Vectorized Finite State Automata

András Kornai
IBM Almaden Research Center
650 Harry Road, San Jose, CA 95120
kornai@almaden.ibm.com

Abstract. We present a technique of finite state parsing based on vectorization and describe the application of this technique to a well-known problem of natural language processing, that of extracting *relational information* from English text. We define Vectorized Finite State Automata, the theoretical model behind the applied system, and discuss their significance.

0 Introduction

One of the persistent problems in building finite automata on the large scale required by actual applications is that the product and powerset constructions routinely used to implement intersection and nondeterminism can, in a few steps, increase the size of the state space beyond reasonable bounds. This paper will describe how to avoid this problem by structuring the state space as a (generalized) finite vector space. Section 1 of the paper introduces the problem by means of a highly artificial but simple example, informally presents the basic idea of Vectorized Finite State Automata (VFSA), and outlines the VFSA solution for this particular problem. Section 2 presents an overview of NewsMonitor, a system extracting relational information from English text, with particular emphasis on the VFSA pattern matching engine around which NewsMonitor is built. Section 3 provides the formal definition of VFSA, discusses their properties, and compares them to Register Vector Grammars (RVGs) [3]. The theoretical implications of the work are discussed in Section 4.

Broadly speaking, there are three ways vectorization can enter the standard setup for finite state language modeling. First, the alphabet itself can be composed of n-tuples, a conceptualization particularly useful for n-ary regular relations and n-way finite automata [9]. Second, the alphabet symbols in a single dimension can be thought of as vectors composed of (binary) features, as is commonly done in Prague-style and in generative phonology [4],[5]. Third, the state space itself can be conceptualized as a vector space, as in RVGs. In this paper we explore this third possibility in the VFSA framework that also encompasses what we take to be the crucial aspects of the first and the second kinds of vectorization.

One way to specify an n-ary regular relation is by an n-way finite state transducer (FST) defined by a finite set S of states, some designated as initial/final, and a finite list L of arcs, where each arc carries an n-tuple of symbols. Other than providing convenient labels for beginning and endpoints of arcs, in practice states play so little role that in tight implementations they are often entirely omitted for the sake of effective storage. Here we will consider the possibility of endowing the state space with a hypercube-like structure by means of a set of (boolean valued) functions called *flags* or *features*. (The term 'flag' is more common in syntax and computer science, the term 'feature' is more common in phonology – here we use them interchangeably.) In phonology, the goal of such an embedding or *feature decomposition* is to achieve a high degree of notational compactness [11]. Here we show that notational compactness is more than an abstract goal, inasmuch as feature decomposition, properly implemented, can lead to appreciable savings in memory requirements. But if tightly organized FSTs do not even mention state information, what difference can encoding the states as feature vectors make? The answer is that with appropriately selected features not only the states, but also the arcs can be omitted from the representation.

1 The problem

As any practicing computational linguist knows, writing and debugging grammars is an expensive, time-consuming process. With (augmented) context-free grammars, as more and more effort is extended in this direction and the coverage of the grammar improves, the runtime performance of grammatical engines becomes worse and worse. While this problem is not unknown in other areas of AI, nowhere is it as acute as in natural language processing. One way out is to strictly limit runtime computation to the basic operation of following a prearranged chain of pointers along the direction dictated by the input. This requires creating all possible chains at compile time, and collecting them in one large state machine which, if memory poses no constraints, can be implemented in a fast and efficient manner by each state pointing to the transition function out of that state, and each transition function returning a pointer to the next state.

Since memory is usually a scarce resource, sophisticated implementations permit runtime operations slightly more complex than dereferencing, e.g. computing byte offsets, in return for significantly more compact storage of the state machine. But, in the end, all efforts to precompute all paths and store them individually are doomed by the ever-increasing complexity of the paths that must be considered as the coverage of the grammar increases. This leaves no alternative but to trade in more runtime complexity in return for space savings. The vectorization method described in this paper relies on bit-

Proceedings of the ECAI 96 Workshop
Extended Finite State Models of Language
Edited by A. Kornai.

field operations that are either directly available as machine instructions or can be built from a few machine instructions, thereby making the system inherently fast.

