ARTIFICIALLY GROWING

A NUMERAL SYSTEM*

James R Hurford

Linguistics Department, University of Edinburgh

1 Introduction

A special kind of language change, not centrally studied in historical linguistics, is language **growth**. Growth is change from a smaller system (or perhaps even from nothing) to a larger system. Grammatical growth can occur by addition of lexical items or, perhaps more interestingly, of rules.

This paper describes attempts to simulate the growth of natural language numeral systems by computational techniques. Similar techniques could in principle by used to model the growth of other subsystems of grammar, and a long-term aim of this strand of research is to model the evolution of many of the organizational features of language. There are two salient factors in favour of using numeral systems as a test bed for this approach, and one salient factor against.

- In favour:

    1. Numeral systems are relatively simple,

    2. Numeral systems are relatively self-contained.

*Thanks to Simon Kirby and Matthew Aylett for helpful and stimulating discussion of this topic. None of it is their fault.

- Against:

    1. Numeral systems are in various ways atypical of other language subsystems.

These properties of numeral systems are known well enough to need no further elaboration here.

2 Artificial Life ('A-Life') Research

The research paradigm inspiring this work is that of Artificial Life ('A-Life'). Two very readable popular introductions to this field are Levy (1992) and Lewin (1993), and there are now major collections stemming from workshops and conferences (Langton et al, (1991, 1994), Brooks and Maes (1994)).

Although terminologically reminiscent of the discipline of Artificial Intelligence (AI), A-Life differs from it in several fundamental philosophical and methodological principles. Whereas AI attempts to model **developed** or **mature** intelligence, the products of millennia of evolution, A-Life attempts to model the evolutionary processes leading to such products. AI systems are typically architecturally complex, embodying the programmer's analysis of the particular manifestation of intelligence under study (such as chess, parsing, visual processing, face recognition). A-Life systems are often, by comparison, architecturally simple, populated in a homogeneous structure by model individuals with (initially) very simple internal structures. In an AI system, the programmer maintains control of as many events in the computational process as possible, little or nothing being left to chance; in an A-Life system, the programmer defines boundary conditions for the evolutionary processes being modelled, and 'sits back' to watch the interplay of random currents. AI typically models the behaviour of a single agent, whereas A-Life typically models multi-agent systems, in which many individuals interact in evolving populations. A-Life, being concerned with complex adaptive systems in general, is broader in its scope than AI, which is restricted to systems to which

one can plausibly attribute intelligence.

Some classic examples of (relatively easily understood) seminal, though not necessarily typical, work in the spirit of A-Life are Conway's 'Game of Life' (Garfinkel, 1983; Seife, 1994) and Ray's 'Tierra' program (Ray, 1991a,b). Both these systems illustrate how the application of very simple principles over many 'generations' can give rise to entities of remarkable complexity. Many of the evolved entities, moreover, show behaviour that can be interpreted as life-like.

Conway's Game of Life is played out by shapes defined by shading squares on graph paper. These change their outline every cycle by simple rules dictating whether squares adjacent to shaded squares get shaded or not, and whether shaded squares stay shaded. Some simple configurations oscillate between two shapes; some shapes effectively glide across the graph paper by changing like a snail's foot. Still more complicated configurations spawn such 'gliders', giving rise to waves of offspring shapes, and some configurations are capable of self-replication. All of this is a surprise, or was to its fascinated discoverers (see Levy (1992) and for accounts).

Ray's Tierra system grows whole life-like eco-systems in the guts of a computer. Original minimally complex structures are defined in terms of configurations of addresses in core memory. Individual instantiations of such structures are 'nourished' by the consumption of computing time, for which they have to compete by carrying out certain actions. If they don't get enough nourishment, they 'die', that is they are removed from the arena of the computation. These 'organisms' are also capable of self-replication, with the possibility of random mutations affecting the definition, in terms of core memory configurations, of the offspring. Allowed to run for many thousands of generations, Ray's Tierra systems evolve populations of diverse 'species', which resemble ecological types in nature, such as parasites and predators. Every so often, there can be a wave of mass extinction, parallel to that evidenced by the real fossil record. The Tierra organisms are reminiscent of computer viruses, the most familiar form of artificial life.

The interest in such work, and in the many more specialized and perhaps less 'toy' projects that have succeeded them, is in general in the emergence of complexity and order, in often in particular in the modelling of the evolution of complex natural systems by basic Darwinian principles from simple beginnings. The thought inspiring the present work is that (the grammars of) languages are complex naturally occurring systems, and one wonders to what extent techniques from A-Life research can be used to model their growth.

The differences between AI and A-Life sketched above are no doubt starkly simplified, but they give a true flavour of the methodological differences between the two approaches. One might further guess that whereas AI is more 'application-driven' (good AI vision systems can be incorporated into useful robots; good parsers can be incorporated into useful machine-assisted translation programs), A-Life is more 'curiosity-driven'. To an extent this is true, but it is now being discovered that some practical problems are so complex that broadly evolutionary, self-organizing, styles of system, in which programs are left to evolve their own solutions, are beginning to be viable in applied fields.

One particular technique in the wider A-Life armoury is Genetic Algorithms (GAs). See Koza (1992), Davis (1991), Goldberg (1989) and Holland (1975) for introductions to Genetic Algorithms. GAs mimic the biology of sexual reproduction. In sexual reproduction, the genes of both parents are randomly mixed; if there is much difference between the parents' genes, the genotype of the offspring will differ from that of both parents and will probably be a new genetic mixture unique in its population. The offspring will also differ from others in the population in the probability of its surviving and reproducing.

Evolving a working computer program by GA involves defining:

- Ordered sets of basic genetic building blocks, which can be small scraps of computer code (roughly analogous, say, to amino acids);
- A reproduction process, by which these elements are mixed into new sets; such assemblages of

computer code could by chance form little subroutines of working programs which do something (analogous to synthesizing proteins).

- Possible random mutations to the system's 'genes';
- A fitness function determining the level of success achieved by any particular set of the basic building blocks at some task fixed by the programmer (analogous to an organism built of proteins surviving in a given environment).

Clearly, in the expression 'Genetic Algorithm', the term *genetic* is used **metaphorically**. The similarity between GAs and biological genetics is **formal**. GA's work because they exploit the same basic mechanisms of selection as biological evolution; the mechanisms themselves, abstracted away from the material on which they operate (say DNA), and the environments in which they are embedded (say the physical world), can be applied to try to model the evolution of any kind of complex adaptive system. It should be clear that GAs may not only model biological evolution, but many different manifestations of evolution, including, for example, the cultural evolution of social conventions. The key elements are selection, innovation by random mixing of the basic 'genetic' units (whatever they may be, e.g. genes or memes), other random innovation (e.g. by biological mutation or small individual acts of cultural invention), faithful transmission of the 'genetic' units between generations, and 'superfecundity' (i.e. having a wide range of choices from which to select).

