MERL – A MITSUBISHI ELECTRIC RESEARCH LABORATORY
http://www.merl.com

Parsing with Finite-State Transducers

Emmanuel Roche
Mitsubishi Electric Research Laboratories
201 Broadway, Cambridge, MA 02139
e-mail: `roche@merl.com`

TR-96-30. Version 1.0   November 1996

Abstract

Accurately parsing natural language sentences requires large scale and detailed lexical grammars. We will see that for the problem of parsing natural language sentences, finite-state models are not an efficient but somewhat inaccurate tool but rather one of the best formalism to represent accurately complex linguistic phenomena. Finite-state transducers should appeal to the linguist looking for precise and natural description of complex syntactic structures while the wide range of formal operations on finite-state transducers provides the designer of parsing programs with powerful tools to improve parsing efficiency. The parsing programs derived from this approach are both simple, precise linguistically and very efficient.

Revisions history.

1 Introduction

Finite-State methods have recently improved computational efficiency for a wide variety of natural language processing tasks; ranging from morphological analysis [15, 26, 2] to phonetic and speech processing [22, 17, 18].

However, finite-state modeling is usually thought as a necessary evil in the sense that more powerful formalisms such as context-free grammars are more accurate but of intractable size for reasonable efficiency. A clear illustration of this view comes from the field of speech recognition in which grammars are often given in a context-free form but the size of the data and the finite-state nature of most representations (phoneme or word lattice for instance) make it difficult and inefficient to use general algorithms such as the Earley parsing. Complex grammars are therefore approximated by finite-state models [21]. In these situations, approximations lead to more efficient and simpler parsing strategies at the cost of a lost of accuracy.

We will see here that for the problem of parsing natural language sentences, finite-state models are not an efficient but somewhat inaccurate tool but rather one of the best formalism at hand to represent accurately complex linguistic phenomena. The use of finite-state transducers for parsing should appeal both to the linguist looking for precise and natural description of complex phenomena and to the implementer of efficient parsing programs.

From a computational point of view, finite-state transducers can be used to parse very large scale lexical grammars. Some of the formal characteristics of transducer parsing include:

- very large scale grammars are represented in a compact form. Parts of different rules that are similar are represented only once,
- factorization, determinization and minimization of transducers can be used to generate more efficient parsers [24],
- the grammar compilation, input sentences and the parsing use homogeneous representations,
- parsing and building the grammar are the same operation, i.e. a rational transduction,
- transforming a CFG into a transducer is yet another transduction,

These properties have been described and illustrated on a large coverage grammar of French in [23].

From a linguistic point of view, we take here a complementary approach to [10] in this volume which shows that FSA is a very natural framework for the task of designing large coverage and lexicalized grammars. We focus on the process of parsing a sentence, therefore assuming that a very precise grammar is already available. While our main point is not grammar design, the discussion should shed light on several important problems and solutions involved at the grammar building stage.

One of the main drawback in using context-free grammars for modeling language is the inability or the difficulty of handling various types of deletion. Consider the sentence

(1) *He expected John to buy a new book.*

In the transformational grammar of Z. S. Harris (see [12] for instance), this sentence is analyzed as the verb *expected* (an *operator*) taking three arguments: (1) the subject *he*, (2) the first complement *John* and (3) the sentence *John buys a new book*; the application of the verb operator deletes the subject of the last argument to transform the sentence into the infinitive clause *to buy a new book*. In order to handle this situation with context-free grammars, each sentence has to be described both with its declarative form, say *N buy N*, and with its infinitive clause form, say *to buy N*. This might not seem to be a difficult problem at first but recall that grammars have to be lexicalized (see [7, 1] for instance), that is, each rule should contain an explicit word, and therefore this type of duplication, which is only one duplication among many others, has to be repeated throughout the whole lexicon.

