Poznań Studies in Contemporary Linguistics 38, 2002/2003 pp. 167-185
© School of English, Adam Mickiewicz University, Poznań, Poland

MINIMALIST FACULTY OF LANGUAGE, COMPUTATIONAL SYSTEM AND LOGIC

JOANNA SADOWNIK
Szczecin University

'Logika tropi prawdę na drzewie gramatyki'
Quine (1977: 56)

ABSTRACT

This paper concentrates on metalinguistic function of logic in the minimalist description of LgF (Language Faculty) and especially CHL (computational system). It is intended to scrutinize the interrelations, i.e. to tease out those elements of the formal language of logic that are directly employed by Chomsky in his desription of the faculty of language and computational system.

1. Introduction

Let me start this presentation with an observation made by Bach (1989) who considers Chomsky's major insight into linguistic studies to be the claim that language is a formal system. According to Bach, the major technical innovation of early generative grammar was to state the combinatorial properties of language in terms of a formal system. Whether such an assertion is right or wrong may be arguable; nevertheless worth considering.

Formalisation seems to confer many advantages. At the deepest level, it permits one to use logical and mathematical techniques to study the consequences of one's hypotheses, for example it helps to determine the expressive power (i.e. strong or weak generative capacity) of alternative hypothesised combinatorial systems (e.g. Chomsky 1957, Chomsky and Miller 1963), or it allows to estimate the learnability of such systems (e.g. Wexler and Culicover 1980). At a more methodological level, formalisation permits one to be more abstract, rigorous and compact in stating and examining one's claims and assumptions. And, as Chomsky stressed in a much quoted passage from the preface to *Syntactic structures* (Chomsky 1957), a formalisation uncovers consequences, good or bad, that one might not otherwise have noticed.

Therefore in what follows, I am going to take a closer look at the relation of logic – an undoubtedly formal system, to Chomsky's minimalist view on language, esp. to his computational system C_{HL}. The reason why I decide to concentrate solely on his minimalist enterprise is twofold. First, the degree and importance of formalisation in the earliest works such as *The logical structure of linguistic theory* (Chomsky 1955) or *Syntactic structures* (Chomsky 1957) are considerably different from those in his *Minimalist program* (Chomsky 1995) or *Minimalist inquiries* (Chomsky 1998). Consequently, the picture of language emerging from them is also different. Second, I believe it to be much more sensible and fruitful to concentrate only on those aspects of logic that stood the trial of time and passed muster.

I would like to stress here, that I am aware of the fact that formalisation is not an unmitigated blessing. An excessive preoccupation with logical formality can (but of course does not have to) overwhelm the search for genuine insight into language. I personally find the proper formalisation of a theory a delicate balance between rigor and lucidity on the one hand and flexibility and openness on the other. That is, enough of it should be retained to spell out carefully and precisely what the theory claims or describes, but not too much in order to keep a more holistic, abstract perspective and to be able to render the theory communicative and commensurate with an intricate network of other theories.

2. Logic as Metalanguage in the description of Language Faculty (LgF) and Computational System (C_{HL})

In order to talk about the metalinguistic function of logic in the description of language faculty (LgF) it is essential to properly define what logic and what LgF are. As far as logic is concerned, I may define it – for the purpose of this section – as a formal system, parallelly to Bach's definition of language:

(1) Logic – a formal system, i.e.: a formalised set of elements or well-formed formulas, creating a certain whole; where formalisability is understood as in (2) below:

(2) Formalisability – the possibility of framing representative elements or formulas, of which the relations can be determined according to certain rules of testing or deduction (cf. a similar though not identical definition in Strawson 1952: 210).

Such a definition of logic is of course very narrow and completely ignores the subject matter of logical study. However, what matters mostly for my purposes is its formal systematicity, and I will only work upon a very small field of logic, such as e.g. Recursion Theory, algorithms, etc. – see the following pages.

As far as the faculty of language (LgF) is concerned, I may propose the following preliminary and general definition, only to plunge into the details immediately after:

(3) Language Faculty – a component of the human mind/brain dedicated to language.

Chomsky (1998: 1)

A particular language Lg is an instantiation of the initial state of the cognitive system of the LgF with options specified. Chomsky views Lg as a generative procedure performed by C_{HL} that constructs pairs (π, λ) that are interpreted at the articulatory-perceptual (A-P) and conceptual-intentional (C-I) interfaces, respectively, as 'instructions' to the performance systems. π is a PF representation and λ is an LF representation, each consisting of 'legitimate objects' that can receive an interpretation (perhaps gibberish). If a generated representation consists entirely of such objects, we say that it satisfies the condition of Full Interpretation (FI). A linguistic expression of Lg is at least a pair (π, λ) meeting this condition – and, under minimalist assumptions, at most such a pair, meaning that there are no levels of linguistic structure apart from the two interface levels PF and LF.

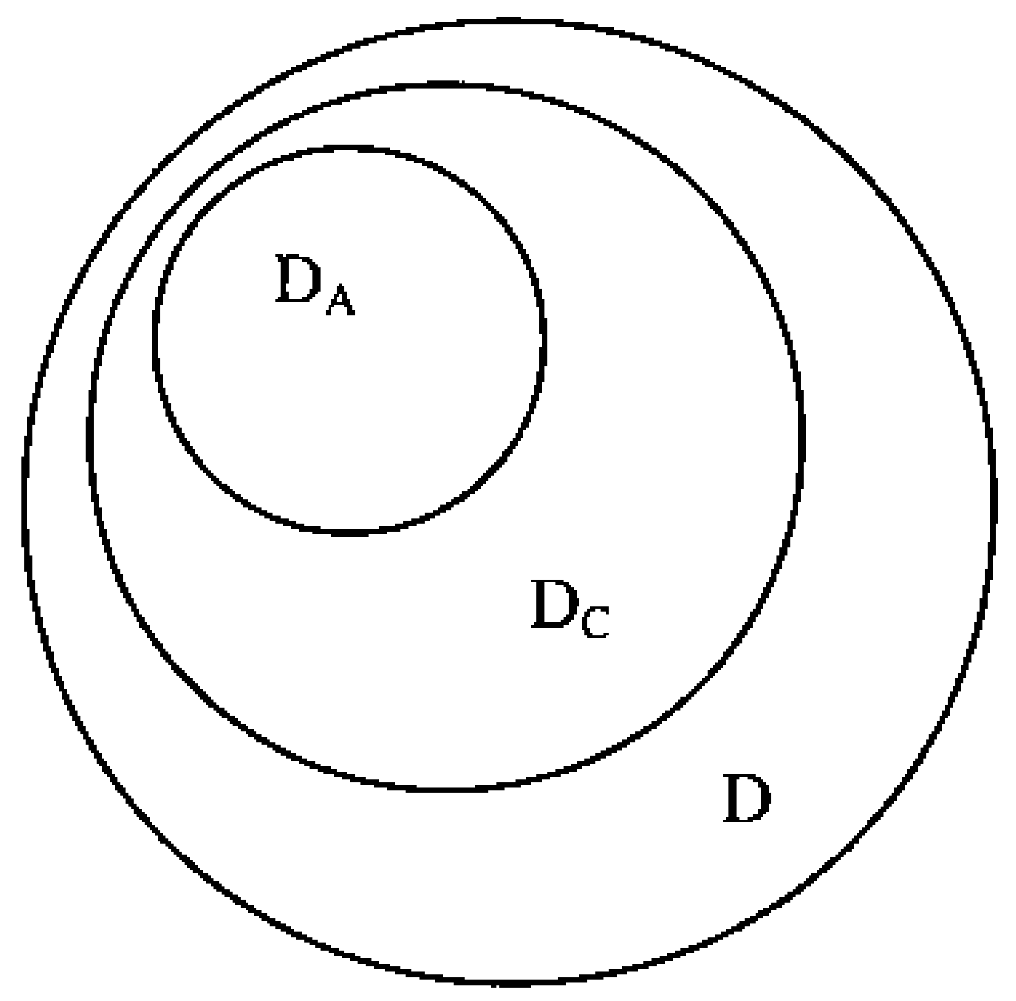
The language Lg determines a set of derivations (computations) done by C_{HL}. A derivation converges at one of the interface levels if it yields a representation satisfying FI at this level, and converges if it converges at both interface levels PF and LF; otherwise it crashes. Thus, Chomsky (1995: 220) adopts the (non-obvious) hypothesis that there are no PF-LF interactions relevant to convergence.