The experiments described later in this paper apply GAs to **grammars**, sets of lexical entries and phrase structure rules, which progressively evolve to express a greater range of meanings (numbers in this case) with maximal coverage, minimal redundancy, and maximal economy of expression.

3 Why? What's the Point?

If a computational system can be host to the evolution of naturalistic grammars, resembling the grammars of known languages, without the rules being directly invented and simply written in by the programmer, then presumably the system must **in some sense** be emulating the forces which gave rise to the real grammars of languages. The cautionary phrase 'in some sense' is to be emphasized. Objects built of metal may always lack some of the properties of objects built of protein, but neither should possible similarities and parallelisms be ignored.

If naturalistic grammars can be evolved in a computer system, the parameters of the system in which this is done may be argued to shed light on the real-world conditions in which real grammars evolved. The grammars of individual languages are not themselves biological objects. The conditions that give rise to grammars are in part biological and in part social. Grammars are shaped both by the innate biological capacities of humans and by the pressures of the social environment. Some of the parameters of a computer system in which life-like grammars arise could be plausibly interpreted as mirroring bio-psychological capacities of individuals; and other parameters can plausibly seen as parallel to social forces operative in the 'Arena of Use'[1].

If, however, repeated attempts fail to construct an artificial system in which grammars simply evolve, then there would be mounting evidence that natural grammars do not arise merely by the interaction of selection, random innovation and faithful transmission across generations. What might an alternative source of grammars be? Presumably something like a creationist account is the alternative to evolution, some kind of deliberate, one-off, large scale invention, masterminded perhaps by some genius. Such an account would see the grammar of a language as something analogous to Esperanto or perhaps Euclid's geometry [2], a whole integrated system springing fully

[1] see Hurford (1990, 1994) and Kirby (1994, 1996)

[2] Pretending, for the sake of the analogy, that Ludwig Zamenhof did not model Esperanto closely on existing

formed from the brow of its inventor, and preserved but scarcely improved by succeeding generations over millennia.

The issue addressed could be expressed as: Did humans (somewhat deliberately) **make** language? or did language just **grow in** humans (and human societies)? The more one tries to flesh out any non-evolutionary explanation of how languages got to be the way they are, the less plausible any creationist or macro-inventionist account seems. Yet on the other hand, the evolutionary style of explanation, which holds that (the grammars of) languages evolved gradually into their present shapes, itself stands sorely in need of fleshing out. It remains to be demonstrated in detail that there could be plausible sets of conditions from which classical evolutionary mechanisms produce grammars of the familiar sort.

Defining and adjusting the parameters of a GA in which naturalistic grammars evolve provides a strictly disciplined framework within which the detailed conditions giving rise to linguistic evolution can be explored. Putting the matter at its least pretentious, all a GA does is emulate a fitness-driven random search of some space. The search space is defined by the properties of the basic genetic units, and the more or less complex configurations into which they may be combined. The strategy guiding the search is defined by the fitness function which selects the more promising configurations for 'survival' and 'reproduction' with each generation.

A specific example, simulating the growth of numeral systems, will be provided in the next sections, after which some due reservations about the complete appropriateness of GAs to the linguistic case will also be set out.

---

natural languages, and that Euclid did not stand at the end of a long tradition, itself the product of evolution

4 Natural Numeral Systems

In this paper, we necessarily set rather humble goals, and concentrate on the cardinal numeral sub-grammars of languages. What, then, are the numeral systems of languages like? It suits our purpose to distinguish between two broad types, which I will call 'primitive' and 'developed'[3]. Naturally, there are some intermediate cases between primitive and developed, but the broad dichotomy is useful. I will characterize the two types of system in the next two subsections.

4.1 Primitive Systems

Not all languages have a numeral system[4]. Some languages have quite simple systems, capable of counting only to about 20 or even lower. In primitive systems, the words have not always fully lost their non-numerical meanings. So the word for 5 might also mean '(left) hand'; the expression for '+ 1' might also mean 'and another'; the expression for 10 might also mean 'man' or 'whole' or 'finished' or 'right hand'. In what follows, I will only mention the numerical meanings.

In these systems, either all the numeral expressions are monomorphemic (or at least do not contain more than one morpheme with a numerical interpretation), or a relatively low number, such as 2, 3, 4, or 5, is used as a basis of addition (or very much more rarely of subtraction or multiplication). Sometimes, after a base number appears in the counting sequence, it is used for all higher numbers. But this is not always so, and there can be what appears to be fairly random interspersing of morphologically complex numerals with monomorphemic numerals.

Examples of the first few numerals in some such simple systems are given below (in some cases, the examples given apparently comprise the whole system). In these examples, a single Arabic digit

[3]No insult is implied to the speakers of languages with primitive numeral systems. It will be apparent that the descriptive terms adopted are apt.

[4]For example, many Australian languages have no numeral system. See Dixon (1980:107-8).

indicates that the number in question has an arithmetically simple, often monomorphemic, numeral; and where several Arabic digits are given, this indicates an arithmetically complex numeral, typically reflecting addition. Only a single example of each type is given here, but most of the types are not uncommon; more examples of them could be given, and from more than one part of the world. These primitive types are widely documented and discussed in a number of works surveying numeral systems (such as Conant (1923), Hymes (1955), Kluge (1937-42), Lean (1985-6), Menninger (1969), Pott (1847), Salzmann (1950), Seidenberg 1960).

4.1.1 Various uses of 2 as a base

Aomie[5] 1 2 2+1 2+2 5 5+1

Yareba[6] 1 2 3 2+2 5 5+1

Korafe[7] 1 2 2+1 4 5 5+1

Hunjara[8] 1 2 2+1 2+1+1

Fuyuge ('Mafulu')[9] 1 2 2+1 2+2 2+2+1 2+2+2 2+2+2+1 2+2+2+2

2+2+2+2+1

[5] Austing and Upia (1975:523-24), quoted in Lean, vol.5:40

[6] Weimer and Weimer (1974), quoted in Lean, vol.5:52

[7] Lean, vol.5:38

[8] Lean, vol.5:28

[9] Williamson (1912:228-9), quoted in Lean, vol.7:42. Some expressions in this counting system have alternatives.

4.1.2 3 as a base

Mawae[10] 1 2 3 3+1 5×1

4.1.3 4 as a base

Tunisian Arabic egg-counting[11] 1 2 3 4

4+1 4+2 4+3 4×2

4×2 +1 4×2 +3 3×4 ... 4×4 ... 5×4 ...

4.1.4 3- and 4-based

Motu[12] 1 2 3 4 5 2×3 7 2×4 2×4 +1

Roro[13] 1 2 3 4 5 2×3 2×3 +1 2×4 2×4 +1

Keapara, Hula dialect[14] 1 2 3 4 5 2×3 2×4 -1 2×4 10-1

Hymes (1955:27) calls this type 'pairing'; this pairing type, and several of the other types illustrated above and below, are found in Athabaskan languages, according to Hymes' survey.

