Rational Transductions for Phonetic Conversion and Phonology

Éric Laporte
Institut Gaspard-Monge
France
laporte@univ-mlv.fr

August 1995

Abstract

Phonetic conversion, and other conversion problems related to phonetics, can be performed by finite-state tools. We present a finite-state conversion system, BiPho, based on transducers and bimachines, two mathematical notions borrowed from the theory of rational transductions. The linguistic data used by this system are described in a readable format and actual computation is efficient. With adequate data, BiPho constitutes the first comprehensive spelling-to-phonetics conversion system for French to take the form of transducers or bimachines.

1 Introduction

Spelling-to-phonetics conversion is one of the most classical problems in natural language processing. Several other conversion problems related to phonetics are interesting in themselves or for their applications. For example, phonetics-to-spelling decoding is a real challenge and has applications in speech processing. Appropriate computational solutions for these conversion problems are provided by finite-state tools: transducers (i.e. automata with input and output) and bimachines, two notions borrowed from the theory of rational transductions. We present a conversion system, BiPho, based on transducers and bimachines. This conceptual and computational framework has two major advantages: the description of linguistic data is carried out in a readable format, and the speed of the conversion algorithm is independent of the size of the set of conversion rules and dominated by the length of input strings. With spelling-to-phonetics conversion data for French, BiPho constitutes the first comprehensive spelling-to-phonetics conversion system for French to take the form of transducers or bimachines.

2 An introductory example

English has one of the most difficult spelling systems. Figure 1 shows the phonetic transcription of *ou* before *gh*. It is a directed acyclic graph which reads from left to right. This graph is a representation of a finite transducer. A transducer is an automaton where each transition is labelled by an input label and an output label. In this figure, input labels are displayed inside the boxes and output labels under the boxes. Input labels are spellings of word parts, output labels are phonetic transcrip-

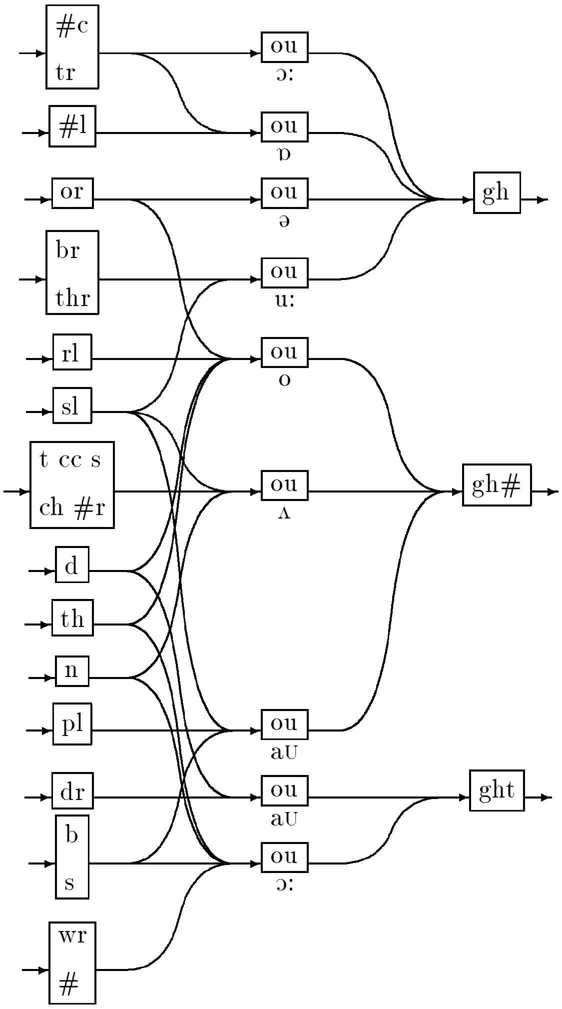


Figure 1: *ou* before *gh*.

tions. Since the figure concentrates on the pronunciation of *ou* before *gh*, a phonetic transcription is displayed only under those boxes which contain *ou*. The other boxes contain the left or right contexts. The special symbol $\sharp$ stands for word limit or morpheme boundary. This graph is rather readable for people but it can also be used in order to compute phonetic transcriptions of words. This type of representation has several advantages.

1. Readability. An error in a graph like that of Figure 1 is easy to detect even for a non-specialist if a word which is an exception to the rules comes across his mind. Metalanguage and conventions are reduced to a minimum and take a graphical form.

2. Formalization. The formal meaning of this type of representation is defined mathematically.

3. Compactness. Various contexts are taken into account in Figure 1, but when similar contexts for different pronunciations are considered, several paths in the graph can often share their common part. For example, final *gh*, i.e. *gh*$\sharp$, appears once for *though, enough, bough* and *thorough*; *th* appears once for *though* and *thought*. When long lists of words or word elements are to be listed, avoiding the repetition of common parts is a substantial economy, whereas making such lists without automata is discouraging and error-prone. The mathematical properties of automata that underlie this practical advantage are minimality properties.

4. Cumulativity. Figure 1 deals with a very specific issue. It should be associated with many other graphs in order to make up the complete data of a phonetic conversion system. The formal mean-

ing of the combination of finite-state transducers can be designed and defined so that the contents of a given graph will not interfere with the contents of another when they are combined. This feature is an improvement upon hierarchies of rules and exceptions which are traditionally used for spelling-to-phonetics conversion: a modification on a particular rule or exception in a hierarchy may have non-local effects.

5. Generality. This introductory example only deals with spelling-to-phonetics conversion of English text. However, the same type of formalism applies to more exotic conversion tasks, involving other languages, phonemics-to-phonetics conversion, phonetics-to-spelling decoding, etc.

6. Efficiency. Finite-state transducers, including ones with dozens of thousands of states, are also an efficient computational tool if they are wisely implemented. This feature is of the utmost importance since the size of the data at stake, in the final analysis, depends on the number of words in the language.

As a matter of fact, this paper describes theoretical and practical work done in this framework on several conversion problems related to phonetics. The phonetic conversion system BiPho exploits complete data for phonetic conversion in French (Laporte, 1993). French spelling is as irregular as English spelling. The output of the conversion constitutes the 600,000-inflected-word phonetic dictionary of LADL[1].

Section 3 states which conversion problems are concerned. Section 4 deals with the problem of designing a transducer to specify a given trans-

[1] Laboratoire d'automatique documentaire et linguistique, University of Paris 7, France.

duction and points out the consequences of decisions made at that stage. Section 5 introduces mathematical properties that relevant transductions usually have and mathematical tools that underly our implementation. Section 6 specifies a readable mode of representing transductions related to phonetics, defines its formal meaning, and describes an efficient implementation of it.