It seems that a linguistic expression of Lg cannot be defined just as a pair (π, λ) formed by a convergent derivation. Rather, its derivation must be optimal, satisfying certain natural economy conditions.[1] Less economical computations are blocked even if they converge.

According to Chomsky, then, the language Lg generates three relevant sets of computations: the set D of derivations, a subset D_C of convergent derivations, and the subset D_A of admissible derivations of D. FI determines D_C, and the economy conditions select D_A. Since economy considerations hold only among convergent derivations (cf. Chomsky 1995: 220-221) then the following reasoning is true: if a derivation crashes, it does not block others; thus, D_A is a subset of D_C:

[1] Chomsky (1998: 9) pursues the question of optimality of Lg much further. He puts forward, not without some reservations, The Strongest Minimalist Thesis which states that language is an optimal solution to legibility conditions. It means that LgF satisfying legibility conditions in an optimal way satisfies all other empirical conditions, too: acquisition, processing, neurology, language change, etc. Accordingly, it would seem that the language organ is a perfect solution to minimal design specifications. However, Chomsky points out that such a conclusion would be a strange and surprising result, therefore interesting to whatever extent it might be true. From such a perspective, the task of a minimalist program is to explore the possibility that Lg approaches the above optimal design.

(4)



$(D_A \subset D_C) \subset D$ – the sets are properly included in one another

From what I have described in the above paragraphs, one may see that Chomsky assumes that the computational system C_{HL} is strictly derivational. This is not, however, the only possible and right characterisation of the syntax. C_{HL} may as well be representational in nature.[2] In any case, however, we have to remember that the ordering of operations is abstract, expressing postulated properties of the LgF of the brain with no temporal interpretation implied. Accordingly, the terms output and input have a metaphorical (metalinguistic) flavour to them.

Another natural condition postulated by Chomsky is that derivational outputs consist of nothing beyond properties of the items of lexicon (lexical features) – in other words, the interface levels consist of nothing more than the arrangements of lexical features. To the extent that this is true, the language meets a Condition of Inclusiveness, where the relevant condition may be defined as in (5) below:

(5) CONDITION OF INCLUSIVENESS (for the perfect Lg):

> Any structure formed by the computation (in particular, π and λ) is constituted of elements already present in the lexical items selected for numeration (N); no new objects are added in the course of computation apart from rearrangements of lexical properties (in particular, no indices, bar levels in the sense of X-bar Theory, etc.).
>
> Chomsky (1995: 228)

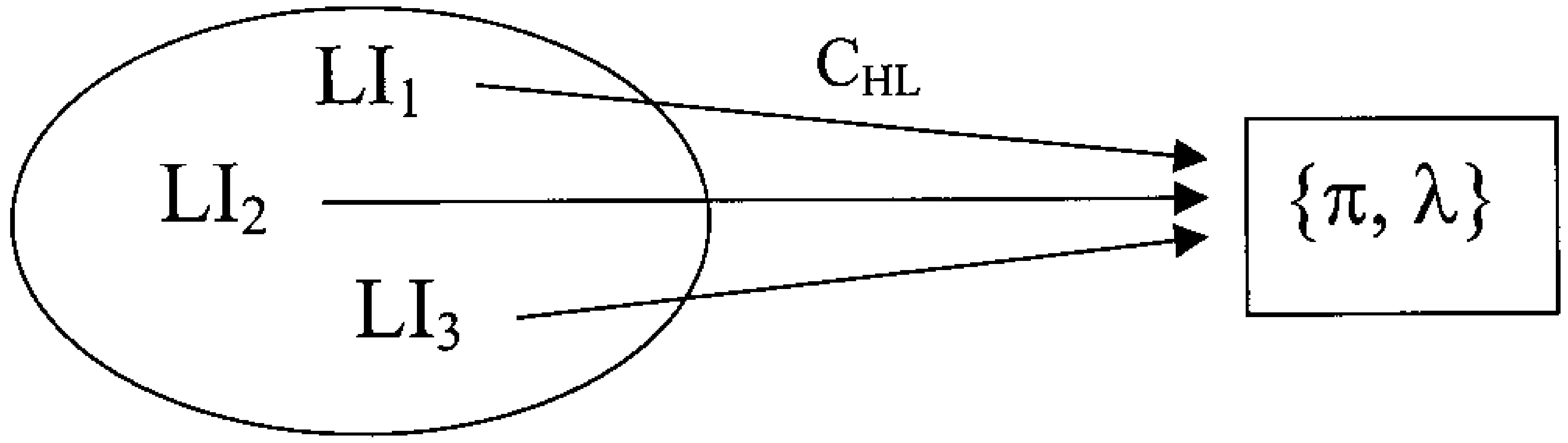
[2] The difference in the two assumptions may be reduced to a question whether C_{HL} involves succesive operations leading to (π, λ) (if it converges) – derivational approach; or it whether it operates in one of any number of other ways, say selecting two given representations and then computing to determine if they are properly paired selecting one and deriving the other – representational approach.

The condition, however, is not fully met. Chomsky notes that this condition holds virtually of the computation from N to LF ($N \rightarrow \lambda$), and that standard theories take it to be radically false for the computation to PF ($N \rightarrow PF$). Anyway, the point is that with sufficiently rich formal devices (say, Graph Theory) counterparts to any such objects (nodes, bars, indices, etc.) can readily be constructed (derived from features). Then, there is no essential difference between admitting new kinds of objects and allowing richer use of formal devices.

With regard to the computational system C_{HL}, Chomsky (1995: 170) assumes that the initial state is constituted of invariant principles with options restricted to functional elements and general properties of the lexicon. A selection Σ among these options determines a language. Lg acquisition involves fixing Σ; the grammar of the language states Σ, nothing more (lexical arbitrariness and PF component aside). In addition, the principles of UG involve only elements that function at interface levels; nothing else can be 'seen' in the course of computation.

In detail, a linguistic expression (π, λ) of Lg satisfies output conditions at the PF and LF interfaces. Beyond that, π and λ must be compatible: it is not the case that any sound can mean anything. In particular, π and λ must be based on the same lexical choices. We can, then, think of C_{HL} as mapping some array A of lexical choices to the pair (π, λ). What is A? According to Chomsky (1995: 225), at least it must indicate what lexical choices are and how many times each is selected by C_{HL} in forming (π, λ). Further on, he assumes a numeration to be a set of pairs (LI, i), where LI is an item of the lexicon and i is its index, understood to be the number of times that LI is selected. This can very roughly be represented by the following drawing:

(6) $A \geq N \quad f : N \rightarrow \{\pi, \lambda\} \;/\; f(N) = \{\pi, \lambda\}$[3]



[3] Curly brackets denote an unordered pair.

