A NEW KIND OF FINITE-STATE AUTOMATON: REGISTER VECTOR GRAMMAR

Glenn David Blank

Lehigh University
CSEE Department
Packard Lab 19
Bethlehem, PA 18015

ABSTRACT

Register Vector Grammar is a new kind of finite-state automaton that is sensitive to context—without, of course, being context-sensitive in the sense of Chomsky hierarchy. Traditional automata are functionally simple: symbols match by identity and change by replacement. RVG is functionally complex: ternary feature vectors (e.g. +-±--++) match and change by masking (+ matches but does not change any value). Functional complexity—as opposed to the computational complexity of non-finite memory—is well suited for modelling multiple and discontinuous constraints. RVG is thus very good at handling the permutations and dependencies of syntax (wh-questions are explored as example). Because center-embedding in natural languages is in fact very shallow and constrained, context-free power is not needed. RVG can thus be guaranteed to run in a small linear time, because it is FS, and yet can capture generalizations and constraints that functionally simple FS grammars cannot.

I INTRODUCTION

Lately there has been considerable impetus among natural language researchers to restrict the computational complexity required by an adequate theory (cf. Gazdar 1981, Church 1982, Langendoen 1984). Whereas FS languages guarantee linear recognition time, those calling for more computational power give rise to a combinatorial explosion with respect to worst recognition time. Certainly, were there no other factors (such as those mentioned by Berwick and Weinberg 1982, Perrault 1983, Pullum 1983 and 1984), grammars with less computational complexity would be preferred, because they are more easily parsed, and probably also learned (see Berwick 1984).

Until recently, however, it has been supposed that grammars with a more desirable recognition time are cursed by a proliferation of rule structures when it comes to representing generalities and particularities of natural languages. The dilemma of computational complexity vs. linguistic generality has resisted a satisfactory solution ever since Chomsky 1957 argued that competence for natural languages requires at least context-free power in order to handle embedding of clauses. Moreover, to capture generalizations about categories that participate in variations of order, he introduced transformations of phrase structure—further increasing computational complexity (cf. Peters and Ritchie 1973). Similarly, Woods 1970 justified recursion plus the manifold tests and registers of ATNs, which make his scheme "equivalent to a Turing machine in power," because "the actions which it performs are 'natural' ones for the analysis of language."

But natural languages can be modelled by a kind of finite-state device that is quite compact. We can avoid excessive duplication of categories, and having to approximate unbounded memory resources. This is possible—,i£ we are willing to abandon the functional simplicity implied by symbols in rule patterns.

II TABLE-DRIVEN FS AUTOMATA

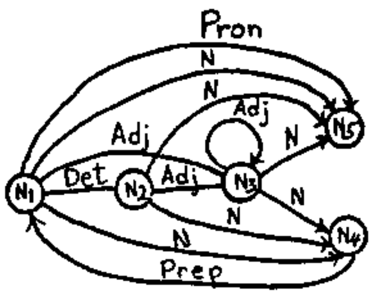
Below are two essentially equivalent ways to design a simple FS grammar: table-driven and network.

Simple (regular) FS automata

Production table

Cond	Cat	Result
N1	DET	N2
N1	ADJ	N3
N1	N	N4
N1	N	N4
N2	PRON	N5
N2	ADJ	N3
N2	N	N4
N2	N	N5
N3	ADJ	N3
N3	N	N4
N5	N	N5
N4	PREP	N1

Transition Network



Examples: production table-driven

word	CSSR	cat	word	CSSR	cat
	N1	init		N1	init
a	N2	DET	she	N5	PRON
sad	N2	ADJ			
old	N3	ADJ			
dwarf	N4	N			
in	N1	PREP			
rags	N5	N			

The table-driven implementation consists of 1) a register (here called the Current Syntactic State Register, or CSSR), 2) a table of syntactic productions (the Synindex) and 3) a machine which initializes the CSSR, makes transitions from state to state, and responds to a final state. The Synindex is a list associating: i) a category symbol, ii) a condition state symbol and iii) a result state symbol. A machine that recognizes sentences makes transitions by first matching a Synindex category symbol with a word's lexical category and the production's condition symbol with the CSSR; it then replaces the symbol in the CSSR with the production's result symbol. Following the example grammar, a transition from N1 to N2 is possible because the CSSR has been initialized to N1, and the category of *a* is DET. The CSSR then gets that production's result symbol, N2. So long as the list of productions is finite, it is a FS grammar. Such a machine has a finite number of possible states, though it can run on indefinitely. The category PREP, for example, iterates back to N1 .