---

[10] Smith (1984:253), quoted in Lean, vol.5:16

[11] Informant, Mukhtaar ben Fraj: These forms were used by the informant's grandmother, and are (or were) used in markets. Friederici (1913, quoted in Lean, vol.4:92) mentions a similar 4-based counting system for yams, taro and coconuts in Bariai (Austronesian).

[12] Lean, vol.7:67

[13] Lean, vol.7:20-22

[14] Lean, vol.7:74-76

4.1.5 3-, 4- and 5-based

Ekoi (Cameroun)[15] 1 2 3 4 5 3+3 4+3 4+4 5+4

4.1.6 5- and 6-based

Onjob[16] 1 2 3 4 5 5+1 5+1+1 5+1+2 5+1+3 5+1+4

4.1.7 2-, 5- and 6-based

Miskito[17] 1 2 3 2+2 5 5+1 5+1 +1 5+1 +2 5+1 +3 5×2

5×2 +1 5×2 +2 5×2 +3 5×2 + 2+2

4.1.8 Quinary systems

A purely 5-based system is referred to as 'quinary'. Such systems are very common, being found in almost all quarters of the world, and taking the shape:

1 2 3 4 5 5+1 5+2 5+3 5+4 10

---

[15] Cauty (1986:138)

[16] Macdonnell (1917:171), quoted in Lean, vol.5:70

[17] Conzemius (1929:81-82); these data were verified in fieldwork in 1971 by Ruth Fowlks (personal communication)

4.2 Developed Systems

The most familiar type of numeral system in better known languages is decimal, and sometimes also partly vigesimal. This canonical type has the following characteristics:

- Single words for 1-10,
- Use of addition to 10 for 11-19,
- Use of multiplication by 10 (or 20), (and addition) for 20-99,
- Single words for higher bases, typically 100, 1000, and sometimes also 20.

Examples of such systems are very familiar.

Further characteristics, common to both primitive and developed types of system, are:

- Complete coverage to some limit – no gaps (although the sporadic use of subtraction suggests there can be temporary stages of a language in which there may be gaps);
- No ambiguity or homonymy (examples of ambiguous numerals are extremely scarce, if they occur at all);
- Little, if any, redundancy or synonymy (from a vast set of arithmetically possible combinations for any given number, typically there is only a single well-formed numeral used in the canonical counting sequence. The occasional exception to this generalization occurs, as in paraphrases like English *one thousand one hundred* versus *eleven hundred*);
- Recursion – expressions for higher numbers typically contain expressions for lower numbers nested within them;

- Packing Strategy – the recursive possibilities are severely constrained by a principle to the general effect that one builds on the highest valued expression available (See Hurford 1975, 1987, for details and discussion).

5 Growing Numeral Systems Artificially

We now bring the artificial techniques of Genetic Algorithms to bear on this diverse range of naturally occurring systems. The basic idea of GAs is applied here to **grammars**; the genetic units making up each 'organism' are **rules**, which may be specifications of lexical items or rules of syntactic combination; each grammar is a set of such rules. The proposal is to selectively 'breed' grammars of numeral systems from an initial randomly generated set of grammars. The interest is in seeing whether the range of systems described above simply emerges after systematic selection of grammars over many generations.

The elementary components of the system, by which rules are defined, are:

- A vocabulary of arbitrary monosyllables (e.g. *ba, be, bi, ca, ce, ci, da, de, di, ...*;
- A set of semantic primitives. These are the concepts of certain small numbers — 1 - 10, which it is assumed are accessible to the mind independently of the existence of a counting system. It seems right to stipulate that some of these numbers are more accessible to the mind than others. In particular, probably 1, 2 and 3 are ranked highest in order of accessibility, followed perhaps by 5 (a whole hand) and 10 (both hands). 6, 7, 8, and 9 are, intuitively, relatively inaccessible. Various possible weightings for the availability of the semantic primitives were experimented with. Numbers higher than 10 were judged to be inaccessible as semantic primitives, without benefit of some linguistic counting system, and were never included among the primitives.
- Basic cognitive operations, such as addition, multiplication and subtraction. These can be

grasped from such concrete operations as placing more objects in a pile, making groups of piles, and taking objects out of a pile. Again, these basic operations are not equally accessible to the mind. It was judged that addition is most accessible, and multiplication and subtraction much less accessible. Various quantitative interpretations of these different degrees of accessibility were experimented with.

- Arbitrary syntactic categories, here labelled s0, s1, s2, ... In order for any grammatical system to arise, the notion of word-class must be available. There is no presupposition of any particular natural connection between any such arbitrary word class and any particular semantic primitive or cognitive operation. These arbitrary syntactic labels are used for the construction of lexical and syntactic rules. In the simulations, the actual number of available syntactic categories was experimentally varied, ranging between 2 and 5; clearly there have to be more than one syntactic category for the concept to be useful.

An arbitrary rule in the system is generated by taking the following steps, in order:

1. Randomly select a 'mother' syntactic category (from the set { s0, s1, s2, ... }).

2. Decide, by the random toss of a computational coin (which may be biased), whether to generate a lexical item or a syntactic rule.

3. If a lexical item is to be generated, select a random semantic primitive, and a random monosyllable. Together, the 'mother' category, the semantic primitive and the monosyllable constitute the randomly assembled lexical entry. For example,

   s1 → fa (3)

   would be a lexical entry for the word *fa*, givings its meaning as the number 3, and assigning it to the syntactic category **s1**.

4. If a syntactic rule is to be generated, select two random syntactic categories (which can be the same), and a random cognitive operation. An example of such a randomly generated syntactic rule is:

   s2 → s1 s2 (addition)

   This rule states that a constituent of category s2 can be constructed from a pair of elements, of categories s1 and s2, respectively, and that the whole is to be interpreted by the arithmetical addition of the values of the parts. In this work, all syntactic rules were binary branching. No extralinguistic significance being associated with the syntactic categories, which are merely a kind of 'glue' for building syntactic rules, the choice of syntactuic categories was never weighted in any way. The choice of cognitive operation was weighted in some, but not all, simulations, reflecting the possible greater accessibility of addition than, say, subtraction.

A grammar is a set of lexical items and phrase structure rules, of the sort just illustrated above. For numeral systems at least, this simple view of what constitutes a grammar is arguably adequate. A **random** grammar is a random-sized set of randomly generated lexical items and rules.

The core routine of the simulations carried out consisted of the following steps:

1. Generate a large set of random grammars, as defined and illustrated above.
2. Select the 'fittest' grammars as 'parents' of the next generation;
3. 'Breed' a new generation of grammars, perhaps with some 'mutation';
4. Return to step 2 and keep recycling.