As an introduction to VFSA in this Section we discuss a simple problem, that of a 32-bit binary incrementer. A brute force statement of the incrementer as a 2-way relation, with 2^{32} disjuncts, would lead to a transducer of unmanageable complexity. What makes this example particularly hard to express by the standard mechanism is its lack of locality: because input is scanned left to right but the carry moves right to left, we need to delay output of the first result bit possibly until after the last input bit is received. Before minimization, we must therefore assign a separate state to each bit-combination, yielding 2^{32} states and the same number of arcs. Needless to say, much more compact solutions are possible, but the size of the unminimized transducer is so overwhelming that using it as a basis for minimization is out of the question. In contrast, vectorization will provide a solution with trivial space requirements.

The informal definition of a Vectorized Finite State Automaton (VFSA) begins with two finite sets of variables $x_0, \ldots, x_k$ and $y_0, \ldots, y_l$ called *internal* and *external* variables respectively. For now we assume these to be boolean variables, the general case will be discussed in Section 3. A full binding of the internal variables is called a *state*: those states that have $x_0 = 1$ are called initial, those that have $x_k = 1$ are called final. A partial binding of the external variables is called a *letter* of the alphabet. In general, the distinction between internal and external variables concerns only the manner in which new input is read, and we will use the term *symbol* to mean partial bindings of the full set of variables.

The *unification* aUb of two symbols a and b is defined in the usual way as the smallest partial binding that extends both a and b. Since unification, a symmetric operation, will fail if a and b are incompatible, we also define *overwriting* a by b, aOb, an asymmetric operation that never fails. If x is free in a and free in b, it remains free in aOb. If x was free in one of a or b it takes the same value as it would in aUb. But if x was bound to different values in a and in b, the latter value prevails. We say that an arc from state u to state v is *sanctioned* by a Pattern-Action (PA) statement $(p; a)$ if uUp exists and $v = uOa$. Transition along a sanctioned arc consumes the external portion of the pattern symbol, but maintains (an extension of) the action symbol as a record of the current state(s) of the VFSA. By replacing pattern and action symbols in the definition by n-tuples we obtain Vectorized Finite State Relations (VFSRs), but this offers no further generality because vectors of vectors in this case are simply longer vectors (or matrices) having fixed dimension.

To model the incrementer we need 67 internal features $x_0, \ldots, x_{66}$, with x_{2i-1} storing the the i-th position of the input and x_{2i} monitoring progress, $(1 \leq i \leq 32)$. A single external feature y serves as the feature encoding of the current bit. We first initialize the progress flags to 1 by the PA statement $(x_0 = 1; x_0 = 0, x_{65} = 0, x_{66} = 0, x_2 = 1, \ldots, x_{64} = 1)$. Note that this statement consumes no input and thus could fire at any time. However, it requires an initial state $(x_0 = 1)$ and produces a non-initial state $(x_0 = 0)$ and will therefore fire only once, at the beginning. To consume the input we have 32 PA statements of the form $(x_{2i} = 1, x_{65} = 0; x_{2i-1} = y, x_{2i} = 0)$. By definition, PA rules in a VFSA are scanned in linear order until a match is found (at which point the rule fires, new input is read, and the rule search restarts at the first rule) so in principle any of these 32 rules could fire at any time.

