Journal of Language and Linguistics
Volume 5 Number 1 2006
ISSN 1475 - 8989

Some 'counting' properties of natural languages and their treatment in the AMS framework

László Drienkó
ELTE University, Budapest

Abstract

The present paper attempts to address the issue of interaction of language and other cognitive capacities. In particular, we would like to highlight some parallelisms between certain linguistic constructions and the basic mathematical capacity to enumerate cardinal numbers. Such parallelisms will be drawn within the generalised agreement framework as outlined in Drienkó (2004a, b). They will be explicitly represented by agreement strategy ABORDER. We discuss such constructions as Old Georgian Genitive, Chinese number names, Dutch Coordination Phrases, and enumeration of digit expressions (numbers). A model for the word-digit interface will also be given.

1. Introduction

The present paper attempts to explicitly address the issue of interaction of language and other cognitive capacities. In particular, we would like to highlight some parallelisms between certain linguistic constructions and the basic mathematical capacity to enumerate cardinal numbers. Such parallelisms will be drawn within the generalised agreement framework as outlined in Drienkó (2004a, b). They will be explicitly represented by agreement strategy ABORDER.

Agreement strategies play a very important role in the Agreement Mapping-System (AMS) model. Their task is to specify the way how agreement properties of linguistics elements should be checked. A strategy can be seen as a mapping (function) from sequences of input elements to two dimensional arrangements – strategy matrices[1] – of attribute-value structures (AVS's). More formally, SF is a strategy iff

[1] We shall also use the term 'checking table' for 'strategy matrix'.

(1.1)

SF $\in$ ((KEL $\cup$ UEL)* $\rightarrow$ **M**), where **M** is the set of $n \times m$ matrices such that $1 \leq n < \infty$, $1 \leq m < \infty$, for any matrix element sm_{ij} of strategy matrix SM$\in$ **M,** $1 \leq i \leq n$, $1 \leq j \leq m$: $sm_{ij} \in$ ALLAVS.

KEL: a collection of AVS's representing 'known' lexical elements, i.e. elements whose values are specified for all (relevant) attributes,

UEL: a collection of AVS's representing 'unknown' lexical elements, i.e. elements with missing attribute values

ALLAVS: a collection of all possible AVS's

For instance, input sequence (1.2) may be mapped onto strategy matrix (1.3), representing a FIRST-TO-FIRST arrangement.

(1.2) $a_1, a_2, a_3, b_1, b_2, b_3$

(1.3)

element:	a	b
$sequence_1$	a_1	b_1
$sequence_2$	a_2	b_2
$sequence_3$	a_3	b_3

The checking table in (1.4) corresponds to strategy LAST-TO-FIRST for the AVS's in (1.2)

(1.4)

element:	a	b
$sequence_1$	a_1	b_3
$sequence_2$	a_2	b_2
$sequence_3$	a_3	b_1

The rows (sequences) of strategy matrices represent couplings of input elements with respect to agreement checking: the attribute values of the elements in the same sequence must be the same for the attribute specified by a corresponding agreement constraint. If, e.g., the *a*'s and *b*'s in (1.2) represented nouns and verbs respectively, an agreement constraint might require that each noun should share some, say, subcategorization feature with a corresponding verb. That which verb corresponds to which noun would be specified by the agreement strategy prescribed by the agreement constraint in question. Strategy FIRST-TO-FIRST, as sketched in (1.3) would then yield a 'cross-serial dependency' effect. Cf. Drienkó (2004a : Section 2.1)

The notion of agreement can be generalised beyond just requiring the identity of some feature(s) between specified linguistic elements. It is also possible to consider values of such elements collectively, i.e. to consider the configuration of attribute

values as a kind of 'meta feature' capable of taking part in what can be termed a kind of 'meta agreement'. For instance, we may demand that the PHF (phonological form) values of a sequence of linguistic elements inputted to the mapping system should constitute a certain sequence, i.e. that the, say, phonological order (PHF_ORDER) value of the input sequence should agree with a predefined value. This type of agreement is represented by strategy ORDER in the Agreement Mapping-System framework. Agreement constraint (1.5), for instance, specifies the phoneme sequence '*t o p*'.

(1.5) 1 2 3 PHF == t o p ORDER

The checking table is similar to that of strategy FIRST-TO-FIRST. The checking process, however differs in that elements in a sequence are considered not one by one, but rather their specified values must constitute the sequence given on the right-hand side of the '==' in the attribute part of the constraint. We represent the checking table for (1.5) with (1.6).

(1.6)

element:	1 2 3	s(PHF_ORDER)
$sequence_1$	$a_1b_1c_1$	s(PHF_ORDER=top)

To be consistent with our definition in (1.1), s(PHF_ORDER) denotes an AVS. This AVS has only one attribute, PHF_ORDER. The PHF_ORDER (meta)value '*top*' is assigned to *s* by constraint (1.5)
It is also possible, just like in (1.3) or (1.4), that there are several sequences made up by the input elements referred to in the constraint, i.e. there can be several occurrences of *a, b,* and *c* type elements respectively corresponding to 1, 2, and 3 in (1.5) or (1.6). The checking table is then given as (1.7).

(1.7)

element:	1 2 3	s(PHF_ORDER)
$sequence_1$	$a_1b_1c_1$	s(PHF_ORDER=top)
$sequence_2$	$a_2b_2c_2$	s(PHF_ORDER=top)
…		
$sequence_n$	$a_nb_nc_n$	s(PHF_ORDER=top)

Note that the PHF_ORDER value is a constant part of constraint (1.5). It is predefined by the creator of the system. Nevertheless, it is also possible to predefine an algorithm (or function) which 'generates' a sequence of attribute values (a 'meta value') that must be matched by the input. We term the corresponding strategy as ABORDER (across the board order).
Strategy ABORDER differs from ORDER in that it regards the sequence of attribute values of all input elements mapped on the representational elements specified in the constraint as a single sequence and this sequence is matched against a sequence of values generated by a function specified by the constraint. Table (1.9) represents the strategy matrix for constraint (1.8).