Implementing this system in detail calls for a number of decisions to be made. It is a virtue of the computational approach that it forces one to specify explicitly many parameters of grammars which are seldom, if ever, contemplated by descriptive linguists, although they are nonetheless real,

and in no way artifacts arising simply out of the approach. In some cases, one can make a plausible intuitive guess at an appropriate value for some parameter, and hold it constant; in other cases, one can use the computational system to experimentally manipulate the values of parameters, within an intuitively plausible range. The relative success with which such experimental settings lead to naturalistic grammars can be taken as evidence pointing to corresponding values in the real-world conditions in which natural grammars evolved. In yet other cases, unfortunately, the settings of parameters are determined by computational convenience, although in no case did it seem that such settings were very counterintuitive. The more salient parameters involved in the current simulations are mentioned below. The first paremeters mentioned are linguistic, having to do with the nature of grammars; the ones mentioned later are 'population-genetic', having to do with the particular mechanics of GAs.

SOME TWEAKABLE PARAMETERS

- **Size of initial grammars.** An upper limit of 50 rules (lexical and syntactic totalled) was put on the initial population of random grammars. The initial grammars varied randomly in size between 1 rule and 50.

- **Lexicon to syntactic rule ratio.** Different ratios of lexical items to syntactic rules were experimented with. In some simulations, the probabilities of random rules being lexical or syntactic were equal. (See step 2 in the rule generation procedure above.) In other simulations, the expected ratio of lexical rules to syntactic rules was as high as 20:1.

- **Weighting of semantic primitives.** In some simulations, all the numbers from 1 to 10 were equally weighted. This implies equal probabilities for lexical items with meanings from 1 to 10. A range of other weightings was tried, in which the numbers 1, 2, 3, 5 and 10 were given greater probability of having their own lexical items.

- **Weighting of semantic operations.** Each syntactic rule generated is associated with a particular arithmetic operation, such as addition, multiplication, or subtraction. Sometimes these operations were assigned to rules with equal probability. In other cases, addition was favoured over subtraction and multiplication.

- **Fitness function.** This is a crucial aspect of the simulation. Here, one has to try to make plausible assumptions about what factors might make one numeral grammar preferable, to a group of human users, over another.

  - **Greater coverage** is, other things being equal, presumably desirable; a counting system that reaches to 100 is more useful than one that reaches only to 20.

  - The **absence of gaps** in the counting sequence is presumably also desirable, as natural numeral systems do not have gaps. But the existence of subtraction in a few natural systems shows that a fitness function that makes a single unbroken sequence a mandatory property for 'survival' is too strict. Presumably in the evolution of natural systems with subtraction the higher numeral (e.g. 10) pre-existed the lower one (e.g. 10-1, 10-2).

  - **Lack of redundancy** is also typical of natural numeral systems, and some penalty for providing several expressions for the same number should be built into the fitness function. Again, however, this cannot be an absolute prohibition, as one occasionally finds a natural numeral system with more than one way of expressing a particular number.

  - The **complexity** of numeral expressions is relevant. Other things being equal, a system with shorter, or less grammatically complex, expressions must be preferable.

- Number of initial grammars. This is largely a matter of computation convenience. For a GA to work well, there needs to be a reasonably large number of initial grammars, providing an original 'gene pool' with enough variants in it for there to be a chance of it containing most

of the required types of rules and lexical items. Obviously, though, these rules are not at the beginning assembled into coherent sets (grammars) that generate naturalistic sequences of numeral expressions. In these simulations, the number of initial grammars varied from 25 to 100.

- Percentage selected. The number of grammars taken each generation as parents of the next generation needs also to be set. In these simulations, the number of grammars selected as breeding stock varied between 5 and 10.

- Mutation rate. This is the rate at which, during the mimicked evolutionary process, random rules were added to grammars or deleted from them. Too high a mutation rate prevents the evolutionary process from settling down to stable solutions, as the gene pool is excessively stirred. Too slow a mutation rate can lead to convergence on solutions which would not, with a higher mutation rate, be stable. Given that the term 'Genetic Algorithm' is no more than a **metaphor** for the computational process outlined here, there is no need, of course, to worry about any biological verisimilitude in setting the mutation rate. These are not biological mutations, but just random innovations in the search space.

- **Evaluation methods.** Given some fitness function, a method needs to be determined for applying it to grammars. Several alternatives are conceivable:

  - **From the grammar.** In this syntax-driven approach, one evaluates a set of products of the grammar, produced either exhaustively or randomly. Having used the grammar to generate a set of expressions, one checks to see how this set is valued in terms of the fitness function, i.e. what coverage it achieves, with how many gaps, with how much redundancy, and so on.

  - **From the meanings** In this meaning-driven approach, one takes a set of 'target mean-

ings', and tests whether the grammar generates expressions for them, and if so, with what redundancy, complexity of expression, and so forth.

6 Results

A number of simulations were run, under different conditions. These are described in order of growing complexity.

6.1 Coverage versus lack of redundancy: a simple lesson

In this simulation, the grammars consisted only of lexical entries. There were no phrase structure rules. This simple experiment teaches us a lesson about the fitness function which applies to all more complex simulations. Which should be paramount in the fitness function, coverage or lack of redundancy? Putting it in terms of real human numeral systems, does the optimal system (i.e. that type found most commonly in human languages)

- favour coverage even at the expense of massive redundancy?
- favour lack of redundancy, even at the expense of gaps in the system (lack of coverage)?
- favour some kind of balance between coverage and lack of redundancy?

Repeated simulations were run in which the fitness function always preferred a grammar generating expressions for the most numbers, regardless of how many such expressions it generated for each number. Only when the coverage of two grammars was equal was relative lack of redundancy invoked to discriminate between them, in favour of the grammar generating fewer expressions.

With this fitness function, there was never any convergence on a naturalistic numeral lexicon, with just one word for each of the numbers from 1 to 10.

Another set of simulations was run in which the fitness function was the reverse of that just described. In this case, lack of redundancy, rather than coverage, was paramount. The fitness function always preferred a grammar which had as few extra expressions as possible for each number, even if some other grammar actually provided expressions for more numbers. Only when two grammars had equally few expressions was the coverage of meanings invoked, so that the grammar with the greater coverage was, then, preferred.

In this case also, after many runs, there was no convergence on a naturalistic numeral lexicon with just one word for each number up to 10.

On brief reflection, it becomes obvious why these simulations do not work to produce language-like results. Large grammars, containing more lexical items, are more likely to cover more meanings; just as a longer sequence of throws of a die is more likely to have at least one throw landing on each possible value than a shorter sequence of throws. If the selection process always favours coverage, large grammars will be selected which have enough lexical items to cover all the possible values. Almost inevitably, such grammars will be massively redundant.

Conversely, if lack of redundancy is selected for, the preferred grammars will be those which manage to express meanings with only one expression each. Small grammars, containing fewer lexical items, are more likely to avoid generating several expressions for the same number; just as a shorter sequence of throws of a die is more likely to avoid repetitions than a longer sequence. But the price of selecting smaller grammars is that they almost inevitably lack some coverage of the set of possible meanings.

