# Intersection of Multitape Transducers vs. Cascade of Binary Transducers: The Example of Egyptian Hieroglyphs Transliteration

**François Barthélemy**
CNAM (Cédric),
292 rue Saint-Martin, Paris, France
INRIA (Alpage),
Rocquencourt, France
francois.barthelemy@cnam.fr

**Serge Rosmorduc**
CNAM (Cédric),
292 rue Saint-Martin, Paris, France
serge.rosmorduc@cnam.fr

## Abstract

This paper uses the task of transliterating an Egyptian Hieroglyphic text into the latin alphabet as a model problem to compare two finite-state formalisms : the first one is a cascade of binary transducers; the second one is a class of multitape transducers expressing simultaneous constraints. The two systems are compared regarding their expressivity and readability. The first system tends to produce smaller machines, but is more tricky, whereas the second one leads to more abstract and structured rules.

## 1 Introduction

In the eighties, two models of Finite State computations were proposed for morphological descriptions: two-level morphology (Koskenniemi, 1983) where simultaneous constraints are described using two-level rules, and rewrite rule systems where rules apply sequentially (Kaplan and Kay, 1994). From a computational point of view, both kinds of rules are compiled into binary transducers. The transducers are merged using *intersection* in the simultaneous model whereas transducer *composition* is used by the sequential model.

Two-level grammars appeared difficult to write because of rule conflicts: the different two-level rules are not independent. Their semantics is not compositional: the semantics of a set of rule is not the composition of the semantics of each rule.

In the last decade, a new kind of simultaneous finite state model has been proposed which does not use two-level rules and avoids rule conflicts. The model uses multitape transducers (Barthélemy, 2007). The present paper is devoted to an application written successively with a cascade of transducers and an intersection of multitape transducers.

The application consists in transliterating Egyptian Hieroglyphs without resorting to a lexicon. The transliteration task is a transcription from the original writing to an extended latin alphabet used to write consonantal skeleton of words. This task is far from obvious, as we explain in 2.2.

The next section gives more details about hieroglyphs. Then, a transliteration grammar using a cascade of weighted rewrite rules is presented. Section 4 presents the *multigrain multitape transducers* and the *Karamel language* used to define them. Then comes a description of a Karamel grammar for hieroglyph transliteration adapted from the rewrite rule cascade. The last section compares the two grammars and their respective strengths and weaknesses.

## 2 Egyptian hieroglyphs

### 2.1 Hieroglyphic encoding and transliteration

The systems we are about to describe take as input an ASCII description of hieroglyphic texts, based on a system called "le manuel de codage" (Buurman et al., 1988), and output a transliteration of the text, that is, a transcription in latin characters of the *consonants* (the Egyptians did not write vowels). Meanwhile, we do also determine word frontiers. An example is given in figure 1.

The letter used in the transliteration layer are conventional signs used in ASCII computer-encoding



*Proceedings of the 9th International Workshop on Finite State Methods and Natural Language Processing*, pages 74–82,
Blois (France), July 12-15, 2011. ©2011 Association for Computational Linguistics

| hierogl. | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **input** | M17 | G43 | D21 | Z1 | N35 | O34 | A1 | Z1 | N35 | N42 | G17 | D36 | I9 | M23 | G43 |
| **output** | iw | | rA | | n | z | | | nHm | | | | f | sw | |
| trans-lation | part | | mouth | | of | man | | | save | | | | it | him | |
| | *the mouth of a man saves him* | | | | | | | | | | | | | | |

Figure 1: example of sentence transliteration

of Egyptian texts. In this encoding, uppercase and lowercase letters note different consonants. Besides, ‘A’, ‘a’ and ‘i’ are used to represent consonantic signs (which the Egyptological tradition renders as vowel in scholarly pronunciation). The above sentence would be pronounced “iu ra en se neḥem ef su”.

## 2.2 The Egyptian hieroglyphic writing system

In this section, we explain the basics of the hieroglyphic system (Allen, 2010) , and we detail a number of problems met when trying to transliterate it.