A FS grammar requires that every category be determined as possible by a function of the immediately preceding category. Space for the register and index are pre-allocated; no external memory is available. This is as true of the equivalent RVG:

RVG Synindex for simple NPs

Cond	Cat	Result	Feature Key:
1222	DET	0222	1-DET
2122	ADJ	0222	2-ADJ
2212	N	0002	3-HEAD
0001	PREP	1112	4-PREP
1111	PRON	0000	

Examples

word	CSSR	cat	word	CSSR	cat
	1111	init		1111	init
a	0111	DET	she	0000	PRON
sad	0111	ADJ			
old	0111	ADJ			
dwarf	0001	N			
in	1111	PREP			
rags	0001	N			

In a RVG, the symbols become vectors of features, each capable of three possible values (+,-,+, or "on", "off", "mask", or henceforward 1,0,2). The match function allows either identity or ambiguity (2 matches any value); the change function allows either literal or masked replacement (2 doesn't change anything). For example, if the CSSR is initialized to 1111, the condition vector of DET (1222) matches, and the result of DET (0222) can be applied, yielding 0111. This is what is meant by functional complexity: whereas in a simple FS automaton match is identity and change is wholesale replacement, in RVG match and change allow ambiguity and masking. Nevertheless the ternary functions are quite simple and determinate.

Note that the RVG Synindex is considerably more compact than the equivalent FS table. Whereas the FS table has three productions for the category ADJ and six for N, the RVG Synindex has just one production per category. The savings—not only of space in the table, but time trying alternatives-will multiply as more permutations of order are introduced.

How do RVGs achieve their efficiency? Functional complexity confers two properties that functionally simple grammars do not have: multiplicity and masking. Condition vectors can convey multiple constraints. For example, PREP (condition 0001) cannot occur until the first three features have been reversed in value. Moreover, result vectors can produce multiple effects: PREP (result 1112) re-enables all NP-opening productions. Note also that disjoint categories (e.g. N and PRON) share the same position in ordering vectors, so that they mutually exclude each other. That is, the result of N and PRON both disable the same feature (governing the HEAD position). (PRON (result 0000) also rules out having a post-modifying prepositional phrase.)

Masking is a consequence of ternary values. The third possible value (2) matches any value and does not change any value. Thus constraints may be passed through intervening states. For example, the category N is oblivious to whether or not the categories DET or ADJ have already occurred, since its condition vector (2212) matches the initial state (1111) or the state after DET (0111) or after ADJ (0011). Thus we can represent options as well as obligations. (Note that the ordering of optional productions is enforced by having successive categories also disable the constraining features, e.g. the result of N is 0002.) Moreover, iteration may be treated as a special case of optionality. Whereas a one-occurrence category disables itself (e.g. the result of DET is 0111), an iterative category does not (the result of ADJ is 0211). Like an optional category, an iterative category is disabled by successive categories up to the next obligatory category (e.g. the result of N is 0002). Finally, masking features allows constraints to be held through any number of intervening states—as will be demonstrated when we consider long-distance dependencies.

Finite functional complexity can thus increase the expressiveness of a grammar considerably. It is not to be confused with "complex symbols" found elsewhere in linguistic theory. The subcategorization symbols of Chomsky 1965 allow for context-sensitive power; the features of Gazdar and Pullum 1982 call for recursive elaboration of symbols. Features attached to GPSG phrase structure nodes themselves take the form of open-ended trees or directed graphs. The feature vectors of RVG, on the other hand, are finite and do not expand. Moreover, RVG is not an attribute-value system requiring piecemeal interpretation of individual features. Whole vectors (a ternary vector can be implemented as a pair of bit vectors), rather than lists of symbols, are the operational unit of on-line processing.

In any case, other researchers have used oomplex features to govern categorization, but never to describe syntactic states.

