Pattern matching: a finite-state approach to generation and parsing

László Kálmán and András Kornai
Research Institute of Linguistics, Hungarian Academy of Sciences

0 Introduction

The first section of this paper outlines a novel parsing method characterized by distributed processing and lack of centralized control. The elementary building blocks of the parser are called *molecules* – these correspond, roughly, to ordinary lexical entries and rewrite rules. The individual molecules behave as finite automata, so the overall recognition capacity of the system does not exceed Type-3 languages. The matter of sentence generation is discussed in Section 2. In the last section we argue that the mechanism proposed for this is equivalent to a finite transducer.

1 Parsing

Usually the lexical component of a parser is but a repository of phonological, morphological, syntactic, and semantic information, and apart from a trivial alphabetic ordering, it has no internal structure whatsoever. This solution to the problem of storing information relevant to the process of parsing is rejected by many people working in the 'semantic net' paradigm. The model to be outlined below owes a lot to the early proponents of that approach, in particular (Quillian1967), but is extremely impoverished compared to later semantic net representations such as KRL. In addition to storing various chunks of information, each molecule acts as a processor of limited capacity. Molecules are linked to other molecules in a unidirectional manner. The flow of information along these links is not regulated by any supervisor program: depending on its own internal state, each molecule can initiate communication, and can receive messages. The internal state of the molecules is the function of their specific information content (or SPINFO), the message received, and their previous state.

Each molecule is divided into three storage areas. SPINFO, the first of these, stores permanent or semi-permanent information that is seldom updated. TEMP, the next one, stores information only for one cycle. The last one, TEMP2, acts essentially as a buffer for TEMP: its content is either processed immediately or lost forever. Messages arrive to TEMP2 (in case of conflict, the last one prevails). Apart from sending messages to neighbors (i.e. dumping the contents of TEMP or SPINFO to the linked molecules), the fundamental operation a molecule can perform is taking the union of two graphs. Both SPINFOs and messages are graphs:

nodes are labelled by the name of one of the molecules or by a special symbol '*', and edges are directed but unlabelled. There can be special (undirected) edges that signify equivalence, and one of the nodes is distinguished – more about these later.

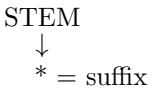
Communication between the molecules is regulated by a protocol based on a distinction between *top level* and other nodes in the message. Those nodes that are at the endpoint of arrows running from to the distinguished node of a message constitute its top level. If the name of the receiving molecule does not appear on the top level of the message received, the molecule stores it on its TEMP, but does not send it any further. The top level lists the intended addressees of the message, therefore if we want to send a message to a particular molecule which is not directly linked to the sender, we have to mention in the message (the name of) every intermediary molecule. The action a molecule takes might depend on its internal state (e.g. activated, learning, inhibited, etc. states) as well: for details see (Kornai1987).

Union of graphs is defined in the ordinary manner; '*' acts as a variable to be substituted by the appropriate subgraph (preserving edges). No two nodes can have the same label in any graph, so identically labelled nodes have to be conflated. The special edges act as conditions on the well-formedness of the graph: unless the nodes connected by such edges are labelled identically, the graph is ill-formed. By combining the special edges with the variable symbol, selective substitution becomes possible. Successful substitution, i.e. the matching of any message that contains no stars in the starred positions of the SPINFO, will render the molecule in question inactive for a full cycle.

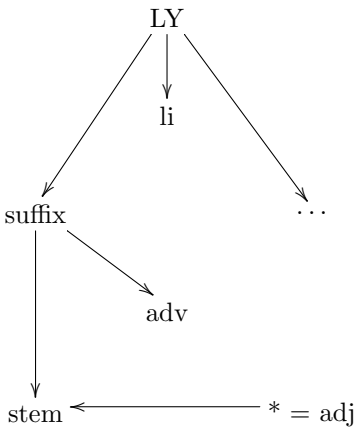
The SPINFO of a molecule stores paths that lead to other molecules: in effect these constitute the only association between molecules that are not immediately linked. Updating a SPINFO is only possible if the molecule is in the learning state. The message a molecule sends is usually the union of its SPINFO with its TEMP, and the distinguished node will be the name of the sending molecule.