Take A to be, at least, a numeration N; C_{HL} maps N to (π, λ). The procedure C_{HL} selects an item from N and reduces its index by 1, then performing permissible computations. A computation constructed by C_{HL} does not count as a derivation at all, let alone a convergent one, unless all indices are reduced to 0 (zero).

Viewing the language Lg as a derivation-generating procedure, we may think of it as applying to a numeration N and forming a sequence S of symbolic elements $(\delta_1, \delta_2, ..., \delta_n)$, terminating only if δ_n is a pair (π, λ) and N is reduced to zero (the computation may go on).[4] S formed in this way (strictly speaking, an n-element finite sequence) is a derivation, which converges if the elements of δ_n satisfy FI at PF and LF, respectively.[5] Economy considerations select the admissible convergent derivations.

Given the numeration N, the operations of C_{HL} recursively construct syntactic objects from items in N and syntactic objects already formed. We have to determine what these objects are and how they are constructed. Insofar as the Condition of Inclusiveness holds, the syntactic objects are rearrangements of properties of the lexical items of which they are ultimately constructed. As an illustration, let me take the computation $N \rightarrow \lambda$.

Suppose that the derivation has reached the stage Σ, which may be taken to be a set $\{SO_1, ..., SO_n\}$ of syntactic objects. Logically speaking, Σ may be regarded as a set being the union of all the elements that appeared in the derivation up to the given stage:

(7) $\Sigma = (SO_1 \cup SO_2 \cup ... \cup SO_n)$[6]

One of the operations of C_{HL} is a procedure that selects a lexical item LI from the numeration, reducing its number by 1, and introduces it into the derivation as SO_{n+1}. Call the operation: Select. At the LF interface, Σ can be interpreted only if it consists of a single syntactic object. Clearly then, C_{HL} must include a second procedure that combines syntactic objects already formed. A derivation converges if this operation has applied often enough to leave us with just a single object, also exhausting the initial numeration. The simplest such operation takes a pair of syntactic objects (SO_i, SO_j) and replaces them by a new combined syntactic object SO_{ij}. Chomsky calls this operation: Merge. Chomsky (1995: 226) states that: "the operations Select and Merge, or some close counterparts, are necessary components of any theory of natural language".

Note that no question arises about the motivation for application of Select or Merge in the course of derivation. If Select does not exhaust the numeration, no derivation is generated and no questions of convergence or economy arise. Insufficient application of Merge has the same property, since the derivation then fails to yield an LF representation at all; again, no derivation is generated, and questions of convergence and economy do not arise. "The operations Select and Merge are 'costless'; they do not fall within the domain of discussion of convergence and economy. Similarly, we do not have to ask about the effect of illegitimate operations, any more than proof theory is concerned with a sequence of lines that does not satisfy the formal conditions that define 'proof' or a chess playing algorithm with evaluation of improper moves" (Chomsky 1995: 226). The terms 'proof' and 'algorithm', used by Chomsky for intentional comparison, are borrowed from logical Proof Theory and Recursion Theory, respectively. For the time being (see following pages) and for the purpose of this presentation I will define them as in (8) and (9) below:

4 Cf. the analogy between forming S and terminating and the notions of 'loops' and 'halt' – see p. 183.

5 An n-element finite sequence is a function defined on a set of all positive integers – dom(f) = 1, ..., n.

6 See the analogy to the notion of a φ-oracle – see p. 181.

(8) Proof of a theorem T is a sequence of formulae such that the last in the sequence is formula T, and every element of the sequence is either an axiom or is derived from previous formulae by the implementation of one or more rules of inference. cf. Marciszewski (1987: 224)

(9) An algorithm over an alphabet A is every finite sequence of formulas of the form $u \rightarrow v$ or $u \rightarrow \bullet v$, called substitutions, in which $u, v \in A^*$.[7] cf. Marciszewski (1987: 193)

A* means here a set of all finite sequences of symbols ('words') over the alphabet A, including the empty sequence ε. A substitution of the form $u \rightarrow \bullet v$ is called a terminal substitution (cf. in our case the pair (π, λ)).

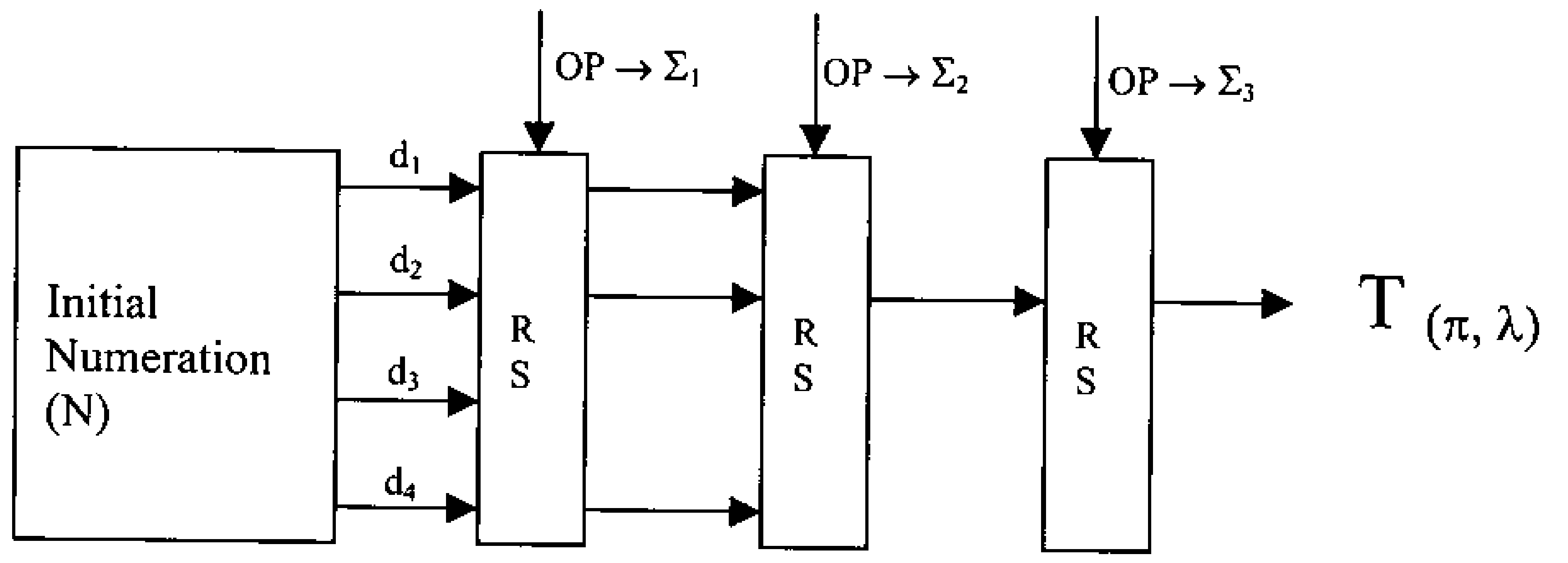
As it was already mentioned, Chomsky wants the initial array A (see the analogue in the definition in (9) above), whether a numeration or something else, not only to express a compatibility relation between π and λ but also fix the reference set for determining whether a derivation from A to (π, λ) is optimal, that is, not blocked by a more economical derivation. He suggests that numeration determines such a set and that only alternative derivations with the same numeration are evaluated for economy. Moreover, the reference set is interpreted in his theory rather locally, i.e. at a particular stage Σ of a derivation, one considers only continuations of the derivation already constructed – in particular, only the remaining parts of the numeration N.[8] Application of the operation OP to Σ is barred if this set contains a more optimal derivation in which OP does not apply to Σ. The number of derivations to be considered for determining whether OP may apply reduces radically as the derivation proceeds (cf. (10) below for illustration).