III WH-QUESTIONS

Wh-questions are perhaps the long-distance dependency par excellence. A wh-word (who, what, etc.) provokes the grammar to watch and wait for a 'gap'—an expected but missing noun phrase. The gap may be filled wherever a noun phrase is expected in a clause:

What is the robot seeking?
WH:what; AUX:is; SUBJ:NP:DET:the; N:robot; VTRANS:seeking; OBJ:NGAP:CCLOSE:?;

What is the wrench on?
WH:what; AUX:is; SUBJ:NP:DET:the; N:wrench; PREPion; NGAP:CCLOSE:?;

What did the robot find the wrench on?
WH:what; AUX:did; SUBJ:NP:DET:the; N:robot; VTRANS:find; OBJ:NP:DET:the; N:wrench; PREPPOST:on; NGAP:CCLOSE:?;

What is a robot?
WH:what; AUX:is; SUBJ:NP:DET:a; N:robot; PREDNP:NGAP:CCLOSE:?;

What did the robot find the wrench?
Synindex search fails at: ?

Delaying, for the moment, discussion of embedded clauses, we can see that wh-questions can be modelled straightforwardly.

Synindex for Wh-Questions

Label	Condition	Result	L?
WH	11111 00002	22222 22211	L
SUBJ	12111 00022	02222 12222	n
AUX	21111 00022	20222 22222	L
VTRANS	02111 00022	20022 22222	L
PREDNP	00111 00022	22000 12222	n
PREP	00111 00022	22000 12222	L
OBJ	00011 00022	22202 12222	n
PREPPOST	00001 00022	22222 12222	L
NP	22222 12222	22222 01122	n
DET	22222 01122	22222 20122	L
N	22222 02122	22222 20022	L
NGAP	00222 12212	22222 02202	n
CCLOSE	00002 00002	11111 00000	L

Feature Key: 1-SUBJ 2-AUX 3-PRED 4-OBJ 5-PREPPOST 6-NP 7-DET 8-N 9-GAP 10-WH
L?: 'L' - consume a lexeme; 'n' - don't

The result of production WH, among other things, switches "on" a feature GAP. Only the production NGAP can turn this feature "off." NGAP can occur in quite a number of places--in fact, just about wherever an NP can occur (except SUBJ). But sooner or later NGAP must occur, inasmuch as CCLOSE requires that GAP have been turned off. The last sentence above fails because GAP is *never* disobligated.

It should be observed that in RVG the recurrence of noun phrases at various positions within a clause does not involve calling a subnetwork. Eschewing recursion, we instead allocate separate sections of the ordering vector type to deal with clause matters and phrase matters. Ternary masking allows the CSSR to preserve its clause-level status while it traverses a phrase. A number of boundary productions—SUBJ, OBJ, IOBJ, PREP, etc.—set a feature which at once temporarily disables subsequent clause-level productions, and enables the boundary production NP. If NP is chosen—there are other possibilities—all of the features ordering a phrase will be reset. One alternative is NGAP, whose condition requires that both the NP and GAP be on, and its result turns both of these features off. Thus NP and NGAP are disjoint.

IV EMBEDDING

Clause embedding was the primary evidence cited by Chomsky 1957 for claiming that natural languages are at least context-free. He discusses, for example, the following kinds of structures:

if S1 then S2
if [either S3 or S4]$_{S1}$- then S2
if [either [George said that S5]$_{S3}$ or S4]$_{S1}$ then *S2*

The argument goes: i) finite-state automata cannot handle $a^n b^n$ (or "mirror image") grammars because they do not have the memory to keep track of n; ii) if..then, either..or constructions suggest that natural languages are of this type; iii) therefore natural languages cannot be finite-state.

There are, however, severe constraints on embedding. The most well-known case is that of object relative constructions:

The mouse the cat chased squeaked.
The mouse the cat the dog bit chased squeaked.

Embedding object relatives once is not unusual, but twice is boggling. The most common explanantion for this problem is that center-embedding causes an overload of short-term memory processing. But as Kac 1980 has pointed out, this cannot be the whole story. For if it were we would expect embedding to break down at a predictable depth. But it does not. For example, though one clause can embed another of a different type, there is no *self*-embedding:

If if the Pope is Catholic then pigs have wings then Napoleon loves Josephine.
That that Dan likes Sue annoys Bob bothers me.

These sort of constructs are plausible, of course, in truly context-free languages such as Pascal or LISP.

Nor are relative clauses a uniform phenomena. First of all, subject relatives are possible indefinitely:

> The fellow who saw the dog that bit the cat that chased the mouse is willing to testify.

Object relatives appear to be limited to a depth of one, but combinations of an object relative and a noun complement are not:

> The statement that the election which he lost ended his career dismays him.
> The only one who the fact that George resigned pleased was Tom.

