

# AUTOMATED TRANSLITERATION OF EGYPTIAN HIEROGLYPHS

*Serge Rosmorduc*

## ABSTRACT

*This article describes a system which is able to give a reasonable transliteration of Middle Egyptian Hieroglyphic texts, using a set of 'rewriting rules'. I give a brief explanation of the inner working of the system, and then proceed to describe the detail of the said rules.*

## INTRODUCTION

A number of years ago, while working on automated syntactic analysis of Middle Egyptian for my doctorate, I considered the possibility of automated transliteration.<sup>1</sup> However, as this task was not central to our work back then, I did not explore the problem in depth. In 1997, I had an engineering student, F. Kerboul,<sup>2</sup> work on the subject again, using the Prolog computer language; she did work on the analysis of isolated words, and, as the results were encouraging, I decided to study the subject in more detail.

---

<sup>1</sup> S. Rosmorduc. (1996). *Analyse morpho-syntaxique de textes non ponctués, Application aux textes hiéroglyphiques*. PhD Thesis, p. 75; id, 'Traitement automatique du langage naturel en moyen égyptien', PIREI X, Bordeaux, 1994, p. 100–101.

<sup>2</sup> F. Kerboul (1997). *Translittération automatique des hiéroglyphes*. Rapport de stage de l'ENSTA.




The system described in this article takes as input a description of a hieroglyphic text as a list of ‘Manuel de codage’ codes, and outputs a transliteration of the text.


This article is written with two audiences in mind, both Egyptologists and specialists of the Natural Language Processing. Thus, it contains a number of rather basic explanations, both on the problem itself and on the method used to solve it.





## ANALYSIS

### Transliteration of Egyptian Hieroglyphs

When an Egyptologist works on a hieroglyphic text, he transcribes it into latin characters which represent its consonantal skeleton. In my case, the input will consist in codes designed to describe the hieroglyphs, defined in the so-called ‘Manuel de codage’<sup>3</sup> (MdC).

|                 |   |   |   |
|-----------------|---|---|---|
| Hieroglyphs     |  |  |  |
| MdC             | I10:D46-M17-N35   | T18-G43-A1  | M17-N29:D21:Y1  |
| Transliteration | <i>ḏḏin</i>   | <i>šmsw</i>   | <i>iqr</i>  |
| Translation     | said  | servant   | excellent   |

The Manuel de Codage describes the hieroglyphic texts with signs codes such as I10 for , and positional codes in the form of ‘?’ for ‘stack above’. The sign values can fall into different categories:

- phonetic signs, which represent one or more consonants, as the vowels are not written. For instance, the I10 snake is *ḏ* (like English ‘j’), while T18 () is *šms* (sh+m+s).
- ideograms, which refer directly to their meaning: for instance,  E1 can be used to write the word *ih*, ‘ox’.
- determinatives, which are a kind of semantic classifier, used at word endings. In our example,  and  (a papyrus roll) are determinatives, respectively for human beings and for abstract concepts.

<sup>3</sup> J. Buurman, N. Grimal, M. Hainsworth, J. Hallof and D. v. der Plas *Inventaire des signes hiéroglyphiques en vue de leur saisie informatique*. Mémoires de l’Académie des Inscriptions et Belles Lettres. Paris: Institut de France (1988)

Now, automated transliteration is not as simple as it might seem. Firstly, signs may have more than one value. For instance, the papyrus roll is also an ideogram in the word *md3:t*, ‘document’. Secondly, signs combination is not straightforward. A multi-consonantal sign is often accompanied by ‘alphabetic’ signs, which correspond to part of its spelling. For instance, Y5  $\equiv$  has the value *mn*, but is often accompanied by N35,  $\equiv$ , which is *n*, and the group Y5:N35  $\equiv$  is *mn*, not *mmm*.

### BASIC APPROACH

To give a first idea of how automated transliteration can work, let us start with a simple example, considering the word  $\text{𓏏} \text{𓏏} \text{𓏏}$ , *3b*, ‘to desire’. In reality, this word is encoded in the ‘Manuel de codage’ as ‘U23-D58-A2.’

