Scherer, K.R., D.R. Ladd & K.E.A. Silverman. 1984. 'Vocal cues to speaker affect: Testing two models.' *Journal of the Acoustical Society of America* 76:5, 1346-1356.

Skinner, M.W., L.K. Holden, T.A. Holden, R.C. Dowell, P.M. Seligman, J.A. Brimacombe & A.L. Beiter. 1991. 'Performance of Postlinguistically Deaf Adults with the Wearable Speech Processor (WSP III) and Mini Speech Processor (MSP) of the Nucleus Multi-Electrode Cochlear Implant'. *Ear and Hearing* 12, 3-22.

Touati, P. 1987. *Structures prosodiques du suédois et du français.* Lund: Lund University Press.

Waltzman, S. & I. Hochberg. 1990. 'Perception of Speech Pattern Contrasts Using a Multichannel Cochlear Implant'. *Ear and Hearing* 11:1, 50-55.

Williams, C.E. & K.N. Stevens. 1972. 'Emotions and speech: Some acoustical correlates'. *The Journal of the Acoustical Society of America* 52:4, 1238 -1250.

Lund University, Dept. of Linguistics
Working Papers 38 (1991), 133-54

A Minimalist Parser for Fast Partial Analysis

Christer Johansson

This work is based on Blank 1985, 1989, presenting a method for making a machine that can provide fast analysis of natural language with bounded resources. The machine is a new kind of finite automaton and the formalism is called Register Vector Grammar, RVG. The machine is claimed to provide analysis of natural language in linear time. Linear time is a considerable improvement compared to polynomial time, which is what is currently available for parsers. In order to make linear time, the depth of the analysis is restricted. The machine has been constructed to model performance rather than competence thus the approach introduces restrictions on the input language. The restrictions can be motivated by the fact that humans can be viewed as limited devices, in particular their short-term memory is limited. As an example, an RVG grammar for agreement in Swedish nominals is developed.

Introduction

A great deal of language analysis can be done with a simplified model that does not have the overhead of context-free grammar. It can be desirable to split up the big analysis module needing context-free power into several smaller analyzers. One reason is that the smaller modules need less computer capacity; another, more important reason, is that a small module is easier to maintain. In any big analysis module there will always be conflicts among the rules and in the lexical database. It is therefore better to build up the analysis in small stages. There have been several experiments with late assigning of grammatical roles and incremental parsing at the Dept. of Linguistics, Lund (Sigurd et al. 1989, Sigurd 1990) and more recently, experiments with the RVG model. These experiments have searched for a psychologically plausible model for human language performance.

The parser which this paper presents is heavily influenced by Register Vector Grammar as described by Blank 1985, 1989. This specific implementation of RVG is, however, constructed to aid limited partial analysis.

The parser is based on production rules, i.e. simple 'if / then' statements, and a special kind of lexicon which connects the surface form with the rules to be applied. For example, the ambiguous lexicon entry '*back* (noun verb)' connects the surface form *back* with either the production rule 'noun' or 'verb'.

The production rules in turn test a global set of features and fire a change on these features if the current values can be accepted.

This paper will first give an overview of some of the peculiarities of the grammar used. After that follows an analysis of agreement in Swedish nominals expressed in feature values, followed by a technical discussion of the implementation of the parser. A complete grammar for Swedish agreement is given as an appendix. (It might be a good idea to read the appendix first to get the feel of RVG grammar.)

The grammar productions

A production rule consists of two parts. The first part defines the condition for the rule to fire; the second part defines the changes which are made if the rule fires. All production rules also have a name attached to them which makes it possible to follow a trace and find out what production rules have been fired. The final output is a trace of production rule names which has led to acceptance of the sentence. Following is an example of production rules illustrating the ternary logic with a scene from the jungle.

The name of the first rule is 'ToughLuck'; if it fires the trace will contain the word 'ToughLuck'. The alternative rule is called 'GoodLuck'.

P means that it is a production rule. The N is specific to the implementation of the grammar and it means that the rule is non-lexical, i.e. it does not consume any word from the lexical input.

```
1)  P ToughLuck N
    IF      +Mowgli  +Tiger  +TigerHungry
    THEN    -Mowgli  +Tiger  -TigerHungry

2)  P GoodLuck N
    IF      +Mowgli  +Tiger  -TigerHungry
    THEN    +Mowgli  +Tiger  -TigerHungry
```

The first rule says that:

if Mowgli *might be* on ('+') **and** Tiger *might be* on ('+')
and TigerHungry *might be* on ('+')
then the rule can fire with the result that:
Mowgli *is* off ('-') **and** Tiger *is* on ('+') **and** TigerHungry *is* off ('-').

What happens if the machine for some reason cannot determine the value of some feature? The production can nevertheless be accepted, because an unknown value (e.g. ?TigerHungry) can in fact be interpreted as either '+' or '-'.

If you want to make a feature unknown, you can do this with the '!' operator in the 'then'-part of the production rule. If the first rule had '!TigerHungry' in its 'then-part' then the machine does not know if TigerHungry was '+' or '-' after the rule has fired (the tiger would perhaps still be hungry since Mowgli is so small).

If, for some reason, it is necessary to test for an unknown value then it can be done by using the '!' operator in the 'if'-part of the production rule. It could be possible for the second rule to have '!TigerHungry' in its 'if-part', which would mean the rule applies if and only if the value of TigerHungry is unknown ('?').

If Mowgli is on ('+') and Tiger is on ('+') but TigerHungry is unknown ('?') then both rules can be accepted as they are, but the 'ToughLuck' rule is tried first.