3 Transductions related to phonetics

In this section, we have a linguistic standpoint about a number of problems for which we will claim that finite-state devices are appropriate formal and computational tools. The prototypical example of these problems is that of spelling-to-phonetics conversion. A given speech utterance can be transcribed orthographically or phonetically; spelling and phonetics can thus be considered as two levels of representation of language. Spelling-to-phonetic conversion refers to two types of problems. First, one is faced with a descriptive problem: which spelling transcriptions are in relation with which phonetic transcriptions? Then, two computational problems can be contemplated: given a spelling transcription, what phonetic transcriptions can be in relation with it? and the reverse problem. In order to pose this kind of problems in an accurate way, we discuss a few issues about some of the levels of representations of speech. The reader who is only interested in formal or computational aspects can go to section 4, page 11.

Spelling needs not be defined, at least for English and other European languages with well-documented, standardized writing systems. Spelling can be considered as a practical level of linguistic representation. It appears as a formal system: transcriptions are coded as sequences of symbols. The set of symbols, the alphabet, is finite. We will consider only lowercase letters and a special symbol $\sharp$ standing for word boundary. The size of the alphabet is thus less than 30 in English. It must be extended for other languages, due to accents and other diacritics. In French, spelling is highly ambiguous with respect to pronunciation, so we use as an intermediate level a disambiguating alphabet where e.g. intervocalic s is marked as s_{15} when it is pronounced [s], like in $paras_{15}\,ol$ (in most words, intervocalic s is pronounced [z]). This disambiguating alphabet has 315 symbols. This method could give interesting results in English also.

The definition of a phonetic level of representation is not so simple. It is connected with three theoretical issues:

- the principle of using a finite set of symbols and of building linear sequences of symbols, is a far from neutral choice linguistically;
- it is usually considered that the elementary units at a phonetic level are not the symbols in the phonetic alphabet but binary feature-value pairs which serve to define these symbols;
- we will make a distinction between narrow transcriptions, which are an observational account of pronunciation, and abstract transcriptions, which are a means of taking into account phonetic variations in the phonology of languages.

3.1 Linear sequences are simple structures

Using an alphabet, i.e. a finite set of symbols, and building linear sequences of symbols, is a familiar principle, but it is not a neutral linguistic choice when it is used to represent speech. If we consider speech as a combination of acoustic and articulatory events, this combination is much more complex than phonetic transcriptions of speech: in the duration of one or two phonetic segments, dozens of acoustic events happen, their chronological order may vary, most of them are continuous variations of continuous parameters, and those which are instantaneous are hardly ever simultaneous. In other words, the most accurate phonetic transcription is only an approximate, partial and imperfect description of speech. However, phonetic transcriptions are an excellent descriptive tool. It is standardized to quite a reasonable degree among linguists, and it is successfully used for speech synthesis (e.g. synthesis by diphones) when associated with prosodic information. This is why we stick to linear sequences of phonetic symbols as one of the convenient and useful representations of pronunciation.

With the development of non-linear phonology, many linguists shifted from one-dimensional to multi-dimensional abstract representations of speech. For example, in spite of the fact that time is essentially one-dimensional, it is unquestionable that some phonetic variations or phenomena involve embedded structures in speech: syllables, coda, etc. However, the level of recursion of such structures has very restrictive bounds, so that they can be coded in linear strings which are a simpler structure than trees.

3.2 Phonetic symbols are readable

The symbols in the phonetic alphabet are usually defined by binary feature-value pairs. In this view, the elementary units at a phonetic level are not the phonetic symbols but the binary features. Phonetic and phonological descriptions make an intensive use of binary features. A set of phonetic symbols, e.g. {pbfv}, may be expressed as $[+labial\ -son]$, which is less redundant. One can also replace a few rules by one. Generative phonology is traditionally much concerned about redundancy, since the best possible grammar should be the least redundant. Using binary features brings about some decrease in redundancy, but also a dramatic decrease in readability: for a human reader, series like {pbfv} are more readable than binary-feature specifications. For such a practical purpose as actual linguistic description, readability and compactness are as important as redundancy. The work described in this paper does not take any advantage of binary features, but the same formal framework could undoubtedly be adapted with only minor modifications in order to express rules by means of features.

Since we use linear strings on a finite alphabet, and we study the relations between these strings, the appropriate formal framework for this study is that of transductions, i.e. relations over two sets of strings. Basic definitions about transductions in the context of phonetics and phonology are given in Kaplan and Kay (1994).

3.3 Phonetics or phonemics

It seems difficult to actually carry out any extensive description of phonetic forms in a language without taking into account the traditional distinction between narrow and abstract transcriptions. Narrow transcriptions are an observational account of pronunciation, whereas abstract transcriptions aim at taking into account phonetic variations in the phonology of languages. For example, the final *s* is pronounced differently in *seats* and *seizes*, and narrow transcriptions reflect this difference: [si:ts], [si:ziz]. If we consider that this [s] and this [iz] are variants of the [z] heard in *sees* [si:z], we can transcribe them by means of the same symbol /z/ in abstract transcriptions: /si:tz/, /si:zz/, /si:z/. We will use the terms *phonetic level* to refer to the level of narrow transcriptions, *phonemic level* to refer to that of abstract ones, and *phonemes* to refer to the elements of the alphabet of the phonemic level.

Phonemic transcriptions are also useful to describe free phonetic variations. For example, in French, *lier* 'link' admits a monosyllabic phonetic form [lje] and a disyllabic one [lije]: we transcribe both as /li+e/ (Laporte, 1989). Several phonetic variants are produced from a phonemic form by a multiple-output transduction. Multiple-output transductions are often defined with optional rules, but the notion of several-output transduction is more general than that of optional rule. For example, if we transcribe *lier* with the phonemic form /lje/ and if we produce the variant [lije] with an optional rule that inserts [i], this rule might produce a wrong variant *[pije] for *pied* [pje] 'foot'. On the other hand, if we transcribe *lier* with the phonemic form /lije/ and if we produce the variant [lje] with an optional rule that deletes [i], this rule might produce a wrong variant *[pje] for *piller* [pije] 'plunder'. Finally, the phonemic form /li+e/ contains an unpronounceable variation mark /+/, so the rule that produces [lje] and [lije] from /li+e/ has to be obligatory.

3.4 From a level to another

Spelling, phonetics and phonemics are three levels of linguistic representation: there are six ways of going from one of them to another, thus six conversion problems for each language. Our experiments on spelling-to-phonemics and phonemics-to-phonetics in French showed that these two problems have much in common: the same computational framework gave good results for both. Another type of conversion problem is also probably very close: the simulation of phonetic changes from a historical state of a language to another or to its present state.

In the following, the transductions whose definition was outlined in this section will be referred to as 'transductions related to phonetics'.