The basic linguistic properties stored in the SPINFO of a molecule corresponding to a lexical entry include its spelling or phonological representation, and categorial information expressing its primary (unmarked) affiliations. This entails that lexical categories are also molecules, containing no phonological form or spelling, but rather primary syntactic functions i.e. names of constituents in which they might occur. The SPINFO of the molecules representing syntactic constituents contains semantic, syntactic, and pragmatic information about the use of the given constituent. Moreover, their SPINFO is *pattern-like,* i.e. it contains empty slots. These slots are represented by variables in the graph; since a message containing such variables on the top level triggers a substitution process in the receiving molecule, it acts as a *request.* The filling in of empty slots may be subject to constraints on linear order. One of the typical cases is the requirement of *absolute adjacency:* for example, a stem has to be immediately adjacent to an ending, a prefix has to be followed immediately by the word to which it is attached, the English verb immediately precedes its direct object, etc. This corresponds to messages that disappear if their top-level variable is not substituted immediately. For instance, let us derive *quickly* from *quick* and *ly.* In the SPINFO of the molecule corresponding to *quick,*

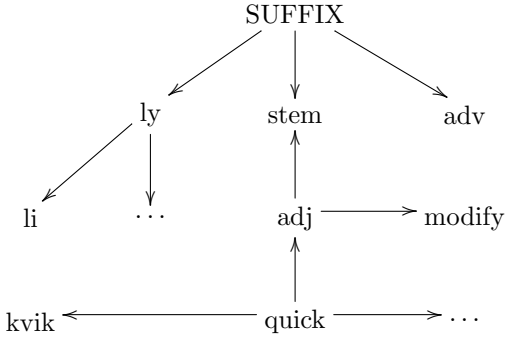
the name of the molecule ADJ(ective) is on top level, which means that *quick* is an adjective. Therefore, the SPINFO of *quick* is sent as a message to ADJ as soon as *quick* is activated. The SPINFO of ADJ contains both STEM and MODIFY on top level, since these are the main functions of adjectives in larger constructions; this SPINFO is unified with the message from *quick* and the result is sent to both MODIFY and STEM. The SPINFO of STEM looks as follows:



This is unified with the message from ADJ, and the result is sent to SUFFIX which, in turn, interprets it as a request. If SUFFIX finds nothing on its TEMP that can be unified with the request, the process will halt because the request disappears. In this case the only chance for *quick* to surface is through the molecule MODIFY. But if the message sent by e.g. LY arrives on time, communication can proceed. The SPINFO of *ly,* namely

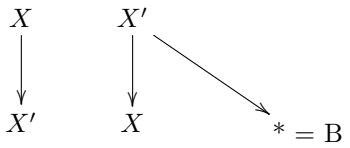


contains the information that *ly* is a deadjectival suffix forming adverbials, that its phonological representation is /li/, etc. This arrives to SUFFIX, which contains no relevant information, and therefore stores the message from *ly* in its original form roughly, with the only difference that the distinguished node will be SUFFIX. When the request from STEM arrives, matching can take place, and the result is



This, then, is sent to the molecule ADV(erb) for further processing. It should be noted that this message will not arrive to *ly,* but STEM will receive a confirmatory message which inhibits it for a cycle.