7 Markov's algorithm, strictly speaking.

8 See the analogy between the optimisation operation and fixing the reference set and S-command – see p. 181.

Where 'global' properties of derivations have to be considered and optimised, e.g. in case of determining the applicability of the principle Procrastinate, Chomsky expects to find some ready algorithm to reduce computational complexity.

(10)



d_1, d_2, d_3, d_4 – competing derivations (i.e. finite sequences of the form $S = (\delta_1, \delta_2, ..., \delta_n)$); RS – reference set; T – termination; OP → $\Sigma_{1,2,3}$ – optimisation operations to particular stages

A core property of C_{HL} is feature checking, the operation that drives movement under the Last Resort. For the purpose of this paper, I might reduce feature checking to deletion. This operation, which is a departure from the above mentioned Inclusiveness Condition, marks some object as 'invisible' at the interface but still accessible within C_{HL}. Chomsky (1995: 280) defines the operation of checking and deletion as in (11) below:

(11a) A checked feature is deleted as possible.
(11b) Deleted α is erased when possible.

A stronger form of deletion is erasure, i.e. eliminating an element entirely so that it is inaccessible to any operation, not just to interpretability at LF. In the logical jargon, especially of the Recursion Theory, to erase means to empty a register on a program counter, i.e. to put a zero (0) in it – cf. Bell and Machover (1977: 233) (see also subsequent discussion). Chomsky's erasure stems from such a formal formulation and has the same flavour to it.

Apart from the above operations, Chomsky distinguishes also the cancellation of a derivation, which is triggered by a mismatch of features (cf. Chomsky 1995: 309, point 108). Importantly, I should distinguish mismatch from nonmatch: thus, the Case feature [accusative] mismatches F′ = [assign nominative], but fails to match F′ = I of a rising infinitival, which assigns no Case. Virtually as well, the cancellation of a derivation under mismatch should be distinguished from nonconvergence. The latter permits

a different convergent derivation to be constructed, if possible. But the point here is literally to bar alternatives. A cancelled derivation therefore falls into the category of convergent derivations in that it blocks any less optimal derivation; mismatch cannot be evaded by violation of Procrastinate or other devices. If the optimal derivation creates a mismatch, it is not permitted to pursue a non-optimal alternative.

Output conditions show that π and λ are differently constituted. Elements interpretable at the A-P interface are not interpretable at C-I, and conversely. At some point, then, the computation splits into two parts, one forming π and the other forming λ. The simplest assumptions, according to Chomsky (1995: 229), are that:

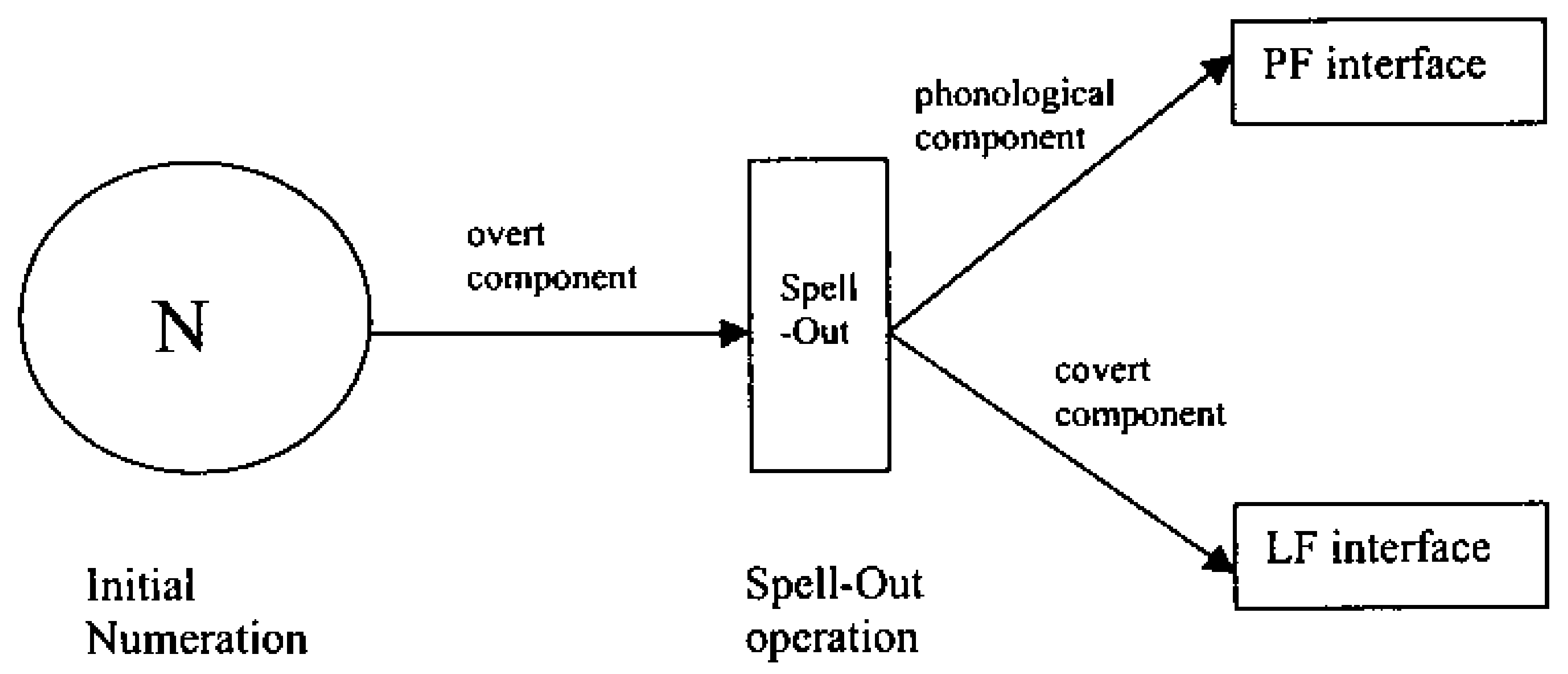
(12a) There is no further interaction between these computations;
(12b) Computational procedures are uniform throughout: any operation can apply at any point.[9]

He adopts (12a) and assumes (12b) for the computation from N → λ, though not for the computation from N → π; the latter modifies structures (including the internal structure of lexical entries) by processes very different from those that take place in N → λ computation. Investigation of output conditions should suffice to establish these asymmetries, which I will simply take for granted here.