In this particular example, progress flags are employed just to make sure that the ith rule can only apply to the ith bit of input. Once the input is consumed, we set x_{65} to 1 by the rule $(x_0 = 0, x_2 = 0, \ldots, x_{64} = 0; x_{64} = 1; x_{65} = 1)$. Starting with x_{64} we reuse the progress flags to monitor the carry. We need 31 PA statements of the form $(x_{2i-2} = 0, x_{2i-1} = 1, x_{2i} = 1, x_{65} = 1; x_{2i-2} = 1, x_{2i-1} = 0)$ to preserve, and another 31 of the form $(x_{2i-1} = 0, x_{2i} = 0, x_{65} = 1; x_{2i-1} = 1, x_{2i} = 1, x_{65} = 0)$ to absorb the carry. Unless we are prepared to reuse the x_0 flag, ordinarily reserved for initial states, the rules affecting the first position needs to be stated separately in each set: for carry we have $(x_0 = 0, x_1 = 1, x_2 = 1, x_{65} = 1; x_1 = 0, x_{66} = 1)$ and $(x_0 = 0, x_1 = 0, x_2 = 1, x_{65} = 1; x_1 = 1, x_{66} = 1)$ and for the case when the carry was absorbed earlier we have $(x_0 = 0, x_2 = 1, x_{65} = 0; x_{66} = 1)$.

Altogether, a set of about a hundred PA rules, each requiring less than a hundred bytes of storage, plus the fixed overhead of the rule interpreter, is all that is required to store and execute the incrementer. Furthermore, the approach taken here smoothly extends to 32-bit (or even longer) adders. A carefully minimized and compressed 32-bit incrementer could possibly be loaded in the gigabyte memory we can reasonably assume for a large computer today. But a 2-way finite automaton with two successive 32-long bitstrings on the upper tape and a single 32-long bitstring, the sum, on the lower tape, would be hard to fit in, no matter how carefully minimized or compressed, while a VFSA performing the same function is still very small.

Though binary incrementers and adders give a good intuitive idea of the amount of space savings made possible by vectorization, they are less than fully convincing in a natural language context, where there is no *a priori* guarantee that a good feature analysis can be found. Binary integers lend themselves naturally to feature decomposition, while morphological or phonological units rarely do, as can be seen from the large number of competing feature decomposition schemes proposed in the linguistic literature. Therefore, a true test of the system is possible only in the context of an actual natural language processing task, such as the 'business intelligence' task to which we now turn.

2 NewsMonitor

In this section we describe NewsMonitor, a high speed high performance shallow semantic analyzer built in 1990 for extracting relational information from English text.

2.1 The task

Starting in the mid-eighties, Brattle Software and later MAD Intelligent Systems attempted to develop an integrated information environment for portfolio managers. In addition to the Dow Jones ticker tape and other real-time indicators of financial performance, the system also contained a batch component called RelationalText [13], which analyzed the Wall Street Journal for relational information deemed highly significant for portfolio managers: *who is where?* and

who bought what? The former task involved the identification of PCT triples i.e. relational triples composed of Person, Company, and Title information, the latter (never implemented) would have identified mergers and acquisitions in the form of a (Buyer, Seller, Terms) triple. RelationalText was already overwhelmed by the PCT task, which involved extracting triples such as `(Carol Dwyer, ETS, senior development leader)` from sentences such as the following:

> "The evaluation may extend over a period of time for a prospective teacher, giving states a much better sense of professional development," said Carol Dwyer, a senior development leader for the Educational Testing Service.

Given the average sentence length in the WSJ, and given the fact that a deep analysis of every sentence was attempted, RelationalText, a system very typical of the sophisticated parsers deployed in the late eighties, took 36 hours for a single issue on a high-end Symbolics. Since the WSJ is available from the Dow Jones newswire three hours before the markets open on the East Coast, 36 hours of parsing time was not acceptable, and an initial target of two hours per issue was set for the NewsMonitor system described here.

2.2 System architecture

In 1990, when we designed NewsMonitor, the idea of replacing RelationalText by a lightweight 'lazy' parser based on finite state pattern matching seemed somewhat heretical, but the results were telling: NewsMonitor required only 30 minutes on a Sparc1 for a single issue of the WSJ. The bulk of this time was actually spent in several separate modules preceding the VFSA system that performed the extraction of PCT data. NewsMonitor had a simple pipeline architecture involving the following stages:

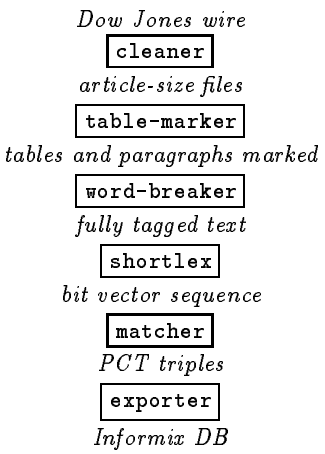

Figure 1. Main stages of the NewsMonitor pipeline

Let us discuss the various stages (given in boxes in Fig. 1) and intermediary representations (given in italics in Fig. 1) in their turn. First, the data coming from the Dow Jones wire must be cleaned of line noise and separated into articles. Devoting a separate `cleaner` module to this task has the advantage that transmission could be restarted at any time (a frequent necessity because of the line noise) and earlier articles can be sent to subsequent processing stages while the later ones are still downloading.

In the second stage numerical tables, very frequent in the WSJ and obviously devoid of PCT information, are removed, and tabs and other formatting are replaced by explicit SGML style marking of the beginning and end of paragraphs. Since sentence boundaries are not indicated by formatting and can not be trivially inferred from punctuation, `table-marker` does not parse the paragraphs into sentences.

In the third stage a lex-based tokenizer called `word-breaker` deals with capitalization, punctuation, numerals, and all non-alphabetic characters, which are replaced by SGML-style tags such as the following:

```
C1  initial uppercase
C2  mixed case
C3  all uppercase
OP  open parenthesis
OQ  open double quote
IO  inner open quote
MO  money: $, US$, HK$, sterling sign, etc.
NU  contains digit(s)
IT  italicized in print
CM  `,´ comma
MD  `--´ em-dash
```

Figure 2. Typical word-level SGML tags

In keeping with the 'lazy' spirit of the design, lexical lookup is divided into two steps. First, in stage four, a very fast hash-based lexicon called `shortlex` is searched for frequent items, function words, the SGML tags that were created in the preceding stage, and in general for all formatives that serve as pivots for the pattern matching performed. The second step of lexical lookup (not shown in Fig. 1) involves consulting the `long lexicons`, including a more traditional lexicon of 80,000 words. This is viewed as a last resort, to be invoked only if the analysis based on shortlex output indicates a definite need. Unlike the short lexicon, which is tightly organized and highly specific to the WSJ (it has only 6,300 words but yields over 85% coverage of dictionary words on any issue), the long lexicons are part of the database system and include general lists of personal names, company names, and place names (the latter not directly required by the task, but necessary for resolving ambiguities) largely absent from standard dictionaries.

The results of `shortlex` lookup are bit-vectors carrying a complex system of flags, and pointers to an array of the ascii strings left by the tokenizer after the removal of punctuation. This array is particularly useful for identifying different occurrences of the same entity, e.g. the *Mr. Henderson* appearing in one paragraph with the *John Henderson* appearing in another. Because the use of such pointers technically takes the system out of the finite state domain, we will concentrate here on the strictly finite state aspects of the pattern matcher and in particular on the system of flags used, and return to the issues raised by the ascii array in Section 3.3.

2.3 The VFSA pattern matcher

The full array of flags is divided into *global, local,* and *workspace* flags. Global flags correspond to lexical properties of the

tokens, e.g. whether they are nouns or verbs, or whether they appear in a particular lexicon. Local flags correspond to positional properties, e.g. whether a token is followed by punctuation, is immediately preceded by a title such as *Mrs*, and so on. Finally, workspace flags are assigned to all properties established in the course of the analysis, e.g. whether a certain word is the last word of a company name. In the incrementer example of Section 1, the odd numbered flags $x_1, \ldots, x_{63}$ would be global, the even numbered flags $x_2, \ldots, x_{64}$ would be local, and x_0, x_{65} and x_{66} would be workspace flags. Presenting the full system of flags and PA rules used in the `matcher` would require more space than we have here, but a few salient aspects are worth noting.