(1.8) 1 2 3 ATTR == :: ABORDER :F_c:

(1.9)

element:	1 2 3	s(ATTR_ABORDER)
$sequence_1$	$a_1 \ldots c_n$	s(ATTR_ABORDER =F_c(IS))

The *ATTR == ::* part of the constraint indicates that the value for attribute ATTR must be provided by a corresponding function, the *ABORDER : F_c:* part specifies this function as F_c
together with the strategy to be used, ABORDER . As usual, numbers in the constraint refer to representational elements in the corresponding linguistic pattern.

Checking table (1.9) tells us that the ATTR_ABORDER metavalue of input sequence IS consisting of input elements $a_1 \ldots c_n$ as mapped onto representational elements 1, 2 and 3 of the given linguistic pattern should be identical with the value provided by F_c for IS.

Employing functions through strategy ABORDER provides a way of incorporating some 'counting' properties of natural languages into the Agreement Mapping-System model. For instance, we may demand that an element should be represented in the input sequence by a growing number of occurrences. We may prescribe, e.g., that any time element *'a'* representing phoneme *'t'* occurs after a *'c'* element, it be followed by one more *'a'*s than at the previous occurrence. Cf. string S$ in (1.10).

(1.10).

S$: t o p t t o p t t t o p…

A straightforward way to attempt to capture the regularity in (1.10) is to look for an algorithm capable of generating such strings. Such an algorithm is sketched in (1.11). The corresponding strategy matrix is (1.12)

(1.11)

F_{PREFIX_t}

String$= "op"
S$=""
Do until condition on length of output is fulfilled:
String$ = "t" + String$
S$=S$+String$

(1.12)

element:	1 2 3	s(PHF_ABORDER)
$sequence_1$	$a_1 \ldots c_n$	
s(PHF_ABORDER=F_{PREFIX_t}(n)="topttop…")		

Note that linguistic patterns in our model without agreement constraints and, consequently, without strategies could only handle a much narrower range of phenomena. It would be possible to allow recursion, but 'counting' properties would

be lost. The system would not be able to differentiate, e.g, between the sequences in (1.13).

(1.13) t o p t t o p t t t o p ….
t t t o p t t o p t t o p ….
t o p t o p t t o p ….

2. Natural language constructions with counting properties

2.1 Old Georgian Genitive

Michaelis and Kracht (1997) cite Old Georgian genitive examples by Boeder (1995) to show that there are natural languages that mildly context sensitive grammar formalisms are not capable of generating.[2] The relevant construction is exemplified by (2.1).

(2.1)

govel-i	igi	sisxl-i	saxl-isa-j	m-is	Saul-is-isa-j.
all-Nom	Art-Nom	blood-Nom	house-Gen-Nom	Art-Gen	Saul-Gen-Gen-Nom

'all the blood of the house of Saul'

Disregarding irrelevant lexical categories, and phonological features we can formulate the basic genitive construction as in (2.2).

(2.2)
N_0-Nom N_1-Gen-Nom N_2-Gen-Gen –Nom ... N_k-Gen^k-Nom

As can be seen, the morphemes for this variant of Old Georgian genitive observe the regularity shown in (2.3).

(2.3) ABC ABBC ABBBC … AB^kC

That is the *i*th occurrence of the sequence *AB…C* contains one more *B*'s than the *(i-1)th* for any *i* such that $0 < i \leq k$.
Sequence (2.3) can be generated by the simple algorithm in (2.4).We shall refer to (2.4) as $Function_{OG}$, or F_{OG} for short.

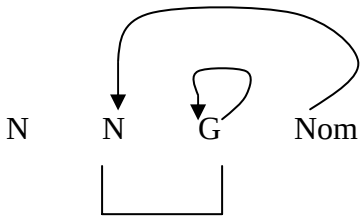
(2.4)
F$_{OG}$
S$=""

[2] Bhatt, R., and Joshi, A. (2004) gives another interpretation of Boeder's data which does not seem to exhibit the property emphasised by Michaelis and Kracht. Since Michaelis and Kracht's interpretation, defying mild context-sensitivity, is more challenging we rely on their analysis.

```
insert$ =""
        For i=1 to k
        insert$ = insert$+"B"
        S$= S$ + "A" + insert$ +"C"
        Next i
```

Drienkó (2004a :Sections 2.3, 2.4) used the ORDER strategy to ensure matching of the order of the attribute values of input elements with a predefined (arbitrary) sequence in connection with Arabic morphology, and Hungarian vowel harmony. However, such a predefined sequence can be thought of as a result of some computational predefining mechanism which can be referred to as an algorithm, or equivalently in our terminology, a function. $\mathbf{F}_{OG}$ in (2.4) can be thought of as such a predefining mechanism. It generates (2.3) and agreement checking will decide whether or not (2.3) and a given sequence of attribute values of input elements are the same.
Unlike in the case of strategy ORDER, where an agreement checking iteration only considers a single sequence – of possibly several – of input elements mapped on a recursive representational element, in the present case all sequences are considered. We call this strategy ABORDER (across the board order).
In much the same fashion, we can use functions like F_{OG} to predefine combinations of category values. Cf. pattern (2.5).

(2.5)



CAT==::
(ABORDER : F_{OG}:)

The '::' stands for the sequence of category values (the CAT_ABORDER metavalue), which sequence is to be provided by the F_{OG} function. For the present example, F_{OG} would provide the representation of an Old Georgian genitive phrase containing *k* nouns. Notation '(ABORDER : F_{OG}:)' explicitly indicates that strategy ABORDER is followed and function $\mathbf{F}_{OG}$ is called.