4 Construction of the transductions

A transduction is an abstract object. A transducer or another formal device that 'realizes' a transduction is an abstract machine that specifies it in a more concrete way, though it does not specify a particular algorithm to perform the conversion. Automata theory provides various mathematically equivalent ways of recognizing the same set or realizing the same transduction. In such a practical enterprise as ours, we have to choose a particular device to realize a transduction. This choice is not neutral:

- the success of the operation depends on the theoretical expressive power of the device;
- this choice may facilitate or hinder the descriptive aspect of the work, namely the elaboration of the conversion rules;
- it may lead to more or less efficient implementations of the computation.

Let us examine the consequences of those requirements on the problem of designing and implementing transducers.

4.1 Alignments

What we will call an alignment of a transduction is a correspondence between input symbols and output symbols in strings. A transduction in itself does not specify any alignment between input symbols and output symbols. However, transducers and other devices do specify an alignment of the transduction they realize. Several transducers that realize the same transduction may specify different alignments, as in Figure 2. The symbol <E> is the empty sequence which is made of no symbols at

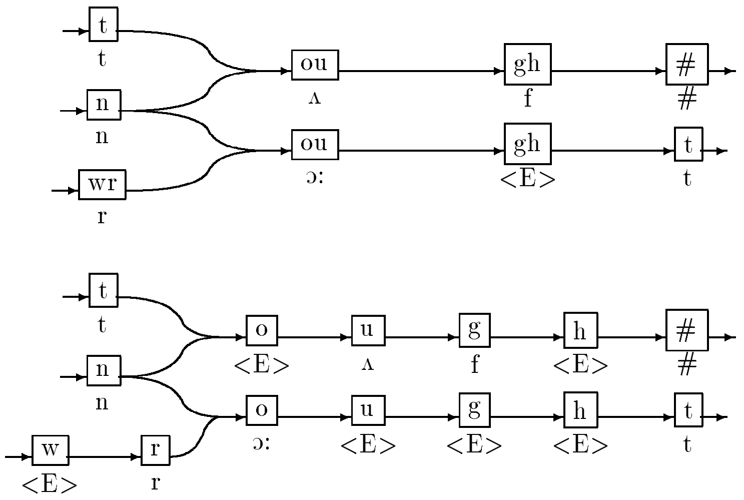


Figure 2: two alignments of the same transduction.

all. Rational expressions and other mathematical constructs used to define rational transductions also specify an alignment. In the case of the transductions related to phonetics mentioned in section 3, page 6, the time correspondence between input and output is a meaningful alignment for all of them, but specifying it in the smallest detail sometimes involves arbitrary decisions. For example, Figure 2 specifies two alignments between spelling and phonetic transcription. They differ only in details and both of them are quite sensible. In order to take full advantage of the partial regularity of spelling-to-phonetics transduction, the transducer that performs the conversion must at least approximately follow the natural alignment.

In a transducer, input and output labels are strings over the input or output alphabet. They can comprise zero, one or several symbols. A reasonable simplification of the problem is to consider alignments where each separate input symbol in the input string has its own counterpart in the output; the output for a given input symbol may still be composed of zero, one or several symbols. Formally, we will say that a device that realizes a transduction is strictly alphabetic if and only if it associates with each symbol in input strings a factor of the corresponding output string. The second transducer of Figure 2 is strictly alphabetic, i.e. each input label is an isolated input symbol. In the case of transductions related to phonetics, a strictly alphabetic alignment is always possible and is usually close to the most natural alignment.

4.2 Divide and conquer

Describing a complex transduction is an intricate task, we need to split it into smaller tasks. The finite-state formal framework provides ways to do that. Individual transductions can be devised for independent subtasks, and combined into a larger transduction that solves the original problem. Two simple principles will help us implement this strategy.

4.2.1 Simultaneous combination

The rules for translating a symbol are often very different from those for translating another. When it is the case, the transduction that will apply to the first symbol can be described independently of the other. Assume a transduction t_1 translates a given input pattern, leaving all the rest unchanged, and a transduction t_2 translates another input pattern that does not overlap the other and also leaves the rest unchanged. Then t_1 and t_2 apply to different places in input strings and can apply (conceptually) simultaneously. In other words, t_1 and t_2 can be implemented (conceptually) in parallel. In section 6, page 23, we will give a formal definition of the 'simultaneous combination' $t_1 + t_2$ of transductions t_1 and t_2 provided that they apply either to different input strings or in different contexts. The result of the simultaneous combination of transductions is independent of the order in which the transductions are given.

4.2.2 Sequential combination or composition

Transductions related to phonetics frequently have a natural expression as a composition of simpler transductions: one describes a finite sequence of transductions in a definite order, and the output of each of them will serve as input for the next. This amounts to defining intermediate levels of representation and going from each level to the next. Expressing a transduction as a composition of simpler ones is a basic method in generative phonology. This concept is called 'rule ordering'. If the output of t_1 serves as input for t_2, the composition of t_1 and t_2 will be noted $t_1 \circ t_2$.

4.3 Deterministic computation

When a transducer is strictly alphabetic, one can make an automaton out of it by deleting all output labels. This automaton is called the projection of the transducer. The projection of a strictly alphabetic transducer may be deterministic or not. If it is, the output strings for a given input string can be produced by a deterministic computation using the transducer. A deterministic computation is more direct, and therefore usually simpler and more efficient than a non-deterministic computation. We say that a transducer is deterministic if it is strictly alphabetic and has a deterministic projection[2]. The transducer of Figure 3 is non-deterministic: when we build the projection by deleting the

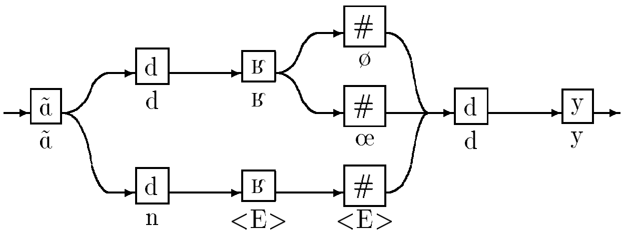


Figure 3: a non-deterministic transducer.

output labels [d] and [n], we leave two transitions with the same label /d/ and different targets. In fact, if we wish to have this transduction realized by a deterministic finite transducer, we will not find any with the same alignment. However, in phonemics-to-phonetics conversion, when this problem occurs, the transduction can usually be expressed as a combination of transductions realized by deterministic transducers. The combination may involve simultaneous or sequential combinations

[2]This terminology is not traditional. There is no standard definition of deterministic transducers.

or both (cf. above). The fact that this simplification of the problem is usually possible is an empirical observation which is not predicted by phonological theories. In the following, we assume it is always the case. For example, the transducer of Figure 3 can be expressed as $t_1 \circ (t_2 + t_3)$. In this expression, t_1, t_2 and t_3 are the deterministic transducers of Figure 4, the symbol $\circ$ refers to sequential combination and $+$ refers to simultaneous combination. An advantage of this representation is that t_1, t_2 and t_3 represent separately three unrelated phenomena. Note that t_1 produces two variants, but is deterministic: when we build the projection by deleting the output labels [ʁ] and <E>, the two original transitions merge into one transition since they have the same input label [ʁ] and the same target.

