ProctoSaurus Rex

The ProctoSaurus repository programs support hidden data in Thesaurus of the Gods text files. This information is located in the *structure* of the text file, and is not visible except through the application of these tools.

The small bit of software titled "Thesaurus of the Gods" re-writes a template message by substituting words from a thesaurus-like index for any word recognized in the input message. In a previous discussion, an encoding referred to as a chain described the specific index locations by which the words in the text template are replaced in the text output. This short paper points to some further useful properties of the template-saurus relation and its representation as a chain.

Using Thesaurus of the Gods to Encode Binary Data

From a slightly abstract viewpoint the Thesaurus of the Gods program performs a repeatable formal decomposition of the input template message text with respect to a given saurus index. That decomposition is represented as the chain. The chain is a sequence of pairs of numbers that correspond to the sequence of symbols (words) from the template message that also occur in the saurus index. The two numbers in the chain pairs refer to the smallest list (lowest in the index hierarchy) in which that symbol appears. Those two numbers are the size of that smallest list and the location in that list where the corresponding word appears.

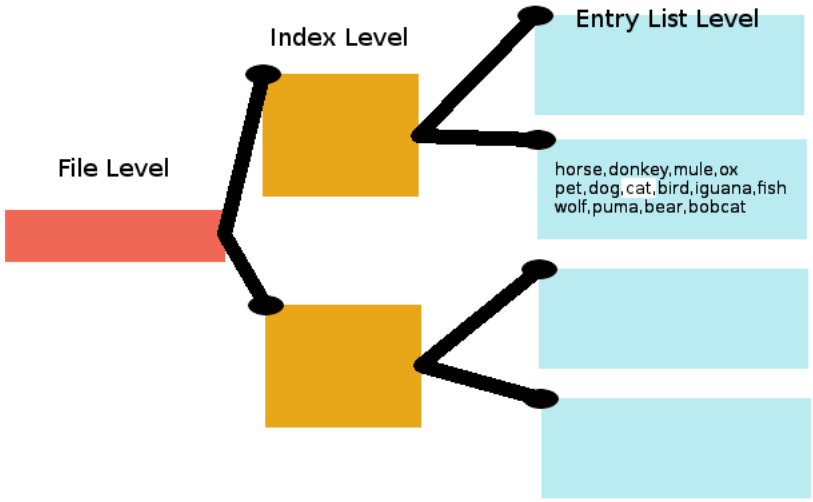
These smallest lists correspond conceptually to the head entries in a dictionary-style thesaurus. For example, the saurus may contain a short list of mutually replaceable words pertaining to pets. The list might be

pet,dog,cat,bird,iguana,fish

The size of this entry list is six. The term I use here for this list size is entropy, that is, the number of values the selection might take on. Any of the six values from zero to five can be used to select a symbol from this list. If the word cat appears in the template text then a corresponding entry appears in the chain with the numbers: 2,6; indicating that cat appears as selection 2 in a list with 6 elements. The diagram in figure 1 shows this relationship. The word "cat" is the second word in the middle entry, which is a list six words long.

When Thesaurus of the Gods processes the template and saurus files it creates an entry like the one just described for every word that matches a symbol in the global master word list. The order in which these entries appear is the same as the order in which those words appear in the template message. Whenever that specific template message is processed with that

Figure 1: The word 'cat' in a saurus hierarchy



same saurus index, the same chain sequence will be produced. The chain is an invariant attribute of the template-index pair.

The ProctoSaurus Number System

In an abstract way, the chain uniquely represents a number. We are accustomed to seeing numbers constructed on a constant number base. Every digit of the number representation is selected from a set of values ranging from zero to one less than the same number base throughout the representation. The digits of a hexadecimal number range from 0 to 15 exclusively. In a decimal number a digit never has a value 12; only 0 to 9. That is not, however, the only possible way to represent numbers.

By interpreting the chain as a number, I mean that the selection index is a digit in the number base given by the entropy value. In the same way that numbers are constructed with a constant number base, the leftmost digit in the chain is the big end, its value being multiplied by the number base of every digit to the right of it. Looking at the digit sequence 7534, it can be interpreted as usual as the base ten number

$$7 * 10^3 + 5 * 10^2 + 3 * 10^1 + 4 * 10^0$$

which, looking at it another way is

$$(((((7 * 10) + 5) * 10) + 3) * 10) + 4$$

Right? The number base gets in there one time fewer than the number of digits. Or if you want to interpret the same digit sequence in base sixteen, then, substituting 16 for 10 gives

$$(((((7 * 16) + 5) * 16) + 3) * 16) + 4$$

namely 30004 in decimal. But, there's no reason you can't mix bases and interepet the digit sequence 7534 as having an associated number base sequence $18, 7, 23, 6$, possibly writing it like a Thesaurus of the Gods chain: $7, 18; 5, 7; 3, 23; 4, 6;$, then calculating its decimal value as

$$(((((7 * 7) + 5) * 23) + 3) * 6) + 4$$

and define it as equal to the decimal number 7474. If that is the definition of the mapping between a chain representation and a numeric value, every chain has a unique value, like other number representations.