Now a third set of simulations was run, in which the fitness of a grammar was calculated from a combination of coverage and lack of redundancy. Simply, and no doubt crudely, the fitness of a grammar in this case was inversely proportional to the sum of the gaps in the number sequence and the number of redundant expressions. To give an example, imagine a grammar which happened to

provide words for numbers as follows:

1	2	3	4	5	6	7	8	9	10
do	*mi*		*la*		*ti*				
re	*fa*				*do*				
	so								

Here there are 6 numbers unexpressed by any word (i.e. 6 gaps), and 4 superfluous words. The combined gap-redundancy score is thus 6+4=10. Grammars with lower gap-redundancy scores were favoured in the simulation.

In these runs, there was always convergence to a naturalistic lexicon with just one word for each number from 1 to 10. Sometimes this convergence waas achieved very quickly, once in as few as 5 generations. In other runs, convergence took as long as 200 generations.

The combined gap-redundancy fitness function took no account of homonymy, and occasionally a simulation converged on a numeral lexicon in which a single monosyllable happened to express more than one number. This homonymy is, as far as I know, never found in real numeral systems. Further simulations should also build homonymy-avoidance into the fitness function. As the method for doing this is straightforward, this line of investigation was not followed. (In the grammars that emerged in later simulations, the occasional instances of homonymy were tolerated; nothing rides on this, as far as I can see.)

In the more complex simulations described below, the fitness function was always some combination of coverage and lack of redundancy, sometimes with other factors as well.

6.2 Effects of other variables

In further experiments with versions generating only simple lexicons, the situation after initial convergence on optimal fitness was studied, to see whether this fitness is maintained for long stable

periods. Both mutation rate and population size were 'tweaked'. Clearly, too fast a mutation rate (e.g. every cycle) stirs up the pool of grammars too much, and optimal fitness is only reached sporadically, with many lapses. But also important is population size. With 20 initial grammars and 5 grammars selected each cycle, there was frequent lapsing from optimal fitness over time, with few periods of stable optimal fitness. With 100 initial grammars and 10 grammars selected each cycle, the situation improved considerably, with long stable periods of optimal fitness, but still the occasional lapse, which was almost always immediately recovered from. With this population size, a mutation rate of 1 mutation per 15 grammars bred was, impressionistically, slightly more stable than a rate of 1 per 5.

The lesson one quickly learns, if one did not know it before, is that there is an intractably large number of possible variables, both linguistic and non-linguistic, which affect the evolution of grammars in this framework. By no means all of the possible variations are mentioned here, for readability's sake. Anyone attempting to replicate these experiments will be (or become) aware of the vast number of theoretical possibilities. It is conceivable that the evolution of real grammars is in fact subject to a small number of real variables. The computational approach described here might in principle lead to the discovery of some very parsimonious 'magical combination' of settings of a few variables, which give rise to naturalistic grammars. Alternatively, lack of progress in finding such a combination might suggest the conclusion that the evolution of real-life grammars is subject to a great many chance vagaries, both psychological and social.

6.3 'Peano-type' results

In the next series of experiments, the random grammars could contain both lexical items and simple phrase structure rules, as described above.

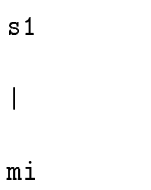
In the weightings for these runs, the random generation of a syntactic rule was made as probable

as the random generation of a lexical item. Many of these simulations converged on numeral systems of a mathematically very elegant type, but a type which is quite unnatural, in the sense of being unrepresentative of natural languages. The systems arrived at were highly economical of both lexical and syntactic resources. They also achieved complete coverage of the number sequence up to any limit, sometimes with no cost in redundancy. One such extreme outcome was:

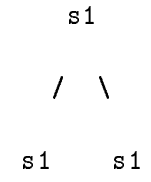
s1 → mi (1)

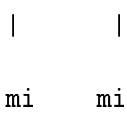
s1 → s1 s1 (addition)

This grammar generates a single structure for the number 1, as in

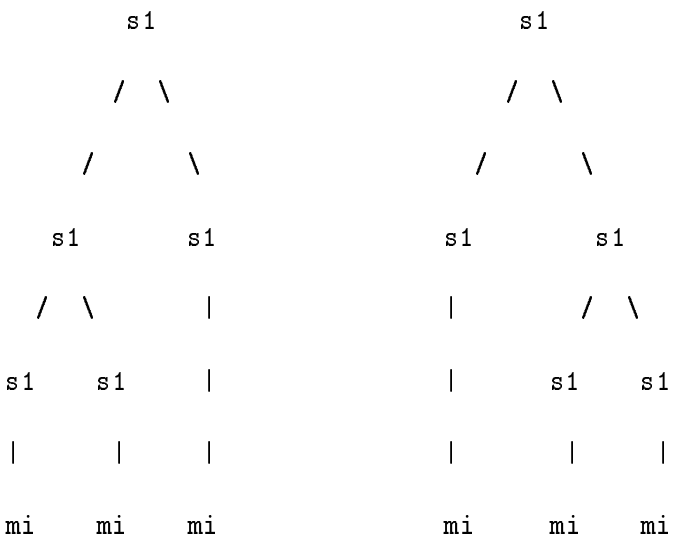


and a single structure for the number 2, namely





but at 3, (structural) redundancy sets in, with two possible structures for 3, namely



The redundancy increases dramatically as one proceeds to higher numbers. In the simulation that gave this result, only binary branching phrase structure rules had been allowed. Without the option of unary, or non-branching, phrase structure rules (or of binary rules which can include a simple lexical item, rather than a syntactic category, on the right hand side), there exists no grammar which can avoid this burgeoning redundancy.

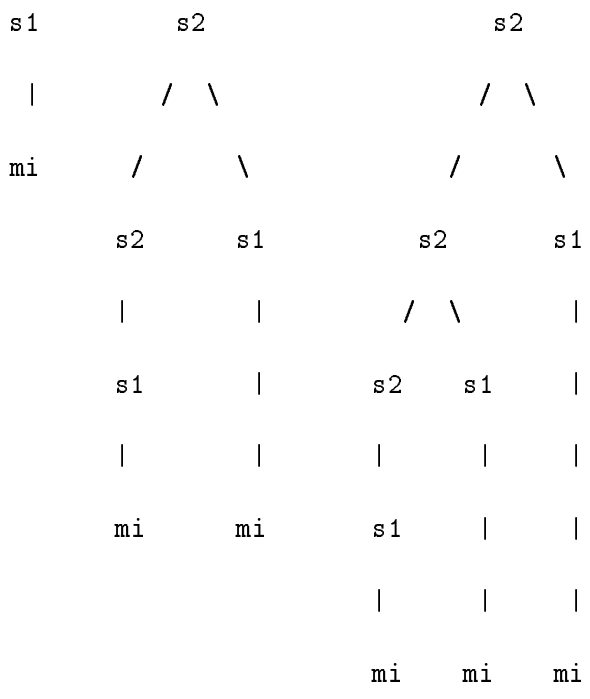
With the possibility of unary, or non-branching, phrase structure rules, a non-redundant grammar exactly parallel to Peano's axiomatic system for defining the number sequence can arise, as follows:

s1 → mi (1)

s2 → s1

s2 → s2 s1 (addition)

This grammar generates a single structure for each number, as shown below for the numbers 1, 2 and 3:



For the short term memories of humans, such a system is highly dysfunctional, and even self-defeating. The value of a counting system is in defining compact expressions which compress information. The psychological burden of counting the instances of the monosyllable *mi* in an expression

for a higher number in this system renders it unusable. Such systems do not occur in spoken human languages. (But note the beginnings of such a system in written Roman numerals, *I, II, III*; such iconic representations of numbers can also be found in the ideographic cunieform writing of the Sumerians and in Egyptian hieroglyphs.)