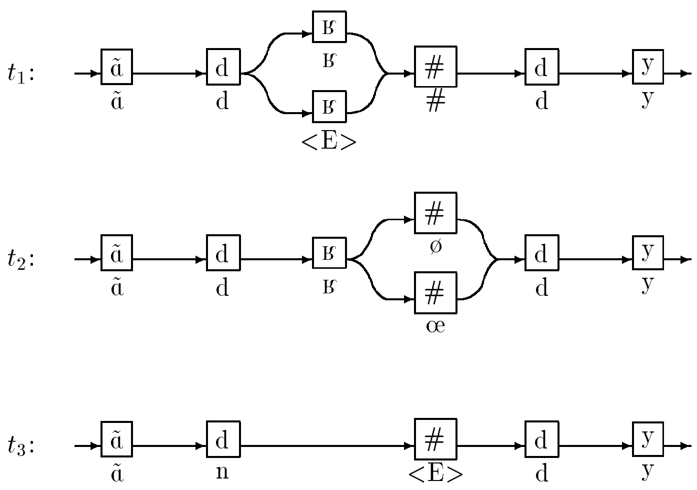


Figure 4: three deterministic transducers.

The phonetic conversion data used with BiPho for French involve 13 levels of representation: the first is spelling, the sixth is phonemics and the last is phonetics. The overall transduction is thus implemented as the composition of 12 transductions. Each of them is in turn the simultaneous combination of 6 to 231 simple transductions realized by deterministic transducers. We need a few more mathematical notions before describing how these elementary transductions are represented, how they are combined, and how the actual conversion is performed.

5 Mathematical properties

The transductions mentioned in section 3 page 6 are usually rational transductions. A transduction[3] over alphabets A and B is rational if it can be specified by a rational expression over $A^\star \times B^\star$. Equivalently, a transduction is rational if it can be realized by a finite transducer. The fact that phonological transductions are usually rational is far from new. It was first noticed by Johnson (1972). It is stated in more standard terms by Kaplan and Kay (1994). In what follows we examine other mathematical properties of transductions related to phonetics. The interested reader will find more details about definitions and algorithms in handbooks of automata theory, e.g. Berstel (1979) or Perrin (1990).

5.1 Transductions realized by deterministic transducers

In a transduction, an input string can be in relation with several output strings. In the case of transductions related to phonetics, this allows us to describe phonetic variants. For example, in Figure 5, the phonemic

[3] In automata theory, the terminology *rational* is preferred to *regular* because it emphasizes the analogy with the theory of rational functions in classical analysis and of rational power series in commuting variables.

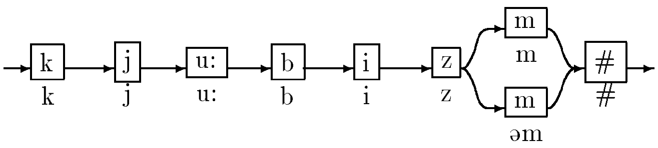


Figure 5: a transduction which is not a function.

string /kju:bizm/ is in relation with two phonetic strings, [kju:bizm] and [kju:bizəm].

When every input string is in relation with at most one output string, the transduction is said to be a function. A transduction realized by a deterministic finite transducer is not necessarily a function (examples: Figures 4 and 5), but it is easy to prove that it is the composition of a rational function and a finite substitution. A finite substitution σ over alphabets A and B is a transduction such that:

- for each $a \in A$, $\sigma(a)$ is a finite subset of $B^\star$,
- $\sigma(<E>) = <E>$ and
- for each $u, v \in A^\star$, $\sigma(uv) = \sigma(u)\sigma(v)$.

Finite substitutions are rational transductions. Figure 6 shows the decomposition of the transduction of Figure 5 into a rational function and a finite substitution which is represented as a one-state transducer in the figure. Recall that transductions related to phonetics can usually be expressed as a combination of transductions realized by deterministic finite transducers. When they can, it follows that they can also be expressed as a combination of rational functions and finite substitutions, which are simple cases of rational transductions. The computational interest of this decomposition stems from the fact that rational functions

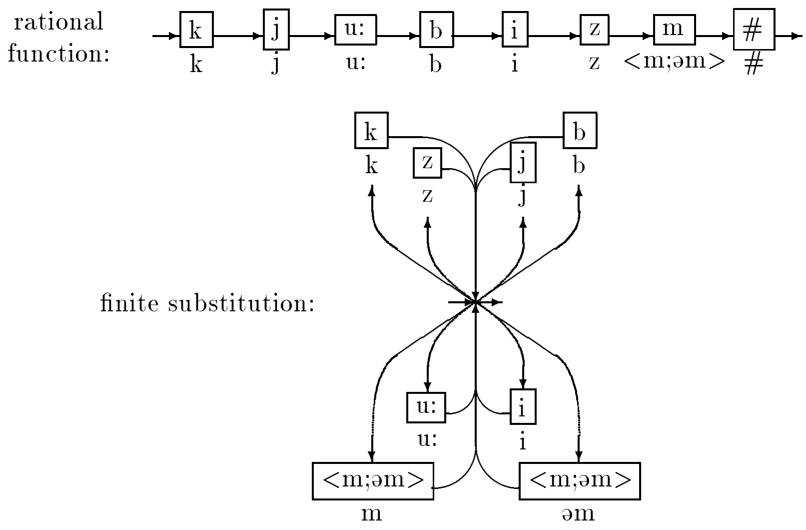


Figure 6: decomposition into a rational function and a finite substitution.

and finite substitutions can be realized by well-known, simple devices for which efficient implementations are known. Finite substitutions are realized by one-state transducers. Rational string functions are realized by bimachines.

5.2 Bimachines

The notion of bimachine was introduced by Schützenberger (1961). It is a strictly alphabetic, deterministic variant of the notion of finite transducer. The set of transductions realized by bimachines is the set of rational string functions (Eilenberg, 1974). Any bimachine can be compiled into an equivalent transducer with the same alignment.