As this parser is customized for grammatical analysis the rules differentiate between the following three types of rules:

1) lexical rules (L) which consume one word from the input
2) non-lexical rules (N) which do not consume input
3) the InitFinal rule (I) which states the condition for acceptance in its 'if'-part and the start values in its 'then'-part. The InitFinal rule always consumes one item from the input, typically a punctuation mark.

The lexical rules are closely linked with the lexicon. The lexicon simply states which lexical rules can be connected with the defining word and in what order the lexical rules should be tried. For example the lexical entry '*flies* (noun_pl verb_sg)' connects the word *flies* with the two production rules 'noun_pl' and 'verb_sg'. The 'noun_pl' rule is tried first and 'verb_sg' is considered an alternative. Those two rules, in turn, are written as production rules.

The non-lexical rules are mainly used to open and close boxes for small phrases, primarily by changing the values of the noun and/or verb acceptance features (i.e. N V). By convention the '+' value of N or V is used to indicate that the box is open and the '-' value is used to indicate that it is closed. Since the parser uses no tree structure, it has to create 'boxes' of

constituents in the trace. This can be done with non-lexical productions. For example, a 'noun box' can be defined as an environment where the feature N is on (+N) and V is off (-V). In this case, the opening production should set N to +N and V to -V and the closing production should set both N and V to '-' in order to take us out of the box.

E.g.: Parse the sentence *Bill loves apples*. The initial value of the features N and V is '-'. The bar sign ('|') indicates how far the parser has analysed.

| Bill loves apples. [-N, -V] (outside a box)
Bill demands +N -V which leads to the opening of a box indicated by '['.
[Bill | loves apples. [+N, -V] (inside a noun box)
loves demands -N +V which leads to the *closing* of the previous box ([-N,-V]) followed by the *opening* of a new box, indicated by ']' and '[' respectively.
[Bill] [loves | apples. [-N, +V] (inside a verb box)
apples in the same way as *Bill* demands +N -V, and forces the machine to take us out of the verb box and into a noun box.
[Bill] [loves] [apples |. [+N, -V] (inside a noun box)
The punctuation mark demands to be accepted outside of a box, and therefore forces the previous box to close.
[Bill] [loves] [apples].| [-N, -V] (outside a box and ready)

Conflict resolution

As the number of production rules grow there will be conflicts between the rules. One cause of rule conflicts is that the same surface form may belong to many different production rules (categories). Another cause of conflicts is that there might be several alternative paths defined by the non-lexical rules which all might lead to the acceptance of the sentence. The parser must have some strategy for determining which rule to try first.

This is particularly important because the number of alternatives that are kept in memory is limited, a fact which is elaborated upon in the next section.

Conflicts between production rules are resolved in the following way:

1. Lexical productions are always tested first.
2. The order between the lexical rules comes from the lexicon.
3. Only if none of the lexical rules that are connected with the word can be accepted will non-lexical rules be tested. The parser tries to stay at the same level for as long as possible. A typical non-lexical rule effectuates a change of level by opening or closing a phrase box.

4. The order between non-lexical rules is the same as their order in the grammar file.
5. Alternatives are supported by keeping a limited backtracking list of the *possible* alternatives at every point where there is a conflict between rules of the same kind.

Problems with backtracking

Connected with conflict resolution are problems with general backtracking. This limited parser tries to overcome the problems by choosing limited backtracking as a support for the conflict resolution. In short, unlimited backtracking demands unlimited memory and unlimited time (in the worst case), which cannot be risked especially if the system is to operate in real time. Limited backtracking can be thought of as functioning in a manner analogous to human short term memory. Information is constantly renewed and the oldest alternatives in memory are forgotten. One might say that forgetful backtracking is an optimistic algorithm; the further an analysis has proceeded, the more likely it is that the first steps were correct. The machine might forget a correct alternative but the same is true for us humans. A piece of evidence is the existence of 'garden-path sentences' (Blank 1989) as in *The horse raced past the barn fell*. Many people have difficulty understanding such sentences when they appear out of context. The verb *raced* might be interpreted as an intransitive verb and therefore it is interpreted as the main verb. When the reader later sees the verb *fell* his intransitive analysis of the verb *raced* falls short. The same sentence could be rewritten into another structure which better guides the reader: *The horse that was raced past the barn fell.*

It seems to be a characteristic of human language to avoid centre embedded structures when it is possible and thus make the sentence easier to understand. The strongest argument for trying finite automata in language analysis is that humans are in fact limited devices. We have limited short-term memory, we do not generate infinitely long sentences and we tend to avoid centre embedded clauses.

It seems therefore unreasonable that practical MT systems spend much time and effort trying to cover (uncommon) constructions like infinite centre embedding and infinitely long sentences. This time could be spent on other things which could improve the overall quality of the translations, e.g. keeping track of referents.

Two relatively new additions to our parser are the possibility to use a filter and the possibility of being able to guess word categories on the basis of word endings. The following two sections will shortly describe these additions before going over to the implementation of agreement in Swedish nominals.

Experiments with a filter

Sometimes it might be easier to let the grammar overgenerate possible alternatives which are later ruled out because they have an impossible or unlikely grammatical structure. This is the purpose of having a filter.

Our filter consists of two parts. The first part is a specification of the production rule names which are considered important for the structure of a sentence. The second part is the possible linear patterns of such production rules. For example, we might have one production rule called NP that opens a nominal and one production rule called VP that opens a predicate. Possible patterns of these might be: NP or NP VP or NP VP NP or NP VP NP NP etc. An impossible pattern would be VP VP.

A positive filter, as implemented, is a filter which states the valid patterns. A negative filter states the invalid patterns that we might have found and it is therefore more permissive – it allows all patterns which have not been found. I made the choice of having a positive filter because I am interested in finding out what patterns actually occur in limited texts.