Embedding also goes to deeper levels in order to attach suspended arguments or adverbial adjuncts:

> A teacher who wants students who persuade their classmates who don't know the material to help them to ask him instead must make himself available.
> Do you see why I wanted to deny that grammar is recursive so vehemently now?

In the first sentence, each infinitive clause i.s projected as an argument of an earlier predicate (wants persuade). All of the relatives are subject relatives, so for each no gap is suspended, only an argument. In the second sentence, adverbs are attached to predicates after intervening complement clauses—so vehemently to deny and now all the way back to the main clause's predicate, see.

Though short-term memory is related to the shallowness of center-embedding generally, it is not an adequate account for the variety of specific constraints. Typically, memory limitations have been regarded as an aspect of performance rather than competence. The performance processor, with limited memory, is said to "approximate" competence, which is said to "idealize" the unlimited memory of a CF automaton. Thus when Church 1982 talks about finite-state processing, his aim is just "to design a parser that approximates competence with realistic resources." But one wonders: since memory constraints are universal to the species, why aren't they of import to models of competence? The various constraints on center-embedding argue against the functionally simple notation for denoting clauses (e.g. S or S-bar), which seems to imply that all clauses are (almost) alike. The versatility possible with the finite functional complexity of RVG re-opens the question whether CF power is needed.

If..then and either..or are better treated as discontinuous constraints than as "mirror-image" syntax. A single feature allocated for each paired construct can give us the obligations and options we want, with the help of a production to enforce the closure of discourse-level units:

Paired Particles

Label	Cond	Result	Feature Key:
IF	00	12	1: if..then
THEN	20	02	2: either..or
EITHER	20	21	
EITH-OR	21	20	
DCLOSE	00	22	

This fragment effectively enforces these constraints: i) IF obligates a THEN clause before Dclose; ii) EITHER obligates an EITH-OR clause before DCLOSE; iii) IF forbids another IF until a THEN re-options it; iv) EITHER forbids another EITHER until EITH-OR re-options it; v) EITHER forbids IF until EITH-OR re-options it. The last case ensures a constraint that Kac 1980 notes but cannot explain:

> Either if [clause] then [clause] or [clause]

(A possible explanation for this constraint is that it avoids many ambiguities that might otherwise be brought on by the conjunction or.)

In general, embedding can be modelled by any FS automata so long as the maximum depth is finite. Conceivably even a simple FS automaton can manage embedding, by duplicating clause syntax at every possible entry point. But such a grammar would be enormous, and is perhaps justifiably scorned as lacking "explanatory adequacy."

But ternary vectors can consolidate matters considerably. Rather than having to respecify for every possible site of embedding, an RVG need only keep track of the current level of embedding.

The data reviewed in this paper can be managed by keeping track of up to three levels of embedding. The top, or main clause level, is treated differently from the bottom two. Where the grammar does allow us to embed more deeply (e.g. right-embedded complements, subject relatives, etc.), it will begin to iterate in the space of the bottom two clauses. Thus we can adjoin arguments or adverbials to the current clause, or one clause higher, or to the main clause, which is always kept. E.g.:

> Do you see [why I wanted [to deny [that grammar is recursive] so vehemently]] now?

The adverbial so vehemently is attached back over an intervening clause; now to the main clause.

There are two styles in which one might implement clause embedding in RVG. They are virtually equivalent in terms of computational efficiency, but reviewing both will perhaps elucidate RVG methodology. The first is in the same spirit as NP embedding. Just as ordering of phrasal constituents is managed by a segment of the complete ordering vector, so each clause level might be treated as a separate segment of a long ordering vector type. Just as several boundary productions (SUBJ, PREP, etc.) suspend further clause-level productions while enabling NP, so other boundary productions may shift attention from one clause level to another. All of the segments are part of the same vector type, so that there is no need for storage beyond the CSSR, which holds a complete vector. Yet it is possible, because of ternary masking, for boundary productions to, as it were, open and close "windows" on relevant segments. The drawback of this technique is that, for each level, the grammar must replicate the features for clauses (the ordering vectors all get wider), and also replicate most of the clause-level productions (the Synindex table gets longer). Replicating productions such as SUBJ, CADJ, et al., is necessary in order to apply different condition and result vectors at each clause level. Increased size is not necessarily a clinching drawback, since it is by a factor somewhat less than three (phrasal and discourse-level features and productions need not be replicated). The alternative approach makes for a smaller Synindex, to be manipulated by a slightly more complicated algorithm.

