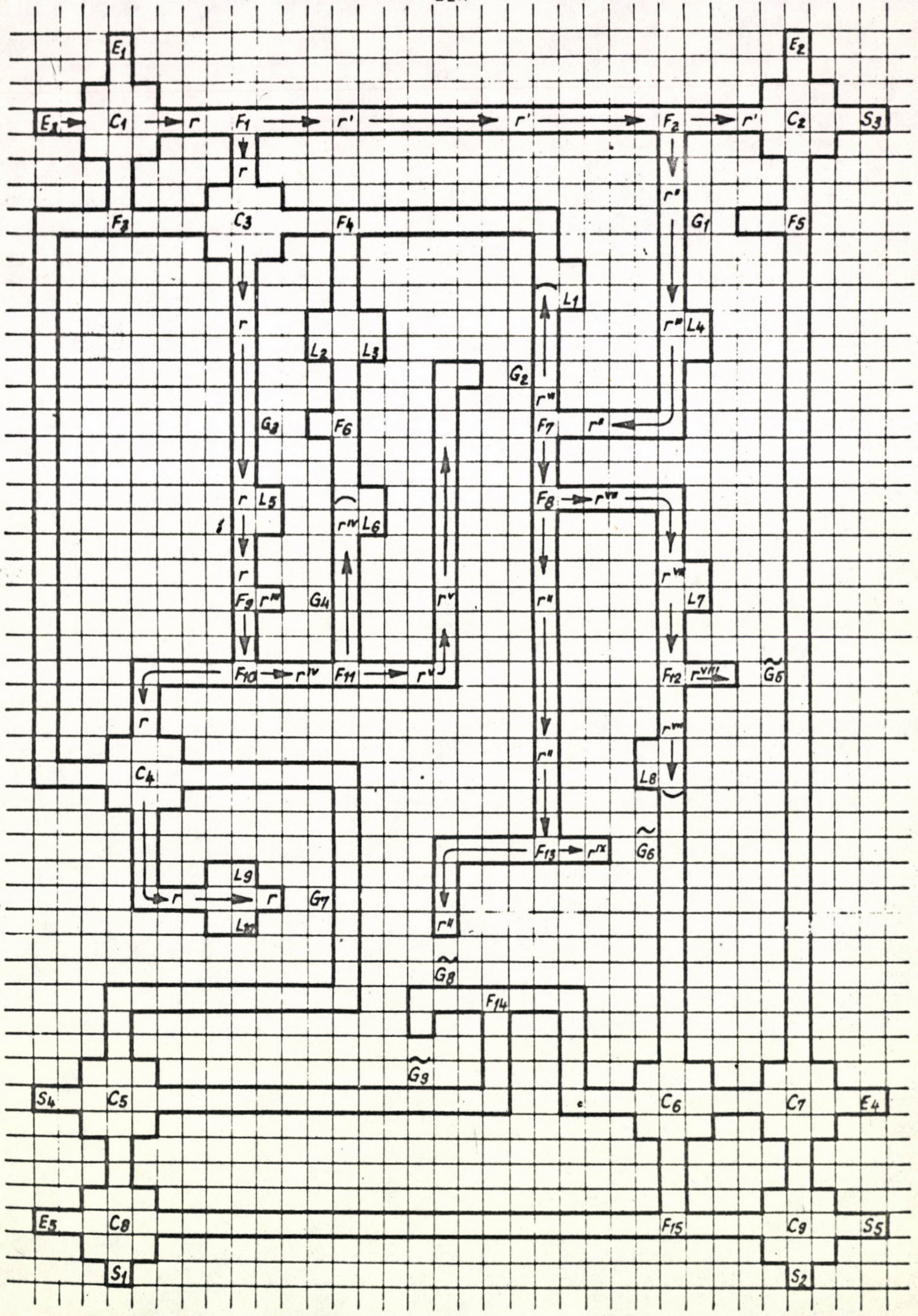


Figure 10.2-1

The resetting of the MAXEL module

REFERENCES

[1] Aladyev, V.: Survey of Research in the Theory of Homogenous Structures and their Applications
Mathematical Biosciences, 22 121-154. (1974).

[2] Bagyinszki, J.: Residue Number System (Decimal) Arithmetic In Parallel Systems Such As Cellular Automata and Magnetic Bubble Memories
KFKI report No. 75-15. Hungary (1975).

[3] Burks, A. W.: Essays in Cellular Automata University of Illinois Press, Urbana, Ill. (1968).

[4] Codd, E. F.: Cellular Automata Academic Press, New York and London. (1968).

[5] Codd, E.F.: A Relational Model of Data for Large Shared Data Banks
Comm. ACM 13, 377-387. (1970)

[6] Codd, E. F.: A Data Base Sublanguage Founded On the Relational Calculus
IBM RJ. June (1971).

[7] Conway, J.: Mathematicel Games (M. Gardner ed.) Sci. Amer. October (1970)

[8] Dettai, I. : Effectivity Problems and Suggestions Concerning CODD-ICRA Cellular Space
KGM ISZSZI technical report, Hungary (1974).