The next section will give a short background about parsing viewed as a string transformation (the formal definitions of the objects manipulated in this chapter are given in the general introduction of this volume). The following section, describing one of the ways context-free languages can be parsed with finite-state transducers, give the general framework for transducer parsing. Section 4 shows that the interaction between morphology and syntax can be viewed as a simple sequence of finite-state operations. Section 5, shows, through a variety of linguistic examples, that transduction parsing is well adapted to transformational grammars and that, in addition to lead to efficient parsing strategies, it also leads to more accurate sentence analysis. We will see in Section 6 that both parsing and grammar design can be viewed as a simple transduction operation; this similarity can be used to precompile the analysis of language sequences that are finite-state in nature. Section 7 details a typical example of tree adjoining grammar to show that there is a natural way of converting a tree adjoining grammar into a single finite-state transducer. The transducer associated to a tree adjoining grammar also defines a parser for this grammar. Since the tree adjoining grammar formalism, which is strictly more powerful than context-free grammars, was designed specifically to render more accurately syntactic phenomena, this is another illustration of the fact that transducer parsing is a tool well adapted to the complexity of natural language.

2 Background

Parsing will be considered here as a string transformation process. Namely, if Σ_g represents the set of symbols, such as $(N$, $N)$ or $(S$, used to mark the syntactic analysis; if Σ_w is the list of words in the language, then a parser is a mapping

$$\text{parser} : \Sigma_w^* \longrightarrow 2^{(\Sigma_g^* \cdot \Sigma_w^* \cdot \Sigma_g^*)^*}$$

such that, for instance, a sequence of words like:

(2) *John left this morning.*

is transformed into a set of of outputs representing the analysis of this sentence:

(3) (S (N *John* N) (V *left* V) (N *this morning* N) S)

Here the set contains only one element but is would contain more than one element if the sentence was syntactically ambiguous.

In practical systems parsing will be broken up into two phases: (1) a morphological analysis and (2) the syntactic analysis. In order to simplify the exposition, in the following section as well as in most of this chapter, we will make the assumption that the syntactic analysis applies directly on the words and that morphological information is available when necessary. This simplification is justified in Section 4.

3 A Top-Down Parser for Context-Free Grammars

One way of parsing context-free grammars with finite-state transducers consists of modeling a top-down analysis. Consider the sentence

(4) *John thinks that Peter kept the book*

and suppose the we have the syntactic data of Table 1.

This data can be seen as a sample of a syntactic dictionary in which the keys are the structures and the information is the type of the structure (sentence structure, noun structure). Such a syntactic dictionary would look like the sample of Figure 1[1].

[1]This grammatical representation is motivated by the fact that grammatical representations, whatever the formalism they are expressed in, should be lexicalized. The linguistic motivation comes from Z.S. Harris [12] and Maurice Gross [6, 7].

N *thinks that* S	S
N *kept* N	S
John	N
Peter	N
the book	N

Table 1: syntactic dictionary for *John thinks that Peter kept the book*

⋮
N think,*S*
N think that S,*S*
⋮
N say N,*S*
N say that S,*S*
N say N to N,*S*
N say that S to N,*S*
N say to N that S,*S*
⋮
John,*N*
⋮
the book,*N*
⋮

Figure 1: Sample of the Syntactic Dictionary

The first step of building a parser consists of transforming each entry of this syntactic dictionary into a finite-state transducer. The finite-state transducer related to an entry will be the machine responsible for analyzing a sentence with this structure. Here, this will lead to the transducers[2] of Table 2 which can be seen as a dictionary of transducers.

Each transducer represents a transduction from Σ^* to Σ^* where $\Sigma = \Sigma_w \cup \Sigma_g$. For instance, the transducer associated to the structure *N thinks that S* (the first transducer of Table 2), that we denote T_{thinks_that}, will map (5) to (6).

(5) [S *John thinks that Peter kept the book* S]
(6) (S [N *John* N] *thinks that* [S *Peter kept the book* S] S)

Formally, $T_{thinks\_that}(5) = (6)$.

[2] On this graph, the symbol *?* stands for any symbol in the alphabet considered and the symbol <*E*> stands for the empty word ϵ.