- The system encourages a grammar-writing style that embodies a qualitative theory of evidence combination. For each flag where probabilistic inferences are critical, particularly for domain delimiters like "end of company name", a small set of *confidence*flags is maintained to encode whether a hard decision has been reached, whether the current setting of the flag is regarded as somewhat probable, very probable, etc. A quantitative theory of evidence combination using e.g. (log) probabilities would be much harder to implement. As the example in Section 1 shows, the VFSA architecture does not make it impossible to perform arithmetic operations on numerical quantities, but it clearly discourages such efforts.
- The system encourages a case grammar or valence-style analysis [14], with separate flags for each slot that can be filled. Out of a universal frame limited to a single byte (8 deep cases, only 7 actually used) the lexical entries of verbs and predicate nominals mask some as optional (this requires a single byte) and some of these are further masked as obligatory (requiring another byte). In the end, results are shipped out to the relational database by inspecting the status (and associated confidences) of the Person, Company, and Title flags, and locating the left and right boundary markers associated with these.
- The system encourages a highly lexical style of analysis, with '$\bar{X}$ features' of lexical categories [6] applied throughout. Over 70 categories are used, and the majority of these are highly specific to the task. For example, the lexical flag "company class" would be set for entries such as *Incorporated* or *Limited* but not for *Telecom*. Though theoretical linguists tend to regard such fine distinctions as irrelevant to Universal Grammar (UG), it is our contention that any theory of UG incapable of furnishing a mechanism for making and maintaining these and similar distinctions is incapable of modeling human linguistic competence.

2.4 Performance

The performance of NewsMonitor surpassed that of RelationalText, which embodied hundreds of rules and several man-years of grammar development, at a very early stage, when NewsMonitor had only only three sets of rules. The first NewsMonitor ruleset centered on the keyword *Mr.*, the second on the keyword *said,* and the third on a different pattern involving *Mr.* Altogether, seven rule-sets were implemented, yielding PCT triples with 97% precision at 65% recall, and requiring less than 3 minutes CPU time per issue (this figure excludes shortlex lookup and the stages preceding, but includes longlex lookups). To put these figures in perspective, note that major system such as RelationalText never progressed beyond 60% precision at 30% recall. GE's SCISOR [7] had at the time 80-90% combined recall and precision, a result that has not been significantly improved upon in the past five years by any system performing detail parsing. The VFSA architecture thus appears competitive with other paradigms of grammar development currently in use.

3 Formal properties of VFSA

In this section we will go beyond the flat arrays that were actually implemented in the matcher stage of the NewsMonitor system and formally define VFSA in a manner embodying other structures as well. In 3.1 we consider the individual dimensions of the vectors on their own. VFSA will be defined in 3.2 simply by gathering the various dimensions together. In 3.3 we compare and contrast VFSA to RVGs.

3.1 Poset-based dimensions

Careful analysis of our first example shows that restricting the variables to boolean is not in fact necessary: within a single dimension, variables can range over any ground set G where pattern matching and overwriting are meaningful operations. Since pattern matching need not succeed, G need not be a semilattice: in the simplest possible definition G would be a finite set composed of elements g_i with unification $g_i U g_j$ defined to yield g_i if $i = j$ and to fail otherwise, and overwriting $g_i O g_j$ defined to yield g_j always. If we consider a monomial x over G, these operations are extend by $g_i U x = x U g_i = g_i O x = x O g_i = g_i, x U x = x O x = x$. We can also add a special symbol $\bar{x}$ to denote failure, so that U is no longer a partial operation. For any set of symbols G, we will denote by G' the set obtained by adjoining these two special elements.

For the boolean case (two-member ground structure) we thus obtain a 4-member unification (join) (semi)lattice $G'_{(2)}$ with x at the bottom, $\bar{x}$ at the top, and the two values $g_0 = 0$ and $g_1 = 1$ at the middle level of the 'diamond' Hasse-diagram. Note that the analogous structure $G'_{(3)}$ built on three ground elements will not be modular. In addition to the $G'_{(n)}$ for $n \geq 2$, we should also mention the degenerate structures $G'_{(1)}$ and $G'_{(0)}$. The former is a three-member chain and the latter is a two-member chain, with x at the bottom and $\bar{x}$ at the top. $G'_{(1)}$ provides a reasonable model of the unary features (privative oppositions) used in phonological theory, and $G'_{(0)}$ is perhaps a good model of the monotonic features sometimes found in computational phonology [1], because the failure symbol can only be turned on once in the course of a derivation.