The signs can have the following value:

1. U23,  $\text{𓏏}$ , can correspond either to the consonants *3b* or *mr*.
2. D58,  $\text{𓏏}$ , is the consonant *b*.
3. A2,  $\text{𓏏}$ , is a determinative for actions linked to the mouth.

Knowing the values of the signs is not enough. We also need to combine them. In reference to the phonetic signs here, two different rules may apply:

- a) we can consider that a biliteral XY sign followed by an uniliteral sign Z can be combined into a group of three consonants, XYZ,
- b) or we can consider that a biliteral sign XY, followed by an uniliteral Y, i.e. followed by its last consonant, can be combined as a group of two consonants, XY.

In all cases, the sign A2 marks the end of the word, and has no phonetic value. Using all possible solutions, we end up with three interpretations:

using the value *mr* for U23, we can only use rule a), which gives the transliteration *mrbb*.

using the value *3b* for U23, we can use either rule a), producing *3bbb*, or b), giving *3b* (the correct solution).

Then, we need a way to represent those ‘rules’, and a way to choose the ‘best’ solution.

**Representation of the rules (simple example)**

The rules will be represented as ‘rewriting rules’, stating that, for a given data, we will obtain a given conclusion. For instance, the two different values of U23 will be represented as:

$$\mathbf{r1.} \text{ U23} \Rightarrow \text{P(A,b)} / 100$$

$$\mathbf{r2.} \text{ U23} \Rightarrow \text{P(m,r)} / 100$$

where P(A,b) means ‘phonetic sign with value A,b’, and 100 is the ‘cost’ of the rule, which is a measure of its correctness. The actual values given to the costs are rather *ad-hoc*.

The other rules for signs values would be something like:

$$\mathbf{r3.} \text{ D58} \Rightarrow \text{P(b)} / 100$$

$$\mathbf{r4.} \text{ A2} \Rightarrow \text{DET(mouth\_action)} / 100$$

Now, this is a first set of rules, which, applied to the entry ‘U23 D58 A2’, can produce the results:

P(A,b) P(b) DET(mouth\_action), for a cost of 300

or P(m,r) P(b) DET(mouth\_action), also for a cost of 300.

A second set of rules, used to combine the signs, will then be applied.

A first rule will state that a uniliteral sign can be read on its own:

$$\mathbf{r5.} \text{ P(\$X)} \Rightarrow \text{L(\$X)} / 100$$

here, \$X is a variable which can be replaced by any consonant. We use two different operators, P() and L() to differentiate sign values (that is P) and word transliteration (that is L).

The same rule is true for biliteral signs:

$$\mathbf{r6.} \text{ P(\$X,\$Y)} \Rightarrow \text{L(\$X), L(\$Y)} / 100$$

But of course, there is the rule which allows to combine a biliteral and an uniliteral sign:

$$\mathbf{r7.} \text{ P(\$X,\$Y), P(\$Y)} \Rightarrow \text{L(\$X), L(\$Y)} / 100$$

Last, we can envisage a rule stating that a determinative can mark the end of a word:

$$\mathbf{r8.} \text{ DET(\$X)} \Rightarrow \text{wordend} / 100.$$

### Applying the rules

The first set of rules, concerning the sign values, is applied to the input text. This creates a set of possible interpretations; then we apply the second set of rules to those interpretations.

The principle is that the ‘cost’ of an hypothesis is the total of the costs of the rules used, and the ‘best’ hypothesis is the one with the least cost.

For instance, to obtain the transliteration *mr̄b*, one has to use rules r2, r3, and r4, and then apply rules r5, r6, and r8, the result being ‘L(m), L(r), L(b), wordend’, with a total cost of 600.

To obtain the transliteration *ꜥb̄*, the rules used will be r1, r3 and r4, and then r7 and r8, with a cost of 500. Hence, *ꜥb̄* is better than *mr̄b*.