Hieroglyphs can be written in lines or columns, right-to-left or left-to-right and are typically grouped to fill the available space, as in this text : . In this work, we will neglect the exact position of signs and deal with their sequence.

Another characteristic is that there are no real word separators. The present work will address this issue.

### 2.2.1 Kinds of signs

The hieroglyphic system used a mix of phonetic and ideographic signs to note the language. Many signs may have more than one possible value.

The *phonograms* note a number of *consonants*, typically one, two or three. For instance, the owl stands for the consonant “m”, and the chessboard for the sequence of consonant “mn”. Vowels are not written, so egyptologists would insert arbitrary “e” between the consonant, and thus, is conventionally pronounced “men”. Those phonograms are classified as uniliteral, biliteral, and triliteral signs, depending on the number of consonants they represent.

*Ideograms* are more or less word-sign. Usually, they are followed by a stroke, to mark this specific use. For instance, writes the word “kA”, bull.

*Determinatives* are semantic classifiers which have no phonetic realisation, but give the general semantic class of a word. As they tend to occur at word endings, they ease the word separation problem too. For instance, in , “mooring pole”, the sign classifies the word as a “wooden thing”.

### 2.2.2 Word formation

Egyptian word spellings tend to contain a phonetic part followed by a determinative.

In the phonetic part, signs which represent more than one consonant come usually with “phonetic complements”, which are uniliteral signs, representing one, two or three consonants of the multiconsonantic sign. For instance, in the group , “nDm”, the phonetic sign has the value n + D + m. But it is nonetheless supplemented by the “m” sign. The group is to be read “nDm” and not “nDmm”.

To give a complete example, the word , “snDm” ,“to seat”, is to be understood as

| | | |
|---|---|---|
| s | nDm+m=nDm | determinative |

### 2.2.3 A few specific problems

Word formation, as we have just described it, is only a general principle. The presence of determinatives is optional, and very usual words (especially grammatical words) have usually none. This makes word separation a complex task.

More, single signs may have multiple values. The frontiers between types of signs are fuzzy; the basis of the phonetic system is the rebus, and an ideogram, which represents a given word with a given consonantic skeleton, can often be used phonetically. For instance, the “house” sign, , can stand both for the word “house, “pr”, or as phonogram in the verb “to go out”, which happens to have the same transliteration, “pr”. The same sign can be abstracted the other way, losing any phonetic value, and be used as determinative for “house-like” places, like , “kAp”, “shelter”.

This fuzziness makes it difficult to give a clear-cut description of signs, even in context. Some signs, which have the behaviour of both determinatives (they tend to stand at word endings) and of phonetic signs (they have still a very definite phonetic value) have been coined as “phonetic determinatives”. They pose specific problems, because the word, as in many languages, may have grammatical inflections standing after the word root. Those inflections are usually written *before* the phonetic determinative, and make the phonetic part non-continuous. For instance, in the word “sr.t”, “what has been predicted”, the root of the verb, “sr”, is written phonetically , then we have an intrusive , “t” , which is a feminine/neutral inflection, then, the phonetic-determinative “sr” , for the verb “to predict” , and finally, the determinative for “mouth actions”. We want to limit the number of rules which deal with this phenomenon.

Other signs have simply a number of unrelated phonetic values. For instance, can be either “Ab” or “mr”. Usually, the context, and in particular, the phonetic complements, help to choose.

Apart from the present work, a number of systems dealing with transliteration have been created: S. Billet (**?**) has written an transliteration agent-oriented system, and M.-J. Nederhof (**?**) has described an algorithm for *aligning* hieroglyphic encoding and transliteration.

## 3 A first solution : a cascade of transducers

In (Rosmorduc, 2008), we introduced a system based on a cascade of weighted finite-state transducers with variables. Basically, each weighted transducer applies a set of rewriting rules of the form:

$$t_1, ...t_n \rightarrow u_1...u_m/c$$

where $c$ is the cost of the rule (a real number) and $t_i$ and $u_j$ are terms. Terms can be either a variable (of the form `$X`), a constant identifier or integer, e.g. `A1` or `100`, or a functional term, whose argument can be either constants or variables, e.g. `P(i, $A, $B)`. Variables on the right side must also appear on the left side.