Instead of widening the ordering vector, we add depth to the CSSR. The new CSSR is multi-leveled:

```
                     CSSR
ClauseLevel --> |Main clause vector |
                |1st embedded vector|
                |2nd embedded vector|
```

The state register is still finite and fully visible; no external memory is needed. Only now it has ordered levels as well as features. It is as if we broke up a long vector, masking inactive segments with an array subscript rather than ternary values. To embed a clause, the shifting facility increments ClauseLevel, and initializes the lower clause from the higher (so that features like GAP can be passed down). To return, it simply decrements ClauseLevel.

This scheme captures generalizations about what clause levels hold in common. It also allows specialized productions to distinguish clause types—with different result vectors. E.g., the complementizer that sets up a complete clause, but the infinitive particle to arranges for a clause starting with a non-finite verb. Similarly, the complex NP constraint (Ross 1967) is easily modelled by different result codes. After shifting clause level, change applies as usual. Complement productions allow feature GAP to be passed down (by masking), whereas relatives reset it:

```
What did you think that the robot which found?
Synindex search fails at: ?

What did you hope to find?
WH:what; TENSE:-past,DO:do; SUBJ:NP:PRON:you
NPCLOSE: NONFIN:-inf,VTRANS:hope; OBJ:
COMPINF:to; NONFIN:-inf,VTRANS:find;
OBJ:NGAP:FILLEDGAP:CCLOSE:?;
```

```
Who were you persuaded to find the wrench by?
WH:who; TENSE:-past,BE:be; SUBJ:NP:PRON:you;
NPCLOSE: PASSIVE:-past par t.VDITRANS: persuade;
COMPINF:to; NONFIN:-inf, VTRANS:f ind; OBJ:NP:
DET:the; NUMBER:-sing,N:wrench; NPCLOSE:
PASSGAP: PASSIVEBY:by; NGAP:CCLOSE:?;
```

Clause-closing productions may or may not insist that GAP be off. RELCLOSE makes this requirement, but does not affect the higher clause, whereas FILLEDGAP does turn off GAP in the higher clause, and PASSGAP leaves GAP alone.

After the shifting facility has embedded two clauses, it resorts to re-use. That is, the second embedded clause is shifted up to the first. Thus right-embedding can iterate in two registers, but preserves the main clause. Two embedded clause registers are enough to allow the parser to resume suspended arguments or adverbials, as in

> A teacher who wants students who persuade their classmates who don't know the material to help them to ask him instead must make himself available.

But its center-embedding capacity has now has been reached. The parser is baffled by this sentence:

> Pamela persuaded the robot who wanted to give the pyramid which was on the blocks which it found to her very much to find a wrench instead.

By the time it encounters very much, the clause to which the adverbial might have been attached (wanted...) has been lost to re-use. But such a sentence is beyond the tolerance of many human speakers as well.

Superficially this embedding scheme resembles a bounded stack. But there are some crucial differences. First of all, a bounded stack scheme (such as that of Church 1982) typically calls for a great deal more storage than just three registers, since it will have to keep track of the full gamut of embedding implied by PS rules—NPs, VPs, X-bars and the like. Second, whereas functionally simple systems treat every level of embedding alike, in RVG each level of embedding is marked—main clause, first embedded, re-usable. Thus in RVG there is no *self*-embedding. Third, CSSR levels are not really organized like a stack, since the lower two registers are re-used iteratively, and the main clause register is always preserved. In fact there is nothing which prevents a RVG from accessing clause levels in another order. For example, we could model cross-serial dependencies in Dutch (see Bresnan et al. 1982), by allowing boundary productions to access storage registers in a queue-like order. (Assuming that cross-serial dependency, like center-embedding, is limited.)

V TOWARD A COMPLETE RVG SYSTEM

RVO syntax may be thought of a general-purpose scheduler, with potential applications wherever such a device would be desirable. Generally, non-syntactic procedures can be scheduled by association with Synindex productions. Indeed, syntactic productions are motivated by non-syntactic requirements. For example, nouns, pronouns and names are very similar posltionally, but they are treated as distinct categories because of semantics. In this section I will briefly describe how an RVO oan, while maintaining modular autonomy to a considerable degree, support integration of processes.