One problem with this system is that we need somehow to consider all the possible combinations of rules, which, at first sight, is a daunting task. For instance, if we consider the word  $\{\downarrow\equiv\wedge$  (V24-D58-F46-D54), *wdb̄*, ‘turn, fold’,

$\{\downarrow$ , V24, can be P(w,D)

$\downarrow$ , D58, can be P(b)

$\equiv$  can be either P(p,X,r), ID(w,D,b) or DET(folding\_action)

$\wedge$  can be ID(i,w), ID(n,m,t,t), or DET(move)

where ID means ‘ideogram’.

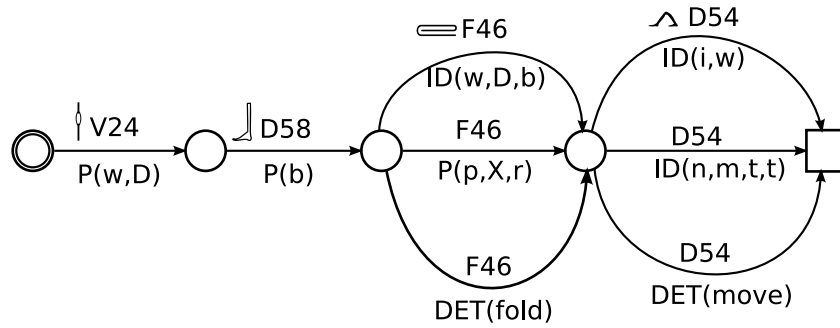
If we simply consider the possibilities here, we have  $1 \times 1 \times 3 \times 3$ , hence 9 interpretations for the sign values. To summarise, the number of hypotheses tends to grow *exponentially* with the length of the text.<sup>4</sup>

Now, rewriting rules is a very well-studied field of computer science, and we have an elegant representation for both the rules and their interpretation, called a *finite state transducer*, itself an extension of the notion of *finite state automaton*.<sup>5</sup> The notion is rather simple to grasp graphically. Here is a transducer representing our sequence of signs:

---

<sup>4</sup> To give another example, if we had four ambiguous signs, each with three interpretations, the total number of combinations would be  $3 \times 3 \times 3 \times 3$ , 81 possibilities!

<sup>5</sup> For a theoretical and practical coverage of the subject in relation with computing and Natural Language Processing, see E. Roche and Y. Schabes, ed. *Finite-State Language Processing*, MIT Press, 1997.



On each link in the transducer, there are two pieces of information: the *input* (on top of the link), which is what can be read by the transducer, and the *output* (below the link), which is what can be produced by the transducer. The input corresponds to the left part of our rules, and the output, to the right part. Any path from the leftmost node of the transducer to its rightmost node (the square one) represents a choice of rule applications. Hence, the transducer permits the representation of a large number of hypothesis in a very compact way. Another important feature of transducers is that they can represent both the entry, the rules and their results.