As (2.4) reflects, $\mathbf{F}_{OG}$ does not spoil the linearity of the agreement checking process, since it should be called only once, the insertion of elements is done in *k* steps, and for any input length, n, $k < n$.

2.2. Chinese number names

Radzinski (1991), in proving that natural languages cannot be generated by multicomponent tree adjoining grammars (MCTAGs), refers to a sublanguage of Chinese numbers names. The words *wu (five)* and *zhao (trillion)* are combined in

such a way that every occurrence of *wu* is followed by a sequence of *zhao'*s, and every sequence of *zhao'*s is shorter (contains less *zhao*'s) than the previous one. Cf. (2.6).

(2.6)

wu zhao$^{k(1)}$ wu zhao$^{k(2)}$ ... wu zhao$^{k(m)}$, k(1) > k(2) > .... > k(m)

We can apply function $\mathbf{F}_{WZ}$ in (2.7) to the situation.

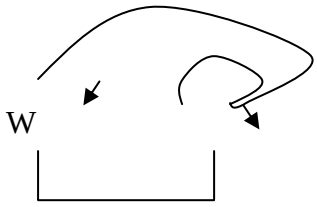
(2.7)

$\mathbf{F}_{WZ}$

S$=""
For i=1 to m-1
If zhao$^{k(i)}$ ⊃ zhao$^{k(i+1)}$ then
S$=S$+ "wu" + zhao$^{k(i)}$
Else
S$=S$+ ""*
Next
S$=S$+ "wu" + zhao$^{k(m)}$

S$, as generated by $\mathbf{F}_{WZ}$, is actually an identical image of the input sequence if the input is a legal 'sentence' of the sublanguage of Chinese number names. Consequently, agreement of input and S$ will evidently follow. If the *i*th sequence of *zhao*'s in the input does not contain the *(i+1)*th, i.e. input is an illegal sentence, $\mathbf{F}_{WZ}$ inserts a 'non-compatible' symbol, say "*", into string S$, thus non-agreement will be trivial.[3]
Pattern (2.8) can handle the 'wu-zhao' phenomenon.

(2.8)



PHF ==::
(ABORDER : F_{WZ}:)

2.3 Dutch Coordination Phrases

Groenink (1996) uses a sublanguage of Dutch coordination phrases to show that Dutch is a non-MCTAL (multicomponent tree adjoining language). The relevant fragment of the language is characterised in (2.9).

[3] Naturally, inserting a non-matching symbol is just one way of many others to block agreement.

(2.9)

dat Jan Piet Marie Fred^{k} (hoorde leren^{k} uitnodigen)$^{+}$ en (zag leren^{k} omhelzen)

For *k=1*, and *k=2*, e.g., we can have (2.10) and (2.11) respectively.

(2.10)

k=1
dat Jan Piet Marie Fred (hoorde leren uitnodigen) $^{+}$ en zag leren omhelzen
'that Jan heard Piet teach Marie to invite Fred and saw Piet
teach Marie to embrace Fred'

(2.11)

k=2
dat Jan Piet Marie Fred Fred (hoorde leren leren uitnodigen) $^{+}$ en (zag leren leren omhelzen
'that Jan heard Piet teach Marie to teach Fred to invite Fred and saw Piet teach Marie to teach Fred to embrace Fred'

The Kleene '$^{+}$' means that the bracketed verb phrase can be repeated an arbitrary number of times, except zero. For explicitness, we write (2.9) as (2.12).

(2.12)

dat Jan Piet Marie Fred^{k} (hoorde leren^{k} uitnodigen)^{m} en (zag leren^{k} omhelzen)

Thus *k=1* and *m=2* could yield something like (2.13).

(2.13)

k=1, m=2
dat Jan Piet Marie Fred hoorde leren uitnodigen, hoorde leren uitnodigen en zag leren omhelzen

Although the grammatical case of the nouns is not explicitly indicated, we assume *Piet*, *Marie*, and *Fred* to be accusative here. Then the order of ARG2CASE values in (2.12) is as in (2.14)[4].

(2.14)

dat Jan Piet Marie Fred^{k} (hoorde leren^{k} uitnodigen)^{m} en (zag leren^{k} omhelzen)
ACC ACC ACC^{k} (ACC ACC^{k} ACC)^{m} (ACC ACC^{k} ACC)

Nevertheless, there may be cases where explicit case-marking determines grammaticality. Consider the Hungarian sentences in (2.15) and (2.16).

(2.15)

Jani Petit Marinak látta segíteni.

[4] In the AMS model, subcategorization is effected by agreement relations between the verb and its arguments. *ARG1CASE=nom* for the verb prescribes that its first argument, the subject, must be in nominative case, i.e. it must also have the *ARG1CASE=nom* feature. Further arguments of verbs are represented by ARG2CASE, ARG3CASE, etc.

Jani-NOM Peti-ACC Mari-DAT see-3rd –SG-PAST-DEF help-INF
'Jani saw Peti help Mari'

(2.16)
?Jani Petit Marinak Fricit látta segíteni meghívni
Fred-ACC help-INF invite-INF
'Jani saw Peti help Mari invite Fred'

As the data show, the verb *segíteni ('help')* requires dative, *látni ('see'),* and *meghívni ('invite')* require accusative. We would not dare to make an explicit statement as to the grammaticality of the Hungarian analogue (2.17) of (2.9), but we suspect that memory training can enhance its intelligibility.

(2.17)
Jani Petit Marinak^{k} Fricit (látta segíteni^{k} meghívni)^{m} és (hallotta segíteni^{k} elzavarni)
(hallotta: hear-3rd –SG-PAST-DEF, elzavarni: chase-away-INF, és: and)

The order of ARG2CASE values is explicitly given in (2.18).