Such Peano-style grammars could result from various different combinations of circumstances. Factors which favour the evolution of such unnatural Peano-type solutions include:

- Fitness defined primarily as economy of storage of grammar, i.e. the fittest grammar has the fewest rules and lexical items;
- Relatively favorable weighting for combinatory syntactic rules, as opposed to lexical items;

6.4 Evolution of 'primitive' systems

One combination of variables was found to give rise to many systems closely resembling the natural 'primitive' systems described earlier. Recall these were typically found in relatively small, and relatively isolated communities.

In these runs, a grammar's fitness was defined as the highest number it could 'count to' with no gaps, and not too much redundancy. Here, the algorithm to determine fitness is, basically: count from 1 upwards, and stop at the first gap, or where there are more than 2 (an arbitrarily chosen limit on redundancy) expressions for some number. The expected ratio of syntactic rules to lexical items was set at 1:8, and the number of abstract syntactic categories permitted was 2.

Starting with a population of 100 random grammars, the fittest 10 grammars were selected as parents grammars for the next generation. These parents bred (all with each other) a generation consisting of a further 100 offspring grammars. From this new generation, the fittest 10 were again selected as parents, and so on. The simulations ran for 1000 generations each, at which stage the fittest (usually the only) remaining grammar was inspected to see what numeral expressions, up to

20, it generated.

Of the 164 runs, over half (86) yielded lexicon-only solutions, summarized as follows:

Words for numbers	Instances of this solution
1 2 3 4 5 6	5
1 2 3 4 5 6 7	52
1 2 3 4 5 6 7 8	21
1 2 3 4 5 6 7 9 (gap at 8)	4
1 2 3 4 5 6 7 8 9	3
1 2 3 4 5 6 7 8 10 (gap at 9)	1

Such simple lexicons may perhaps be regarded as the basis on which somewhat more elaborate systems, with some syntactic rules, are founded.

In the remaining 78 runs, all solutions in some way resembled the natural primitive systems illustrated earlier. Solutions included base-2, base-3, base-4, base-5, base-6, and various mixed-base systems. Some examples are given below (note the occasional gaps and redundancies in these systems):

BASE-2

(The first five expressions here are the same as those of Yareba, cited above.)

1 = 1

2 = 2

3 = 3

4 = [2 2]

5 = 5

6 = [2 [2 2]]

6 = [[2 2] 2]

8 = [[2 2] [2 2]]

MIXED 2- 5- and 6-BASE

(The first four expressions here are the same (in mirror image) as those of Hunjara, cited above.)

1 = 1

2 = 2

3 = [1 2]

4 = [1 [1 2]]

5 = 5

6 = 6

7 = [5 2]

8 = [1 [5 2]]

8 = [5 [1 2]]

8 = [6 2]

9 = [1 [6 2]]

9 = [6 [1 2]]

12 = [5 [5 2]]

13 = [5 [6 2]]

13 = [6 [5 2]]

14 = [6 [6 2]]

MIXED 2- 7- and 9-BASE

(The first six expressions here are the same as those of Fuyuge ('Mafulu'), cited above.)

1 = 1

2 = 2

3 = [2 1]

4 = [2 2]

5 = [2 [2 1]]

6 = [2 [2 2]]

7 = 7

8 = [7 1]

9 = 9

```
10 =  [ 2 [ 7 1]]

10 =  [ 7 [ 2 1]]

10 =  [ 9 1]

11 =  [ 7 [ 2 2]]

12 =  [ 2 [ 9 1]]

12 =  [ 9 [ 2 1]]

13 =  [ 9 [ 2 2]]

14 =  [ 2 7]

14 =  [ 7 2]

15 =  [ 7 [ 7 1]]

16 =  [ 2 [ 2 7]]

16 =  [ 2 [ 7 2]]

17 =  [ 7 [ 9 1]]

17 =  [ 9 [ 7 1]]

18 =  [ 2 9]

18 =  [ 9 2]

19 =  [ 9 [ 9 1]]

20 =  [ 2 [ 2 9]]

20 =  [ 2 [ 9 2]]
```

MIXED BASE-3 and -10

```
1 =  1
```

```
2 =  2

3 =  3

4 =  [ 1 3]

5 =  [ 2 3]

6 =  [ 3 3]

7 =  [ [ 1 3] 3]

8 =  [ [ 2 3] 3]

9 =  [ [ 3 3] 3]

10 =  10

11 =  [ 1 10]

12 =  [ 2 10]

13 =  [ 3 10]

13 =  [ 10 3]

14 =  [ [ 1 3] 10]

14 =  [ [ 1 10] 3]

15 =  [ [ 2 3] 10]

15 =  [ [ 2 10] 3]

16 =  [ [ 3 3] 10]

16 =  [ [ 3 10] 3]

16 =  [ [ 10 3] 3]

20 =  [ 10 10]
```

MIXED 3- 4- and 5-BASE

(The first nine expressions here are the same as those (or their mirror images) of Ekoi, cited above, labelled by Hymes the 'pairing' type.)

1 = 1

2 = 2

3 = 3

4 = 4

5 = 5

6 = [3 3]

7 = [3 4]

7 = [4 3]

8 = [3 5]

8 = [4 4]

8 = [5 3]

9 = [4 5]

9 = [5 4]

10 = [5 5]

BASE-4

(The first seven expressions here are the same as the Tunisian egg-counting system, cited above.)

1 = 1

2 = 2

3 = 3

4 = 4

5 = [1 4]

6 = [2 4]

7 = [3 4]

8 = [4 4]

9 = [[1 4] 4]

10 = [[2 4] 4]

11 = [[3 4] 4]

12 = [[4 4] 4]

MIXED BASE-4 and -8

(This solution (or its mirror image, in which, for example,

11 = [[3 4] 4]) occurred 8 times. The first seven expressions are the

same as the Tunisian egg-counting system, cited above.)

1 = 1

2 = 2

3 = 3

4 = 4

5 = [4 1]

6 = [4 2]

7 = [4 3]

8 = 8

9 = [4 [4 1]]

10 = [4 [4 2]]

11 = [4 [4 3]]

12 = [4 8]

16 = [4 [4 8]]

BASE-5 (quinary/decimal)

(The first ten expressions here are the same as those of typical quinary systems, mentioned above.)