A bimachine over alphabets A and B is composed of two finite sets $\overrightarrow{Q}$, $\overleftarrow{Q}$, two initial states $\overrightarrow{q_-}\in\overrightarrow{Q}$, $\overleftarrow{q_-}\in\overleftarrow{Q}$, two transition functions

$\overrightarrow{\delta} : \overrightarrow{Q} \times A \to \overrightarrow{Q}$ and $\overleftarrow{\delta} : \overleftarrow{Q} \times A \to \overleftarrow{Q}$, and an output function

$$\gamma : \overrightarrow{Q} \times A \times \overleftarrow{Q} \to B^\star.$$

In fact, $\overrightarrow{Q}$, $\overrightarrow{q_-}$ and $\overrightarrow{\delta}$ constitute a left-to-right deterministic automaton without final states, and $\overleftarrow{Q}$, $\overleftarrow{q_-}$ and $\overleftarrow{\delta}$ constitute a right-to-left deterministic automaton without final states. The transition functions are extended to $\overrightarrow{Q} \times A^\star$ and $\overleftarrow{Q} \times A^\star$ by setting $\overrightarrow{\delta}(\overrightarrow{q}, \text{<E>}) = \overrightarrow{q}$, $\overleftarrow{\delta}(\overleftarrow{q}, \text{<E>}) = \overleftarrow{q}$, $\overrightarrow{\delta}(\overrightarrow{q}, ua) = \overrightarrow{\delta}(\overrightarrow{\delta}(\overrightarrow{q}, u), a)$, $\overleftarrow{\delta}(\overleftarrow{q}, ua) = \overleftarrow{\delta}(\overleftarrow{\delta}(\overleftarrow{q}, u), a)$. For an input string $a_1 a_2 \ldots a_n$, the output for a_i is defined as

$$\gamma(\overrightarrow{\delta}(\overrightarrow{q_-}, a_1 a_2 \ldots a_{i-1}), a_i, \overleftarrow{\delta}(\overleftarrow{q_-}, a_n a_{n-1} \ldots a_{i+1}))$$

The output string for $a_1 a_2 \ldots a_n$ is the concatenation of the output strings for $a_1, a_2, \ldots a_n$. Thus, a bimachine realizes a string function.

Bimachines are a convenient tool both for linguistic description and computation.

The linguistic description of a transduction related to phonetics generally takes the form of a set of conversion rules. Usual rules comprise a 'context part', which recognizes whether the rule applies, and an 'action part', which translates symbols. Rules are often stated in the form $a \longrightarrow u/L\_\_\_R$, where the context part is $L\_\_\_R$ and the action part is $a \longrightarrow u$. In the usual sense, the context refers to the input string, which is known before the rules apply, and not to the output string. This is the most straightforward convention and makes rules readable and easy to design.

The structure of a bimachine is quite similar. The two deterministic automata correspond to the left and right context parts of the rule, and the output function constitutes the action part; the context part refers to the input string only. In section 6, page 23, we describe a readable, graphic mode of representation of bimachines, and algorithms for loading a bimachine from this format and running it.

The structure of a finite transducer is not so directly similar to that of a usual rule. Contexts and actions are mixed up in transition labels. Since transition labels combine input and output symbols, contexts may refer both to input and output labels (see Koskenniemi (1983), Kaplan and Kay (1994) for examples).

The computation of a bimachine is deterministic, hence simpler than that of a non-deterministic device. On the other hand, the inversion of a transduction (swapping input and output) is probably easier to implement on a transducer than on a bimachine.

5.3 Locality

The translation of a symbol usually depends on its context, but this dependency is usually very local. This is probably the reason why phonologists are so fond of counter-examples with unbounded dependencies. Intuitively, a conversion rule is local if the length of the context needed to apply the rule is bounded for all input strings. Typical values of this bound are small, about ten or even five symbols. Contexts of unbounded length are frequently used by phonologists, but in most cases they are easy to replace with bounded contexts. For example, <Cons>* in a context apparently matches any number of consonants, but since sequences of consonants with no intervening vowel hardly go beyond five symbols in French, the pattern of Figure 7 has the same effect as a rule that converts /e/ into [ɛ] before <Cons><Cons>*#.

Formally, the notion of locality is defined with respect to automata. Let s, d be positive integers such that $0 \leq d \leq s$. An automaton is

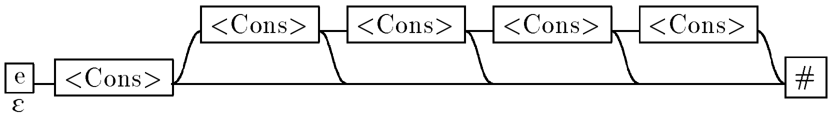


Figure 7: bounded context for sequences of consonants.

(s, d)-local if for each pair of paths of length s, $(q_0, a_1, q_1, \dots a_s, q_s)$ and $(q'_0, a_1, q'_1, \dots a_s, q'_s)$, labelled by the same sequence $a_1 a_2 \dots a_s$, we have $q_d = q'_d$. An automaton is local if there exist s and d such that it is (s, d)-local. If so, the smallest possible value for s is called the scope of the automaton.

This notion of locality applies to the left-to-right and right-to-left deterministic automata of a bimachine. Let l, r be positive integers. We say that a bimachine is (l, r)-local if its left-to-right automaton is (l, l)-local and its right-to-left automaton is (r, r)-local. The maximal length of relevant left contexts is l and the maximal length of relevant right contexts is r. If a bimachine is (l, r)-local, the function that it realizes is also realized by a finite transducer whose projection is an $(l + r, l)$-local automaton.

Recall that with BiPho, the transduction is expressed as a combination of rational functions and finite substitutions (cf. page 18). In the phonetic conversion data for French, this decomposition of the problem could be done in such a way that all rational functions are realized by local bimachines, i.e. all contexts have bounded length.

6 Implementation

The rational functions used by BiPho are realized by local bimachines. They are created in a graphic form like that of Figure 8, which comprises a context part which recognizes whether the rule applies and an action part which translates symbols. Each batch of graphs that must apply simultaneously is read and combined into a bimachine. The resulting bimachines apply sequentially to input strings. The finite substitutions that should apply to the output of the bimachines are not implemented yet.

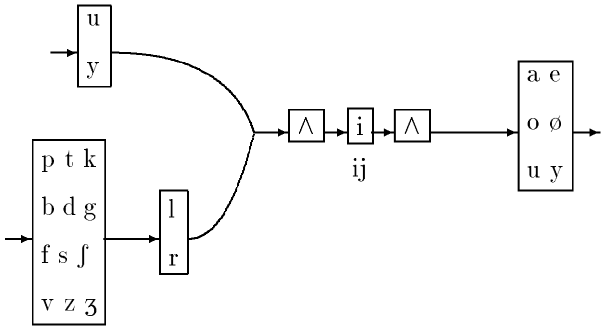


Figure 8: a conversion rule.