(2.18).
Jani Petit Marinak^{k} Fricit (látta segíteni^{k} meghívni)^{m} és (hallotta segíteni^{k} elzavarni)
ACC DAT^{k} ACC (ACC DAT^{k} ACC)^{m} (ACC DAT^{k} ACC)

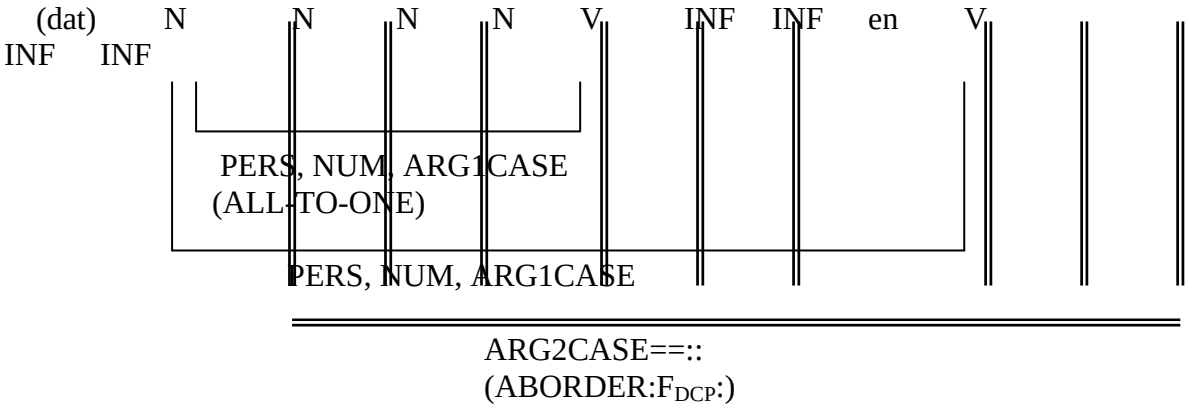
As (2.14) and (2.18) reveal, the basic logic behind the constructions in questions can be represented by the sequences of case values. Such sequences can be generated by function $\mathbf{F}_{DCP}$, as given in (2.19).

(2.19) $\mathbf{F}_{DCP}$
A2$=ARG2CASE_N2
A3$=ARG2CASE_N3
A4$=ARG2CASE_Nk
A$= A2$ + A3$^{k} + A4$
S$= A$ +A$^{m} + A$

ARG2CASE_N2 and ARG2CASE_N3 are the respective ARG2CASE values of the second and the third noun, ARG2CASE_Nk is the ARG2CASE value of the last noun. The output S$ of $\mathbf{F}_{DCP}$ is a sequence of ARG2CASE values of the nouns and the verbs as predetermined by (2.12) or (2.17).[5] Thus agreement pattern (2.20) can handle both (2.9) and (2.17).[6]

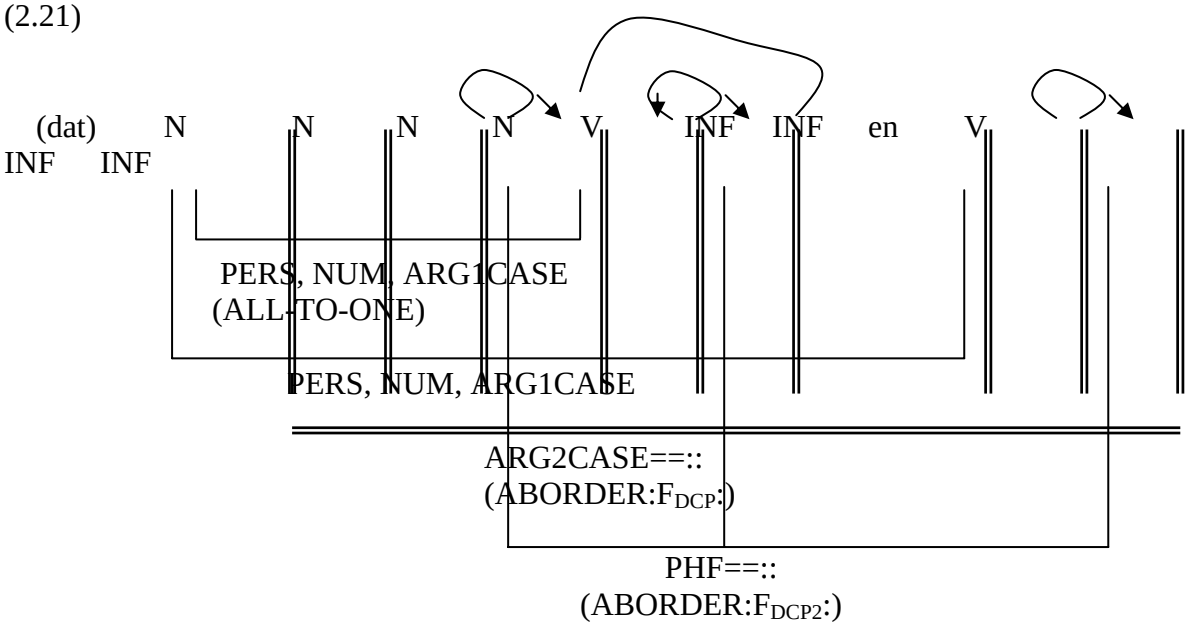
(2.20)

[5] Note that ACC ACC ACC^{k} = ACC ACC^{k} ACC. That is why $\mathbf{F}_{DCP}$ is applicable to both the Dutch construction and its Hungarian counterpart.
[6] We use double lines just for better readability.



Pattern (2.20) does not distinguish between individual verbs or nouns. If we choose to insist on having *k* occurrences of identical nouns (e.g. Fred's), and the corresponding *(m+1)k* occurrences of the same verb (e.g. leren) as (2.12) might suggest, we can add an across-the board PHF (phonological form) constraint to (2.20). The ABORDER constraint can utilise function $\mathbf{F}_{DCP2}$. Cf (2.21) and (2.22).