1 = 1

2 = 2

3 = 3

4 = 4

5 = 5

6 = [5 1]

7 = [5 2]

8 = [5 3]

9 = [5 4]

10 = 10

11 = [5 [5 1]]

12 = [5 [5 2]]

13 = [5 [5 3]]

14 = [5 [5 4]]

15 = [5 10]

20 = [5 [5 10]]

BASE-6

1 = 1

2 = 2

3 = 3

4 = 4

5 = 5

6 = 6

7 = [6 1]

8 = [6 2]

9 = [6 3]

10 = [6 4]

11 = [6 5]

13 = [6 [6 1]]

14 = [6 [6 2]]

15 = [6 [6 3]]

16 = [6 [6 4]]

17 = [6 [6 5]]

The similarities between these artificially generated systems and the 'primitive' systems described earlier, are striking.

6.5 Systems get stuck in local optima

It is noteworthy that, of all the artificial grammars generated in the runs described above, not a single pure decimal system emerged. We will explore why this is so.

Recall that what is simulated here is a quasi-social process whereby grammars are formed from a pool of variant lexical items and syntactic rules which are somehow ambient in the community. Over time, the pool of available lexical items and rules becomes restricted so that only one grammar may be constructed from them, and this grammar is the 'fittest' that the community has happened upon. After that point of convergence, only random 'mutations' to the pool of lexical items and rules may disturb the situation. And such mutant lexical items or rules will only persist in the pool if they happen to fit in with existing rules and lexical items to form a grammar which is fitter than the one previously converged upon.

As readers familiar with any complex adaptive system will know, there can exist many 'local optima', that is solutions to a problem (such as finding an effective counting system) which are fitter than their close neighbours in the possibility space, but not the fittest solutions overall, in any global sense. It is clear that the artificial systems illustrated above are local (near-)optima, in this sense. In other words, any small mutation to one of the grammars arrived at tended very strongly to produce a grammar that was less fit

If a particular grammar converged upon by the algorithm is at a local optimum, then any random mutation applied to it, in the form of a randomly added or deleted syntactic rule or lexical item, will result in a grammar less fit (by whatever definition of fitness was used to arrive at the tested

grammar) than the tested grammar. If such random mutations do not always, but nevertheless tend strongly to, result in less fit grammars, then the tested grammar is near a local optimum.

The 13 first grammars converged upon by the algorithm decribed above were tested for local optimality in this way. To each of these grammars, 100 random mutations were applied (not serially, but always starting with the tested grammar). Thus for each tested grammar, 100 one-step mutants were produced. The results for the 13 tested grammars are aggregated below.

1 mutation produced a less fit grammar	in 444 cases.
1 mutation produced an equally fit grammar	in 776 cases.
1 mutation produced a fitter grammar	in 80 cases.

Thus the probability of a random mutation producing an improvement to one of the converged-upon grammars is low, 0.062 (80/1300). These grammars are at or near local optima.

To test whether the algorithm would recognize a globally optimal solution if it saw one, another series of runs was carried out in which the initial population of 100 random grammars was 'seeded' with a small number (10) of copies of a grammar which (I thought) was globally optimal. There were no nasty surprises here, and the algorithm always quickly converged, after just one or two cycles, on the globally optimal solution. The algorithm did however administer one sobering little lesson, in that it discovered a better grammar than the one with which I had seeded it. My 'optimal' grammar was not quite optimal, as it allowed both 1-deleted and non-1-deleted variants of 10 (i.e. $10 = 10$ and $10 = 1 \times 10$), which occurred in all the -teen expressions, giving redundancy.

6.6 Evolution of 'developed' systems

The algorithm explored here never succeeded, in a limited number of trials, in converging on a complete and pure decimal numeral system such as is the basis of the counting systems of most of the world's major languages. On several occasions it got close.

One its better efforts was as follows:

```
LEXICON:

Value   Syntactic       Word
        Category

1       s0              re
2       s0              ga
3       s0              ko
4       s0              se
5       s0              ji
6       s0              ge
7       s0              ci
8       s0              ti
9       s0              le
10      s2              di

SYNTACTIC RULES

s0 --> s2   s0  (Addition)

s1 --> s2   s2  (Addition)
```

COMPLEX EXPRESSIONS GENERATED

```
11 = 10+1

12 = 10+2

13 = 10+3

14 = 10+4

15 = 10+5

16 = 10+6

17 = 10+7

18 = 10+8

19 = 10+9

20 = 10+10
```

As can be seen, this system is like a developed decimal system up to 20, but goes no further.

Another near miss (though at first blush it may not look like it) is as follows:

1 (GAP)

2 = 2

3 = 3

4 = 4

5 = 5

6 = 6

7 (GAP)

8 = 8

$9 = 9$

$10 = 10$

11 (GAP)

$12 = 10{+}2$

$13 = 10{+}3$

14 (GAP)

15 (GAP)

$16 = 10{+}6$

17 (GAP)

$18 = 10{+}8$

$19 = 10{+}9$

$20 = 2{\times}10$

21 (GAP)

$22 = 2{\times}10 + 2$

$23 = 2{\times}10 + 3$

24 (GAP)

25 (GAP)

$26 = 2{\times}10 + 6$

27 (GAP)

$28 = 2{\times}10 + 8$

$29 = 2{\times}10 + 9$

31 (GAP)

$32 = 3{\times}10 + 2$

$33 = 3{\times}10 + 3$

34 (GAP)

35 (GAP)

$36 = 3\times10 + 6$

37 (GAP)

$38 = 3\times10 + 8$

$39 = 3\times10 + 9$

$40 = 2 \times 2\times10$

41-59 (GAPS)

$60 = 2 \times 3\times10\ 3 \times 2\times10\ 6\times10$ (REDUNDANCY)

61 (GAP)

$62 = 6\times10 + 2$

$63 = 6\times10 + 3$

64 (GAP)

65 (GAP)

$66 = 6\times10 + 6$

67 (GAP)

$68 = 6\times10 + 8$

$69 = 6\times10 + 9$

70-79 (GAPS)

$80 = 8\times10$

81 (GAP)

82 = 8×10 + 2

83 = 8×10 + 3

84 (GAP)

85 (GAP)

86 = 8×10 + 6

87 (GAP)

88 = 8×10 + 8

89 = 8×10 + 9

90 = 9×10 3 × 3×10 (REDUNDANCY)

91 (GAP)

92 = 9×10 + 2

93 = 9×10 + 3

94 (GAP)

95 (GAP)

96 = 9×10 + 6

97 (GAP)

98 = 9×10 + 8

99 = 9×10 + 9

The main problem with this system is the lexical gaps for values 1 and 7, and the gaps caused by the fact that the words for 4 and 5 are not of the appropriate syntactic category to fit into the higher-valued syntactic constructions. If those gaps were filled with appropriately categorized lexical

items, the system would be close to a modern decimal system. But note the hint of a vigesimal system in the various ways of expressing 60.