Let us consider a few more rules to permit us to go further:<sup>6</sup>

$$ID(\$A,\$B) \Rightarrow L(\$A), L(\$B) / 100$$

$$ID(\$A,\$B,\$C,\$D) \Rightarrow L(\$A), L(\$B), L(\$C), L(\$D) / 100$$

$$ID(\$A,\$B,\$C) \Rightarrow L(\$A), L(\$B), L(\$C) / 100$$

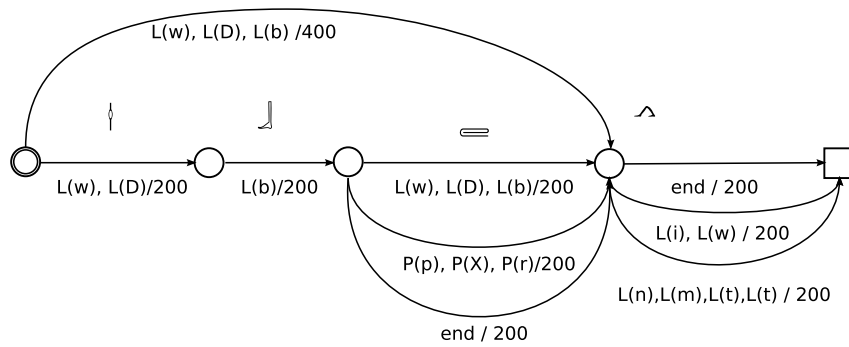
$$P(\$A,\$B), P(\$B), ID(\$A,\$B,\$C) \Rightarrow L(\$A), L(\$B), L(\$C) / 100$$

$$P(\$A,\$B,\$C) \Rightarrow L(\$A), L(\$B), L(\$C) / 100$$

Once applied, those rules will give us the following transducer:

---

<sup>6</sup> Those rules are in reality rather over-simplistic to allow the whole analysis to be performed, but I wish here to underline the mechanisms of the system.



Each path in the transducer corresponds to a possible analysis (of course, only a few of which are reasonable). If we count them, we find that there are now twelve different possibilities, but the transducer represents them all with only nine links. An interesting feature is that choices which are independent from one another are clearly separated by the automaton.

**Problems to solve**

This being said, there are a number of problematic points to deal with if we want to perform a reasonable automated transliteration of a hieroglyphic text. We want to be able to separate the various words. To do this, we need somehow to model the form of an Egyptian word.

Firstly, on the linguistic level: a biliteral or triliteral word is usual, but longer words tend to have specific forms, either because they contain grammatical endings, or because they are derived from simpler roots by the addition of prefixes or reduplication.

Second, on the graphemic level: a word tends to contain a phonetic part, followed by an ending with a determinative and various marks, like plural indicators. The structure of a word ending, and in particular the determinatives, needs to be described thoroughly.

This is a reasonable description for most words, but not all. Some are written ideographically, but they are not very usual in hieratic texts. More annoying is the problem of function words which often do not follow the general pattern. In this case, the reasonable thing to do is to introduce some lexical information in our system, and have specific rules for those words.

## ARCHITECTURE OF THE SYSTEM

To design a system able actually to deal with texts, we needed to choose a corpus. The principles of the hieroglyphic system is the same for all texts, but the details vary a lot. For instance, the orthography changed considerably from the 12th to the 19th dynasty. Even if we keep a synchronic corpus, the orthography of hieratic texts, or cursive texts in general is much more explicit than that of monumental texts. The system we are about to describe was tailored for Middle Kingdom Hieratic texts.

To achieve our goal, we need to make the system somewhat more complex. We will need, not two, but five layers of rules. Each layer will be applied to the result of the previous one. All layers are implemented as transducers, but some layers do not follow the model above.

## LAYERS OF THE SYSTEM

### Sign Normalisation

This first layer takes as input the MdC codes, and outputs a simplified representation thereof. The ‘phonetic codes’ of the MdC are replaced with Gardiner codes, and some sign variants are replaced by their base sign (for instance, F47  $\equiv$  and F48  $\equiv$  are variants of F46  $\equiv$  and will be replaced by ‘F46’).

### Sign values

The next level of rules takes as input normalised codes, and replaces them with all their possible interpretations. We keep the information about biliteral signs and so on, in order to combine the signs values in the next layer.

Let us give a few typical rules, about the sign D36  $\text{𓄓}$  :

D36  $\Rightarrow$  P(a) / 100

D36  $\Rightarrow$  ID(a) / 200

D36  $\Rightarrow$  DET(action) / 500

D36 can be understood as a uniliteral sign, which is the preferred interpretation (with a cost of 100); but it does also behave as an ideogram, when writing the word ‘arm’ or the word ‘condition, state’.<sup>7</sup> The cost for this interpretation is slightly higher. The idea is that, in later rules, the combination D36 and Z1 ( $\text{𓄓}$ ) will be interpreted as an ideographically written word



at a relatively low cost. The third rule deals with a confusion between D36 and D40 𓄀, as a determinative for actions. Note that this could have been dealt with in the first level instead.