Consequently, I assume that at some point in the (uniform) computation to LF, there is an operation Spell-Out that applies to the structure Σ already formed. Spell-Out stripes away from Σ those elements relevant only to π, leaving the residue Σ_L, which is mapped to λ by operations of the kind used to form Σ. Σ itself is then mapped to π by operations unlike those of the N → λ computation. I call (after Chomsky) the subsystem of C_{HL} that maps Σ to π the phonological component, and the subsystem that continues the computation from Σ_L → LF the covert component. The pre-Spell-Out computation I call overt. It may be assumed further that Spell-Out delivers Σ to the module of Morphology, which constructs word-like units that are then subjected to further phonological processes that map it finally to π, and eliminates features no longer relevant to the computation.
The computation path sketched above may be represented as in (13) below:

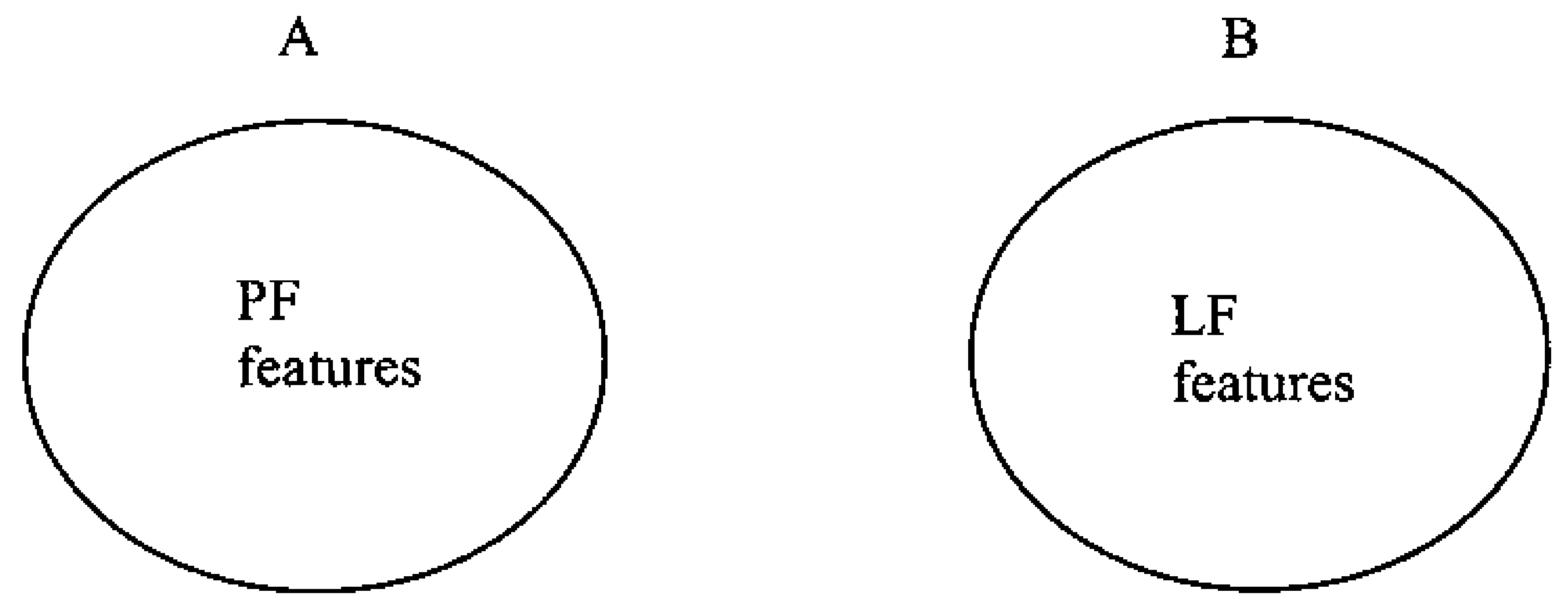
[9] All depends on the procedure, i.e. a program – see subsequent discussion.

(13)



To complete the picture of the structure of LgF, and C_{HL} in particular, I should perhaps finally add that Chomsky distinguishes two types of lexical features: those that receive an interpretation only at A-P interface and those that receive an interpretation only at the C-I interface. He assumes further that these sets are disjoint, given the very special properties of the phonological component and its PF output:

(14)



disjoint sets – $A \cap B = \varnothing$

Now, since I have presented Chomsky's minimalist views on language as a generative procedure, it is time to step back and concentrate on those elements of its description that stem from logic. After a closer look, it is possible to distinguish the following logical notions used metalinguistically:

(15a) Generative procedure (Lg as a derivation-generating procedure)
(15b) Pair (π, λ)
(15c) Set (optimal set, reference set, sets of computations, disjoint set)
(15d) Instructions: input, output
(15e) Derivations = computations/computational system
(15f) Optimal design (economy conditions)

(15g) Operations (Select, Merge, Spell-Out)
(15h) Inclusion (Condition of Inclusiveness, proper inclusion of sets)
(15i) Numeration N
(15j) Mapping (of an array A to the pair (π, λ))
(15k) Array A
(15l) Sequence S (of derivations)
(15m) Stage Σ (union of set, language states)
(15n) Recursive operations
(15o) Proof (used only for comparison)
(15p) Algorithm (for reducing derivational complexity, applying Procrastinate)
(15q) Deletion, erasure, cancelling operations
(15r) Convergence & crashing (of derivations)
(15s) Checking of features
(15t) Representation
(15u) Program (minimalist)

All the above mentioned notions and terms can be traced back in formal logic (logical calculus), mainly in those domains dealing with Recursion Theory, automata, algorithms and Proof Theory (but also in Set Theory and Theory of Relations). In what follows, I am going to describe briefly the above aspects of logical study, with the exception of Set Theory and Theory of Relations which, being of secondary importance for the main topic of this paper and perhaps requiring separate treatment, I take for granted here.

The primary task of Recursion Theory is to characterise and study the class of all algorithmic functions of natural numbers (N_L – non-negative integers). Roughly speaking, a function *f* is algorithmic if there is an algorithm (i.e. a deterministic mechanical procedure) for calculating the value $f(\partial)$ for any ∂ belonging to the domain of *f* (where ∂ stands for an n-tuple $\partial = < a_1, a_2, \dots, a_n >$). Algorithmic functions represent a class of recursive functions. Recursive functions have been singled out of all other functions having their arguments and values in N_L as those that are effectively computable.[10] Effective computation means that there exists a finite description of a function. This description may be regarded as a finite sequence of symbols. Let us order these symbols into a sequence ($a_1, a_2, \dots, a_n, \dots$) that might be infinite. Now, we are able to find unambiguous correspondence between each description (thus, each element of the domain) and some natural number. Such a method of finite sequence coding has been introduced by Gödle and the numbers matched to sequences of symbols are called Gödle's numbers (for details see e.g.: Van Heijenoort 1967).

[10] The fact that all algorithmic functions are recursive is known as Church's Thesis or Church-Turing Thesis (for details see e.g. Hopcroft and Ullman 1994: 192ff).

(Generally) recursive functions $\Re$ are the smallest class of total functions that have unrestricted number of arguments from N_L and have values from N_L, and that fulfil the following conditions:

(16a) Consequent function S, i.e. $S(x) = x+1$, zero function Z, *i.e.* $Z(x) = 0$ for every x, and functions U_n^i, where $U_n^i(x_1, \ldots, x_n) = x_1$ for any n and $i \leq n$, all belong to $\Re$.

(16b) If an n-argument function f and k-argument functions $g_1, g_2, \ldots, g_n$ belong to $\Re$, then the function h obtained by superposition: $h(x_1, \ldots, x_n) = f(g_1(x_1, \ldots, x_k), g_2(x_1, \ldots, x_k), g_n(x_1, \ldots, x_k))$ also belongs to $\Re$.

(16c) If an n-argument function f and (n+2)-argument function g belong to $\Re$, then the function h obtained from f and g by a simple recursion: $h(0, x_1, \ldots, x_n) = f(x_1, \ldots, x_n)$ also belongs to $\Re$.