Ok, ok. Certain people will object to the fact that although a chain represents a unique value, the representation of a given value is not unique, that is, that the mapping relation is not a bijection. I'll write about that another time, because it's cool, but for the moment I'll just refer to the classic "On Numbers And Games" (ONAG) by John Conway, or Donald Knuth's "Surreal Numbers" as very competent presentations of other non-bijective number systems.

Continuing the previous pattern, for a digit sequence $DCBA$ written as a chain with number bases dcba: $D, d; C, c; B, b; A, a;$, then the numeric value is

$$(((((D * c) + C) * b) + B) * a) + A$$

or, another way of looking at it is

$$Dabc + Cab + Ba + A$$

Let's call the chain C and the nth element of the chain C_n with the selection component of element n being s_n and the entropy component e_n for purposes of this exposition.

A concise and more or less formal statement of the value of the chain is

$$v = \sum_{i=0}^{n} s_i (\prod_{j=0}^{i-1} e_j)$$

Algorithmically the preceding nested parenthesis form is closer to the code. The pattern

$$(((((s_3e_2) + s_2) * e_1) + s_1) * e_0) + s_0$$

Without the redundant parenthesis:

$$((s_3e_2 + s_2) * e_1 + s_1) * e_0 + s_0$$

is calculated from the inside out starting with the high-order digit and accumulating the product of entropy components as the calculation proceeds rather than computing them separately.

A Brief Look at the Code

Obviously these are some fairly large numbers, even for small to moderate message sizes. The ProctoSaurus C code handles that by linking to the excellent GNU Mult-Precision (GMP) library and utilizing the mpz big integer formats and functions.

The text representation of the chain used in Thesaurus of the Gods is read and translated into an array of the simple binary structures shown in code fragment 1.

Code Fragment 1 Binary data type of the chain array elements

```
// binary chain data type
typedef struct _EntSeq
{
  int Sel;
  int Ent;
} EntSeq, *pEntSeq;
```

Calculating the EChain array into a single decimal number is done in the simple loop shown in code fragment 2.

Code Fragment 2 Calculation of a single decimal value from a chain

```
  int i;
  pEntSeq EChain; // binary chain array
  int EChainLen; // number of elements in the EChain array
  mpz_t ChainBin; // GMP big integer variable for chain value
...
// the conversion loop
  mpz_init_set_ui(ChainBin, EChain[0].Sel);
  for(i=1; i < EChainLen; i++)
  {
    mpz_mul_ui(ChainBin, ChainBin, EChain[i].Ent);
    mpz_add_ui(ChainBin, ChainBin, EChain[i].Sel);
  } // for

  gmp_printf("Chain content:\n%Zd\n", ChainBin);
```

Arbitrarily, the leftmost (high order) digit of the chain is defined to be the zeroth element of the chain array (like a string representation of any number). The big number variable ChainBin is initialized to the selection number of this leftmost digit. The loop just multiplies ChainBin by each succeeding entropy component and adds the selection component. The gmp_print handles very large numbers and long strings easily. As an example, the output of this loop when calculating the value of the 136 element chain produced from the BunchOfNamesMsg.out.chain.txt example from the Thesaurus Of The Gods example script, produces the following output (broken into smaller segments to make it conveniently printable here in code fragment 3):

Code Fragment 3 BunchOfNamesMsg.out.chain.txt example file value

```
Chain content:
730377420665243306871696757646480103582724868292317464295047205489059083915040618210255365940570106378526000572192082244155110273542093544000642952688608110601483786080712133920939769451344924290589281380110487706393962710044058004851793
49809102
```

Reversing the Process of Representation

Given a number in standard numeric form, such as the output value in code fragment 3, how do you convert it back into a chain representation? Let's start by verifying the translation by returning the value shown in code fragment 3 to the string representation of the chain from the Thesaurus of the Gods example file BunchOfNamesMsg.out.chain.txt. What does that look like? It's very similar to the conversion of a binary number to a printable digit string. In the case of base ten conversion, the low order digit is the remainder of the variable mod ten. Now divide the variable by ten and find the mod ten remainder of that result. Repeat this until the variable is reduced to zero. This procedure gives us the individual digits of the variable in reverse order, that it, starting with the low digit and working up to the high.