(2.21)



(2.22)

$\mathbf{F}_{DCP2}$

P1$=PHF_N4
P2$=PHF_INF1
S$= $P1\$^k + (P2\$^k)^m + P2\$^k$

If the PHF value of the last noun is "Fred", i.e. PHF_N4="Fred" and PHF value of the first infinitive is "leren", i.e. PHF_INF1="leren" then the output of $\mathbf{F}_{DCP2}$, S$, is (2.23) as required by (2.12).

(2.23) $S\$ = Fred^k + (leren^k)^m + leren^k$

2.4 Counting

In all previous examples there were some linguistics elements which were recursive in the sense that they could be repeated any number of times, and there were also some elements whose occurrence was dependent on the occurrence of other elements. The dependency was determined by the function in the agreement constraint. For instance, in (2.3), repeated here as (2.24), the number of 'B's is dependent on which 'A' they follow, or, equivalently, which 'C' they precede, i.e on *k*; and $\mathbf{F}_{OG}$ specifies that the sequence of 'B's must grow by 1 each time.

(2.24) ABC ABBC ABBBC … AB^kC

When counting from a certain number to another, in a sense, we do something similar. We repeat some elements and change others. In this case, however, it is not only the length of the string representing a number that grows in a systematic way, but also the digits must observe a predefined order.

(2.25) 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110,

Considering, e.g, (2.25) may lead to the conclusions in (2.26).

(2.26)

1. Digits observe the order '0, 1, 2, 3, 4, 5, 6, 7, 8, 9'
2. When the last digit is '9' a new 'cycle' begins, which means that the next number is constructed in the following way:
   Change '9' to '0' for the last digit and for all '9' digits immediately
   preceding it
   Change the first 'non-9' digit according to order '0, 1, 2, 3, 4, 5, 6, 7, 8, 9'
   If the first digit is '9' then suppose there is a '0' before it , so that order '0, 1, 2, 3, 4, 5, 6, 7, 8, 9' can be observed (i.e the new number will begin with '1').
3. When the last digit is **not** '9', next number is obtained by changing only the last digit (according to '0, 1, 2, 3, 4, 5, 6, 7, 8, 9')

We give a corresponding pseudo-code in (2.27).

(2.27)

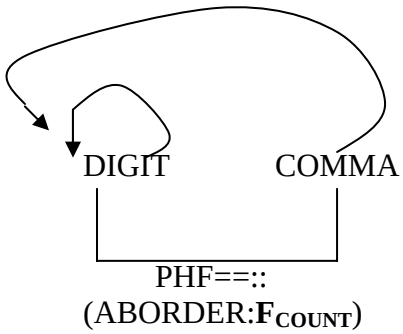
$\mathbf{F}_{COUNT}$

1. Set order of digits: OD$(-1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
2. First number = substring to the left of first comma in input string
3. N$=First number
4. S$=N$+","
5. Digit_step_back = 0: nine$="false"
6. For i=1 to Number_of_commas-1

7. Flag
8. If Last_digit - Digit_step_back=0 then N$= “0”+N$
9. E1$= the digit at position Last_digit - Digit_step_back in N$
10. If E1$ = “9” then E1$=-1: nine$=”true”
11. E2$=the digit immediately following E1$ in OD$
12. Replace E1$ with E2$ in N$
13. If nine$=”true” then Digit_step_back= Digit_step_back+1:nine$=”false”:goto Flag
14. S$= S$ + N$ + “,”
15. Digit_step_back=0
16. next

The output of $\mathbf{F}_{\text{COUNT}}$, S$, is a string containing numbers in increasing order separated by commas. Counting can start from any number. Thus pattern (2.28) corresponds to a ‘language’ whose ‘sentences’ are enumerations of ordered sets of cardinal numbers.

(2.28)



In (2.27) we used ‘–1’ to mark the start of a ‘0, 1, 2, 3, 4, 5, 6, 7, 8, 9’ cycle. Of course, other symbols could also do the job
Commas – or equivalent symbols - are necessary to indicate the end of a given number. Such separating elements are employed both in written languge – spaces, commas, new lines, etc. – and in oral speech – pauses, intonation.

3. Transition between words and digits: the word-digit interface

In the previous section we modelled the abstract capacity to enumerate numbers. Nevertheless numbers are an integral part of natural language. Words can be combined to refer to the various numbers, and at the same time there are numeric expressions corresponding to such word combinations. In other words, words can be transformed into sequences of digits and sequences of digits correspond to linguistic utterances. Below we present a model of associating linguistic expressions with digital sequences.

3.1 Words to digits

While (2.28) represents the abstract capacity to process – to produce, or perceive – an infinitely long sequence of numbers, ‘linguistic counting’ can go on in a systematic way only before we reach the order of milliards/billions. Accordingly, our model will be concerned with only a finite set of numbers in the rest of this paper.

In order to obtain the necessary patterns for numbers we divide words denoting numbers into the following basic categories. Cf. (3.1).

(3.1) one, two, three, four, five, six, seven, eight, nine: CAT = one
ten... nineteen: CAT = teen
twenty ... ninety: CAT = ty
hundred: CAT = hundred
thousand: CAT = thousand
million: CAT = million

Now cardinal numbers 1 to 999 999 can be mapped on patterns (3.2).