(16d) If an (n+1)-argument function f belongs to $\Re$ and it is regular, i.e. for any $x_1, \ldots, x_n$ there exists such y that $f(x_1, \ldots, x_n, y) = 0$, then the n-argument function g obtained from f by a minimum operation: $(\min y)\, f(x_1, \ldots, x_n, y) = 0$ also belongs to $\Re$.

cf. Marciszewski (1987: 185)

In other words, a function is recursive, if it can be obtained from initial functions mentioned in (16a) above by a finite number of subsequent applications of operations mentioned in (16b), (16c) and (16d).

As it was already mentioned, Recursion Theory characterises algorithmic functions. By algorithm, logicians (mathematicians, computer scientists) as well as Chomsky generally mean a computation procedure whose application leaves nothing to chance and ingenuity but requires a rigid stepwise mechanical execution of explicitly stated rules.[11] An algorithm is presented as a prescription consisting of a finite number of instructions. It can be applied to any one of a set of possible inputs – each input being a finite sequence of symbolic expressions. Once any particular input has been specified, the instructions dictate a succession of discrete simple operations, requiring no recourse to ingenuity or chance. The first operation is applied to the input and transforms it into a new finite sequence of symbolic expressions. This outcome is, in turn, subjected to a second operation (dictated by the instructions of the algorithm). It may happen that after a finite number of steps the instructions dictate that the process must be discontinued and an output be read off ('spelled out' – in Chomsky's terminology, compare previous pages) in some prescribed way from the outcome of the last step. For some inputs, however, the process may never terminate and there is no last step and hence no output.

Speaking in somewhat more formalised language, I should recall the definition in point (9) (see page 9 above) according to which an algorithm is a finite sequence

[11] See e.g.: Marciszewski (1987), Bell and Machover (1977), Hopcroft and Ullman (1994).

of formulas. I ought to add here that the order of occurrences of given formulas is important. Moreover, the substitutions may be exemplified in the following fashion. Let me take a linguistic example. It can be said that the substitution $s = u \rightarrow v$ (or $s = u \rightarrow \bullet v$) is applicable to a word: $x \in A^*$, if u is a root-word of the word x. Furthermore, a word y is derived from the word x by a substitution s (applicable to x), if y is created from x by a substitution of the first-from-the-left occurrence of the root-word u in the word x by a word v.

If MA is a Markov's algorithm, then for every word $x \in A^*$ it is possible to define, in the following manner, exactly one sequence (finite or infinite) $x_1, \ldots, x_k, \ldots$ of words over the alphabet A – marked as $COMP_{MA}(x)$ (see an analogue to a natural language's generative capacity):

(17a) $x_0 = x$

(17b) Suppose that for some $k \geq 0$, x_k has been specified. Two instances are to be considered:

I. If there exists a substitution applicable to x_k in MA, then x_{k+1} is a word derived from x_k by the first substitution (in a sequence of substitutions forming MA) applicable to x_k.

II. If there exists no substitution applicable to x_k in MA, then x_k is the last element of the sequence $COMP_{MA}(x)$.

cf. Marciszewski (1987: 194)

A partial function (i.e. 1-argument function) $F: A^* \rightarrow A^*$ is called algorithmically computable (see an analogue to Chomsky's derivational procedure), if there exists a normal Markov's algorithm MA over the alphabet A such that for every $x \in A^*$ $F(x)$ is defined iff $COMP_{MA}(x)$ is finite and $F(x)$ is equal with the last element in $COMP_{MA}(x)$ sequence.[12]

For every finite alphabet $A = \{a_1, \ldots, a_k\}$ there exists a mutual unambiguous correspondence, called numeration N, (see an analogue for Chomsky's term) between the set A* of all words over the alphabet A and the set of all non-negative integers. For example:

(18) $N(\varepsilon) = 0;$
$N(a_1) = i;$
$N(a_i, \ldots, a_i) = i_0 + i_1 k + i_n n^k, \qquad 1 \leq i_j \leq k$

It is said that a partial function $f: N_L \rightarrow N_L$ represents a partial function $F: A^* \rightarrow A^*$ in the numeration N, if for every $x \in A^*$:[13]

[12] iff = if and only if.

[13] Recall that N_L stands for non-negative integers – see p. 177.

(19a) F (x) is defined iff f (N (x)) is defined,
(19b) N (F (x)) = f (N (x)).

Taking a more general standpoint, it can be said that a function f is representable in a calculus if the statements of the form '$f(\partial) = b$' (with this particular f) which do have such formal proofs are precisely the correct ones. If the calculus is set up in a suitable way, all functions representable in it will be algorithmic.[14] Now, I would like to recall the definition of a proof, given in point (8) in the context of Chomsky's discussion – see page 9 above. Having observed that syntactic notions, such as claims and derivability are defined in terms of inference rules and can be expressed with the use of some formal calculus (and other formal devices), Chomsky's comparison turns out to be quite apt and intentional. More to that, rules of inference can be applied not only to axioms and theorems, but also to any set of formulas. Let me take Chomsky's sequence $S = (\delta_1, \delta_2, ..., \delta_n)$, (see page 7 above), in short (δ_n), and the pair (π, λ) being the result of subsequent application of inference rules, beginning with (δ_n). Then, we can say that (π, λ) is derivable from (δ_n). This can be written symbolically as:

20) $\delta_n \vdash_{ST} (\pi, \lambda)$[15]

As it was already mentioned, recursive functions (hence algorithmic ones as well) are effectively computable. An important analysis of effective computation was conducted by Turing and Post.[16] They introduced 'ideal' machines, called Turing's machines, which operate in a deterministic fashion, step by step, without errors, and which are not restricted by memory limitations and working time (i.e. they are finite but as capacious as demanded). Needless to say, functions computed by such machines are recursive functions. Some very simple functions can be computed by some ideal machines less complicated than Turing's machines. They are called finite automata and are closely related to the notion of a formal grammar introduced by Chomsky in the 1950s. Formal grammars turned out to be very useful not only for natural language studies but also for studying artificial languages of computer programming. More importantly from the point of view of this work, though, Chomsky's minimalist idea of the computation of derivations proceeds analogously to the above mentioned machines – in a deterministic way selecting freely from the lexicon at any stage.

[14] For this it is sufficient that there should be some effective procedure for enumerating, one by one, all the deductions of the calculus. Then, if f is a representable n-ary function and ∂ is any n-tuple, we can calculate $f(\partial)$ by systematically searching for a formal proof of a statement of the form '$f(\partial) = b$'. This yields an algorithm for f – cf. Bell and Machover (1977).

[15] ST is a system to which the terms are relative. For the form of symbolic script see Marciszewski (1987: 224f).

[16] See e.g.: Post (1936, 1943, 1946), Turing (1936).