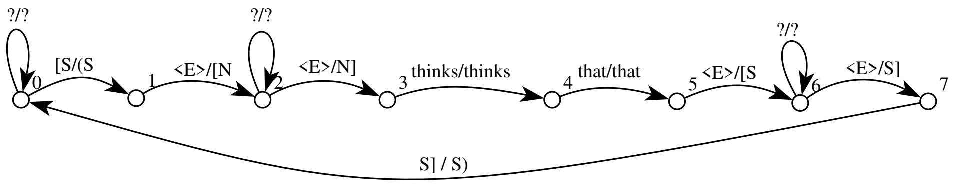
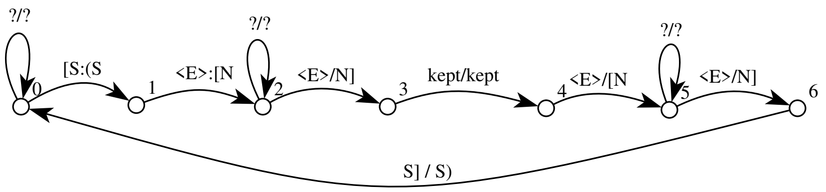
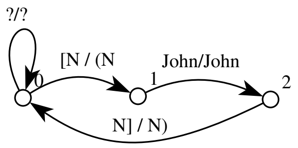
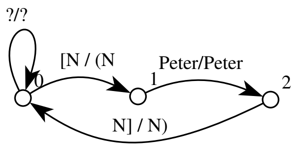
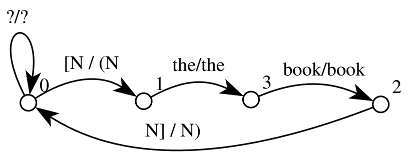
<table>
<tr><td>S</td><td>N <i>thinks that</i> S</td><td>[S <i>a</i> thinks that <i>b</i> S] → (S [N <i>a</i> N] <V <i>thinks</i> V> <i>that</i> [S <i>b</i> S] S)</td></tr>
<tr><td colspan="3">
</td></tr>
<tr><td>S</td><td>N <i>kept</i> N</td><td>[S <i>a</i> kept <i>b</i> S] → (S [N <i>a</i> N] <V <i>kept</i> V> [N <i>b</i> N] S)</td></tr>
<tr><td colspan="3">
</td></tr>
<tr><td>N</td><td>John</td><td>[N John N] → (N John N)</td></tr>
<tr><td colspan="3">
</td></tr>
<tr><td>N</td><td>Peter</td><td>[N Peter N] → (N Peter N)</td></tr>
<tr><td colspan="3">
</td></tr>
<tr><td>N</td><td>the book</td><td>[N the book N] → (N the book N)</td></tr>
<tr><td colspan="3">
</td></tr>
</table>

Table 2: Transducers associated to each structure

Given the dictionary we define the grammar

$$T_{dic} = \bigcup_{T_i \in dic} T_i$$

as being the union of all the transducers defined in the dictionary. For instance, if dic_1 is the dictionary defined in Table 2, then T_{dic_1} is the transducer represented of Figure 2.

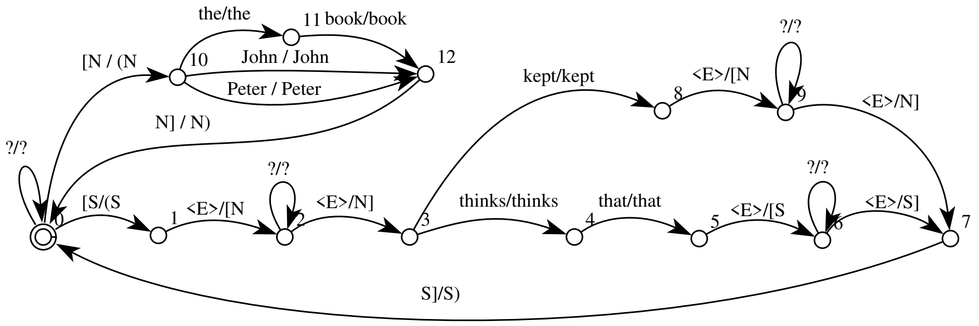


Figure 2: Transducer T_{dic_1} representing the syntactic dictionary dic_1

This transducer being given, parsing simply consists of applying the transducer on the input and checking whether the output is different from the input; if it is, then the transducer is applied again. The process is repeated until the input is not modified. Formally,

$$\text{parser} = T_{dic}^{\infty}$$

On the example of the input (5), the parsing is illustrated below:

[S *John thinks that Peter kept the book* S]
$\downarrow (T_{dic_1})$
(S [N *John* N] *thinks that* [S *Peter kept the book* S] S)
(S [N *John thinks that Peter* N] *kept* [N *the book* N] S)
$\downarrow (T_{dic_1})$
(S (N *John* N) *thinks that* (S [N *Peter* N] *kept* [N *the book* N] S) S)
$\downarrow (T_{dic_1})$
(S (N *John* N) *thinks that* (S (N *Peter* N) *kept* (N *the book* N) S) S)

The input sequence is given on top and the first application of T_{dic_1} results in two outputs. We reapply the transducer on each one, the first one leads to the next sequence while the second one has no output. Finally, the latest sequence is:

(7) (S (N *John* N) *thinks that* (S (N *Peter* N) *kept* (N *the book* N) S) S)

which is also the analysis of the input.