(3.2)

ONE
TEEN
TY
TY ONE
ONE HUNDRED
ONE HUNDRED (AND) ONE
ONE HUNDRED (AND) TEEN
ONE HUNDRED (AND) TY
ONE HUNDRED (AND) TY ONE
ONE THOUSAND
ONE THOUSAND (AND) ONE
ONE THOUSAND (AND) TEEN
ONE THOUSAND (AND) TY
ONE THOUSAND (AND) TY ONE
ONE THOUSAND ONE HUNDRED
ONE THOUSAND ONE HUNDRED (AND) ONE
ONE THOUSAND ONE HUNDRED (AND) TEEN
ONE THOUSAND ONE HUNDRED (AND) TY
ONE THOUSAND ONE HUNDRED (AND) TY ONE

TEEN THOUSAND
TEEN THOUSAND (AND) ONE
TEEN THOUSAND (AND) TEEN
TEEN THOUSAND (AND) TY
TEEN THOUSAND (AND) TY ONE
TEEN THOUSAND ONE HUNDRED
TEEN THOUSAND ONE HUNDRED (AND) ONE
TEEN THOUSAND ONE HUNDRED (AND) TEEN
TEEN THOUSAND ONE HUNDRED (AND) TY
TEEN THOUSAND ONE HUNDRED (AND) TY ONE

TY THOUSAND
TY THOUSAND (AND) ONE
TY THOUSAND (AND) TEEN
TY THOUSAND (AND) TY
TY THOUSAND (AND) TY ONE
TY THOUSAND ONE HUNDRED
TY THOUSAND ONE HUNDRED (AND) ONE
TY THOUSAND ONE HUNDRED (AND) TEEN
TY THOUSAND ONE HUNDRED (AND) TY

TY THOUSAND ONE HUNDRED (AND) TY ONE

TY ONE THOUSAND
TY ONE THOUSAND (AND) ONE
TY ONE THOUSAND (AND) TEEN
TY ONE THOUSAND (AND) TY
TY ONE THOUSAND (AND) TY ONE
TY ONE THOUSAND ONE HUNDRED
TY ONE THOUSAND ONE HUNDRED (AND) ONE
TY ONE THOUSAND ONE HUNDRED (AND) TEEN
TY ONE THOUSAND ONE HUNDRED (AND) TY
TY ONE THOUSAND ONE HUNDRED (AND) TY ONE

ONE HUNDRED THOUSAND
ONE HUNDRED THOUSAND (AND) ONE
ONE HUNDRED THOUSAND (AND) TEEN
ONE HUNDRED THOUSAND (AND) TY
ONE HUNDRED THOUSAND (AND) TY ONE
ONE HUNDRED THOUSAND ONE HUNDRED
ONE HUNDRED THOUSAND ONE HUNDRED (AND) ONE
ONE HUNDRED THOUSAND ONE HUNDRED (AND) TEEN
ONE HUNDRED THOUSAND ONE HUNDRED (AND) TY
ONE HUNDRED THOUSAND ONE HUNDRED (AND) TY ONE

ONE HUNDRED (AND) ONE THOUSAND
ONE HUNDRED (AND) ONE THOUSAND (AND) ONE
ONE HUNDRED (AND) ONE THOUSAND (AND) TEEN
ONE HUNDRED (AND) ONE THOUSAND (AND) TY
ONE HUNDRED (AND) ONE THOUSAND (AND) TY ONE
ONE HUNDRED (AND) ONE THOUSAND ONE HUNDRED
ONE HUNDRED (AND) ONE THOUSAND ONE HUNDRED (AND) ONE
ONE HUNDRED (AND) ONE THOUSAND ONE HUNDRED (AND) TEEN
ONE HUNDRED (AND) ONE THOUSAND ONE HUNDRED (AND) TY
ONE HUNDRED (AND) ONE THOUSAND ONE HUNDRED (AND) TY ONE

ONE HUNDRED (AND) TEEN THOUSAND
ONE HUNDRED (AND) TEEN THOUSAND (AND) ONE
ONE HUNDRED (AND) TEEN THOUSAND (AND) TEEN
ONE HUNDRED (AND) TEEN THOUSAND (AND) TY
ONE HUNDRED (AND) TEEN THOUSAND (AND) TY ONE
ONE HUNDRED (AND) TEEN THOUSAND ONE HUNDRED
ONE HUNDRED (AND) TEEN THOUSAND ONE HUNDRED (AND) ONE
ONE HUNDRED (AND) TEEN THOUSAND ONE HUNDRED (AND) TEEN
ONE HUNDRED (AND) TEEN THOUSAND ONE HUNDRED (AND) TY
ONE HUNDRED (AND) TEEN THOUSAND ONE HUNDRED (AND) TY ONE

ONE HUNDRED (AND) TY THOUSAND
ONE HUNDRED (AND) TY THOUSAND (AND) ONE
ONE HUNDRED (AND) TY THOUSAND (AND) TEEN
ONE HUNDRED (AND) TY THOUSAND (AND) TY
ONE HUNDRED (AND) TY THOUSAND (AND) TY ONE
ONE HUNDRED (AND) TY THOUSAND ONE HUNDRED
ONE HUNDRED (AND) TY THOUSAND ONE HUNDRED (AND) ONE
ONE HUNDRED (AND) TY THOUSAND ONE HUNDRED (AND) TEEN

ONE HUNDRED (AND) TY THOUSAND ONE HUNDRED (AND) TY
ONE HUNDRED (AND) TY THOUSAND ONE HUNDRED (AND) TY ONE

ONE HUNDRED (AND) TY ONE THOUSAND
ONE HUNDRED (AND) TY ONE THOUSAND (AND) ONE
ONE HUNDRED (AND) TY ONE THOUSAND (AND) TEEN
ONE HUNDRED (AND) TY ONE THOUSAND (AND) TY
ONE HUNDRED (AND) TY ONE THOUSAND (AND) TY ONE
ONE HUNDRED (AND) TY ONE THOUSAND ONE HUNDRED
ONE HUNDRED (AND) TY ONE THOUSAND ONE HUNDRED (AND) ONE
ONE HUNDRED (AND) TY ONE THOUSAND ONE HUNDRED (AND) TEEN
ONE HUNDRED (AND) TY ONE THOUSAND ONE HUNDRED (AND) TY
ONE HUNDRED (AND) TY ONE THOUSAND ONE HUNDRED (AND) TY ONE

Input elements in the agreement model are represented as AVS's. Words denoting numbers then can be characterised as in (3.3)-(3.5).