The actual filter construction is aided by the program. The user specifies the important productions in a file and chooses the alternative to update a filter for a specific text. The filter created is a specification of the structure rules and all of the patterns of those rules that occurred in the text.

The user, who might be somebody else than the grammar constructor, has to revise the filter and erase all the 'wrong' patterns. Those patterns which our parser should not recognise must not be included in our positive filter. The filter allows the grammar writer to write more general production rules, but the grammar writer should try to keep the number of ambiguous rules as low as possible. If the parser can not accept a sentence because of the filter then the parser will still try to backtrack.

Guessing categories

One of the most tiresome tasks in MT is to construct a lexicon. It would be most welcome if the machine could guess word categories by itself.

The meaning of the individual words is of course needed for MT but not until we make the transfer. When it is time for transfer the parser will already have gathered much information about the word, which narrows down word ambiguity. This information can be added to the surface form of the word and form a more accurate key for the lexical look up. This is the purpose of a Guess Table.

A Guess Table can be viewed as a collection of 'word ends' which are each paired with a list of possible categories (production rule names). It has, in fact, been shown that it is possible to predict word category (in Swedish) to a high degree by just looking at the last letters of the word (Elenius 1991). The Guess Table works by first looking at the last five letters and if (and only if) no guess can be made from that it looks at the last four and so on until it looks at the last letter.

One point in favour of having a Guess Table is that it costs almost nothing in computational power compared with a complete morphological analysis. The Guess Table does not interfere with the lexicon in the sense that words which are listed, in their full form, in the lexicon will not be subject to the Guess Table.

Development of RVG grammar for agreement

Features in Swedish nominal agreement

As a demonstration of Feature Grammar we can look at agreement in Swedish nominals. Swedish nominals (usually) have agreement in the three features number, gender and definiteness. Some examples:

Permissible:

en grön kvarn 'a green mill'	ett grönt barn 'a green child'	gröna kvarnar 'green mills'	gröna barn 'green children'
den gröna kvarn**en** 'the green mill'	det gröna barn**et** 'the green child'	de gröna kvarn**arna** 'the green mills'	de gröna barn**en** 'the green children'
Kvarn**en** är grön. 'The mill is green'	Barn**et** är grönt. 'The child is green'	Kvarn**arna** är gröna. 'The mills are green'	Barn**en** är gröna. 'The children are green'

Nonpermissible:

*grönt kvarn 'green mill'	*gröna kvarn 'green mill'	*grön barn 'green child'	*ett gröna barn 'green child'

A special case occurs when the article is replaced by a genitive determiner. When this occurs the noun has the same form as an indefinite noun and the adjective has the '*a*-ending' form. For example:

sin gröna kvarn !	sitt gröna barn !	sina gröna kvarnar	sina gröna barn
'his/her green mill'	'his/her green child'	'their/his/her green mills'	'their/his/her green children'

The analysis of ordinary agreement is done by comparing the surface form with the three agreement features. This is done for articles, adjectives and nouns. For a more elaborated study on Swedish attributes see Teleman 1969.

Determiners in Swedish

Swedish has three features: number, gender and definiteness which can have the values ±singular, ±utral and ±definite, respectively. This gives rise to 8 possible groups.

Determiners are used to express definiteness but also set demands on agreement in Swedish. In the most common case the determiner is an article but it can be more complex consisting of demonstrative pronouns, words for selection and enumeration, etc. (compare Teleman 1969). The table below shows articles.

Utral determiners

	Singular	Plural
Definite:	Art[+singular, +utral, +definite] **den** 1 'the'	Art[-singular, +utral, +definite] **de** 2 'the'
Indefinite:	Art[+singular, +utral, -definite] **en**/_ 3 / 6 'a'	Art[-singular, +utral, -definite] _ 8

Neutral determiners

	Singular	Plural
Definite:	Art[+singular, -utral, +definite] **det** 4 'the'	Art[-singular, -utral, +definite] **de** 2 'the'
Indefinite:	Art[+singular, -utral, -definite] **ett** /_ 5 / 7 'a'	Art[-singular, -utral, -definite] _ 8

Group 1: art[+singular, +utral, +definite]
Group 2: art[-singular, ±utral, +definite]
Group 3: art[+singular, +utral, -definite]
Group 4: art[+singular, -utral, +definite]
Group 5: art[+singular, -utral, -definite]

The empty article forms 3 groups:

6: art[+singular, +utral, -definite]
7: art[+singular, -utral, -definite]
8: art[-singular, ±utral, -definite]

Condition table for the different determiner groups:

group	*singular*	*utral*	*definite*
1 ART_SG_UTR_DEF	+	+	+
2 ART_PL_DEF	-	?	+
3 ART_SG_UTR_IND	+	+	-
4 ART_SG_NEU_DEF	+	-	+
5 ART_SG_NEU_IND	+	-	-
6 SG_UTR_IND	+	+	-
7 SG_NEU_IND	+	-	-
8 PL_IND	-	?	-

Adjectives in Swedish

Utral adjectives

	Singular	Plural
Definite:	Adj[+singular, +utral, +definite] den gröna kvarnen 1a 'the green mill-the' den gröne jätten 1b 'the green giant-the'	Adj[-singular, +utral, +definite] gröna 1 'green'
Indefinite:	Adj[+singular, +utral, -definite] grön 2 'green'	Adj[-singular, +utral, -definite] gröna 1 'green'

Neutral adjectives

	Singular	Plural
Definite:	Adj[+singular, -utral, +definite] gröna 1 'green'	Adj[-singular, -utral, +definite] gröna 1 'green'
Indefinite:	Adj[+singular, -utral, -definite] grönt 3 'green'	Adj[-singular, -utral, -definite] gröna 1 'green'

Three main groups emerge:

Group 1a: Adj[±singular, ±utral, ±definite]
Group 1b: Adj[+singular, +utral, +definite] [+masculine]
Group 2: Adj[+singular, +utral, -definite]
Group 3: Adj[+singular, -utral, -definite]

The +masculine feature (in group 1b) is especially common in southern Sweden. Swedish can have the following endings on adjectives:

1a *a*-ending
1b *e*-ending (is on the way out because of a spelling reform)
2 0-ending
3 *t*-ending

There also exist some adjectives with the same form in all cases; these are to be treated as exceptions to the rule and functionally belonging to ending 1 because no features can be determined from this group of exceptions.