The parsing is described here as the application of a transducer on a string however, in practice, the input strings are represented by FSAs and the transducers therefore apply on these FSAs directly. For instance, the input sentence is initially represented by the flat automaton of Figure 3.

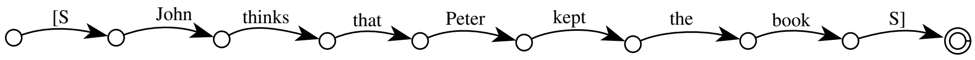


Figure 3: Automaton representing the input sentence

Once T_{dic} is available, the parsing algorithm is therefore extremely simple to describe, it is given on Figure 4 in which the function *Apply_transducer*, takes a transducer and an automaton as input and outputs an automaton. Each string of this output automaton is an output of a string represented by the input automaton through the transducer.

TransducerParse
Input: T_{dic}, sentence
sent_1 = sentence;
while $((\mathrm{sent}_2 = \mathrm{Apply\_transducer}(T_{dic}, \mathrm{sent1}))! = sent_1)$
    $\mathrm{sent}_1 = \mathrm{sent}_2$;
Output: sent_1

Figure 4: Parsing algorithm

A trace of the parsing of the example above is given on Figure 5. The figure shows that each transduction is applied on a finite-state automaton, a directed acyclic graph here, representing a finite set of strings.

4 Morphology

The transducers representing the sentence structures in the previous section apply directly on the inflected words like *thinks* or *kept*. Hence, in order build a full parser, one would have to duplicate each transducer for all the inflected forms of a given noun or verb. For instance, the transducer representing the sentence structure *N kept N* should a priori be duplicated into the transducer representing *N keep N* and *N keeps N*. Obviously, such an approach is highly redundant and space consuming and for languages with higher inflexion variability (French, Italian, Spanish for instance) this type of duplication is not feasible. A more practical approach consists of decomposing the parsing process into a morphological analysis and the syntactic analysis per se. The morphological analysis will map each inflected word into a unique canonical form (the infinitive form for a verb or the singular form for a noun) plus a list of morphological features such as tense, person or number. The individual