The second type of information about linear order that is supposed to be encoded in molecules is *precedence.* The difference between precedence and strict adjacency is that in the case of precedence the request message does not disappear if it is not satisfied immediately. This storing of messages is possible if (at least) two molecules work in close collaboration (i.e. there are bidirectional links between them), and are able to send messages to each other *ad infinitum.* Such a ping-pong effect will be stopped by one of the molecules' getting inhibited. That is, for any linear precedence statement $A < B$, the model employs two interlinked molecules X and X' having SPINFOs of the following form:

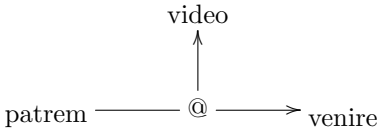


Additionally, A has a link to X, and B has a link to X'. In such cases, the message activating A will be enriched by the SPINFO of A and sent to X. X, in turn, will enrich it with its own SPINFO, and send it to X'. X' will send a request to B and to X as well, and it is this latter message that will come back to X' in the next cycle. This way, information will be requested from B at every turn, and this will go on until X' gets inhibited by a confirmatory message from B.

The third type of linear ordering is *free word order.* This is expressed by patterns containing several variables on the top level. This way, simultaneous requests are sent to several molecules.

2 Generation

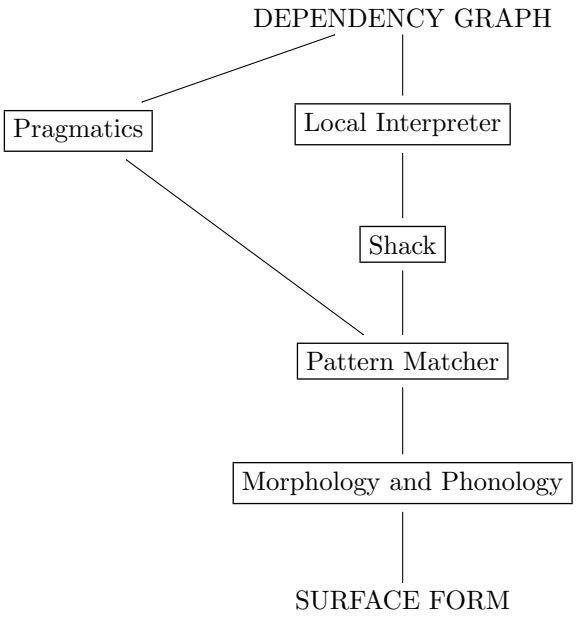
The starting point of a derivation is the semantic representation which is made up from the SPINFOs of the contentive items. The edges run from dependents to heads: these are usually (names of) molecules. In addition to these, the graph might have "ideal" vertices (here depicted as @-signs), which do not correspond to lexical items: such vertices provide a starting point or an endpoint for arrows that run from (or to) edges:[1]



[1] This formalism has strong conceptual links with the ideas of dependency grammars (Tesniére1959) and relational grammar (see (Perlmutter1980), and the references cited therein): for a rigorous definition, see (Kornai1987).

This picture is inaccurate in one very important respect: the vertices should be lexical stems rather than fully formed words. We shall return to this question below.

The main task of our generator is to encode such graphs in linear sequences (surface strings). This linearization is an essentially local process: in any given moment, only one edge of the graph is under scan. If this edge contains an ideal vertex, the situation is somewhat more complex: see below. One component of the grammar is the *Local Interpreter* (LI) which encodes the relationship expressed by the edge under scan in the manner appropriate for the language in question. For instance, the possessive relationship is encoded as morphological marking on the possessed element in Hungarian, and as morphological marking on the possessor in English. But in English, it is also possible to encode this relationship with a grammatical formative *of* that has to precede the possessor and follow the possessed item. In general, LI can specify elements for morphological marking, linear order and nearness, but can also leave these unspecified. This is very similar to the behavior of the molecules discussed in Section 1: the step-by-step action of the LI is intended to capture the spreading activation of the various molecules.