Categories. In earlier versions of RVG, categories, like syntactic ordering and semantic constraints, were represented in terms of ternary feature vectors. Ternary values supported cross-categorization well—lexemes in more than one category could allow them with 2's. But since RVG holds down duplication of productions, it is feasable to associate, with each lexeme, a list of Synindex production numbers. Moreover it is possible, with some pre-processing, to let these production numbers be generated from labels, and also to infer the set of non-lexioal productions that could precede each lexical production. Thus, for example, the lexeme kitten is categorized by the label N, which is converted to a production number; the non-lexioal production numbers for SUB J, OBJ, NP, etc., are kept in a precedes set associated with the Synindex entry for N. This approach is at once more convenient, because categories are kept functionally simple with respect to notation, and more efficient, because the lexicon is 'wired' directly to the Synindex, obviating any on-line processing of categories.

Semantics. RVG syntax can in fact support Just about any form of semantics. But I present our biases. Since RVG permits a relatively flat, direct treatment of permutations of order, there is no reason to complicate matters—as trees do, since they imply recursion. Rather than transform syntactic structures, or propose meta-rules or lexical-dependency subtrees which have similar effect, why not let categorized actions operate upon semantic structures directly? The problem of mapping constituent structures into lexical structures is simplified by just eliminating context-free constituent structures. There is no need to move or unify sub-trees in RVG; there are no trees at all. Moreover, semantic and morphological agreement can be simplified: there is no need to 'percolate' or 'inherit' features up and down trees, nor to design special filters or powerful constraints to regulate such activity. Functional complexity allows RVG to be sensitive to context without being context-sensitive in the sense of the Chomsky hierarchy.

In RVG, we allocate a fixed configuration of registers (the Current Predication State Registers, or CPSR) for managing grammatical relations. The CPSR and CSSR together comprise the state of a clause (or a phrase in a clause), and as shown above, RVG keeps track of up to three clauses. Permanent semantic representation is built up dynamically (it is here that we allow open-ended structures); CPSR slots hold addresses of proposed semantic referents.

Categorized actions (associated with syntactic productions) have the responsibility of mapping new lexical material into existing semantic structure. In the spirit of RVG, lexemes are viewed as standardized in structure. All semantic features are organized in a single vector type, the first-order semantic vector. Every lexeme has an INTRINSIC first-order code. Constraints on arguments of predicates are specified, as needed, in terms of additional first-order vectors. First_Order__Pred, a generalized procedure called by many categorized actions, for all arguments, first matches vectors (thus checking selectional restrictions), then unifies them. E.g., 22201 matches 01202 and yields 01201. Thus in RVG features are not used merely to validate, but also to define semantic structures.

Morphology. Kunst and Blank (1982) show that morphology oan be efficiently implemented as a retrieval tree, with provision for morphological paradigms as nodes encountered during lexical lookup. Currently, we represent morphemes as lexemes in their own right, each with its own list of Synindex production numbers and semantic structure. Thus, for example, lookup recognizes oats as two lexemes: -plur and cat. The lexeme -plur has an INTRINSIC vector (distinguishing it from-sing). It also lists the subscript for the Synindex production NumberER, which maps the INTRINSIC of the lexeme (-.plur) to CPSR[Head], and then calls First__Order_Pred: thus enforcing number agreement. Another morphologically categorized production, TENSE, enforces 'subject-verb' agreement. Actually, The kittens is playing fails at TENSE, whereas The bricks are playing violates constraints at VTRANS. But note that all agreement is managed by uniform operations upon a single vector type.

Letters must have been being written.
SUBJ:NP:NUM3ER:-plur,N:letter; NPCLOSE:
TENSE:-pres,MODAL:must; NQNFIN:-inf,HAVE:have;
PERF:-pastpart,BE:be; PROG:-progpart,BE:be;
PASSIVE:-pastpart,VTRANS:write; CCL0SE:.;

Discourse, etc. RVG does not require that 'S' be the root of syntax. The examples in this paper suggest that a disco use-level syntax could be integrated as part of the Synindex, or as a separate production table interfacing with the Synindex. Paired particles were modelled as disoourse-level discontinuity, and wh-questions are marked by a feature switched on by the production WH. Generalizations about syntax, expressed by features in vectors rather than by complex rule patterns, are thus more readily available to other systems.