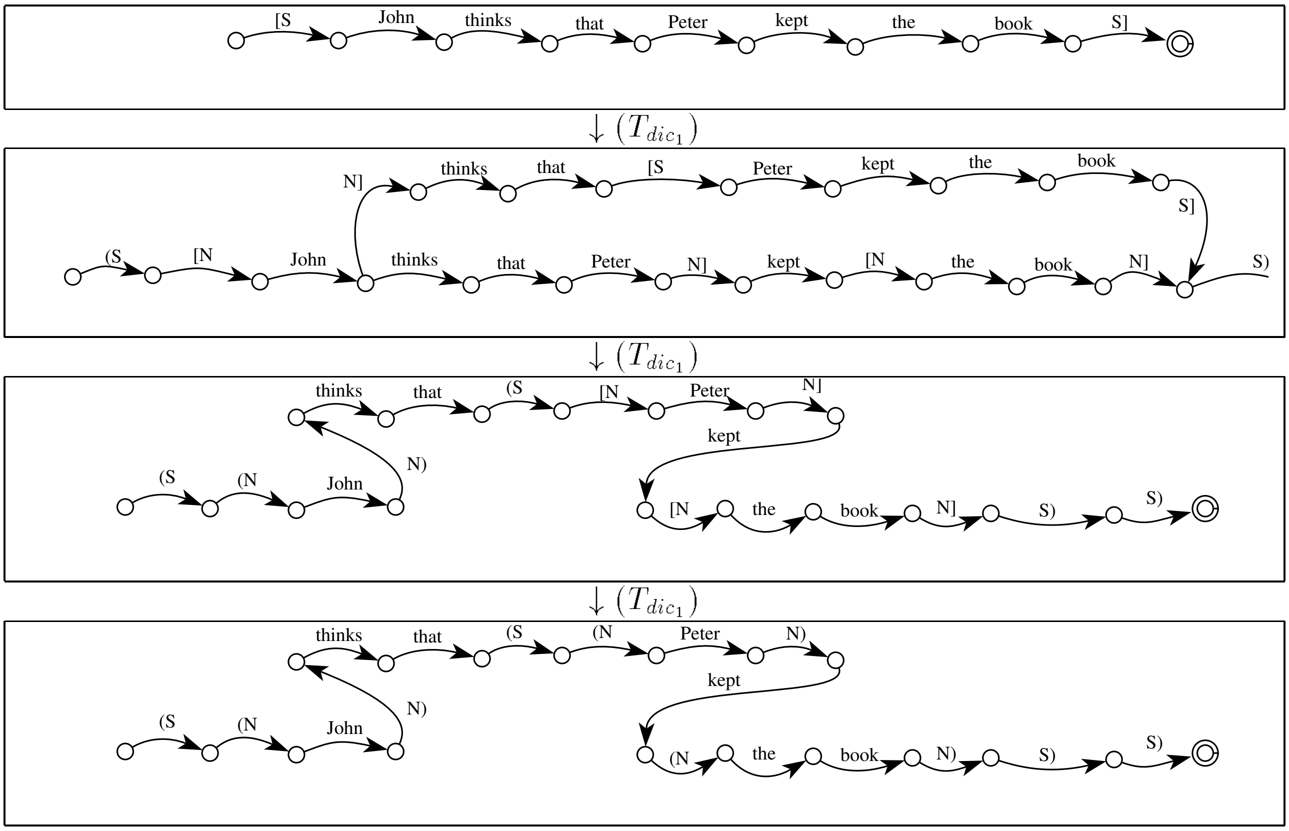


Figure 5: Parsing trace of the example

syntactic transducers will now be described as applying on the canonical forms rather than on the inflected words.

In first approximation, the morphological analysis is a dictionary lookup which can also be formalized as a string transformation process. Indeed, consider the sentence

(8) *John left this morning.*

Each of the four words of this sentence should be an entry in a morphological dictionary[3]. The information related to each entry is a list of morphological interpretations for this entry. For instance, the ambiguous string *left* could be a noun, a verb or an adjective and the information associated to *left* should therefore give all these interpretations. More formally, each interpretation can be written as a sequence of symbols. For instance, the interpretation of *left* as the past tense of the verb *to leave* will be the sequence:

(9) # v pt *leave left*

[3]The precise description of such dictionaries is described in [3]. See also [26] and [27] in this volume for other kinds of precise treatment of morphological analysis.

with the convention that the symbol # marks the starting point of the interpretation, that *v* stands for verb, that *pt* stands for past tense and that the word *leave* is the canonical form of the word (infinitive form here) whereas the last symbol is the word itself. With this representation, the entry of the string *left* in the morphological dictionary will be the following[4]:

left	# v pt *leave left* # v pp *leave left* # ns *left* # adv *left* # adj *left*

With these conventions, each interpretation is a string of

$$\{\#\} \cdot \Sigma_m^* \cdot \Sigma_w$$

in which

- Σ_w is the list of words and symbols such as *eat, John, left, leave* that can appear in the text being analyzed,
- Σ_m contains symbols representing morphological features (such as *pt, ns* or *adv*) and the canonical forms of the words (such as *leave*).

Moreover, since the information associated to a word is a list of interpretations, the information is an element of

$$2^{(\{\#\} \cdot \Sigma_m^* \cdot \Sigma_w)}$$

and the dictionary lookup is therefore a mapping from words to morphological information, that is:

$$lookup : \Sigma_w^* \rightarrow 2^{(\{\#\} \cdot \Sigma_m^* \cdot \Sigma_w)}$$

The whole dictionary can therefore be viewed as a string transformation. For instance, the sub-dictionary necessary to analyze Sentence (8) can be represented as follows:

[4]We use the additional following conventions: *pp*: past participle, *ns*: singular noun, *adv*: adverb, *adj*: adjective.

$$
\begin{array}{lcl}
John & \rightarrow & \#\ \mathrm{pn}\ John \\
 & & \\
 & & \#\ \mathrm{v\ pt}\ leave\ left \\
 & & \#\ \mathrm{v\ pp}\ leave\ left \\
left & \rightarrow & \#\ \mathrm{ns}\ left \\
 & & \#\ \mathrm{adv}\ left \\
 & & \#\ \mathrm{adj}\ left \\
 & & \\
this & \rightarrow & \#\ \mathrm{det}\ this \\
 & & \#\ \mathrm{det}\ this \\
 & & \\
morning & \rightarrow & \#\ \mathrm{ns}\ morning
\end{array}
$$