In a phonological setting we would also find gradual oppositions such as vowel height, which are best modelled as chains (fully ordered sets) of ground values. In the system described in Section 2, finite chains of linearly ordered values between 0 (no confidence) and 1 (full confidence) have repeatedly shown themselves to be useful as coefficients to hard boolean flags, because matching everything at or above a prescribed confidence value is often the key to rules that add significantly to recall but could not be stated on hard flags without great loss in precision. With these uses in mind we define a single

dimension of a VFSA to be any finite partially ordered set G, and define G' as containing two additional elements, x which is smaller than all elements of G and $\bar{x}$ which is larger. When we consider several dimensions G_i (or H_j) each endowed with its own partial order, we keep the adjoined symbols $x_i, \bar{x}_i$ (or $y_j, \bar{y}_j$) distinct from one another.

3.2 Definition and discussion

Formally, a *Vectorized Finite State Automaton* is defined as a quintuple (V, F, S, A, P) where $V = \prod_{i=0}^{k} G'_i$ is the set of state *v*ectors, $F \subseteq V$ is the set of *f*inal (accepting) states, $S \in V$ is the *s*tart state, $A = \prod_{j=0}^{l} H'_j$ is the *a*lphabet, and P is a linearly ordered list of *p*attern-action statements (productions) in the form $(p; a)$, where $p \in V \times A$ and $a \in V$. For a VFSA in state s (note that s includes the symbol currently scanned), either $s \geq p_i$ for some rule in P, or the automaton halts (failure). If $s \geq p_i$ is true for several patterns in P, the action part of the first of these is executed and new input is read. To execute an action a on state s we inspect the dimensions G_i individually: either s_i and a_i are comparable, in which case the result is $\max(s_i, a_i)$, or they are not, in which case the result is a_i.

To bring this formal definition in harmony with the informal definition given in Section 1, we would need to introduce two more flags for distinguishing initial and final states, a trivial excercise. Since they are defined here in the manner of finite automata, VFSA are obviously FSA. Conversely, every ordinary FSA can be equivalently defined as a VFSA by assigning separate flags to every state and to every symbol in the alphabet, and adding a PA rule for each transition. Since VFSA operate on vectors having fixed dimensions in a length-preserving manner, there is no need to keep VFSA and VFSR separate: both are weakly equivalent to ordinary FSA, and length-preserving FSRs, the same closure properties obtain, and the same constructions can be carried through in the vectorized case as in the standard case.

Needless to say, no computational advantage could be claimed for mechanically generated vectorization that assigns separate flags for everything. Rather, the claimed advantage rests on the observation that natural language constructs are amenable to feature decomposition to begin with, and that VFSA can exploit this decomposition while standard FSA can not. In particular, the intersection of machines, normally leading to a multiplicative increase in the state space, will cause only an additive increase in the size of the flag arrays employed in VFSA. To make sure that the production set will also grow additively, we need an extra flag for each intersection. As for nondeterminism, the primary cause for exponential increase in the state space is the need to maintain partial results during the computation. FSA state space is a very ineffective memory device: to keep just eight bits around we need to increase its size by a factor of 256. In contrast, VFSA state space is designed to encourage keeping partial results around, with no more than two bits required to keep a single flag, a *content* bit which is 0 or 1 depending on whether the flag is negative or positive, and a *control* bit which is 0 for free and 1 for bound variables. (As a practical matter, content and control bits are best kept in separate arrays, rather than alternating.)

In principle, the VFSA architecture would permit overloading the flags, so the 50% overhead entailed in the use of separate content and control bits could be decreased. But in practice overloading serves only to protect a resource, flag dimension, which is not particularly scarce, and generates savings only at the cost of considerably decreased maintainability. The constituting factor of a VFSA is that the alphabet, the states, and the arcs are subject to one and the same feature analysis, and by overloading the flags the clarity and purpose of this analysis would be lost.