The text entry (which is variable-free) is used as input for the first transducer in the cascade. Variables values are lazily matched with the input - as a result, the rules representation is compact, while the final transducer can be quite large (e.g. for a text of 1300 words, the last resulting transducer contains over 1 000 000 nodes).

The costs are attached to the last link of a rule representation. When the cascade is computed, they are combined by a simple addition. When all transducers have been used to process the input, one of the least-cost path is selected as “the” best path, and the transliteration can be read on the output layer. The actual costs are quite ad-hoc, with low values indicating likely rules, and high values, unlikely ones. With equally valued rules, the system tends to prefer the rules which encourage the grouping of signs.

The current system is compounded of five transducers. We will first explain the main purpose of each one, and then concentrate on exceptional uses.

**First transducer, normalization:** this is a rather simple technical layer, as we need to normalize the entry, because the encoding often proposes a choice of codes for the same sign.

**Second layer, sign values:** The second layer replaces the sign codes with their values. Values are expressed using functors. Each functor corresponds to a particular kind of value, and the functor’s arguments represent the value.

**phonetic values** are expressed as P(X), P(X,Y) or P(X,Y,Z), depending on the number of consonants in a given sign. For instance, the following rules.

```
A17 => P(X,r,d) / 100
A17 => P(n,m,H) / 300
```

states that sign A17 can have the phonetic value “Xrd” or “nmH”, “Xrd” being preferred with a cost of 100 over nmH (cost 300).

**determinatives** are expressed as DET(MEANING), where “MEANING” is (occasionally) used to keep track of the determinative’s value. E.g.

```
A17 => DET(child) / 100
```

states that A17 can be a determinative for “child”.

**phonetic determinatives and ideograms** are expressed with the same system as phonetic signs:

```
A17 => IP(X,r,d) / 100
D56 => ID(r,d) / 100
```

**plural and ideogram markers** which can be found as word endings, are expressed using `END()`, which takes as argument “P1” if the word is singular, or “P3” if it is plural.

**Third layer, groups:** this layer build “groups” aggregating a complex phonetic sign with its phonetic complements. Note that groups are not words, as a word can contain more than one group.

```
P($x,$y), P($x), P($y) =>
        G($x,$y), endGroup / 10
```

states that a biliteral sign of value `P($x,$y)`, can be completed with two uniliteral signs representing both `$x` and `$y`, and that they form a group of phonetic value `G($x,$y)`. The symbol `endGroup`, which is inserted after the group, will be used in the next layer when gluing the phonetic parts and the word endings.

The layer also recognises the possible forms of word endings, which include some inflections, determinative, and possibly plural markers :

```
P(w), P(t), DET($x), END($y) =>
  b3,L(w),L(t),DET($x),endWord/100
```

The ”b3” is a marker which will be combined with ”endGroup” in the following layers.

**Fourth layer, words phonetics:** This forth layer deals mainly with the phonetic shape of words. Not all consonantic sequence, nor all group sequences, can form a word with equal probability. Egyptian has mostly bi and tri-consonantal roots, so shorter and longer words (ignoring the inflections) are not that likely. Two typical rules are:

```
G($x,$y), endGroup, G($z) =>
     L($x), L($y), L($z) / 100
G($x,$y), endGroup, G($x,$y) =>
     L($x), L($y), L($x), L($y)
     / 100
```

The first states that building a triliteral word with two groups, a biliteral one and a uniliteral one, is quite possible. The other concerns quadriliteral rules. Arbitrary combinations resulting in a quadriliteral root are usually given a high cost, but in this rule, we represent a reduplicated[1] root of the form XYXY, which is a rather usual way of building intensive words in Egyptian.

Word endings and groups are also attached by rules which erase at no cost the sequence `endGroup, b3`, which ensures that a “phonetic part” followed by a word ending is the favoured way of building a word. Erasing `endGroup` on its own, which amounts to allowing a word with only a phonetic part (which is still possible), is given a large cost of 1000.