In the current version of the software, we distinguish between different kinds of values:

Phonetic values, coded by ‘P’

Ideographic values, coded by ‘ID’

Phonetic-ideographic values, coded by ‘IP’.<sup>8</sup>

Some signs are also real ‘word-signs’, which differ from the usual ideogram in that they write a word on their own, whereas the typical Egyptian ideogram is usually combined with Z1 𓏏 to do so (for instance 𓏏𓄀). In this case, we decide to generate the full interpretation of the word right away. For instance, O3, 𓏏𓄀, *prt-brw* will be treated by the rule:

O3 => L(p), L(r), L(t), wordend, L(x), L(r), L(w), wordend / 100

where ‘wordend’ is a marker for word endings. This data will be copied as is by the following layers.

This is also a good place to deal with a kind of hiatus which occurs sometimes in the MdC. For instance, the group R22:R12 𓏏𓄀 is in fact to be understood as a whole, i.e. it should be considered as one sign.<sup>9</sup> The same is true for the determinative T14-G41 𓏏𓄀, which is not a ‘double’ determinative, but represents in fact a bird being hit by a throw-stick, as shown in the variant G84: 𓏏𓄀.

Finally, as some signs are rather strong markers of some specific words, especially of function words, but may be combined with phonetic complements, we decided that, in the next layer, it might be interesting to keep the sign codes themselves, so we have a ‘copy rule’:

\$X => \$X / 0

<sup>7</sup> In this last case, it is not of course an ideogram *stricto sensu*, but it does behave like one.

<sup>8</sup> The relevance of the category of phonetic-ideographic value is somewhat dubious, as was pointed to me by P. Vernus. They could probably be replaced with ideographic values.

<sup>9</sup> W. Schenkel ‘Gesichtspunkte für die Neugestaltung der Hieroglyphenliste’, *Göttinger Miszellen*, vol. 14, 1974, p. 31–45.

### Simple Groups

This layer takes its input from the previous layer, i.e. sign values, and combine those signs. It does not produce words yet, as words can be composed of multiple and rather complex groups of signs. Basically, this level of rules does two things:

- a) it combines phonetic complements with other signs, saying that  $\overline{mn} + \dots n$  is  $mn$  and not  $mnn$  (dealing with exceptions like D4:D21  $\overline{ir}$  which is  $irr$  and not  $ir$  in our corpus).
- b) it tries to ascertain word endings

Regarding case a), we obtain as input phonetic values like P(m,n) or P(n), ideographic values, and ideographic-phonetic values. We need to combine them to realise the values of the resulting groups.

However, as we are interested in word formation, we need to keep the information that all those consonants belong to the same group, as a group belongs to only one word. Hence, our rules have the form:

$$P(\$X,\$Y), P(\$Y) \Rightarrow G(\$X,\$Y), \text{groupend} / 10$$

which means that a phonetic sign with value  $\$X\$Y$ , followed by an unilateral sign of value  $\$Y$  should be read  $\$X\$Y$ . The fact that we grouped those signs is marked by the G() symbol, and also by the groupend, which is used in the following layers to deal with group combination.

Regarding word endings, we model possible combinations of signs. The idea is to represent the way the various determinatives and grammatical markers may be combined in words. In particular, a number of phonetic signs, coding the plural, the feminine, or various verbal values, may be mixed with the determinatives. We have decided to re-order them at that stage, which is convenient for the next layer, where we will build the word. Let us look at two significant rules for this purpose:

$$\text{DET}(\$x) \Rightarrow \text{wordendstart}, \text{DET}(\$x), \text{wordend} / 100$$

$$\text{DET}(\$x), P(t) \Rightarrow \text{wordendstart}, L(t), \text{DET}(\$x), \text{wordend} / 100$$

The first rule states that a determinative may end a word by itself. The second takes a group consisting of a determinative and a 't', and states that this can be a word ending. However, the 't' is output in front of the determinative for the next layer.