Group 1 can be further divided into two groups: ADJ_SG_DEF, and ADJ_PL. The purpose of dividing the ending into two groups is that now the groups have distinguishing features. (Note that the underslash ('_') connects the different parts (e.g. SG UTR) into one concept (one 'word') like SG_UTR.) Condition table for the different adjective groups:

group	*singular*	*utral*	*definite*
1) ADJ_SG_DEF	+	?	+
1) ADJ_PL	-	?	?
2) ADJ_SG_UTR_IND	+	+	-
3) ADJ_SG_NEU_IND	+	-	-

When looking at the examples it can be seen that the feature we are most interested in, namely the singular or plural feature, can be derived from agreement phenomena. This shows that agreement phenomena can be useful for finding important features, without the need of a full lexicon in the analysis part of machine translation.

Nouns in Swedish

Utral nouns

	Singular	Plural
Definite:	Noun[+singular,+utral,+definite]	Noun[-singular,+utral,+definite]
	kvarnen 1	kvarnarna 3
	'mill-the'	'mills-the'
Indefinite:	Noun[+singular,+utral,-definite]	Noun[-singular,+utral,definite]
	kvarn 2	kvarnar 4
	'mill'	'mills'

Neutral nouns

	Singular	Plural
Definite:	Noun[+singular,-utral,+definite]	Noun[-singular,-utral,+definite]
	barnet 5	barnen 1
	'child-the'	'children-the'
Indefinite:	Noun[+singular,-utral,-definite]	Noun[-singular,-utral,-definite]
	barn 2	barn 2
	'child'	'children'

5 groups from these 8 can be derived from the word endings.

Group 1: Noun [±singular, ±utral, +definite] with specific endings.
Group 2: Noun [±singular, ±utral, ±definite] with no (specific) ending.
Group 3: Noun[-singular, +utral, +definite] with specific endings.
Group 4: Noun[-singular, +utral, -definite] with specific endings.
Group 5: Noun[+singular, -utral, +definite] with specific endings.

Groups 1 and 2 are further divided in order to get distinguishing features in gender and / or number.
Condition table for the different noun groups:

group	*singular*	*utral*	*definite*
1 NOUN_SG_UTR_DEF	+	+	+
1 NOUN_PL_NEU_DEF	-	-	+
2 NOUN_SG_UTR_IND	+	+	-
2 NOUN_NEU_IND	?	-	-
3 NOUN_PL_UTR_DEF	-	+	+
4 NOUN_PL_UTR_IND	-	+	-
5 NOUN_SG_NEU_DEF	+	-	+

Note that there is no ending that differentiates noun group 2 from adjective group 2. Perhaps this explains why some Swedish speakers actually use the group as if it were an adjective group; especially when the noun is part of a compound noun (*barn matsedel* instead of *barnmatsedel* 'children's menu'). An other, more widespread, explanation is that this is due to the influence of English.

Grammar productions for the nominal

The information gathered about nouns, adjectives and determiners can easily be transferred into lexical production rules (see appendix). The special case (a genitive determiner) is explained by a special category of adjectives which only occur after a genitive determiner. This category has the same surface form as the adjectives ending with *a*.

GENITIVE	+	GEN_ADJ	+	HEAD
if		if +GEN		if -DEF
then [+GEN -DEF]				

The -DEF feature says that *after* a 'GENITIVE' we can only have the indefinite form of the noun and we also know that there is only one form of the adjective that can be used, namely the form with agreement in the GEN feature. This is not to say that the noun *is* indefinite, but the nouns surface form corresponds to the indefinite form. The genitive adjective has agree-

ment in the GEN feature and the noun has agreement in the SG, UTR and DEF features. Barbara Gawrońska pointed out to me that there are some other words in Swedish which act as the 'genitive determiner' I have tried to describe. In some cases perhaps we must accept that Swedish does not have agreement in its nominals.

	...sin	gröna	kvarn
BEFORE	?	+GEN +SG +UTR -DEF	+SG +UTR -DEF
AFTER	+GEN +SG +UTR -DEF	+SG +UTR -DEF	+SG +UTR -DEF
	...sitt	gröna	barn
BEFORE	?	+GEN +SG -UTR -DEF	+SG -UTR -DEF
AFTER	+GEN +SG -UTR -DEF	+SG -UTR -DEF	+SG -UTR -DEF
	...sina	gröna	barn
BEFORE	?	+GEN -SG -DEF	-SG ?UTR -DEF
AFTER	+GEN -SG -DEF	-SG -DEF	-SG -UTR -DEF
	...sina	gröna	kvarnar
BEFORE	?	+GEN -SG -DEF	-SG ?UTR -DEF
AFTER	+GEN -SG -DEF	-SG -DF	-SG +UTR -DEF

In some very special cases there is no agreement between the article and the head noun. Cooper 1986 indicated that agreement in Swedish might not obey the Head Feature Convention proposed in the literature on Generalized Phrase-structure Grammar. (The Head Feature Convention says that a certain set of features, the HEAD features, are the same in the mother node and its head daughter for all local subtrees). The following examples 1-4 illustrate the situation.