Same thing for hexadecimal conversion, or base 64 representation. If the variable is already in binary format (which is very likely in an ordinary computer) then you have the added advantage that a four-bit and mask is the same as mod sixteen, a six-bit mask is mod 64, right shift four bits is divide by sixteen, and right shift six bits is division by 64.

Converting the value V, to the digit sequence base B, with zero being the low-order position, and calling the nth digit D_n, we have

$$D_n = (V/B^n) \ mod \ B$$

The process of converting a GMP variable to a chain is very similar, except that where an ordinary digit conversion uses the same base repeatedly, hence the B^n term, the chain conversion uses the entropy value from each chain element as it steps through the sequence. The "digit" value is the selection value for each item in the chain. The conversion of a numeric value requires specification of a sequence of entropy values used in the new chain. Again using the previous notation, the selection components s_n of the constructed chain are

$$s_n = (V / \prod_{i=0}^{n-1} e_i) \ mod \ e_n$$

Algorithmically, again, we work from the low digit to the high, dividing by each entropy component and saving the quotient for the next step. A snipet of the code is shown in code fragment 4.

Code Fragment 4 Converting a numeric value to a chain

```
pEntSeq OutSeq;
mpz_t InNum;
mpz_t Rem; // remainder that is used multiple times for mod calls
mpz_t Tmp; // generic temp variable to conserve allocations

mpz_set(Tmp, InNum);
for(i=0; mpz_sgn(Tmp) != 0; i++)
{
  OutSeq[i].Ent = GetEnt();
  mpz_fdiv_qr_ui(Tmp, Rem, Tmp, (unsigned long) OutSeq[i].Ent);
  OutSeq[i].Sel = (int) mpz_get_ui(Rem);
} // for
```

The snipet shows the chain OutSeq[] being created from the GMP numeric value InNum given some sequence of entropy values acquired by repeated calls to GetEnt(). A temporary big numeric variable is initially assigned the value of InNum, which is the value that will create the chain. The GMP function mpz_fdiv_qr_ui() repeatedly divides Tmp by the unsigned integer entropy into a quotient and remainder. The remainder is converted to an unsigned integer which is the selection component (or digit) of the chain element. The result is a chain array OutSeq[] whose zeroth element is the low order digit. Notice that this is the opposite of the order of the code snippet in code fragment 2. The arithmetic is clearer when shown this way, but in practical code you need to reverse one or the other.

The Non-uniqueness of the Chain Representation

From the two preceding sections it should be clear that the selection of the entropy sequence is somewhat arbitrary. It is simply a sequence that describes the partitioning of the numeric value into several parts. It can be

partitioned into many small parts, a few larger parts, a large and small mix, or, as in the case of ordinary representations in constant number bases, it can be partitioned into same-size pieces. The following are few simple examples using the 32-bit decimal value 5432123.

```
Chain: 5,10;4,10;3,10;2,10;1,10;2,10;3,10;
```

$$(((((5 * 10 + 4) * 10 + 3) * 10 + 2) * 10 + 1) * 10 + 2) * 10 + 3 = 5432123$$

```
Chain: 5,16;2,16;14,16;3,16;3,16;11,16;
```

$$((((5 * 16 + 2) * 16 + 14) * 16 + 3) * 16 + 3) * 16 + 11 = 5432123$$

```
1,2;0,3;1,5;5,7;6,11;8,13;12,17;4,19;
```

$$((((((1 * 3 + 0) * 5 + 1) * 7 + 5) * 11 + 6) * 13 + 8) * 17 + 12) * 19 + 4 = 5432123$$

```
Chain: 10,19;10,17;11,13;6,11;1,7;3,5;2,3;1,2;
```

$$((((((10 * 17 + 10) * 13 + 11) * 11 + 6) * 7 + 1) * 5 + 3) * 3 + 2) * 2 + 1 = 5432123$$

On the other hand, just slightly mixing up the previous chain:

```
Chain: 6,11;10,19;1,7;11,13;3,5;2,3;1,2;10,17;
```

$$((((((6 * 19 + 10) * 7 + 1) * 13 + 11) * 5 + 3) * 3 + 2) * 2 + 1) * 17 + 10 = 5767481$$

The last two sequences use the same set of entropy numbers to partition the numeric value (the first eight prime numbers) but the selection components are very different. The two chains are very different. They are not just the same chain elements in a different order. Revisiting the way chains originate in the Thesaurus of the Gods, two different chain sequences that have an entropy set in common can be generated from different template messages using the same saurus index. Notice that a given numeric value can be encoded in very, very many different ways from a given saurus index. Notice also, however, that a few changes to the order in which the template words occur, or the ordering of the saurus causes large changes in the numeric value.