[9] Dettai, I.: A Version of von Neumann's Cell KGM ISZSZI technical report, Hungary (1975)

[10] Domán, A.: A Three-dimensional Cellular Space (A challenge to CODD-ICRA)
KGM ISZSZI technical report, Hungary (1974).

[11] Domán, A.: A Flexible Three-dimensional Cellular Space International Congress of Cybernetics and Systems, Bucharest, Rumania, Aug 25-29 (1975).

[12] Domán, A.: Parallel Computing Systems Doctoral thesis at the Technical University of Budapest, Hungary (in Hungaian)

[13] Fáy, G.: Circuitry Realization of Codd's Automation's Cell First International Conference on Computer Science,
Székesfehérvár, Hungary, May 21-26. (1973).

[14] Fáy, G.: DPL ALPHA and Codd's Cellular Space Second European Meeting on Cybernetics and Systems Research, Vienna, Austria April 16-19. (1974).

[15] Fáy, G.: A Basic Course on Cellular Automata University lecture notes, Budapest, Eötvös Lóránd University (In Hungarian) (1975).

[16] Fáy, G., D.V. Takács: Survey of de CODD-ICRA In: LINDENMAYER (1975).

[17] Fazekas, B.: Some New Components and Phenomena in CODD-ICRA Space. Graduate thesis in mathematics, Budapest Eötvös Lóránd University (In Hungaian) (1975).

[18] Golze, U.: Destructioı of a Universal Computer-Constructor in Codd's Cellular Space Report. Dept. of Comp. and Comm. Sci. Univ. of Michigan, Ann Arbor, Michigan 48104 (1972).

[19] Herman, G.T.: (org) Conference on Bibliogically Motivated Automata Theory MITRE Corporation, Mc.Lean, Virginia, USA June 19-22 (1974).

[20] Herman, G.T. and Rozenberg, G.: Developmental Systems and Language North Holland Publ. Co. Amsterdam. (1975).

[21] Hoare, C. A. R.: An Axiomatic Basic for Computer Programming Comm. ACM. 12, 576-583. (1969)

[22] Huszár A.: Testing equipment for Automaton Cells Graduate thesis in electrical engineering. Technical University of Budapest (In Hungarian) (1974).

[23] ICRA TEAM: Studies in Cellular Automata("Sejtautomaták")
To be publisehed by Gondolat, Budapest, first in Hungarian later in English. (1978.)

[24] Ippolito, J.C.: Contribution a l'implantation de functions logiques sur des réseaux cellulaires
These, Grade de Docteur es Sciences Physiques, Université Paul Sabatier de Toulouse. (1972).

[25] Kaszás, O.: Character Generation in CODD-ICRA Space Graduate thesis in electical engineering. Kandó Kálmán Technical Highschool, Budapest (In Hungarian) (1975).

[26] Legendi, T.: Simulation of Cellular Automata
Inner report, Szeged, József Attila University, MTA group of Mathematical logic and automata theory. (In Hungarian) (1975).

[27] LINDENMAYER, A. (org): Conference On Formal Languages, Automata and Development
Noordwijkerhout, The Netherlands 31 March – 6 april. (1975).

[28] Martoni, V.: Investigations in Lindenmayer space. Inner report, KGM ISZSZI Budapest, (In Hungarian) (1975).

[29] Riguet, J. (org): Conference on Automata and Related Topics Université René Descartes Sorbonne, Paris, May 16-19. (1974).

[30] Riguet, J.: Automates Cellulaires a Bord et Automates CODD-ICRA I.-II. Computers redus de L'Academie les Sciences de Paris SIRIE A t. 282. (19. Janvier 76.) pp. 167-170, 239-242. (1976).

[31] Smith, III. A.R.: Cellular Automata Theory
Technical report, Sztanford Electronic Laboratories No. SU-SEL 70-016. (1969)

[32] Szőke, M. (Mrs): Destruction in CODD-ICRA Graduate thesis in mathematics. Budapest Eötvös Lóránd University (In Hungarian) (1975).

[33] Takács, D.V. (Mrs): A Bootstrap for Codd's Cellular Space
First International Conference on Computer Science,
Székesfehérvár, Hungary, May 21-26. (1973).

[33] Takács, D.V. (Mrs): Galois Connections and DPL ALPHA System Second European Meeting on Cybernetics and Systems Research Vienna, Austria (1974)

[34] Takács, D.V. (Mrs): Cellular Automaton as a new tool for computer science
KGM ISZSZI report, Budapest. Hungary (In Hungarin) (1974)

[35] Takács, D.V. (Mrs): Un Automate Cellulaire CODD-ICRA Pour la Recherche de Numbers
These Presenté a Université René Descartes de Paris Sorbonne Doctorat d'Université (1975).

[36] Toffoli, T.: Backward steps in cellular spaces
In: Lindenmayer (1975).

[37] Von Neumann, J.: The General and Logical Theory of Automata. pp. 1-41 of Cerebral Machanismus in Behavior. The Hixon Symposium, (1948) ed by L.A. Jeffress, published by John Wiley, New York, 1951. (1948)

[38] Von Neumann, J.: Theory of Self-reproducing Automata
University of Illinois Press, Urbana edited and completed by A. W. Burks. (1966).

ISBN 963 311 067 X

XC 46410