3.3 VFSA and RVGs

Though VFSA are equivalent in generative capacity to FSA, their closest conceptual relatives are not the standard finite automata but the Register Vectors Grammars (RVGs) introduced in [3] by Blank, who discusses how a number of important syntactic phenomena including wh-movement and limited self-embedding can be analyzed by keeping partial results around. The central computational mechanism of VFSA, namely the use of asymmetric (overwriting) operations guided by template matching is already present in [3] (and possibly already in the unpublished work of Kunst cited there). We did not follow Blank in his use of explicit side-effecting (called actions there), and we use $\bar{X}$ features for lexical categories. But these are mostly cosmetic differences, and for the most part VFSA can be thought of RVGs without the restriction to boolean features.

Is it really necessary to move away from the boolean restriction? Outside the domain of names, all partial orders that had some practical use seem to lead to a notion of (co)unification that satisfies the usual distributivity axioms, meaning that Stone's theorem (see e.g. [2]) guarantees that a feature decomposition will exist. So for the most part fixed groups of RVG flags can be used for encoding multi-valued features with very little overhead. With names, however, a great deal more arbitrariness is present, and feature decomposition offers no leverage in capturing facts such that a relation obtains between *Bank of America* and *BOFA* but not between *Commonwealth of Massachusetts* and **COFM*. In other words, something like NewsMonitor's ascii arrays will always be necessary, and VFSA make a virtue out of this necessity.

Another important difference between the two systems follows from the fact that our goal is not detail parsing but the extraction of relational information. Because relational information is often spread over several sentences, the matcher makes the computationally convenient, but psychologically obviously unrealistic assumption that the whole text, more precisely the flag structures associated with every word, can be kept in memory for the whole time of the analysis. In effect, state is kept not in a one dimensional array, the register, but in a large array of as many vectors as there are words. This has great impact on the style of the grammar. A single register encourages a local view and a temporal metaphor of *updating the state,* while the simultaneous use of several vectors encourages a global view and the spatial metaphor of *spreading information across states.* For RVGs, the kind of macro expansion system described in [3] makes the best sense as rule compiler, since the goal there is to collapse more complex state changes into a single unit. In contrast, the pattern matcher used in NewsMonitor employs awk/sed style rules for manipulating the state space.

4 Theoretical implications

In this paper we defined Vectorized Finite State Automata and described NewsMonitor, a system extracting relational information from English text using VFSA-based pattern matching. VFSA could also be used to perform detail parsing of English sentences, as demonstrated by the RVG work, or even binary arithmetic, as the example of Section 1 shows, but information extraction is a particularly challenging domain where, as the high performance of NewsMonitor shows, the techniques appropriate for sentential syntax are neither sufficient nor necessary,

In earlier work ([8], [10]) we argued that the storage device required for keeping the dependency information between the spoken/parsed and the yet unspoken/unparsed parts of a sentence need not be structured as a stack of potentially unbounded depth, but can in fact be limited to hold only a few data structures of the size and complexity of lexical entries. Here we permit unlimited interaction between distanct locations, meaning that a linear workspace (one fixed size vector for each input word) is kept by the matcher, in principle making the system a Linear Bounded Automaton (LBA). Since many researchers will no doubt have the immediate reaction that the use of LBA (full context-sensitivity) entails intolerable complexity, it is worth taking a closer look at this matter.

A complexity measure such as the Chomsky-hierarchy is only as good as its predictions about the complexity of actually carrying out the computation. Conceptualizing the matcher as an LBA is misleading, because it updates the array of vectors in strictly limited finite state transduction steps and therefore in effect remains a finite state device for any fixed array. Since in practice the search patterns are largely orthogonal, and the individual searches are blindingly fast, the standard worst-case results concerning LBA are simply meaningless here. A much better measure of complexity, in the sense of corresponding to actual computational difficulty, is provided by the dimension of the state space. At the sentence level, it is always possible to update the array of vectors in a single left-to-right pass, storing only a handful of vectors (in a fixed set of registers) at any given time [12]. According to the measure of complexity proposed here, this means constant dimensionality i.e. real-time operation. At the text level, the matcher requires dimensionality linear in the input length, indicating performance linear in the length of the input text. This is what we actually observe.