Encoding Various Data Sources as Chains

From a practical standpoint, where would you find large numeric values that might be interesting to encode in bogus text messages? Certainly you might occasionally want to send an SHA256 hash code, or something similar without

making it accessible to everyone who sees your tweets. A plain text password that's only valid for brief time, might be another example.

A hex number can be imported directly into a GMP integer variable by simply specifying the number base. Ordinary 8-bit ASCII text can be converted to a single binary value in much the same way a numeric text string is converted. The very simple function in code fragment 5 shows one way to perform the conversion. This example takes advantage of the fact that most ASCII text does not use the high-order bit of the byte. If you are encoding special characters you should use 256 in place of 128 in the example.

Code Fragment 5 Converting an ordinary text buffer to a GMP number

```
int EntPackText(mpz_t OutNum, char *InText, int InLen)
{
  int i;

  for(i=InLen-1; i >= 0; i--)
  {
    mpz_mul_ui(OutNum, OutNum, (unsigned long) 128);
    mpz_add_ui(OutNum, OutNum, (unsigned long) (InText[i] & 0x7f));
  }

  return(0);
}
```

Similarly, unpacking the large GMP number into a text buffer is like converting any integer into a string. Another short function shown in code fragment 6 is one approach.

Code Fragment 6 Converting a GMP number back to an ordinary text buffer

```
int EntUnpackText(mpz_t InNum, char *OutText)
{
  int i;
  unsigned long URem;

  mpz_set(Tmp, InNum);
  for(i=0; mpz_sgn(Tmp) != 0; i++)
  {
    mpz_tdiv_qr_ui(Tmp, Rem, Tmp, (unsigned long) 128);
    URem = mpz_get_ui(Rem);
    OutText[i] = (char) URem;
  }
  OutText[i] = 0;

  return(i);
}
```

Clearly, a short message like the BunchOfNames examples can encode a password, or even a short passphrase, but the same technique can be used to convert much longer text sequences to message chains.

If you were writing a book and storing it privately in a directory full of innocuous looking bogus text, for example, more than one GMP number

might be required to hold it all. Even shorter entries like those of a private diary might overflow a single numeric value. You might also want to keep a variety of template and saurus file variations that would make extraction of the original data less obvious to a potential eavesdropper. So, how do you determine whether a given template and saurus will overflow?

The Data Capacity Of a Message

As was previously explained, a given template and saurus uniquely determine a chain that describes the relationship between the pair. The selection components of the chain describe the index locations of the substitutable words in the template. A text replacement generated from that template will have the same entropy components but different selection components, therefore it will have a different numeric value associated with the chain. The maximum that this second value can have is determined by the entropy components of the chain.

This can be easily seen by considering the maximum value that can be held in (for example) a 16-bit word. Every 16-bit word can be expressed as tho bytes, written as four hex digits. The maximum value of each hex digit is 15, one less than the number base, written as 'f', so the maximum value of the 16-bit number is written ffff, or 6435; one less than 16^4.

If every selection component of a chain had its maximum value, they would all be one less than the corresponding entropy component. The maximum value that can be encoded in a given length chain with given entropy components is one less than the product of all those components. More formally:

$$v_{max} = \prod_{i=0}^{n} e_i$$

A code snippet in code fragment 7 shows the calculation.

Code Fragment 7 Calculating the capacity of a chain

```
  pEntSeq EChain;
  mpz_t ChainCap;
...
  mpz_init_set_ui(ChainCap, EChain[0].Ent);
  for(i=1; i < EChainLen; i++)
    mpz_mul_ui(ChainCap, ChainCap, EChain[i].Ent);

  gmp_printf("Chain capacity:\n%Zd\n", ChainCap);
```

It is convenient to know the binary value that will be encoded in a chain first, then select a template and saurus pair that has capacity at least one

greater as that value. If the encoded value is significantly larger, then multiple encodings or selection of a larger or higher density template/saurus pair will be required.

By density I mean the ratio of the number replaceable words to the total number of words in the template. The density can be increased either by adding more words to the template from the lists in the saurus, or by increasing the number of words in the saurus that occur in the template text. The template may have coherent readability or some other property that makes changing it undesirable. Density might still be increased by improving the saurus file. Using the tools and adding name list files to the multi-file input, an enlarged saurus can be easily created. Keep in mind that when a value is inserted with a given template/saurus pair, the same saurus must be used to extract it.