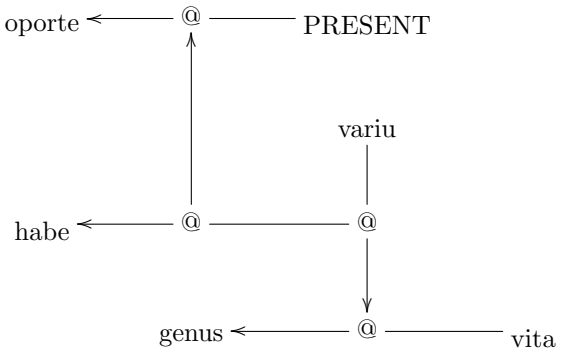


The results of the local interpretation are stored in a special short-term memory called the *Shack,* which will be discussed later. The most important step in the linearization process is called *pattern matching.* This is performed by a component operating on the contents of the shack. The *Pattern Matcher* (PM) drains the shack whenever possible; in general, LI, PM, and the Shack operate concurrently. At present, the patterns (templates) are stored in the long-term memory of the PM; in the future, we plan to divide them among the SPINFOs of the molecules constituting it, and to replace the matching operation by graph unification. The actual choice of patterns is determined by the content of the shack and by pragmatic information.

(In principle, postlexical suprasegmental information is also part of the patterns – in practice, neither this nor a proper handling of pragmatics is implemented.)

In general, pattern matching happens in a local manner much as the context-free rewriting of a node in a phrase structure tree. Matching is left to right (without backtracking): this means that if adjacency constraints and constituency information are incompatible, i.e. in the case of discontinuities, then the surface template takes over and no phrase structure rule is applied. The resulting forms are fully specified for morphological and phonological features and thus can drive the phonological component of the grammar.

We shall illustrate the linearization process with the derivation of the Latin sentence *oportet varium habere vitae genus* from the underlying representation



It should be mentioned here that e.g. *genus vitae* could well be a single word in another language. In this respect the same meaning can correspond to different dependency graphs in different languages.

The Local Interpreter first goes to the head of the whole construction, i.e. the vertex that has out-degree zero. This vertex can be reached by following the edges between normal vertices and switching to the out-arrow at ideal ones. In our example, the head is *oporte.* The aim is to interpret every edge in a contiguous manner if possible, with vertices with smaller degree having priority. Thus, *oporte* ← PRESENT has to be interpreted first. In this case, local interpretation means the addition of the feature ⟨TENSE PRES⟩ to the stem *oporte,* and the result is stored in the shack. The following edge is the one pointing to *oporte* ← PRESENT. This arrow, however, both originates in and points to an ideal vertex and therefore cannot be interpreted directly. For arrows starting in ideal vertices, we apply the *Principle of Chain Forming* (PCF), which shifts the starting point of the arrow in question to the farthest "descendant" of the ideal vertex. In other words, the PCF moves the tail of an arrow along as many arrows as possible until it reaches a normal vertex. This vertex will be *habe* in our example.

If an arrow points to an ideal vertex, this means that its starting point is a dependent of a whole construction *(oporte* ← PRESENT in our case). In such cases the agreement resulting from local interpretation usually holds between the dependent *(habe...)* and the head of the construction *(oporte).* Thus the LI interprets an edge between *habe* and *oporte* (the farthest descendant of the ideal vertex) as an *agreement pipe* : the infinitive form of *habe* has to agree with *oporte* in tense. *Habe* ⟨INF

TENSE *A⟩ and *oporte* ⟨TENSE *A⟩ are stored in the shack, linked together by the pipe. Since the LI interprets edges rather than vertices, it puts certain elements in the shack more than once. But items in the shack are unique, much as nodes in SPINFOs and messages.

In addition to pipes, the elements in the shack can also be linked by nearness-marking pointers, which we shall call *bands.* (Bands are probably immaterial in Latin and in 'free word order' languages.) In the unmarked case, edges between normal vertices are interpreted as pipes and bands at the same time. The same holds for arrows starting from an ideal vertex, with the only difference that the starting point of these shifts to the farthest descendant of the ideal vertex, as required by the PCF.

In the case of arrows pointing to an ideal vertex, the situation is somewhat more complex: the head of the head-construction is linked to the starting point of the arrow by an agreement pipe, whereas the nearness pointer runs between the starting point and the farthest ancestor of the ideal vertex. In our example, the arrow pointing to *oporte* ← PRESENT would be interpreted as an agreement pipe between *habe* and *oporte* plus a nearness band between *habe* and PRESENT. (Of course, this is only relevant if *oporte* and PRESENT are encoded as separate words in the given language.)