A more general objection to keeping everything in memory could be made on psychological grounds: a system that keeps large newspaper articles in working memory is clearly unable to address the issues of human memory limitations, and is therefore liable to use startegies that are, from a cognitive standpoint, unjustifiable. We take this objection very seriously, and would like to avoid the simplistic answer 'hey, it works'. But in our opinion, psychologically realistic parsing is only a remote goal at the present time, and the scientific validity of any computational system is not measured by its direct relevance to cognitive processes, but rather by the ability of the system to leverage linguistically significant generalizations into efficient computational blocks. Since the pattern matching approach encourages writing grammars in a case/construction style that is very familiar to grammarians, VFSA meet the adequacy criteria that linguistic theory currently imposes on a computational system.

If our goal is to build systems incorporating the insights of theoretical linguistics we should keep in mind that linguistic theory does not end with sentential syntax. In logical semantics the need to maintain and update memory locations devoted to individuals and to locations (both spatial and temporal) has long been recognized, and the extraction of relational information is in this sense a semantic task. The VFSA architecture, by broadening the definition of ground sets from the strictly boolean to anything finite, creates a mechanism for maintaining and updating for example person information by storing them directly in flags. The ascii array employed in NewsMonitor is best thought of as a crude implementation of a more general system of flags tracking individuals. If in any given discourse only a limited number (determined by constraints on medium- rather than short-term memory) of individuals can be actually individuated, the road is open for an implementation that uses only a single register.

Acknowledgements

Codevelopers and contributors to the NewsMonitor system included Bich Nguyen <bich@acuson.com>, Darin Okuyama <okuyama@netcom.com>, and Josef Schreiner. Special thanks to Stanley Peters <peters@csli.stanford.edu>.

REFERENCES

[1] Steven Bird, *Computational Phonology,* Cambridge University Press (1995)

[2] Garrett Birkhoff and Thomas Bartee, *Modern applied algebra* McGraw-Hill, New York (1970)

[3] Glenn David Blank, 'A finite and real-time processor for natural language', *Communications of the ACM* **32**(10) 1174-1189 (1989)

[4] Colin Cherry, Morris Halle, and Roman Jakobson, 'Toward the logical description of languages in their phonemic aspect', *Language* **29** 34-46 (1953)

[5] Noam Chomsky and Morris Halle, *The Sound Pattern of English,* Harper & Row, New York (1968)

[6] Ray S. Jackendoff, $\bar{X}$ *Syntax: A Study of Phrase Structure,* MIT Press, Cambridge MA 1977

[7] Paul F. Jacobs and Lisa F. Rau, 'SCISOR: extracting information from on-line news', *Communications of the ACM* **33**(10) 88-97 (1990)

[8] László Kálmán and András Kornai, 'Pattern matching: a finite-state approach to parsing and generation', ms, Institute of Linguistics, Hungarian Academy of Sciences (1985)

[9] Ronald M. Kaplan and Martin Kay, 'Regular models of phonological rule systems', *Computational Linguistics* **20**(3) 331-378 (1994)

[10] András Kornai, 'Natural Languages and the Chomsky Hierarchy', in: M. King (ed): *Proceedings of the 2nd European Conference of the Association for Computational Linguistics* 1-7 (1985)

[11] András Kornai, 'The generative power of feature geometry', *Annals of Mathematics and Artificial Intelligence* **8** 37-46 (1993)

[12] András Kornai and Zsolt Tuza, 'Narrowness, pathwidth, and their application in natural language processing', *Discrete Applied Mathematics* **36** 87-92 (1992)

[13] David McDonald, 'Recovering relational information from text', Brattle Software Technical Report (1989)

[14] Harold L. Somers, *Valency and Case in Computational Linguistics,* Edinburgh University Press, 1987