1) Den gröna jätte *som jag känner* är snäll
[+DEF] [±DEF] [-DEF]
'The green giant whom I know is kind'

2) Den gröna jätten är snäll
[+DEF] [±DEF] [+DEF]
'The green giant is kind'

3) Den gröna jätten *som jag känner* är snäll
'The green giant, whom I know, is kind'

4) **Den gröna jätte är snäll*
'The green giant is kind'

The problem is that in the context of example 1 there is no agreement between the definiteness of the article *den* and the head noun *jätte*. Note that it is only in certain contexts (when e.g. the subject noun is modified by a relative phrase) this is true as can be seen from example 4. Example 1 is very hard to explain in terms of agreement.

I also want to point out some cases where the congruence depends on other factors such as ellipsis or conflict between natural and grammatical gender.

Permissible:
1) Jordgubbar är goda.
'Strawberries are good.'

2) Jordgubbar är gott.
'Strawberries (that) is good.'

Example 1 refers to the individual berries while in example 2 the reference is to the taste of strawberries in general. Both examples are permissible in Swedish.

Sometimes there is a conflict between natural gender and grammatical gender as in the following two examples (Cooper 1986):

1) Statsrådet är sjuk.
'The minister (he) is ill.'

2) Statsrådet är sjukt.
'The minister (it) is ill.'

Technical discussion

Outline

Register vector grammar, RVG, is equivalent to a finite automaton in computational power. But RVG benefits from the use of local parallelism which allows the machine to check the values of many ternary features in one go. This means that RVG is more efficient and compact than an ordinary finite automaton (Blank 1985, 1989). The features are stored in vectors and backtracking is supported by keeping information in registers.

Vectors

Ordinary finite automata represent states and categories as separate symbols – nodes and arches in transition diagrams. According to Blank, most modern syntactic formalisms abstract over category symbols, including non-terminals and tree structures. RVG abstracts over state symbols (Blank 1985, 1989) through the use of vectors of ternary valued logical units. These vectors contain a set of features that can be 'on', 'off' or 'either/or'. Vectors help to reduce the redundancy in the grammar description. In this specific implementation, a vector can hold 80 features. This size is chosen on the ground that it is unlikely that we will need more features in our RVG-grammar (reality might demand more features).

Registers

RVG uses registers to keep track of alternative states. RVG guarantees linear time because it only has a limited number of registers. Structural ambiguities are supported by allowing the machine to return to machine states stored in these registers. Note that the number of registers never

grows, instead RVG re-uses registers thereby forgetting many but not all ambiguities. In my implementation, registers are a part of the backtracking handling.

Comparison with a finite automaton

The definition of RVG is equivalent to that of a finite non-deterministic automaton (Blank 1985, 1989). The RVG automaton is a 5-tuple (S, C, I, F, T) where S is a finite set of machine states, C is a finite set of categories, I is the initial state, F is the final state and finally T is a transition relation mapping S x C onto S. The difference lies in how S and T, the states and transitions, look. T corresponds to the 'if-then' rules of the grammar.

Ternary logic

RVG can handle ternary logic; 'on', 'off' and 'either/or'. Machine state is represented by a vector of ternary logical values where every value points to a (grammatical) feature. **f** is a vector, $[f_1, f_2, \ldots f_n]$, of ternary valued features. The transition relation T uses two functions, match and change (Blank 1985, 1989). To be able to express match and change effectively I want to express them in terms of fast binary operations, such as **and**, **or**, **xor**. These binary operations are performed in parallel, on the number of binary units contained in a machine word. I will show one way to accomplish this; by simulating match and change in binary logic. The three values '+' (on), '-' (off) and '?' (either or) are coded as 01, 10, 00 respectively. f_i and g_i are ternary values belonging to the vectors **f** and **g**.

The match function

Match checks if two vectors are compatible and returns an ordinary logic value (true or false). Match is the function associated with the 'if-part' of the production rule.

e.g. match ([+ - ?], [? - +]) is true, because '?' matches everything, while match([+ + +], [+ ? -] is false, as '+' and '-' do not match.

For separate values match is defined as:
match(f, g) =
TRUE if f=g or f=? or g=?
FALSE otherwise.

For two vectors **match(f, g)** is defined as:
match(f, g) =
TRUE if match(f_i, g_i) = TRUE for all relevant i
FALSE otherwise.

Match is a projection from a ternary logic value onto an ordinary Boolean value. As match is symmetric it can be constructed out of symmetric functions, such as the binary functions **xor** and **and**, with the aid of a truth table.

Truth table 1: Defining match for Boolean logic
(**xor** is a binary exclusive or; **and** is a binary and.)

f' g'	f' **xor** g' **and** f'	f g	match(f,g)
00 00	00	? ?	TRUE
01 00	00	+ ?	TRUE
10 00	00	- ?	TRUE
00 01	00	? +	TRUE
01 01	00	+ +	TRUE
10 01	10 !!	- +	FALSE
00 10	00	? +	TRUE
01 10	01 !!	+ -	FALSE
10 10	00	- -	TRUE

From the table it can be noted that when match is true the formula (f' **xor** g' **and** f') has the value 00. The match of two vectors is true in the case that all features f_i, g_i gave rise to a 00 value; this means that the machine can compare with the nil vector **0** = [00 ... ... 00]:

Match(f, g) := (**0** = ((**f xor g**) **and f**)).

The change function

Change is the function that takes the state of the machine from one state to another. It is connected with the then-part of a production rule. Change takes two vectors and creates a third. E.g.,

change([+ - ? - + ?], [- + + ? + ?]) produces the vector [- + + - + ?].

For separate values change is defined as:

change(f, g) = g if g≠?
f if g=?.

For two vectors **change(f, g)** is defined as:

change(f, g) = change(f_i, g_i) for all relevant i.