The arrow pointing to *habe* originates in an ideal vertex, therefore its starting point has to move to its farthest descendant, which happens to be *genus.* The edge *habe* ← *genus* is interpreted as *habe* + *genus* ⟨CASE ACC⟩, and the result is stored in the shack. The next edge is *genus* ← *vita,* the proper encoding being *genus* + *vita* ⟨CASE GEN⟩. In the next step *genus* ← *variu* gets interpreted as a pipe: *variu* ⟨GENDER *A PLURAL *B CASE *C⟩; *genus* ⟨GENDER *A PLURAL *B CASE *C⟩. By virtue of the PCF, a nearness band between *variu* and *vita* might also be established. The shack differs from an ordinary stack memory in many important respects. First of all, the stack is finite: we suppose that at any given moment at most five elements can be stored in it. Secondly, the shack is unordered (random access). The items in the shack can be identified immediately. This means, as we said above, that no element can be stored in more than one copy in the shack. In addition to providing convenient storage area, the shack also performs certain operations on its content. Information concerning the gender, number etc. of elements can flow along the pipes, and in the case of syntactic concord this is obligatory. We shall say that the shack is well-formed whenever these operations have all been carried out. We suppose that the PM always operates on a well-formed shack.

In general, items do not remain in the shack for long. In any given moment the PM scans the contents of the shack, and whenever it finds an item which is consistent with the leftmost slot of the template, matching takes place. If an item is matched, it is deactivated in the dependency graph but can remain in the shack. In fact, it must remain in the shack as long as some of its neighbors in the dependency graph are active but do not appear in the shack. This means that the shack contains every dependency information between those parts of the sentence which have already been spoken and those parts which have not. In particular, we suppose that items that have been matched are inaccessible unless they are still present in the shack.

In our example, the template is (VP →) V⟨AUX⟩ S. As soon as *oporte* ⟨TENSE PRES⟩ appears in the shack, matching can take place. But as all other vertices (excepting PRESENT) are still active, *oporte* ⟨TENSE PRES⟩ must remain in the shack. The next appearing element is *habe* ⟨INF⟩ which has to agree with *oporte* in tense and thus gets fully specified: *habe* ⟨INF TENSE PRES⟩. Now *oporte* ⟨TENSE PRES⟩ can disappear from the shack.

In the expansion of S the surface pattern Adj V NP is chosen. Since the leftmost slot in this template cannot be filled in by anything in the shack, matching must be delayed.[2] The next element to appear in the shack is *genus* ⟨CASE ACC⟩, which does not change the situation. Then *vita* ⟨CASE GEN⟩ appears in the shack, but the PM has to wait until *variu* arrives.

The well-formedness of the shack requires the gender, number and case features of *genus* to be piped to *variu*. After that, *variu* ⟨GENDER NEUTR PLURAL - CASE ACC⟩ can be matched. The next slot in the pattern can be filled in by *habe* ⟨INF TENSE PRES⟩. Again, for some reason or other, we chose the pattern (NP →) NP⟨CASE GEN⟩ NP rather than NP NP⟨CASE GEN⟩, and we match *vita* ⟨CASE GEN⟩. In the last step, *genus* ⟨CASE ACC⟩ is matched and the shack empties because there are no active vertices in the graph.