RVGs are highly reversible. A parser and a generator can be implemented (and have been) using the same Synindex and basic table-driving algorithm. Indeed, all of the major data structures of RVG work in either direction as is; only the procedures need actually be reversed. While parsing, discourse-level features are set by syntax for higher-level consideration. While generating, discourse-level features are set from above to choose particular sentence forms.

RVGs are computationally simple and compact. The basic algorithm is that of a table-driven finite-state device, modified to invoke ternary match and change functions. Ternary vectors can be represented on a binary computer by paired bit vectors. Ternary match and change are implemented by combining logical operations (exploiting the low-level parallelism of bits in computer words). The most recent version of RVG, in Pascal (earlier versions were in SNOBOL and Icon), does so, with great gains in speed. On a DEC-20, the syntactic parser averages about 5 milliseconds per lexeme. Further improvement is possible—on ternary circuitry.

Expansion to fuller coverage (a substantial fragment of English syntax has been implemented) will have a minimal slowing effect, since each new category motivates one new production, rather than many new rules (or a meta-rule that generates many rules). Each revision is on features already allocated in all productions. Occasionally new features must be allocated, but most frequently early in grammar-making, and least frequently late.

Ambiguous parsing to be sure slows processing down, though not unbearably. The combination of well-specified category vectors, semantic constraints and possibly a form of bounded parallelism (under investigation) can hold syntactic processing time to a small linear time. Thus parsing of natural languages appears to be feasable by machines in real time. Indeed, syntax should; be fast, if it is to facilitate the many other processes—from phonology to reasoning—which all go on in real time.

ACKNOWLEDGEMENT

RVG originates in unpublished work by A. E. Kunst, to whom I am also grateful for help with this paper. In an unpublished paper, Professor Kunst compares RVGs to Petri nets (rather than simple transition networks). Petri nets also allow functional complexity. Specifically, he compares of safe Petri nets (which are known to be weakly equivalent to FS automata) and RVGs.

REFERENCES

[1] Berwick, R. & A. Weinberg. Parsing efficiency and the evaluation of grammatical theories. Linguistic Inquiry 13:4 (1982) 135-191.

[2] Berwick, R. Bounded context parsing and easy learnability. Proc. 22M ACL, Stanford, Palo Alto, 1984, pp. 20-23.

[3] Blank, G. Lexicallzed metaphors: A cognitive JttQdfii In in framework of Register Vector semantics. PhD thesis, University of Wisconsin-Madison, 1984.

[4] Bresnan, J., R. Kaplan, S. Peters and A. Zaenan. Cross-serial dependencies in Dutch. Linguistic Inquery 13:4 (1982) 613-35.

[5] Chomsky, N. Syntactic structures. Mouton, The Hague, 1957.

[6] Chomsky, N. of the therory? *syntex* MIT Press, Cambridge, MA, 1965.

[7] Church, K. On memory limitations i,q natural language processing. IU Linguistics Club, Bloomington, Indiana, 1982.

[8] Gazdar, G. &G. K. Pullum. Generalized phrase Structure grammar: a theoretical synopsis. IU Linguistics Club, Bloomington, Indiana, 1982.

[9] Kac, M. On the recognition of complex NP's. Workshop .on Language Processing and Asauisitlon. Brown University, Providence, RI, 1980.

[10] Kunst, A. E. Petri net theory and the representation of natural languages. Unpublished paper, Comparative Literature, University of Wisconsin-Madison, 1983.

[11] Langendoen, D. T. and Y. Langsam. The representation of constituent structures for finite-state parsing. Proc. 22nd ACL. Palo Alto, 1984, pp. 24-27.

[12] Perrault, C. On the mathematical properties of linguistic theories. Proc. 21st ACL, MIT, Cambridge, MA, 1983, pp. 98-104.

[13] Peters, P. S. and R. W. Ritchie. On the generative power of transformational grammars. Information *Sciences.* 6 (1973) 49-63.

[14] Pullum, G. Context-freeness and the computer processing of human languages. Proc. 21st ACL MIT, Cambridge, MA, 1983.

[15] Pullum, G. Syntactic and semantic parsability. Proc. 22nd ACL, 1984, pp. 112-22.

[16] Ross, J. Constraint? on variables in syntax. PhD thesis, MIT, 1967.

[17] Woods, W. Transition network grammars for natural language analysis. CACM 13:10 (1970) 591-606.