I will take advantage of this rule to point out why it is interesting to distinguish 'phonetic signs' from ideograms in our system. If we consider X2,  $\vartheta$ , which for our period is an ideogram for  $t$ , 'bread', it cannot be used

as a phonetic marker for feminine words. And that is exactly what our system does, as there is no rule 'X2 => P(t)' in the 'sign values' level.

The set of rules in this level is rather large, as we need to deal with all possible configurations of signs. Except for the most obvious ones, this was achieved through analysis of the corpus.

### Word formation

The task of the next set of rules is to combine the various groups into words. It uses the markers created by the previous level.

First, the wordendstart and the groupend markers allow us to express that a word should in general contain both a phonetic part and a determinative part (for those which do not, we have more specific rules). Thus, we have a rule

$$\text{groupend, wordendstart} \Rightarrow \text{epsilon} / 0$$

which means that a group ending followed by a typical 'end of word' part is normal, erasing the sequence groupend, wordendstart, with a cost of 0 ('epsilon' stands for 'nothing').

$$\text{groupend} \Rightarrow \text{wordend} / 10000$$

which means that a group ending can be a word ending too, but at a very high cost in general. Words for which this is normally the case are few, and can be dealt with using specific rules.

Next, we have rules to combine the various groups. For instance, a word may be written with a biliteral group:

$$G(\$x, \$y) \Rightarrow L(\$x), L(\$y) / 100$$

Or it can be written with two uniliteral groups:

$$G(\$x), \text{groupend}, G(\$y) \Rightarrow L(\$x), L(\$y) / 100$$

In this last rule, the group ending which corresponds to the first group is 'consumed' for a small cost (much less than the 10000 needed previously), which means that the system will prefer to group two uniliteral groups in a biliteral word than to make a word with the first group on its own (once again, function words like suffix pronoun have specific rules which short-cut this one).

Long words are not very usual in Egyptian, and they tend to follow specific patterns. For instance, a word can be formed on a biliteral root by reduplicating it, with often a strengthening effect. For instance the root *qn*,

'to be strong' generates the verb *qnqn*, 'to beat'. We have described this formation with the rule:

$$G(\$x, \$y), e2, G(\$x, \$y) \Rightarrow L(\$x), L(\$y), L(\$x), L(\$y) / 100$$

which allows reduplicated roots of the form ABAB (both groups need to have the same consonants).

### Phonetic Determinatives and general cleanup

The last set of rules deals mainly with phonetic determinatives. They are currently considered as part of the end of the word, but they bring with them a phonetic value, which should be matched by the beginning of the word.

### PROBLEMS SOLVED

#### Function and frequent words

A number of words, mainly very common ones, and above all function words, tend not to take a determinative at all. Hence, they would break our system. Fortunately, although those words are commonly found in texts, they represent a small part of the lexicon. Hence, we can write specific rules for them. In the current system, those rules are kept in the 'group formation' layer, because in that layer, we have the phonetic values of the signs at our disposal, and we have also copied their codes specifically for that purpose.






The rule for the preposition *hn* is:

$$P(H), P(n), P(a) \Rightarrow \text{wordstart}, L(H), L(n), L(a), \text{wordend} / 1$$

here, wordstart and wordend will prevent any attempt to group this word with another one later on.

However, there are a few problems, especially with suffix pronouns, and especially with 's', which can be either a feminine suffix pronoun, or be used as a causative prefix. A point which must certainly be improved in our system is that it sometimes outputs a sequence of suffix, which is obviously impossible. It would be possible to create a transducer to prevent this behaviour, but it has not been done yet.

### Multiple determinatives

In Middle Egyptian orthography, words tend to have only one determinative, but they sometime take more. As we decided to extract our rules from the corpus, we needed to list those cases. In fact, they are not arbitrary. One of those determinatives tends to be a very general one, as Y1  (abstractions), A1  (man), or N35A  (liquid), the other being much more specific, and in some cases, one might be tempted to consider it as an ideogram. For instance, in the word , *ktt*, small one, the first determinative () corresponds to small things, whereas the second one corresponds to women.