**Fifth layer, cleanup and ending attachment** This last part does some cleanup, and removes data which was copied from layer to layer, in order to keep only relevant analysis. It also deals with *phonetic determinatives,* for which the previous layer is a bit too early.

### 3.1 Cross layers issues and discussions

#### 3.1.1 The so-called phonetic determinatives

The problem of phonetic determinative is that we are going to combine them, not with a group, but with the word’s phonetics. Let’s consider the following example :

| glyphs | | | | | |
|---|---|---|---|---|---|
| codes | O34 | D21 | X1 | E27 | A2 |
| groups | s | r | t | IP(s,r) | det |
| | phonetics | | word ending | | |

Here, the root part of the word phonetics is compounded of two uniliteral signs, making two groups. The word ending contains a “t” which is the feminine inflection, the phonetic determinative E27, and the determinative of mouth actions.

The problem is that we need to combine the first part with the ending, while keeping a low number of rules. This is done in two steps. First, in the group layer, we re-order the signs, in order to put the phonetic determinative before the inflections. We introduce a token, “b4”, which will be consumed in the last layer.

```
P(t), IP($x,$y), DET($a) =>
```

[1]this is the technical word used by the scholars, even though *duplicated* would probably suffice.

```
    b4, IP($x,$y), L(t), DET($a),
    e4 / 10
```

The word layer will simply copy the result of the group layer. Then, in the "cleanup" layer, as we do have a representation of the word, we can combine it with the IP if needed :

```
L($X), L($Y), b4, IP($X,$Y) =>
    L($X), L($Y) / 10
```

Note that this rule is agnostic about the way the L() readings were produced.

#### 3.1.2 Word separation

Word separation is a by-product of our system. Basically, we explicitly mark certain sequences of signs as word endings, plus, we can transform any group ending into a word ending. The possible phonetic structures of a word (as a group sequence) are also listed. The combination of those systems, along with their associated cost, is used to produce a reasonable words separation.

#### 3.1.3 Exceptions

The idea of the system was to try to experiment on transliteration without a lexicon, which can be useful for unknown word. Basically, no extensive lexicon is used; however, grammatical words, and some very frequent verbs don't respect the usual word-formation rules. For instance, the verb "Dd" "to say" is written with (I10) and (D46), which are respectively uniliteral signs we will render "D" and "d". The nice thing with automata here is that they lend to a graceful representation of those exceptions. We directly map the signs to the final output in one of our levels, and the result will be copied by each level until the last one :

```
I10, D46 => startWord, L(D),
            L(d), endWord / 100
```

## 4 Intersection-oriented multitape transducers

This section is devoted to Karamel, a language used to define multitape finite state transducers (Barthélemy, 2009).

The definition of multi-tape machines is done using regular expressions extended with *tuples*. Tuples are somehow Cartesian products which glue together independent regular expressions read on different tapes, but unlike Cartesian product, tuples are not distributive with respect to concatenation. The theoretical basis of the language is the *multi-grain relations* (Barthélemy, 2007). The tuples used in regular expressions are instances of *tuple types* which must be declared beforehand. The tuples are written using curly braces and begin with the type name, followed by the components. Components of the tuples are both named and ordered, so two syntaxes are allowed to write them: with the name or using the order. Default values are defined for each component.

Embedded tuples are used to give tree-structure to tuples in the relations. There is a constraint: a tape appears in *at most one* of the components of the tuple. Recursive structures are therefore not allowed. The regular sets defined by regular expressions extended with tuples are closed under intersection and difference. So these two operations are available for writing extended regular expressions.

Here are a couple of concrete examples written using Karamel syntax. The value of a sign is expressed using a tuple type called `val` which has 4 tapes: for hieroglyph signs, for phonetic values written with the latin alphabet, for the semantic value and one for a subtype of values used in composition rules. The tapes are called respectively `tsig`, `tphon`, `tsem` and `vtype`. An instance of a purely phonetic sign value:

```
{val: tsig=<P17>, tphon=nmh,
      tsem=<>, vtype=<phon>}
```

The notation <> stands for the empty string and <P17> for the single symbol `P17`, whereas `nmh` is a string of three symbols. The notation `{val}` is used when no value is specified for the components of a `val` tuple. In this case, all the components take their respective default values.