Truth table 2: Comparison between change and Boolean **or**

The truth table for change(f, g)			The truth table for Boolean **or**			
f	g	result	f	g	result	
00	00	00	00	00	00	
01	00	01	01	00	01	
10	00	10	10	00	10	
00	01	01	00	01	01	
01	01	01	01	01	01	
10	01	01	10	01	11	– !
00	10	10	00	10	10	
01	10	10	01	10	11	– !
10	10	10	10	10	10	

A wrong result is characterized by the value 11. This might be corrected with a boolean **and** with the g-value.

r	g	r **and** g
11	01	01
11	10	10

This is incorrect if g = 00 but it works as it should for the other values. In this case g is known but f can take on different values. In order to handle the case g = 0 a mask is constructed as follows:

IF g=00 THEN mask:=11 ELSE mask:=g.

Now, the formula (f **or** g **and** mask) covers all cases.

As every ternary unit in a vector does not affect the other units in the same vector, the function can be carried out in parallel on two full vectors.

The fourth bit pattern 11, call it '!', can be used as follows:

match(X, !) = TRUE if X=? else FALSE.
change(X, !) = ?.

This makes it possible to check that a value is '?' and change a '+' or '-' into a '?'.

Complexity

One of my goals is to support Blank's claim that the RVG-parser has linear time complexity. The key to why the machine can operate in linear time is limited backtracking. In (Blank 1989) there is a similar but perhaps more

formal analysis. First note that RVG analyzes left to right and depth first. In every step an alternative might be saved in the backtracking list. The number of alternatives for any word is determined by the grammar and by the lexicon. Both the grammar and the lexicon are fixed in size and thus the maximum number of alternatives for any word is constant. The number of times a word, in the worst case, can be analyzed is determined by two things: firstly, by how ambiguous the word can be and secondly, by how long the backtracking trace |B| is. The ambiguity of a word is determined by how many productions that can take the word; in the worst case all of the productions. The ambiguity is thereby proportional to the length of the grammar |G|. By allowing the backtracking to forget alternatives, as described in the previous section, there exists a constant, K, such that every word cannot be passed more than K times in an analysis; where K depends on |G| and |B|. Suppose that every word has |G| alternatives and that the machine is forced to backtrack when the backtracking list is filled. Every word might then be tested, at most, $|G|^{|B|}$ times. The time needed to go through all the words is proportional to $|G|^{|B|}N$, i.e. a constant term times the number of words. The constant term is most affected by the length of the backtracking list |B|. In order to assure that the constant term cannot dominate the time consumption the length of the backtracking list must be limited. Limiting the length of the backtracking list makes the RVG-algorithm a linear time algorithm for every grammar. The fact that the machine does not handle all possibilities is as it should. If it had been able to find all paths in any grammar then it could also find the shortest (in linear time). Finding the shortest path in a graph (graphs can be described by grammars) is comparable to 'The Traveling Salesman Problem' which is an intractable problem (Hopcroft & Ullman 1979) that cannot be solved with an algorithm in less than polynomial time today. The dependency $|G|^{|B|}$ is not characteristic of the RVG-algorithm; all algorithms that use backtracking suffer from the same, or similar, problems. RVG can guarantee to stop within a short time if the length of the backtracking list is short enough. But the greatest advantage of RVG is that it uses an iterative method which does not need more than a small memory space which is known from the start.

Conclusion

For a parser to be able to guarantee low time consumption it must limit the backtracking trace. This means that possible solutions to the parse problem are lost. RVG is an example of a limited parsing algorithm, there will be

similar effects in other parsing systems if the backtracking is limited to a small fixed depth. The biggest advantages with RVG consist in the fact that it uses an effective, iterative algorithm which only demands a small memory space and the fact that it uses a very efficient ternary logic which allows for micro-parallelism in the match and change functions. RVG is also a very robust algorithm in the sense that it can guarantee to stop within a time interval known from the start of the analysis. The parser is also able to deal properly with many linguistic problems without the need for large resources.

Finally, the grammar grows very slowly (Blank 1989) compared to other formalisms thanks to ternary logic. A disadvantage of RVG is that the notation used in the grammar is original and quite difficult to grow accustomed to.

Although this parser does not attempt to make an exhaustive natural language analysis, it can provide a fast (but admittedly crude) analysis of many cases in natural language. Remembering that natural language analysis, in general, demands context-free power, we must be ready to sacrifice something to gain the speed of finite state design. The sacrifice is that we cannot analyse infinite centre embedding and perhaps a correct assignment of grammatical roles is also too difficult to handle at once. Other designs will most probably have problems with the fact that the computer has limited memory and limited speed and it cannot, because of these limitations, analyse all sentences which are in theory possible for it to analyse. The usefulness of our RVG parser is somewhat uncertain because not many grammars have been written using our RVG parser and it is not tested on any larger amounts of texts. Johansson 1991 presents some examples of RVG grammar for handling unbounded dependencies in English.

The grammar for agreement in Swedish might be further elaborated into an 'agreement checker' for Swedish. It would probably be very welcome in a word processor, since current 'spelling checkers' do not handle agreement mistakes. The RVG parser could also be useful as a pre-processor in large MT systems (because of its speed and the little cost in computer power) on the way to the construction of the functional structure which a system like SWETRA (Sigurd et al. 1990) uses to translate from.

References

Blank, G.D. 1985. 'Register Vector Grammar; A new kind of finite automaton'. *Proceedings of the Ninth International Conference on Artificial Intelligence* (UCLA Aug 18-23, 1985), 749-756.

Blank, G.D. 1989. 'A finite and real-time processor for natural language'. *Communications of the ACM*, October 1989 32.10.

Cooper, R. 1986. 'Swedish and the Head-Feature Convention'. *Topics in Scandinavian Syntax*, 31-52. Dordrecht: Reidel Publishing Company.