3 Conclusions

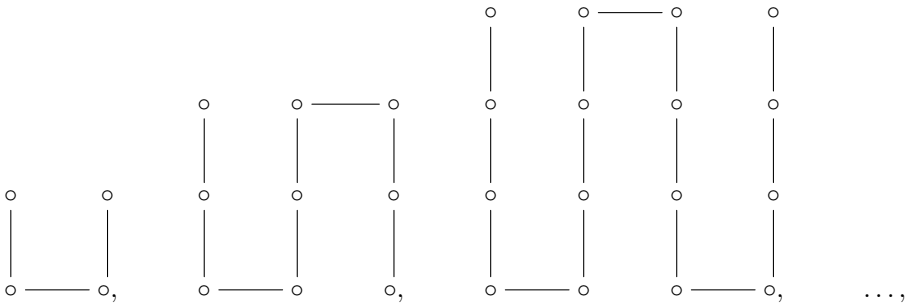
It would not be proper to say that our model 'explains' the well-known performance restrictions of human speakers, because certain parts of our grammar, most notably, the shack, were expressly designed to model these very restrictions. Nevertheless, it should be mentioned that our model gives the right predictions not only for center-embedded constructions with large depth, but also for purely left-branching constructions. Examples like *John's stepmother's father's uncle* were problematic for the classic model of (Yngve1961), because their depth can be arbitrarily large without affecting their acceptability. In our model such examples can be generated only with the aid of pre-planned templates: this seems to explain their rareness in everyday speech. Since in parsing, the current possessor need not remain in the TEMP of the POSSESSION molecule for long, the short-term memory of the hearer is not overburdened, and the construction is acceptable.

Thus, we have a three-way classification of syntactic constructions: purely right-branching constructions are easy both to produce and to understand, purely left-branching ones with large depth are easy to understand, but hard to produce, and mixed ones with large depth are impossible to produce or to understand. Logically, there is a fourth possibility, namely, constructions that are easy to produce but hard to understand. Our model cannot describe these and thus predicts that no such construction will ever be found in natural languages.

The idea of generating surface strings from semantic networks as underlying

[2] It appears that in general slot-filling is delayed until the next element appears in the shack. Thus ordinary derivations seem to involve a certain (strictly limited) amount of preplanning.

structures is certainly not new in computational linguistics (see e.g. (Shapiro1982) and the references quoted therein), but in the generative process usually ATNs or other equally powerful mechanisms are used. Since the present model employs pipes, bands, and a number of other devices, it is far from obvious that its generative power is any smaller. For instance, if the well-formed underlying representations are



and this cannot be ruled out without constraining the semantic component in some manner, then a trivial (identity) rule of local interpretation, a shack with no pipes, a single (regular) phrase-structure template S → aS, and a trivial (band-free) PM component will generate the language $\{a^{n^2} | n > 1\}$, which is properly context-sensitive.

Since it is impossible to prove the finite-stateness of our model rigorously without a well-defined notion of semantically correct dependency graph, pragmatic rules, etc., the only expedient is to present the broad outlines of such a proof. (Naturally, the details will have to be spelled out eventually.) The above example makes it clear that at present we cannot prove the whole grammar to be finite-state: the only thing we can reasonably claim is that the syntactic component behaves as a finite transducer. If dependency graphs turn out to be regular[3] and if the pragmatic component can be shown to be well-behaved, then the regularity of the stringsets generated by our model will follow as a direct consequence of the classic theorem asserting that the GSM image of a regular language is regular. Here GSM stands for Generalized Sequential Machine (finite transducer): these can be defined with an *input alphabet* V_i, an *output alphabet* V_o, a finite set S of *internal states* containing a unique *initial state* s (from S), and a set S_f contained in S of *final states.* The GSM scans an input string from left to right; at any given point it will emit a string of V_o^* depending only on the input letter under scan and on its current internal state, and move (possibly non-deterministically) to another internal state. The GSM image of an input string is simply the concatenation of the strings emitted in the process, and the GSM image of a language L is defined to be the set containing the images of the strings in L (and only these).[4] We do not want to establish the desired result by appealing to the finiteness of the lexicon and/or to the length limits of actual

[3] This presupposes some definition of regularity for infinite sets of dependency graphs – obviously, we want to call every finite set of such graphs a regular set.