6.1 Construction of a bimachine from a rule

The conversion rule of Figure 8 reads as follows. The left and right context parts are delimited from the action part of the rule by the symbol $\wedge$. The left and right contexts take the form of finite automata that read from left to right. The transitions in the context parts are boxes with only input labels (inside the boxes). The action part is the only transition which has both an input label and an output label (below the box). The input label is one input symbol, but the output label may be a string of zero, one or several output symbols. In case of variants, the output label stands for the list of variants. The semantics of the rule is straightforward: whenever the input label of the action part occurs between the left and right contexts, substitute the output label for it, otherwise leave it unchanged. This rule converts /i/ into /ij/ in certain contexts, e.g. for *plier* [plije] 'fold', it converts /plie/ into /plije/.

The context part of the graph contains only the part of the context which is relevant to the transduction; if the action part of the rule must take place no matter what the left context is, then the left context part of the graph is empty. The left and right context parts of the automaton are converted into finite automata which are then determinized and minimized with the aid of standard algorithms. Let L (resp. R) be the set of sequences recognized by the left (resp. right) context part of the graph: the left-to-right deterministic automaton of the bimachine must recognize $A^\star L$, and the right-to-left automaton must recognize $A^\star \widetilde{R}$, where the elements of $\widetilde{R}$ are the elements of R read in reverse order. The only algorithm needed for the construction of these automata is the construction of a finite automaton recognizing $A^\star L$ from an automaton recognizing L, and the same for $\widetilde{R}$. Since relevant contexts are bounded in length, L and R are finite. We apply to them a variant of the algorithm of Aho and Corasick (1975). The original version of this algorithm makes use of the set of prefixes of a finite set L. This set can be replaced with the set of states of the minimal deterministic finite automaton recognizing L. The algorithm has to be adapted (Mohri, 1994), but it produces an automaton with less states than in the original version. However, the resulting automaton is not necessarily minimal, so we minimize it. There is no notion of final states in a bimachine. In our implementation, the automata for $A^\star L$ and $A^\star \widetilde{R}$ do have final state sets: they are used in the definition of the output function.

The output function γ of the bimachine is defined as follows: if $a \in A$ is the input label of the action part, and if $u \in B^\star$ is the output label, then

$\gamma(\overrightarrow{q}, b, \overleftarrow{q})$ `:=`
 `if` b `=` a `and` $\overrightarrow{q}$ `is final and` $\overleftarrow{q}$ `is final`
 `then` u
 `else` b

The two deterministic automata are implemented with two-dimensional tables whose rows are indexed by states and whose columns are indexed by input symbols. The content of the table at line $\overrightarrow{q}$ and at column a is the state $\overrightarrow{\delta}(\overrightarrow{q}, a)$.

6.2 Simultaneous combination of bimachines

Several transductions realized by bimachines can apply simultaneously to the same input provided that they do not conflict. A conflict is defined as follows. Let $a_i \longrightarrow u_i/L_i \_\_\_ R_i$, for $1 \leq i \leq n$, be n bimachines over A and B defined as above: $L_i \subset A^\star$ and $R_i \subset A^\star$ are the left and right context parts, $a_i \in A$ is the input label of the action part and $u_i \in B^\star$ is the output label of the action part. A conflict arises if and only if two bimachines apply to the same input symbol of an input string, i.e. if there are two indices i and j, with $1 \leq i < j \leq n$, such that $A^\star L_i \cap A^\star L_j \neq \emptyset$, $a_i = a_j$ and $R_i A^\star \cap R_j A^\star \neq \emptyset$. This condition is checked for each pair of rules by computing the intersections of contexts.

If there are no conflicts, the simultaneous combination is possible. Let $\overrightarrow{Q_i}$, $\overleftarrow{Q_i}$ be the state sets of the n bimachines, $\overrightarrow{q_{i,-}} \in \overrightarrow{Q_i}$, $\overleftarrow{q_{i,-}} \in \overleftarrow{Q_i}$ the initial states, $\overrightarrow{\delta_i}$: $\overrightarrow{Q_i} \times A \rightarrow \overrightarrow{Q_i}$ and $\overleftarrow{\delta_i}$: $\overleftarrow{Q_i} \times A \rightarrow \overleftarrow{Q_i}$ the transition functions, and $\gamma_i : \overrightarrow{Q_i} \times A \times \overleftarrow{Q_i} \rightarrow B^\star$ the output function. Then the combined bimachine is defined as follows:

$$\overrightarrow{Q} = \overrightarrow{Q_1} \times \overrightarrow{Q_2} \times \ldots \times \overrightarrow{Q_n},\ \overleftarrow{Q} = \overleftarrow{Q_1} \times \overleftarrow{Q_2} \times \ldots \times \overleftarrow{Q_n},$$

$$\overrightarrow{q_-} = (\overrightarrow{q_{1,-}}, \overrightarrow{q_{2,-}}, \ldots \overrightarrow{q_{n,-}}),\ \overleftarrow{q_-} = (\overleftarrow{q_{1,-}}, \overleftarrow{q_{2,-}}, \ldots \overleftarrow{q_{n,-}}),$$

$$\overrightarrow{\delta}\ ((\overrightarrow{q_1}, \ldots \overrightarrow{q_n}), a) = (\overrightarrow{\delta_1}\ (\overrightarrow{q_1}, a), \ldots \overrightarrow{\delta_n}\ (\overrightarrow{q_n}, a)),$$

$$\overleftarrow{\delta}\ ((\overleftarrow{q_1}, \ldots \overleftarrow{q_n}), a) = (\overleftarrow{\delta_1}\ (\overleftarrow{q_1}, a), \ldots \overleftarrow{\delta_n}\ (\overleftarrow{q_n}, a)).$$

With this definition, $\overrightarrow{Q}$ and $\overleftarrow{Q}$ may contain states which cannot be reached from the initial states, but it is not necessary to actually create such states. Before defining the output function, note that for each $u, v \in A^\star$ such that $\overrightarrow{\delta}\ (\overrightarrow{q_-}, u) = \overrightarrow{\delta}\ (\overrightarrow{q_-}, v)$,

$$\forall i \in [1, n]\ (u \in A^\star L_i \Longleftrightarrow v \in A^\star L_i).$$

For each state $\overrightarrow{q} = \overrightarrow{\delta}\ (\overrightarrow{q_-}, u)$ we can therefore define

$$\texttt{left(}\overrightarrow{q}\texttt{)} \texttt{ := } \{i \in [1, n] \mid u \in A^\star L_i\}.$$