We decided to add a mark, ‘GENDET’, to those signs. A rule of the form:

$$\text{DET}(\$X), \text{wordend}, \text{GENDET}, \text{DET}(\$Y) \Rightarrow \text{DET}(\$x), \text{DET}(\$Y), \text{wordend}/1$$

allows then the combination of a generic determinative with another one, while preventing the combination of two run-of-the-mill determinatives.

### Robustness

Software is said to be robust if it can cope with ‘bad’ or unexpected input. The very principle of our system makes it quite easy to do so. The idea is that each layer must accept any possible input, but always produce an output. This is done by ‘catch-all’ rules, which have a very high cost.

## CONCLUSION

### Evaluation

We have developed our set of rules on the tale of the story of the *Shipwrecked Sailor* (3500 signs, 1419 words), and we have about 9% of erroneous words. A word was considered to be erroneous if the interpretation given by the system was not possible; in some cases, it might be considered somewhat optimistic, as the grammatical context might for instance invalidate the software’s analysis.

The software was then tried on another text, *P. Westcar* (9480 signs, 4000 words), for which the set of rules had not been made. The error rate raised to 18%, which is rather high. However, most errors were caused by unforeseen groups of signs, which is rather easy to fix within our formalism.

A good feature of the system is that one can add complex rules quite freely, without interfering with the already correct parts of the software, because a complex rule matches a complex and peculiar entry, and so will interfere with more general rules only in specific cases, where it will normally improve the result. Of course, rules which accept very simple entries are definitely more difficult.

Now, if we take a Late Egyptian text, far removed from the training corpus, the result is rather bad, but quite interesting. Our system, when trying to transliterate the *Instruction of Amenemope*, makes the same kind of errors as a student trained in Middle Egyptian would do. In particular, he is baffled by syllabic orthography and multiple determinatives. In a way, this indicates that we have modelled the performance of a student trained in the classical language, but not in its more recent successors.

On a more practical side, the software is quite fast on a modern computer. The speed is currently in the region of 30s for 3000 signs, which seems reasonable (and much faster than a human would achieve!). A number of rather simple optimisations would permit considerable improvements in the results.

### Possible improvements

A number of technical points can certainly be improved in the system; in particular, the sign value layer could benefit from a considerable increase in speed. The rule system could probably be improved, both by adding a few layers (in particular layers to filter out multiple determinatives), and by cleaning up the existing ones. The cost system itself is of course very *ad-hoc*, and it would be interesting to see if statistics could be applied to this problem,<sup>10</sup> but this would require a major revision of the system, with a much simpler structure. The down-side of statistics in this case is that we would no longer be able to model human expertise.

The system as a whole is rather independent of the actual training corpus, and one could imagine using different sets of rules for different types of texts: hieroglyphic texts, Late Egyptian texts, and so on.

Of course, the addition of a lexicon would increase the precision of the system, and is indeed rather easy, as is demonstrated by the work on function words.

---

<sup>10</sup> F. Pereira, M. Riley. *Speech recognition by composition of weighted finite automata*. In Roche & Schabes, ed., p. 431–453.

**Possible uses**

Our system can of course be used to produce a 'reasonably good' transliteration of a text, which can speed up the work of a scholar (although we are not sure that correcting an existing transliteration is much faster than writing one from scratch).

The result itself, a transducer, can be used as input for further processing, for instance as the entry of a syntactic analyser.

An interesting property of the system is that it analyses the word structure while transliterating it. This could be of some use, for instance in text searches. The intermediary levels, with the sign group, can be thought of as 'normalized forms' of the words, and could be used for searching and indexing.

And, last but not least, the creation of rules in itself is rather stimulating, and a quite interesting exercise in the study of Ancient Egyptian spellings.