A Word About Eve

The preceding example of disguising one's private diary pages in daily saurus re-writes of a few templates - possibly from a thumb drive copy of Thesaurus of the Gods - raises some additional issues. In the presence of an **Eve**sdropper who (hopefully) doesn't have a copy of the saurus files (but might understand the principles of Thesaurus of the Gods well enough to attempt to guess the entropy chain) a couple of factors increase in importance.

Consider a saurus file composed from many tiny lists with only a few entries each. Eve might start by eliminating the obvious articles, prepositions, pronouns, etc. (a, an, and, but, the, he, she...) and try brute-force combinations of small entropy numbers. Especially if numerous examples of TotG output texts are available that were created with the same saurus, it would be fairly easy to guess a workable entropy chain and reconstruct a partial saurus that matches those files. For any identified substitution there would only be a few likely entropy components and only a few selections to try within that range.

At the other end of the spectrum, a saurus file with many entries in every list makes guessing significantly harder, but requires many times more words to make up the saurus. Every selection contributes a larger chunk to the numeric value, since the entropy is larger, so the number of replacement words in the template is reduced. That also has the effect of reducing the number of guesses Eve has to make. If one of the numeric values embedded in the output files becomes known (i.e., a plaintext attack - perhaps only partial) then merely guessing the entropy components of the chain, since they are relatively few in number, reveals the structure of the saurus index.

A middle-of-the-road option is provided in the Thesaurus of the Gods

tools code. For a text list file containing unstructured data, that is, the data is not necessarily a collection of synonyms, the function XformIndexToFull-Partition() transforms an index component into a particular structure. It collects all the words in the index into a single pool, then partitions the pool into a collection of lists whose lengths are the counting sequence starting with the number two. However many entry lists are required to cover the size of the pool, that is the number of lists created. This maximizes the variation in the entropy components available while also using the fewest items from the pool to do so. The calculation is shown in code fragment 8.

The variable PartCnt is the number of partitions required to cover the size of the pool. The IdealSum variable is the sum of the consecutive integer sequence of that length starting with the number two. For the unbeliever, the sum of the first n integers is

$$x = \frac{n^2 + n}{2}$$

so, if x is the pool size

$$n = \frac{\sqrt{8x + 1} - 1}{2}$$

We truncate this real number into the largest integer less than n (floor), so the last partition will be a little larger to catch the leftovers. Since we are starting with 2, not with 1, we want

$$\frac{x(x + 3) + 2}{2} - 1$$

because we really want

$$x = \frac{(n + 1)^2 + n + 1}{2} - 1$$

All right... Maybe contorted logic. It's how I think these days. If you don't like it, re-write it.

Code Fragment 8 Calculating the Full Partitioning Parameters

```
PartCnt = (int) ((sqrt((double) (TotalWords * 8 + 1)) - 1.0) / 2.0);
IdealSum = (PartCnt * (PartCnt + 3) + 2) / 2 - 1; // sum of consecutive integers starting with 2
```

The eavesdropping situation is too complex to examine deeply here. The essential element that Thesaurus of the Gods provides is that every text, whether TotG output or not, has a corresponding chain and a numerical

value with respect to any saurus that contains a few (or more) words that appear in that text. That fact makes the files on your hard drive and the entire internet a target-rich environment for anyone who wants to read your private diary. It makes it easy to create plausible deniability that a heavily encrypted binary file may not have. (You can always encrypt it and use that as the binary numeric value.) It also requires the specific pairing of files in an environment in which many are very similar to each other. The proverbial maze of twisty passages, every one the same.

Further

This little exposition has covered the basic aspects of a simple application. It is detailed in an attempt to be clear, not to be tedious. The concepts are simple enough to be manipulated by fairly inexperienced programmers, but interesting and useful enough to appeal to a more sophisticated taste. It is hoped that many applications in many variations will appear in the hands of others.

The emphasis in the ProctoSaurus and Thesaurus of the Gods has been simple text files. The same principles apply to denser data structures, such as image files. They apply to more abstract substitution systems - in particular to deeper substitution systems. The multi-base number system also has many interesting properties, both related and unrelated to this general theme. Furthermore, the properties of thesauruses, lists, and collections of lists have fascinating properties with little or no relevance to the applications at hand. In particular, possible applications to the analysis of genetics are interesting. I continue to explore these themes and if there seems to be interest, I'll post some more code. Hopefully, others will, too.