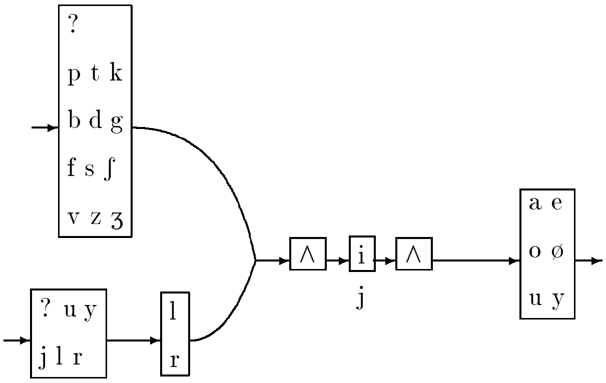
Similarly, for each $\overleftarrow{q} = \overleftarrow{\delta}\ (\overleftarrow{q_-}, u)$

$$\texttt{right(}\overleftarrow{q}\texttt{)} \texttt{ := } \{i \in [1, n] \mid u \in R_i A^\star\}.$$

Now let $\overrightarrow{q} \in \overrightarrow{Q}$, $a \in A$ and $\overleftarrow{q} \in \overleftarrow{Q}$. There is at most one $i \in [1, n]$ such that $i \in \texttt{left(}\overrightarrow{q}\texttt{)}$, $a = a_i$ and $i \in \texttt{right(}\overleftarrow{q}\texttt{)}$. (If there were two, take $\overrightarrow{q} = \overrightarrow{\delta}\ (\overrightarrow{q_-}, u)$ and $\overleftarrow{q} = \overleftarrow{\delta}\ (\overleftarrow{q_-}, v)$: there would be two i's such that $u \in A^\star L_i$, $a = a_i$ and $v \in R_i A^\star$, in contradiction with the fact that there are no conflicts.) If there exists such an i, define $\gamma(\overrightarrow{q}, a, \overleftarrow{q}) = u_i$, otherwise

$\gamma(\overrightarrow{q}, a, \overleftarrow{q}) = a$. This completes the definition of the combined bimachine which will simulate the behaviour of the n bimachines whenever one of them applies to an input symbol.

For example, let us combine the rules of Figures 8 and 9. The rule

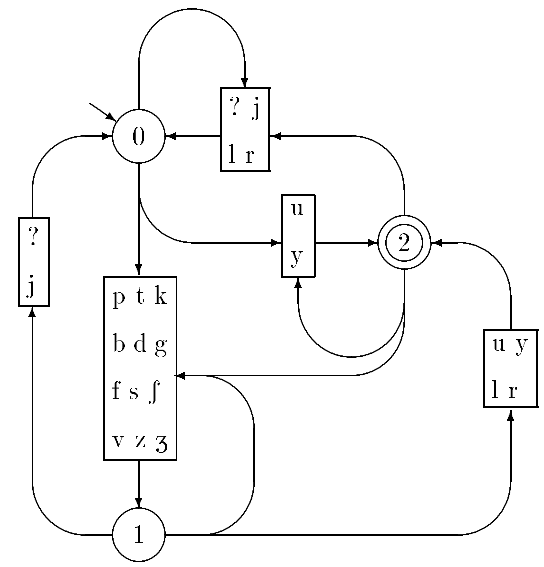


'?' stands for all symbols except p t k b d g f s ʃ v z ʒ l r j u y.

Figure 9: a conversion rule.

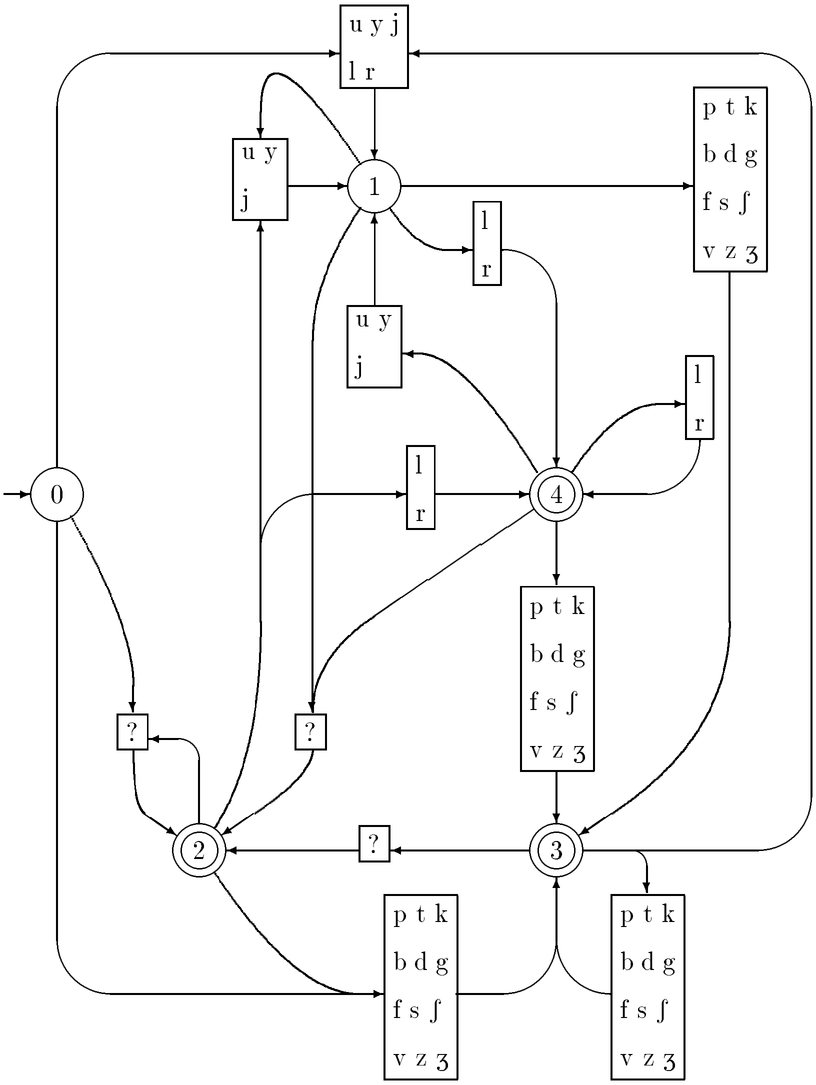
of Figure 9 converts /i/ into /j/ in certain contexts, e.g. for *allier* [alje] 'ally', it converts /alie/ into /alje/. The minimal deterministic automaton for $A^{\star}L_1$ is in Figure 10 and the one for $A^{\star}L_2$ is in Figure 11.

These rules do not conflict. If you build their simultaneous combination, you will obtain the left-to-right automaton of Figure 12, and a two-state right-to-left automaton. The states $\overrightarrow{q}$ which are marked as final in Figure 12 are those for which `left(`$\overrightarrow{q}$`)` is nonempty. With the French phonetic conversion data for BiPho, the deterministic automata of the 12 bimachines have 3 to 144 states. The output function is implemented with two tables, `BimSet` and `Output`. `BimSet` is a two-dimensional ta-



'?' stands for all symbols except p t k b d g f s ʃ v z ʒ l r j u y.