Let me take a closer look on such a machine, an ideal, imaginary one – being a Turing's version or its close counterpart – in order to highlight further meta linguistically used words from the chart in (15). I will call my machine a 'computer ALM' – an algorithm-(program, instruction)-computing machine. I assume that the computer has a finite sequence of registers R_i (i = 1, 2, ...). I call the positive number i the address of the ith register R_i. The registers are designed to be storage places for the inputs, outputs and intermediate stages of a computation. I assume that at each moment in time every register stores some natural number. In addition, my ideal computer has a program counter K, which also contains at each moment some natural number.[17] It is said that a register or the counter K is empty when the number stored in it is 0.[18]

I shall assume also that any number, however large, can be stored in each register and in the program counter. From this, it would seem that I require ALM to be able to store an infinite amount of information. However, in fact, I shall always assume that at any moment almost all (i.e. all but a finite number of) the registers are empty. Moreover, each program will only make use of a finite number of registers, whose addresses can be read off directly from the program itself. Thus, I need only an unlimited – but in each case finite – number of registers (see an analogue to a natural language). Even so, the amount of information that can be stored in the computer, albeit finite, is unbounded.[19]

I furthermore assume that the computer can obey four commands as follows:

(21a) *Z commands.* For each positive i, the command Z_i is obeyed by erasing the ith register (i.e. causing R_i to become empty) and adding 1 to the computer program counter K (i.e. causing the number k be replaced by k+1). The other registers remain unchanged.[20]
(21b) *S commands.* For each positive i, the command S_i is obeyed by adding 1 to both R_i and K. The other registers remain unchanged.
(21c) *A commands.* I suppose that ALM has been linked up to a φ-oracle.[21] Let i be a positive number and suppose that at a given moment the number stored in R_i is r. Then the command A_i is obeyed by putting $\varphi(r)$ in

[17] This number will not be a part of a computation but merely a position marker for book-keeping purposes.

[18] The term 'empty' might be traced in Chomsky's Minimalist Program in the context of categories; an empty category may be a metalinguistic analogue of an empty register.

[19] This term occurs in its metalinguistic sense in Chomsky's Binding Theory.

[20] Notice another term 'command' that is taken from logical jargon and employed by Chomsky (c-command, m-command) in his description of phrase structure.

[21] φ-oracle: if φ is any sequence (i.e. a total unary function), then a φ-oracle is an agency able to supply the value φ (r) for each r. It must be stressed that an oracle is not assumed to be a computer – in fact, it cannot be a computer unless φ happens to be algorithmic – cf. Bell and Machover (1977: 233). I do not consider oracles to be a part of my ALM but external to it.

place of r in R_i and adding 1 to K. The other registers remain unchanged. The number $\varphi(r)$ is to be obtained by the computer from the φ-oracle.

(21d) *J commands*. Let i and j be positive and let k be any number. Suppose that at a given moment the numbers stored in R_i and R_j are r_i and r_j. Then if $r_i = r_j$, the command $J_{i,j,k}$ is obeyed by putting k in K instead of the number that was stored there previously. If $r_i \neq r_j$, then $J_{i,j,k}$ is obeyed by adding 1 to K. In either case, all other registers remain unchanged.

Bell and Machover (1977: 234)

All together, I have commands Z_i, S_i, A_i and $J_{i,j,k}$ for all positive i and j and all k.

Importantly for this paper, I should observe the similarity between Chomsky's derivational procedure done by C_{HL} (an analogue of the computer) and the above program. Precisely speaking, the following notions from logical and linguistic descriptions can be identified or at least compared: my ALM ≈ C_{HL}, registers are linguistic items and addresses LIs' indexes, command *Z* bears resemblance to Chomsky's operation Select, command *S* has the function of 'steps-supervising' and might constitute sort of a reference set in the optimisation operations thus playing a role in determining most economical derivations, command *A* can be identified with operation Merge, φ-oracle may be regarded as a set $\Sigma(SO_n)$ being the union of all the elements that appeared in the derivation up to a given stage, and finally command *J* is similar to feature checking.

I have just called the above commands a program. I should be more specific now. By a program I mean any finite sequence of commands:

(22) $\Pr = < C_0, C_1, \ldots, C_{h-1} >$

Bell and Machover (1977: 234)

Here h is the length of Pr, i.e. the number of its commands. For any $k < h$, I call C_k the kth command of Pr. Thus, Pr begins with its 0th command, not with its 1st. The addresses occurring in a program Pr are precisely those positive numbers i and j for which Z_i, S_i, A_i or (for some k) $J_{i,j,k}$ are in Pr.

If I observe here that the functioning of the whole Chomsky's C_{HL} is constructed along the lines of the above logical formulation of a program, which should become quite obvious by now, it is also possible to attribute to the name: 'minimalist program' a profound dimension. One can see that under the surface of that name – given to mean a new direction of development of linguistic studies – there lies a pretty technical, formal layer.

On the previous pages, I have described how Chomsky's linguistic 'program' of LgF wc-ks, i.e. I showed the stages of derivation performed by C_{HL} – a minimalist 'computer' wuhin the brain. Now, it is time to present how its logical counterpart (that was Chomsky's model) operates, making use of the commands from (21). Now

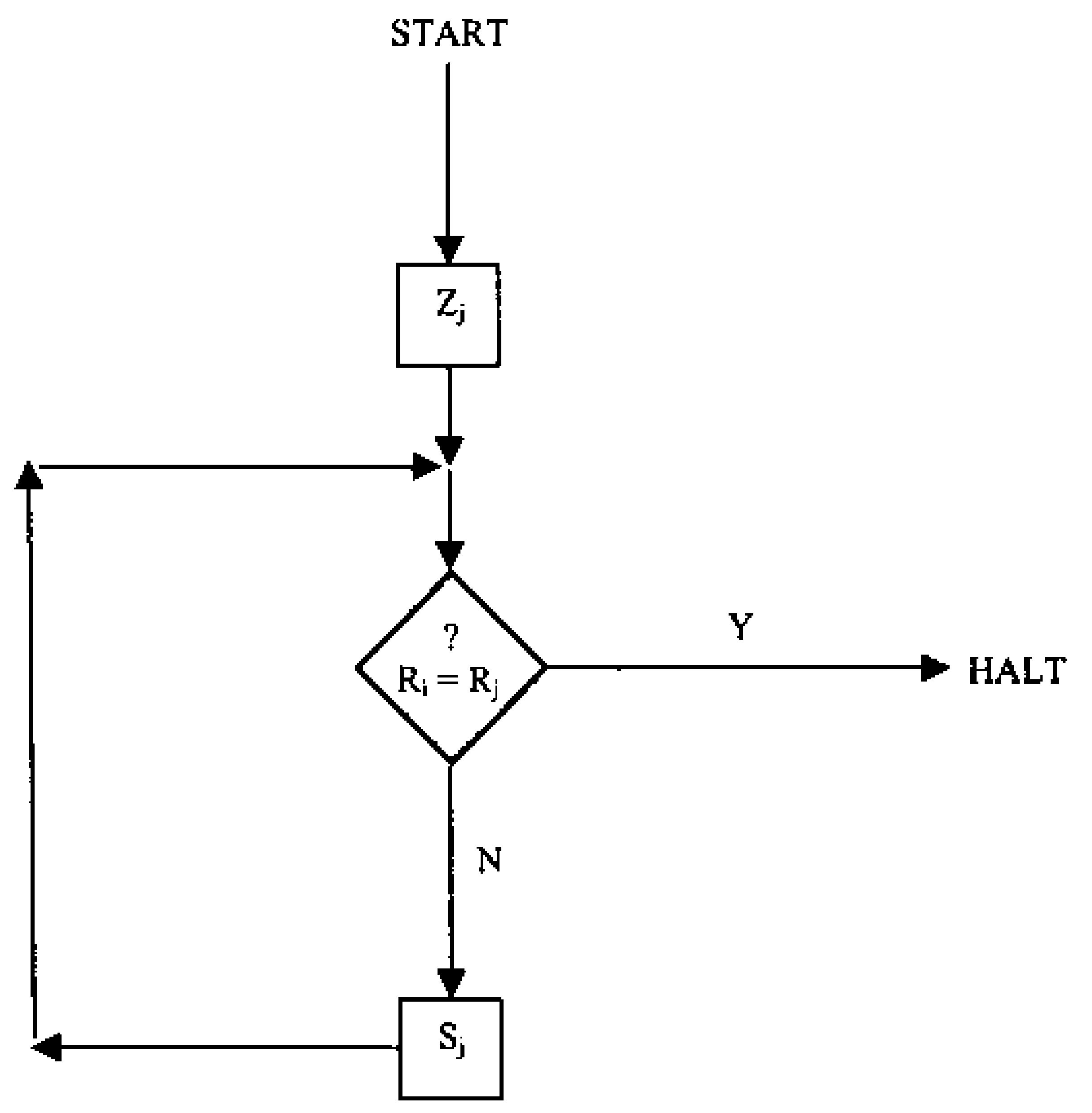
then, I suppose that the computer operates under a given program Pr and that it has been linked up to some oracle. When 'switched on', ALM will go through a finite (or infinite) succession of steps. Each step consists of obeying some command of Pr. Furthermore, at a given moment, the number stored in K is k. If $k <$ length of Pr, the next step will be to obey the kth command of Pr. However, if $k \geq$ length of Pr (so that Pr has no kth command), then the computer halts – it 'switches itself off' and does not perform any more steps.