Phonetic values are sometimes composed in groups where some consonants are redundantly written. Groups are implemented by 2-tuples where the first component contains a string of phonetic values and the second one is the transliteration (on tape `trans`). Here is an example:

```
{group: vals=
    {val: tsig=<F28>, tphon=ab,
                vtype=<phon>}
    {val: tsig=<D58>,tphon=b,
          vtype=<phon> },
```

```
    trans= ab}
```

The examples given above are string tuples. Extended regular expression may use regular operators to describe regular sets of tuples, like in the following example: `{wend: ({det}|{phon}|{tend})+}` which describes the set of all tuples of type `wend` where the first component is a non-empty string of tuples of types `det`, `phon` and `tend`.

A Karamel grammar begins with some declarations: symbols, classes of symbols, tapes and tuple types. A class of symbol is a finite set of symbols which has a name. A class name may be used in regular expressions and stands for the disjunction of the symbols in the class. Variables are available and take values in such classes. Occurrences of a variable within a given regular expression must take the same value. Variables express long-distance dependencies. Here is an example of expression using a variable:

```
{group:
    vals={phon: $x in (<letter>)},
    trans= $x}
```

This may be read: the letter found in the `phon` tuple on tape `tphon` is the same as the letter found on the `trans` tape of the same `group` tuple.

It is possible to define abbreviations for tuples where the order and the default value of the components may be different from the ones in the type definition. For example, an abbreviation called `phon` is defined for instances of the type `val` where the component `tsem` is set to the empty string and the component `vtype` is set to the value `<phon>`. This abbreviation is used to define purely phonetic values. For instance `{phon: tsig=<F28>, tphon=ab}` is just another notation for the tuple `{val: tsig=<F28>, tphon=ab, vtype=<phon>, tsem=<>}`.

A machine already defined may be used in an extended regular expression, using its name written between << and >>. E.g. `{group: <<some_vals>>*}` uses the machine `some_vals`.

The extended regular expressions may include weights which are arbitrary floating numbers. Weights are written in expressions enclosed by two exclamation marks. The operations on transducers combine weights using the *tropical semiring*. Operations such as concatenation, composition and intersection compute sums of weights of both operands. The *n-best* operation is available to select the paths having the smallest weights in a machine.

The *external composition* is a binary operation which combines a multitape machine and a regular language, considered as an input on a given tape. The *external projection* projects a multitape machine on one tape and then removes all the tuple boundaries. These two operations are the interface of a multitape machine with the outer world.

## 5 Transliteration using multitape transducers

A Karamel Grammar has been written by translating the cascade of binary transducers presented in section 4. The classes of symbols defined include the hieroglyphs (class `sign`), the letters used in the transliteration (class `letter`), a set of semantics values (class `sem`), divided in two subclasses, generic values (class `gensem`) and regular values (class `regsem`). There are also several classes of auxiliary symbols such as subtype names. The tapes include one tape for the text written with hieroglyphs (tape `tsig`), one tape contains the transliteration (tape `trans`), two tapes contain respectively the phonetic and the semantic values of the signs (tapes `tphon` and `tsem`). There are several auxiliary tapes for information such as subtypes and rule identifiers.

There are a number of different tuple types with up to four levels of embedding. A tuple type is used to represent sign values on four tapes. One tape is used for the hieroglyphs, possibly a sequence of several signs, another for the phonetic value which is a sequence of latin letters, another tape contains the semantic value. The last tape called `vtype` contains a value type which is important to separate subclasses of values which play a different role and appear in different places of the forms. There are six subtypes: pure phonetic values, ideograms, phonetic ideograms, determinatives and numbers. Six abbreviations are defined for these subtypes, which set the `vtype` value and some other tapes. For instance, for the determiners, the phonetic value (tape `phon`) is set to the empty string.

The tuple type group is used for groups of phonetic values where some consonants are written redundantly. A sequence of such groups is used to write the phonetic part of a form which is represented using a core tuple.