Figure 10: the minimal deterministic automaton for $A^{\star}L_1$.



'?' stands for all symbols except p t k b d g f s ʃ v z ʒ l r j u y.

Figure 11: the minimal deterministic automaton for $A^{\star}L_2$.

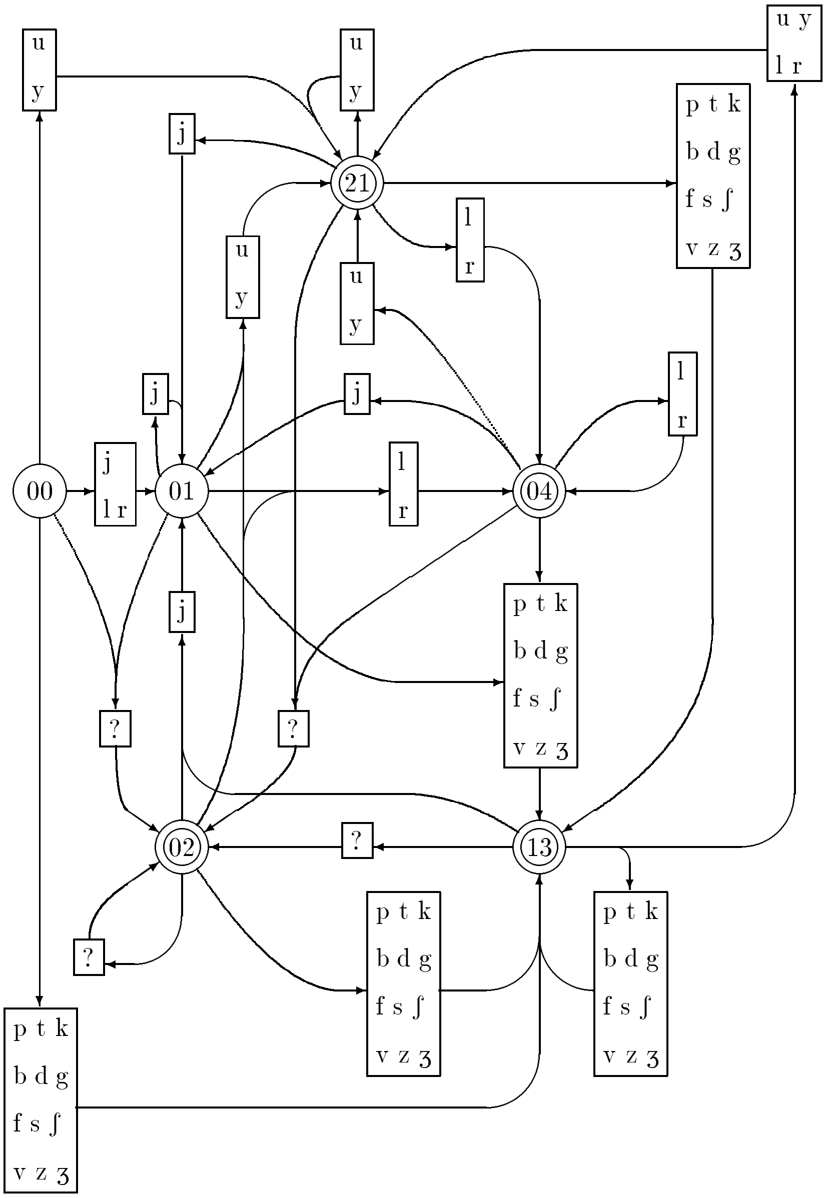


Figure 12: the left-to-right automaton of the simultaneous combination.

ble whose rows are indexed by left-to-right states and whose lines are indexed by right-to-left states. The content of the table at line $\overrightarrow{q}$ and at column $\overleftarrow{q}$ is a key that gives access to the set `left(`$\overrightarrow{q}$`)` $\cap$ `right(`$\overleftarrow{q}$`)`. `Output` is a two-dimensional table whose rows are indexed by the keys of the sets `left(`$\overrightarrow{q}$`)` $\cap$ `right(`$\overleftarrow{q}$`)`, and whose lines are indexed by input symbols. The content of the table at line k and at column a is the output sequence $\gamma(\overrightarrow{q}, a, \overleftarrow{q})$ defined in section 6.2.

6.3 Running the bimachine

When running the bimachine on an input string, the string is first processed in reverse order: we compute the values of the states of the right-to-left automaton for each symbol in the input string and store them in a one-dimensional array. Then, for each symbol from left to right, the state of the left-to-right automaton is computed. This value is used with the value of the right-to-left state, the input symbol and the tables `BimSet` and `Output` in order to retrieve the output sequence. The complexity of this algorithm is independent of the number of states and transitions of the bimachine: the time of the conversion is dominated by the length of the input sequence.

7 Conclusion

The finite-state formal devices described in this chapter and tested in the context of phonetics and phonology proved to be both convenient for linguistic description and adapted for efficient implementation. The conversion system BiPho was tested with complete phonetic conversion data for French. Since phonetic conversion of most languages is simpler than for French, and since BiPho makes only minimal assumptions about the mathematical properties of the conversion, we believe that it can be used for virtually any conversion task related to phonetics.

References

Aho, Alfred and Margaret Corasick. 1975. Efficient string matching: an aid to bibliographic search. *CACM*, 18(6):333–340.

Berstel, Jean. 1979. *Transductions and Context-Free Languages.* Stuttgart: Teubner.

Eilenberg, Samuel. 1974. *Automata, Languages and Machines*, volume A. New York/San Francisco/London: Academic Press.

Johnson, C. Douglas. 1972. *Formal Aspects of Phonological Description.* Mouton.

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

Koskenniemi, Kimmo. 1983. *Two-level morphology: a general computational model for word-form recognition and production.* Publication no. 11. Dept. of General Linguistics, University of Helsinki.

Laporte, Éric. 1989. Quelques variations phonétiques en français. *Lingvisticae Investigationes*, XIII(1):43–116.

Laporte, Éric. 1993. Phonétique et transducteurs. Technical report, Université Paris 7-Denis-Diderot, June.

Mohri, Mehryar. 1994. Syntactic Analysis by Local Grammars Automata: an Efficient Algorithm. Papers in Computational Lexicography (COMPLEX). Research Institute for Linguistics, Hungarian Academy of Sciences, Budapest, pp. 179–191.

Perrin, Dominique. 1990. Finite automata. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*. Elsevier, chapter 1, pages 3–57.

Schützenberger, Marcel-Paul. 1961. A remark on finite transducers. *Inform. and Control*, 4:185–196.