Elenius, K. 1991. 'Comparing a connectionist and a rule-based model for assigning parts-of-speech'. *Speech Transmission Laboratory Quarterly Progress and Status Report. April 15, 1991.* Dept. of Speech Communication & Music Acoustics, Royal Institute of Technology, Stockholm.

Hopcroft, J. & J. Ullman. 1979. *Introduction to automata theory, languages, and computation.* Addison Wesley.

Johansson, C. 1991. *En implementation av en parser för naturligt språk.* Dept. of Computer Science, Lund University.

Sigurd, B. 1990. 'Implementing the generalized word order grammar of Chomsky and Diderichsen'. *Coling 90*, vol. 2, 336-40. Helsinki University.

Sigurd, B., M. Eeg-Olofsson, B. Gawrońska-Werngren et al. 1989. *Lingvistik och fonetik på dator. Praktisk Lingvistik 12.* Dept. of Linguistics, Lund University.

Sigurd, B., M. Eeg-Olofsson, B. Gawrońska-Werngren & P. Warter. 1990. 'SWETRA – a multilanguage system for special purposes. Documentation and progress report'. *Praktisk Lingvistik 14.* Dept. of Linguistics, Lund University.

Teleman, U. 1969. *Definita och indefinita attribut i nusvenskan.* Lundastudier i nordisk språkvetenskap. Lund: Studentlitteratur.

The parser is developed on a Macintosh and the compiled program is available for Macintosh computers on one diskette.

Acknowledgment

Thanks to Bengt Sigurd for support, patience and remarks; to Sheila Dooley Collberg for drawing my attention to the article by Cooper, to Merle Horne and Barbara Gawronska for comments.

Appendix

This is an example of the input files that the parser needs. It needs one *Lexicon* file, one *Guess Table* file, one *Filter* file, one *Grammar* file and one *Input* file. All files must be in **TEXT** format.

The Lexicon

• This is the actual lexicon that was used.
• It contains only words which are not properly handled
• by the Guess Table. In this case there is only one
• alternative for each defined word in the lexicon.

kvarn (NOUN_SG_UTR_IND)	barn(NOUN_NEU_IND)
den (ART_SG_UTR_DEF)	de (ART_PL_DEF)
en (ART_SG_UTR_IND)	det(ART_SG_NEU_DEF)
ett (ART_SG_NEU_IND)	grön (ADJ_SG_UTR_IND)
är (COP)	. (close)

The Guess Table

• The Guess Table defines which categories are associated
• with what word endings.

Ending	*Categories*
en	(NOUN_SG_UTR_DEF
	NOUN_PL_NEU_DEF) • 2 alternatives
arna	(NOUN_PL_UTR_DEF)
ar	(NOUN_PL_UTR_IND)
et	(NOUN_SG_NEU_DEF)
t	(ADJ_SG_NEU_IND)
a	(ADJ_PL
	ADJ_SG_UTR_DEF
	ADJ_SG_NEU_DEF) • 3 alternatives

The Filter

• The filter is used to construct patterns out of pattern
• building productions.

```
TOKENS
  NOM  COP              • These are 'structure building' productions
END
```

• These two patterns are the two valid patterns:
(NOM) (NOM COP NOM)