(3.3)
PHF = three
CAT = one
DIGITFORM1=3
'three'

(3.4)
PHF = eleven
CAT = teen
DIGITFORM1=11
'eleven'

(3.5)
PHF= twenty
CAT = ty
DIGITFORM1=20
DIGITFORM2=2
'twenty'

Next we establish the connection between words of English (a natural language) and the symbolic language of digits. First we model the conversion of sequences of words into sequences of digits.

When input elements are successfully mapped on a pattern, we say that the pattern is *saturated*. A saturated pattern can be thought of as representing a grammatical sentence, or phrase. However, a sentence can also be regarded as a linguistic unit on its own right, its features being determined by the elements it consists of. Sentence (3.6), for instance, can be understood as a 'higher level' feature structure with attribute values like, e.g. those in (3.7).

(3.6) Joe sleeps

(3.7)
PHF =joe sleeps
SUBJ = joe

PRED = sleep
TYPE = declarative

The composition of such feature structures from saturated patterns can be done in an automatic way by specifying instructions/algorithms for the individual patterns.
It can be prescribed, for instance, that the SUBJ value for (3.7) should be identical with the PHF value of the input element mapped on the first element of the pattern licensing (3.6), or it can be declared that the TYPE of the sentence is declarative, etc.

Analogously, patterns in (3.2) can be interpreted as standing for 'number sentences'. (3.8) can saturate (3.9).

(3.8) Twenty one thousand nine hundred and eleven

(3.9) TY ONE THOUSAND ONE HUNDRED AND TEEN

For (3.9) we can then provide an algorithm to obtain the feature structure representation of number (3.8). Cf. (3.10). This representation can serve as input element to patterns of other cognitive modules.[7]

(3.10)

$$\begin{pmatrix} \text{PHF} = \text{twenty one thousand nine hundred and eleven} \\ \text{DIGITFORM} = 2\ 1\ \ 9\ 1\ 1 \end{pmatrix}$$

An algorithm that outputs AVS (3.10) is sketched in (3.11).

(3.11)

New structure: s
s(PHF = the concatenation of the PHF values of all the elements mapped on (3.9))
s(DIGITFORM = DIGITFORM2 of the first element + DIGITFORM1 of the second element + DIGITFORM1 of the fourth element + DIGITFORM1 of the seventh element)

Note that words with *CAT = ty* have both DIGITFORM1 and DIGITFORM2. The reason is that the digit representation of these words consists of one digit when they are followed by a *CAT = one* word, and it consists of two digits in other cases. Cf. e.g. (3.12) and (3.13).

(3.12) twenty 20
twenty nine 29

(3.13)
TY
TY ONE

[7] E.g. DIGITFORM values, interpreted as sequences of digits, can serve as input to pattern (2.28).

3.2 Digits to words

It would be quite idyllic for the system to be able to use patterns like those in (3.2) both ways, i.e. both for word-to-digit and for digit-to-word mapping. However this could be done for one-digit numbers only. Letting the category values be the same, say, *one* both for the word and for the digit representing it would make it possible to map 'zero', 'one', 'two', 'three' 'four', 'five', 'six', 'seven', 'eight', 'nine', and '0','1', '2', '3', '4', '5', '6', '7', '8', '9' on pattern (3.14).[8]

(3.14) ONE

For more digits there are two basic problems. First, the number of digits does not equal the number of words in the linguistic form. Cf. (3.15).

(3.15)

eleven	11	→	one word, two digits
twenty thousand	20000	→	two words, five digits

The second problem is that digits can have different linguistic forms at different positions.
For instance,'2' is pronounced as two in '200' or '32', but it is 'twenty' in '20', or '0' is pronounced when alone – zero, nought -, but not in other cases[9].
Now what our system will do is use digital patterns as in (3.16) to initialise 'higher level' feature structures like (3.10).

(3.16)

D
D D
D D D
…
D D D D D D

Recall our goal is the converse of what we did in Section 3.1. Then linguistic elements were mapped on a linguistic pattern, their PHF values made up the PHF value of the input, and its
DIGITFORM value was calculated according to the algorithm linked to the pattern. In the present case the input is a sequence of digits, and our aim is to obtain a linguistic form.

As usual, input elements are feature structures. Cf. (3.17) –(3.19).

[8] This method could work for all numbers up to a boundary as well if the inner structure of numbers were not to be considered. We could then just enumerate all the numbers less than the boundary, both in linguistic and digital form, and assign the same category to all forms. However, the inner structure of number is our primary concern, so we need a more insightful analysis.
[9] Recall we do not discuss (decimal) fractions.

(3.17)

PHF = nine DCAT = d DIGITFORM = 9	‘9’

(3.18)

PHF = eleven DIGITFORM =11	‘11’

(3.19)

PHF= twenty three DIGITFORM =23	‘23’

We divide input elements into two categories. Those with *DCAT = d*, representing numbers 0-9, can be directly mapped on patterns (3.16). On the other hand, those with the missing DCAT value, representing numbers 10-99, cannot.

Now suppose digit sequence (3.20) is mapped on (3.21).

(3.20) 9 1 1

(3.21) D D D

The feature structure representation of (3.20) is then (3.22) as provided by algorithm (3.23).[10]

(3.22)

DIGITFORM = 9 1 1 PHF = nine hundred (and) eleven

(3.23)

New structure s:

s(DIGITFORM = the concatenation of the DIGITFORM values of all the elements mapped on (3.21)
Indirect_element$ = the concatenation of the DIGITFORM values of the second, and the third elements
s(PHF = the PHF value of the first element + “hundred (and)” + the PHF value of Indirect_element$)

Thus the usefulness of indirect elements is reflected in the effect that input elements mapped on different representational elements of a pattern can be considered together, as a single element.[11]

[10] ’(and)’ indicates that in British English ’and’ should be pronounced, and in American English it should not.