[4] For a more rigorous definition, and for the proof of the theorem mentioned above, see e.g. (Salomaa1973) p30ff or (Harrison1978) p206ff.

sentences. To the contrary, we think that no arbitrary bounds should be put on sentence length and that the long-term memory of humans is so vast that it is a reasonable idealization to take it to be *infinite*. In addition to this, we cannot constrain the rules of local interpretation in any principled manner. Therefore, the starting point of our argumentation has to be the shack: at any rate, the regularity of our generation model must be the result of the finiteness restriction on the shack. The regularity of the parser follows from a classical theorem of (Kleene1956).

The pattern matcher operates on the contents of the shack, but it is not the individual items themselves but rather their lexical categories that determine its behavior. Items with different inflectional markers or with different subcategorization frames must be considered different, since these matters can (and do) affect the choice of pattern. Since the number of lexical (sub)categories is finite (probably less than 10^3) in any natural language, and the number of paradigmatic forms a word can have is also strictly limited (by, say, 10^4), at most 10^7 essentially different forms can appear in one position in the shack (probably 10^5 would be more realistic here).

Since the shack is unordered, there are at most $(10^7)^5 = 10^{35}$ ways to fill it completely. If we include a blank symbol with the 10^5 forms mentioned above, and take it into account that the shack hardly ever contains more than 3 items, we get $(10^5)^3 = 10^{15}$ as a 'more realistic' estimate on the number of possible configurations in the shack. But no matter which estimation we use, only a fraction of these configurations will be well-formed: we think that the number of well-formed shack configurations is actually less than 10^6.

Now, the input alphabet of the GSM modelling the pattern matcher can be identified with the set of these configurations. (It should be kept in mind that the input language over this alphabet will not be regular for 'unnatural' languages, like the 'square' language above.) The output alphabet of the PM contains the actually filled slots, which correspond to terminals fully specified for subcategorization and paradigmatic form. The experience with context-free models of natural languages shows the number of such terminals to be less than 10^5, in accordance with our 'realistic' estimation above.[5]

The states of the GSM correspond to the patterns that can be used to fill in the template: if we take the PM to be a *non-deterministic* GSM, the number of its states will equal the number of phrase structure rules plus the number of surface patterns necessitated by discontinuities. Thus, the number of states need not exceed 10^3 for any language; again, experience with context-free models shows that quite detailed grammars can be written with a few hundred CF rule schemata.

Since deterministic and non-deterministic finite transducers are equivalent, our 'proof' is complete. But on real-life computers, truly non-deterministic computation is impossible to implement, and it should be kept in mind that in the actual

[5] This number cannot be reduced if the verbs of the language in question have cca. 10^3 paradigmatic forms (as is the case with Sanskrit and many agglutinative languages), and the number of verbal subcategories is cca. 10^2 (which seems to be universally true).

model the 'non-deterministic' choices of the PM are determined (or at least probabilistically governed) by the pragmatics component.

References

Michael A. Harrison. 1978. *Introduction to Formal Language Theory.* Addison-Wesley.

Stephen C. Kleene. 1956. Representation of events in nerve nets and finite automata. In C. Shannon and J. McCarthy, editors, *Automata Studies*, pages 3–41. Princeton University Press.

András Kornai. 1987. Finite state semantics. In U. Klenk, P. Scherber, and M. Thaller, editors, *Computerlinguistik und philologische Datenverarbeitung*, pages 59–70. Georg Olms, Hildesheim.

David M. Perlmutter. 1980. Relational grammar. In Wirth and Moravcsik, editors, *Current approaches to syntax*, pages 195–229. Academic Press.

M. Ross Quillian. 1967. Semantic memory. In Minsky, editor, *Semantic information processing*, pages 227–270. MIT Press, Cambridge.

Arto Salomaa. 1973. *Formal Languages.* Academic Press.

S.C. Shapiro. 1982. Generalized augmented transition network grammars for generation from semantic networks. *AJCL*, 8:12–26.

Lucien Tesniére. 1959. *E1lements de syntaxe structurale.* Klincksieck, Paris.

Victor H. Yngve. 1961. The depth hypothesis. In R. Jakobson, editor, *Structure of Language and its Mathematical Aspects*, pages 130–138. American Mathematical Society, Providence, RI.