There are also tuples to write frozen forms, directly from hieroglyphs. These forms do not use embedded val tuples. The type wend is used to describe the word endings which usually follow the phonetic parts. Word endings contain the inflection written phonetically and determinatives. The types idform, number and gram describe forms and their main component is a sequence of values (val tuples). They describe respectively an ideographic notation, a number and a grammatical word such as a pronoun or a preposition. With all these tuple types, there are several possible structures for forms :

```
{cpform: {core: {group: {val}*}*}
         {wend: {val}*}+}|
{idform: {val}*}|{number:{val}*}|
{gram: {val}*}|{frozen};
```

An instance of the most complex structure :

```
{cpform:
  {core:
    {group: {phon: s, <S29>}, s}
    {group: {phon: nDm, <M29>}
           {phon: m, <G17>}, nDm}}
  {wend:{det: <seat>, <A17>}}}
```

This corresponds to the analysis of nDm :

| S29= | M29+G17= | A17= |
|---|---|---|
| s | nDm+m=nDm | determinative |

Each tuple type is described in the grammar using two transducers: one which describes all the possible values for one occurrence of the tuple type, the second one describes the context in which sequences of the tuple types may appear to make a form. The second transducers uses the definition of the first one. For instance, the word endings follow a number of patterns including phonetic values and determiners. These patterns are described in a machine called all_word_endings. The context of the patterns is described in a machine called actual_word_endings. Part of the code of the two machines is given below. Note that the second machine uses the first one in its definition.

```
let all_word_endings =
  {wend: seq = {phon: y},
         trans = y};!1000!
 |{wend: seq = {det}{phon: y},
         trans = y};!100!
 |{wend: seq = {phon: y}{det},
         trans = y};!100!
 | ...

let actual_word_endings =
  {cpform:{core:{group:{val}*}*}
       <<all_word_endings>>+}|
  {idform:{val}*}|{number:{val}*}|
  {gram:{val}*}|{frozen};
```

Each pattern in a machine all_XXX corresponds to one rule of the rewrite rule system of the original grammar. The excerpt above translates the rules :

```
P(y) => b3, L(y), endWord / 1000
DET($x), P(y) => b3, L(y),
           DET($x), endWord / 100
P(y), DET($x) => b3, L(y),
           DET($x), endWord / 100
```

The auxiliary symbols b3 and endWord used in the rewrite rules correspond to the opening and closing of a wend tuple in Karamel.

The description of all written forms is potentially given by the intersection of all the machines actual_XXX, one machine for each tuple type. This intersection is statically computable, and the result is a large transducer. It is also possible to compute the intersection dynamically: the text represented by a sequence of hieroglyphs is first combined with one of the machines using an external composition operation, then the result is intersected successively with all the other transducers. The best transliteration is computed by an n-best computation and the transliteration is finally extracted using an external projection.

## 6 Comparison of the two approaches

The two grammars represent the structure of forms using different means: the cascade grammar uses pairs of auxiliary symbols whereas the Karamel grammar uses tuples. The structure described is almost identical in both grammars. Some tuples of the grammar are not represented in the cascade grammar: it is the case of the smallest tuples (type val) and some of the largest tuples (type cpform).

Values in the cascade grammars are rewritten either phonetically or semantically. The sign and the corresponding value are never simultaneously represented in the intermediate strings.

The multitape grammar puts homogeneous information on each tape: there is a tape for hieroglyphs, two for phonetic values, one for semantics, several for auxiliary values. The cascade concatenates different kind of symbols, especially at the intermediate levels. The input consists in hieroglyphs and the output in latin letters, but the intermediate strings have not only both kind of representations (hieroglyph and latin letters), but also semantic values and auxiliary symbols.

Some of the rewrite rules change the order of signs with respect to the text order. This is used for two purposes: in word ending, determinatives which sometime appear before the inflection marked using phonetic values, are pushed to the end of the word in such a way that all the determinatives of a word ending are contiguous. This is important for the next layer of the cascade which rewrites pairs of determinatives with various weights, depending on semantics constraints. The other case of reordering deals with the phonetic determinatives which are put at the beginning of word endings. This is done to check that the phonetic value of the sign is coherent with the transliteration of the phonetic part.