I shall always assume that initially – i.e. before ALM begins to operate, the program counter K is empty. Thus, the commands of Pr are obeyed one by one in order;

except that when a command $J_{i,j,k}$ is obeyed and the numbers stored in R_i and R_j happen to be equal, then the next command to be obeyed will be the kth command of Pr (or, if $k \geq$ length of Pr, the computer will halt). In other words, if the numbers in R_i and R_j are equal, a command $J_{i,j,k}$ makes ALM jump to the kth command rather than proceed in order.

I shall also assume that initially almost all the registers are empty. It is easy to see that if my computer is operating under the program Pr, the only registers that might affect the process or be affected by it are those whose addresses occur in Pr. Thus, almost all the registers remain empty throughout and play no role at all.

Let me consider the following example (cf. Bell and Machover (1977: 235ff)). Let i and j be positive. I construct a program $\Pr_{i,j}$ whose effect is to copy the number initially stored in R_i into R_j. In linguistic terms the example is to illustrate a simplified version (for the sake of clarity) of a derivation prior to merger of linguistic items and introduction of the φ-oracle. All registers other than R_j are to be left unchanged. In particular, R_i will retain its initial contents. If $i = j$, then the empty program will do the job. Now, let $i \neq j$. I begin by setting up a flow chart which shows how I want the required program to work (see fig. in (23) below).



A diamond-shaped box represents a question (in the present case: 'Are the numbers stored in R_i and R_j equal?' ≈ 'Are the features of LIs selected from the numeration matching?'). Two arrows lead away from the diamond corresponding to the answers 'Yes' and 'No'; they are labelled accordingly 'Y' and 'N'. The rest is self-explanatory.

By following the chart, the reader can see how my program is supposed to work. First, R_j is erased – LIs are selected for derivation. Then, the computer enters a 'loop' and goes round and round as long as the numbers in R_i and R_j remain different – features are matched.[22] Each time round, it adds 1 to R_j – through a reference set optimisation is applied. When the number in R_j becomes equal to that in R_i, ALM gets out of the loop and halts – features are matched; and computation could now go on, had there been some further command in the Pr, e.g. Merge.

Now, I will convert the chart into an actual program. I start at the entrance of the chart (marked 'START') and work out way long, following the arrows. My 0th command is Z_j. Next, the first command, which corresponds to the diamond, must, of course, be of the form $J_{i,\,j,\,\_}$. I leave the third index blank, to be filled later on; this third index will determine the jump that the computer will have to make when the numbers in R_i and R_j are found to be equal.

In the meantime, I follow the arrow labelled 'N', which corresponds to the case where the numbers in R_i and R_j are different, and it does not involve a jump. The next command will thus be S_j. After this, I have to make ALM jump back to the diamond command $J_{i,\,j,\,\_}$, which was the 1st command. I therefore take my next command to be $J_{1,1,1}$: the number in the register R_i will, of course, be found to be equal to itself, and the machine will therefore jump back to the 1st command, as required. (Instead of $J_{1,1,1,Jp,p,1}$ can be used with any positive *p*.)

Now, that I have come back to the diamond, I have all the commands I need. Since the length of my program is 4 (there are 4 commands), I can fill the blank in $J_{i,j,\,\_}$ with 4 (or with any number > 4). Thus, I have constructed the program $Pr_{i,j}$:

(24) $Pr_{i,j} = < Z_j, J_{i,\,j,\,4}, S_j, J_{1,1,1} >$

Bell and Machover (1977: 326)

Finally, I ought to observe that this description of the above machine and the program bears considerable similarity to Chomsky's procedure of determining the proper pair (π, λ). More to that, one can find some resemblance between the flow chart in (23) and the computation path in (13), although this fact may not seem that straightforward at first.

[22] For linguistic counterparts of a loop and halt see note 4.

2. Summary

Summing up this paper on metalinguistic function of logic in Chomsky's minimalist description of LgF and C_{HL}, it has to be said, first of all, that his usage of certain logical notions is unquestionable (which, I hope, I have lucidly shown). Secondly, thanks to employing logic – viewed as a formal system – he enriched his description with additional layer and precision. Finally, and most importantly, he formalised LgF and C_{HL}, i.e. made it possible to frame these abstract entities of mind within the boundaries of logical calculus.

REFERENCES

Allwood, J., L.G. Andersson and O. Dahl, O. 1977. *Logic in linguistics*. Cambridge: CUP.

Bach, E. 1989. *Informal lectures on formal semantics*. Albany, N.Y.: SUNY Press.

Bell, J. and M. Machover, M. 1977. *A course in mathematical logic*. Amsterdam: Elsevier Science Publishers.

Chomsky, N. 1957. *Syntactic structures*. The Hague: Mouton.

Chomsky, N. 1995. *The minimalist program*. Cambridge, Mass.: MIT Press.

Chomsky, N. 1998. *Minimalist inquiries: The framework*. Ms.

Chomsky, N. and G. Miller. 1963. *Introduction to the formal analysis of natural languages*. In Luce, R.D. et al. (eds.). Vol.2. 269-322.

Hopcroft, J.E. and J.D. Ullman. 1994. *Wprowadzenie do teorii automatów, języków i obliczeń*. Warszawa: PWN.

Luce, R.D., R.R. Bush and E. Galanter (eds.). 1963. *Handbook of mathematical psychology*. New York: Wiley.

Marciszewski, W. (ed.). 1987. *Logika formalna*. Warszawa: PWN.

Post, E. 1936. "Finite combinatory processes formulation". *Journal of Symbolic Logic* 1. 103-105.

Post, E. 1943. "Formal reductions of the general combinatorial decision problem". *American Journal of Mathematics* 65. 197-215.

Post, E. 1946. "A variant of a recursively unsolvable problem". *Bulletin of AMS* 52. 264-288.

Quine, W. 1977. *Filozofia logiki*. Warszawa: PWN.

Strawson, F. 1952. *Introduction to logical theory*. London: Methuen.

Turing, A.M. 1936. "On computable numbers with an application to the Entscheidungsproblem". *Proceedings of London Mathematical Society* 42, 2. 230-265.

Van Heijenoort, J. (ed.). 1967. *From Frege to Gödel: A sourcebook in mathematical logic 1879-1931*. Cambridge, Mass.: Harvard University Press.

Wexler, K. and P. Culicover. 1980. *Formal principles of language acquisition*. Cambridge, Mass.: MIT Press.