For (3.24) algorithm (3.23) should produce phonological form (3.25).

(3.24) 9 0 1
(3.25) nine hundred (and) one.

This can be effected by assuming (indirect) input elements like (3.26).

(3.26)

$$\begin{pmatrix} \text{PHF} = \text{one} \\ \text{DIGITFORM} = 01 \end{pmatrix} \quad \text{‘01’}$$

For (3.27) this method would result in (3.28)

(3.27) 9 0 0
(3.28) nine hundred (and) zero

Here we assume (3.29).

(3.29)

$$\begin{pmatrix} \text{PHF} = \text{zero} \\ \text{DIGITFORM} = 00 \end{pmatrix} \quad \text{‘00’}$$

In order to eliminate the ‘(*and) zero*’ part we modify algorithm (3.23). Cf. (3.30).

(3.30)

New structure s:
s(DIGITFORM = the concatenation of the DIGITFORM values of all the elements mapped on (3.21)
Indirect_element$ = the concatenation of the DIGITFORM values of the second, and the third elements
P$ = the PHF value of the first element + “hundred (and)” + the PHF value of Indirect_element$
If there is an “(*and) zero*” segment in P$ then replace it with “”, (the empty string)
s(PHF = P$)

If we choose to consider (3.31) and (3.32) as legal three-digit numbers, we have to cancel the first zero as well.

(3.31) 0 9 0
(3.32) 0 0 9

This involves further modification of (3.23). Cf. (3.33).

[11] Note that Indirect_element in (3.23) represents the string ‘11’ , and this string in turn represents AVS (3.18).

(3.33)

New structure s:
s(DIGITFORM = the concatenation of the DIGITFORM values of all the elements mapped on (3.21)
Indirect_element$ = the concatenation of the DIGITFORM values of the second, and the third elements
P$ = the PHF value of the first element + “hundred (and)” + the PHF value of Indirect_element$
if there is a “zero hundred (and)” segment in P$ then replace it with “”, (the empty string)
if there is a “hundred (and)” segment in P$ and there is also a “zero” segment then replace “zero” with “”, (the empty string)
s(PHF = P$)

Furthermore, for (3.34) algorithm (3.33) yields phonological form (3.35).

(3.34) 0 0 0
(3.35) zero

‘Interface algorithms’ like (3.33) can be worked out for all patterns in (3.16) in similar fashion.
For simplicity, we do not distinguish between indirect elements ‘00’-09’, ‘10-19’, or ’20’ ‘30’ …’90’. Our analysis shows that no such distinction is needed, however it could be possible to assign different statuses to the different types. It would, in turn, entail the modification of our present analysis.

Recall that we placed an arbitrary upper bound (one million) on the scope of our discussion.
Although we do not see a sharp boundary, the case seems to be that (very) large numbers require the activation of other cognitive modules. For example, one can add additional value to certain numbers by repetition, like in (3.36).

(3.36) a million million …

On the one hand, such repetitions can be regarded as a linguistic means to give emphasis. Cf. sentence (3.37).

(3.37). It’s very very good

On the other hand they represent mathematical multiplication, since a million million is understood as a million times million. Indeed, further exact characterisation of numbers is only possible by mathematical means. Cf. the exponential expressions in (3.38).

(3.38) 10^{30}
$1{,}11111111111111111 \times 10^{17}$

.

4. Conclusions

This paper investigated the interaction of language and some basic mathematical capacities. We presented a model that allows parallel analyses for the various phenomena. A central part was played by strategy ABORDER, and the functions characterising individual constructions.
As was shown, 'interface algorithms' can establish the connection between a natural language and the symbolic language of digits by converting saturated patterns into feature structures that can, in turn, serve as input elements for other cognitive patterns. We did not emphasise the computational complexity aspect of the functions and interface algorithms employed in this work, however a closer look would reveal that we attempted to keep to computational simplicity, i.e. our algorithms do not go beyond linearity in the length of the input sequence.

References

Bhatt, R., Joshi, A. (2004) Semilinearity Is a Syntactic Invariant: A Reply to Michaelis and Kracht 1997 *Linguistic Inquiry* - Volume 35, Number 4, Fall 2004, pp. 683-692

Boeder W. (1995) Suffixaufnahme in Kartvelian. In . *Double Case. Agreement by Suffixaufnahme*. OUP. New York. pp. 151-215

Drienkó, L. (2004a). Agreement Mapping System Approach to Language. *Journal of Language and Linguistics.* Vol.. 3. No. 1. 38-61.

Drienkó, L. (2004b). Outlines of Agreement Syntax. *Journal of Language and Linguistics.* Vol.. 3. No. 2. 154-181.

Groenink, A.V. (1996). Mild context-sensitivity and tuple-based generalizations of context-free grammar. Report CS-R9634, Centrum voor Wiskunde en Informatica (CWI), Amsterdam

Michaelis, J, Kracht, M. (1996). *Semilinearity as a Syntactic Invariant*.. Paper presented at the conference *Logical Aspects of Computational Linguistics (LACL `96)*, Nancy, September 23-25, 1996. Appeared in C. Retoré (ed.), *Logical Aspects of Computational Linguistics*, pp. 329-345, Springer, Berlin, 1997. Online version: http://tcl.sfs.uni-tuebingen.de/~michael/papers.html

Radzinski, D. (1991) Chinese number-names, tree adjoining languages, and mild context-sensitivity. *Computational Linguistics. 17: 277-299.*