A sample run on the computer

```
• All features which are used in the
• grammar must be declared first in the
• grammar file. The '•' is a comment
•  marker that makes the rest of the line
• a comment.
DECLARE
N V
Art     • Article
Head    • Head word
SG UTR DEF          • Congruence
                    • Features
NP1                 • In nominal 1
END • of declare
MACROS
• Initialize is the start vector
Initialize
 -N  -V  !SG !UTR !DEF -HEAD
 -ART -NP1
• Feature values to start a nominal
ClearNominal
 +N  -Art -HEAD
• No changes to the feature values
Nothing
END • of macros
• Begin non-lexical productions
• A non-lexical production
• to open a box for a nominal
P NOM N
IF     -N  -V  -NP1
THEN %ClearNominal +NP1
• % means use a macro
• To close an open nominal
P  NEND N
IF      +N  +NP1
THEN  -N  +NP1
• An empty 'article' that handles plural
• congruence with an adjective
P PL_INDEF N
IF  +N  -SG
  -ART   • no article found
THEN -SG -DEF !UTR
    +ART • this is an article
• Handle singular congruence with an
• adjective
P SG_INDEF_UTR N
IF  +N  +SG +UTR -ART
THEN +SG -DEF +UTR +ART
P SG_INDEF_NEU N
IF  +N  +SG -UTR -ART
THEN +SG -DEF -UTR +ART
• Begin lexical productions
• The InitFinal production
P close I
IF -N  -V • All boxes must be off
THEN %Initialize
• A copula verb
P COP L
IF -N          • not in a nominal
THEN  -NP1   • prepare for NOM
• The various articles.
• There is only one article
• in a nominal  (-ART -> +ART)
P ART_SG_UTR_DEF L
IF     +N  +SG  +UTR  +DEF -ART
THEN +N  +SG  +UTR  +DEF +ART
P ART_PL_DEF L
IF     +N  -SG  +DEF  -ART
THEN +N  -SG  +DEF  +ART
P ART_SG_UTR_IND L
IF     +N  +SG  +UTR  -DEF  -ART
THEN +N  +SG  +UTR  -DEF  +ART
P ART_SG_NEU_DEF L
IF      +N  +SG  -UTR  +DEF  -ART
THEN +N  +SG  -UTR  +DEF  +ART
P ART_SG_NEU_IND L
IF      +N  +SG  -UTR  -DEF  -ART
THEN +N  +SG  -UTR  -DEF  +ART
• The various adjectives.
• There must be some kind of article in
•  front of an adjective (+ART)
P ADJ_SG_DEF  L
IF     +N  +SG  ?UTR  +DEF +ART
THEN +N  +SG  +UTR  +DEF
P ADJ_PL  L
IF     +N  -SG  +ART
THEN +N  -SG
P ADJ_SG_UTR_IND L
IF      +N  +SG  +UTR  -DEF  +ART
THEN +N  +SG  +UTR  -DEF
P ADJ_SG_NEU_IND L
IF      +N  +SG  -UTR  -DEF  +ART
THEN +N  +SG  -UTR  -DEF
• The various nouns.
• There is only one head noun
•  in a nominal (-HEAD -> +HEAD)
P NOUN_SG_UTR_DEF L
IF      +N  +SG +UTR +DEF -HEAD
THEN  +N  +SG +UTR +DEF +HEAD
P NOUN_PL_NEU_DEF  L
```

```
IF      +N -SG -UTR +DEF -HEAD
THEN +N -SG -UTR +DEF +HEAD
P NOUN_SG_UTR_IND L
IF      +N +SG +UTR -DEF -HEAD
THEN +N +SG +UTR -DEF +HEAD
P NOUN_NEU_IND L
IF      +N    -UTR -DEF -HEAD
THEN +N    -UTR -DEF +HEAD
P NOUN_PL_UTR_DEF L
IF      +N -SG +UTR +DEF -HEAD
THEN +N -SG +UTR +DEF +HEAD
P NOUN_PL_UTR_IND L
IF      +N -SG +UTR -DEF -HEAD
THEN +N -SG +UTR -DEF +HEAD
P NOUN_SG_NEU_DEF L
IF      +N +SG -UTR +DEF -HEAD
THEN +N +SG -UTR +DEF +HEAD
```

```
• The Input to the machine
• (a text file):
En grön kvarn.
Den gröna kvarnen.
Kvarnen är grön.
De gröna kvarnarna.
Kvarnarna är gröna.
Kvarnen är grön.
Det gröna barnet.
Barnet är grönt.
Barnet är gröna.
```

```
• The Output:
•    (a text file)
NOM
   ART_SG_UTR_IND:=EN
   ADJ_SG_UTR_IND:= GRÖN
   NOUN_SG_UTR_IND:= KVARN
NEND
   CLOSE:=.
ALT_NO
• = No more alternatives
```

```
NOM
   ART_SG_UTR_DEF:=DEN
   ADJ_SG_UTR_DEF:= GRÖNA
   NOUN_SG_UTR_DEF:=KVARNEN
NEND
   CLOSE:=.
ALT_NO
```

```
NOM
   NOUN_SG_UTR_DEF:=KVARNEN
NEND
   COP:=ÄR
NOM
SG_INDEF_UTR
   ADJ_SG_UTR_IND:= GRÖN
NEND
   CLOSE:=.
ALT_NO
```

```
NOM
   ART_PL_DEF:=DE
   ADJ_PL:=GRÖNA
   NOUN_PL_UTR_DEF:=
       KVARNARNA
NEND
   CLOSE:=.
ALT_NO
```

```
NOM
   NOUN_PL_UTR_DEF:=
       KVARNARNA
NEND
   COP:=ÄR
NOM
PL_INDEF
   ADJ_PL:=GRÖNA
NEND
   CLOSE:=.
ALT_NO
```

```
NOM
   NOUN_SG_UTR_DEF:=
       KVARNEN
NEND
   COP:=ÄR
NOM
SG_INDEF_UTR
   ADJ_SG_UTR_IND:= GRÖN
NEND
   CLOSE:=.
ALT_NO
```

```
NOM
   ART_SG_NEU_DEF:=DET
   ADJ_SG_NEU_DEF:= GRÖNA
   NOUN_SG_NEU_DEF:= BARNET
NEND
   CLOSE:=.
ALT_NO
```

```
NOM
   NOUN_SG_NEU_DEF:= BARNET
NEND
   COP:=ÄR
NOM
SG_INDEF_NEU
   ADJ_SG_NEU_DEF:= GRÖNT
NEND
   CLOSE:=.
ALT_NO
```

```
NO
  *Barnet är gröna.
```

Lund University, Dept. of Linguistics
Working Papers 38 (1991), 155-67

On the Starting up of UTFÖR

Eva Magnusson and Kerstin Nauclér

In this paper we will describe UTFÖR (Utveckling, utprövning och utvärdering av träningsmetoder för att förebygga läs- och skrivsvårigheter hos språkstörda barn; 'Development, trial and evaluation of training methods in order to help prevent reading/writing problems in language-disordered children'), a research project aiming to prevent later reading and writing problems by developing training methods for language-disordered preschool children. The effect of such training methods on the development of reading and writing/spelling will be evaluated in grade 1. We will start by giving the background for the project and reviewing previous research in the field. Then we will outline the procedure and describe the work done during the first year of the project.

Background

For a long time researchers have been busy trying to find the causes of reading and writing problems, and dedicated teachers have been working hard at adapting their teaching methods to the latest research findings. In spite of all these efforts, the number of poor readers and spellers is said to have increased. An important reason for this lack of success is that the relationship found between reading/writing and certain other abilities has been given a causal interpretation. Other poorly-developed abilities that have been observed to co-occur with reading and writing problems have mistakenly been identified as the cause of these difficulties and have consequently been considered as something that should be trained in order to eliminate the reading and writing problems. However, other deficiencies can just as well be an effect of the reading and writing problems as a cause for them. A third possibility is that they, as well as the reading and writing problems, may be the manifestations of a common underlying factor.

It is not possible to identify the causes of reading and writing problems by studying other deficiencies that appear simultaneously in individuals who have developed into poor readers and writers. Nor is it possible to trace the