7 Conclusion

In the artificial approach described here, a numeral system resembling the dominant type found in the world's languages can emerge only very rarely, though it is not actually impossible. On the other hand, 'suboptimal systems', resembling the systems found in a number of isolated language communities throughout the world, emerge frequently. If a developed system is artificially imposed on a community with such a suboptimal system, the developed system is quickly adopted. If the approach outlined here has any verisimilitude, we may conclude that the natural primitive systems have an internal stability but are highly vulnerable to invasion (through language contact) by the developed decimal system which prevails throughout much of the world. This fits very well with the facts of language contact; in fact, 'exotic' numeral systems as have been shown above are typically abandoned in favour of a 'modern' decimal system. The simulations here show that this replacement may be due to some kind of real linguistic superiority (in coverage, lack of redundancy, and suitability to specifically human memory constraints) of the decimal system, and not just a consequence of the superior economic or military power of the invading culture.

REFERENCES

Austing, J. and Upia, R., 1975 "Highlights of Ömie Morphology", in T.E.Dutton (ed.), pp.513-598.

Brooks, Rodney A. and Pattie Maes (eds) 1994 *Artificial life IV* (proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems, MIT Press, Cam-

bridge, Mass.

Cauty, Andrè, 1986 "Taxinomie, Syntaxe et Economie des Numèrations Parlèes", *Amerindia* 11:87-141.

Conant, Levi Leonard, 1923 *The Number Concept: its Origin and Development*, MacMillan and Co., New York.

Conzemius, Eduard, 1929 "Notes on the Miskito and Sumu Languages of Eastern Nicaragua and Honduras", *International Journal of American Linguistics*,5:57-116.

Davis, Lawrence (ed.), 1991 *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York.

Dixon, R.M.W. 1980 *The Languages of Australia*, Cambridge University Press, Cambridge.

Dutton, T.E. 1975 *Studies in Languages of Central and South-East Papua*, (*Pacific Linguistics*, C-29.

Garfinkel, S.L. 1983 "The Game of Life on the IBM PC", *DR Dobbs Journal of Software Tools*, 8,6:42.

Goldberg, David E. 1989 *Genetic Algorithms in Search Optimization and Machine Learning*, Addison-Wesley Publishing, Inc., Reading, Massachusetts.

Holland, John H. 1975 *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, Michigan.

Hurford, James R. 1990 "Nativist and Functional Explanations in Language Acquisition", in I.Roca (ed.) *Logical Issues in Language Acquisition*, Foris Publications, Dordrecht, Holland. pp. 85-136.

Hurford, James R., 1994 "The Study of Language Systems", *Journal of Quantitative Linguistics*, 1,1:43-55.

Hymes, Virginia Dosch, 1955 "Athapaskan Numeral Systems", *International Journal of American Linguistics*, XXI, No.1:26-45.

Kirby, Simon, 1994 "Adaptive Explanations for Language Universals: a model of Hawkins' performance theory", *Sprachtypologie und Universalienforschung* 47: 186-210.

Kirby, Simon, 1996 "Competing Motivations and Emergence: explaining implicational hierarchies", *Language Typology*

Kluge, Theodor, 1937-42 I. *Die Zahlenbegriffe der Sudansprachen*; II. *Die Zahlenbegriffe der Australier, Papua und Bantuneger*; III. *Die Zahlenbegriffe der Voelker Amerikas, Nordeurasiens, der Munda und der Palaioafrikaner*; IV. *Die Zahlenbegriffe der Dravida, der Hamiten, der Semiten und der Kaukasier*; V. *Die Zahlenbegriffe der Sprachen Central- und Suedostasiens, Indonesiens, Micronesiens, Melanesiens und Polynesiens*, published by the author, Berlin.

Koza, John R., 1992 *Genetic programming : on the programming of computers by means of natural selection*, MIT Press, Cambridge, Mass.

Langton, Christopher G., C. Taylor, J. D. Farmer, and S. Rasmussen, (eds) 1991 *Artificial life II* (proceedings of the workshop on artificial life held February, 1990 in Santa Fe, New Mexico), Santa Fe Institute studies in the sciences of complexity v.10. Addison-Wesley, Reading, Mass.

Langton, Christopher G. et al. (eds) 1994 *Artificial life III* (proceedings of the workshop on artificial life held June, 1992 in Santa Fe, New Mexico), Santa Fe Institute studies in the sciences of complexity v.17. Addison-Wesley, Reading, Mass.

Lean, Glen A. 1985-6 *Counting Systems of Papua New Guinea* (volumes 1-12 and research bibliography), Department of Mathematics, Papua New Guinea University of Technology.

Levy, Steven, 1992 *Artificial Life* Pantheon Books.

Lewin, Roger, 1993 *Complexity: Life at the edge of chaos*, J.M.Dent, London.

Macdonnell, F. 1917 "Vocabularies – Cape Nelson Station, North-Eastern Division", *Papua, Annual Report for 1914-15*, pp.171-174, App.4.

Menninger, Karl, 1969 *Number Words and Number Symbols*, M.I.T.Press, Cambridge, Massachusetts.

Pott, August Friedrich, 1847 *Die Quinare und Vigesimale Zählmethode bei Völkern aller Welttheile*, Dr Martin Sändig oHG, Wiesbaden.

Ray, Thomas. S. 1991a "An approach to the synthesis of life". In: Langton, C., C. Taylor, J. D. Farmer, and S. Rasmussen (eds), *Artificial Life II*, Santa Fe Institute Studies in the Sciences of Complexity, vol. XI, 371-408. Redwood City, CA: Addison- Wesley.

Ray, Thomas S. 1991b. "Evolution and optimization of digital organisms" In: Billingsley K. R., E. Derohanes, H. Brown, III (eds.), *Scientific Excellence in Supercomputing: The IBM 1990 Contest Prize Papers*, Athens, GA, 30602: The Baldwin Press, The University of Georgia. Pp. 489-531.

Salzmann, Zdeněk, 1950 "A Method for Analyzing Numerical Systems", *Word*, Vol.6,No.1:78-83.

Seidenberg, A., 1960 *The Diffusion of Counting Practices, University of California Publications in Mathematics*, Vol.3,No.4:215-300. University of California Press, Berkeley.

Seife, C. 1994 "Mathemagician – Presenting the Legendary Conway, John, Horton – Trickster, Group Theorist, Inventor of the Game of Life", *Sciences - New York*, 34,3:12-15.

Weimer, H., and Weimer, N., 1974 *Yareba Language, Dictionaries of New Guinea, Vol.2*, Ukarumpa: Summer Institute of Linguistics.

Williamson, R.W. 1912 *The Mafulu, Mountain People of British New Guinea*, London: MacMillan.