The Karamel grammar does not need to change the order of symbols. The constraints on multiple determinatives and the coherence between phonetic ideograms and the phonetic part transliteration are expressed as long-distance dependencies using Karamel variables.

The formalism used in the cascade consists in rewrite rules without context: a center is rewritten regardless of the surrounding symbols. There is no way to express long-distance dependencies within the formalism. On the other hand, long-distance dependencies are costly: they result in larger transducers.

There are also some cases of changes in the structure proposed for a form in the cascade grammar: two word endings are collapsed into one by removing the symbols marking the end of the first and beginning of the second. In the Karamel grammar, this rule is not implemented as a change of the structure, but by allowing under conditions a second word ending after the first one. The structures are never changed once built.

| step | states | | arcs | |
|---|---|---|---|---|
| step | binary | multitape | binary | multitape |
| values | 75 | 12 723 | 2 095 | 61 581 |
| groups | 8 675 | 165 286 | 20 234 | 277 812 |
| words | 10 383 | 104 844 | 23 605 | 7 458 881 |
| cleanup | 923 | 821 031 | 5 001 | 924 514 |

Figure 2: sizes of binary and multitape machines

The multitape transducers are larger than binary transducers for a couple of reasons. They contain more information because they keep all the information whereas in the cascade, some symbols are forgotten after they have been rewritten. Another reason is that there is an overhead due to representation of tapes and tuples, which are compiled using auxiliary symbols. The third reason is that some long distance dependencies are implemented in the multitape machines. These long-distance dependencies do not appear in one binary transducer, but they appear when statically composing the transducers of the cascade. Figure 2 gives the sizes of comparable machines of the two grammars.

## 7 Conclusion

The comparison done here is not completely fair because the second grammar has been translated from the first one, almost rule by rule. This does not give the best possible implementation of the application in Karamel. Some features available in Karamel are not used.

The Karamel language provides a more abstract description of the forms, using an explicit tree structure and separating the different pieces of information on different tapes, according to semantic criteria. On the other hand, the Karamel machine is much larger. Karamel is a high-level declarative formalism whereas non contextual rewrite rules are an efficient low-level language.

Some trade-off is possible: cascade of transducers may be expressed using a richer language (e.g. XFST (Beesley and Karttunen, 2003)) whereas the Karamel language has some contextual rewrite rules which have not been presented in this paper because they are not used in the Egyptian transliteration grammar.

## References


James P. Allen. 2010. *Middle Egyptian: An Introduction to the Language and Culture of Hieroglyphs*. Cambridge University Press.

François Barthélemy. 2007. Multi-grain relations. In *Implementation and Application of Automata, 12th International Conference (CIAA)*, pages 243–252, Prague, Czech Republic.

François Barthélemy. 2009. A testing framework for finite-state morphology. In *Implementation and Application of Automata, 14th International Conference (CIAA)*, volume 5642 of *Lecture Notes in Computer Science (LNCS)*, pages 75–83, Sydney, Australia.

Kenneth R. Beesley and Lauri Karttunen. 2003. *Finite State Morphology*. CSLI Publications.

Jan Buurman, Nicolas Grimal, Michael Hainsworth, Jochen Hallof, and Dirk Van Der Plas. 1988. *Inventaire des signes hieroglyphiques en vue de leur saisie informatique*. Mémoires de l'Académie des Inscriptions et Belles Lettres. Institut de France, Paris.

Ronald M. Kaplan and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics*, 20:3:331–378.

Kimmo Koskenniemi. 1983. Two-level model for morphological analysis. In *IJCAI-83*, pages 683–685, Karlsruhe, Germany.

Serge Rosmorduc. 2008. Automated transliteration of egyptian hieroglyphs. In Nigel Strudwick, editor, *Information Technology and Egyptology in 2008, Proceedings of the meeting of the Computer Working Group of the International Association of Egyptologists (Informatique et Egyptologie), Vienna, 811 July 2008*, pages 167–183. Gorgias Press.