The function *lookup* takes individual words as input, it can be generalized into the function *morpho* that associates to each sequence of words, e.g. sentence, its morphological interpretations. Formally,

$$morpho = lookup^*$$

that is,

$$morpho(a \cdot b) = lookup(a) \cdot lookup(b)$$

and therefore *morpho* is a function mapping a sequence of words to a set of sequences of words and morphological information:

$$morpho : \Sigma_w^* \rightarrow 2^{(\{\#\}\cdot\Sigma_m^*\cdot\Sigma_w)^*}$$

For instance, the following sequence, which is the correct interpretation of (8),

(10) # pn *John* # v pt *leave left* # det *this* # nn *morning* # .

is an element of *morpho*(8). Obviously, *morpho*(8) doesn't contain only the correct morphological interpretation, it contains all the possible interpretations. At this point it is important to notice that number of interpretations of a sentence is typically very high. For instance (8) contains 5×2 interpretations, and this number grows exponentially with the length of the sentence. Hence, the result of the mapping of *morpho* will not be the explicit list of strings representing each interpretation, this would be unmanageable, but rather a finite-state automaton representing this set of interpretations. For instance, *morpho*(8) will be represented by the automaton of Figure 6.

The syntactic analysis will run on the result of the preliminary morphological analysis. In other words, the whole parsing process, which is a mapping:

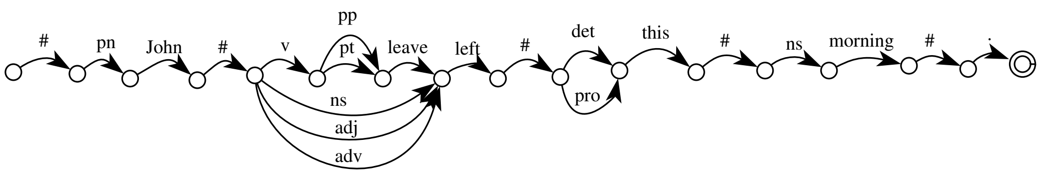


Figure 6: Automaton representing a morphological analysis.

$$\text{parser} : \Sigma_w^* \longrightarrow 2^{(\Sigma_g^* \cdot \Sigma_w^* \cdot \Sigma_g^*)^*}$$

will be broken as indicated in the following diagram:

$$\begin{array}{ccc}
\Sigma_w^* & & 2^{(\Sigma_g^* \cdot \Sigma_w^* \cdot \Sigma_g^*)^*} \\
\downarrow {\scriptstyle morpho} & & \uparrow {\scriptstyle morpho^{-1}} \\
2^{(\{\#\} \cdot \Sigma_m^* \cdot \Sigma_w^*)^*} & \xrightarrow{syntax} & 2^{(\Sigma_g^* \cdot \{\#\} \cdot \Sigma_m^* \cdot \Sigma_w^* \cdot \Sigma_g^*)^*}
\end{array}$$

In the example of sentence (8), we can illustrate this decomposition as follows:

John left this morning.

$\downarrow$ ($_{morpho}$)

$\vdots$

\# pn *John* \# v pt *leave left* \# det *this* \# nn *morning* \# .

\# pn *John* \# v adj *left* \# det *this* \# nn *morning* \# .

$\vdots$

$\downarrow$ ($_{syntax}$)

(S (N \# pn *John* N) (V \# v pt *leave left* V) (N \# det *this* \# nn *morning* N) S) \# .

$\downarrow$ ($_{morpho^{-1}}$)

(S (N *John* N) (V *left* V) (N *this morning* N) S)

The morphological analyzer applies directly on the input string to produce a set of morphological interpretations for this sentence. The syntactic analysis is then performed by *syntax* on each morphological interpretation, that is on the automaton representing the set of interpretations. A final stage, denoted $morpho^{-1}$, removes the symbol representing the morphological features as well as the canonical forms introduced by the first step.

In order for the syntactic analysis to apply on the result of the morphological analysis, each syntactic transducer should be adjusted to handle inputs of the correct shape, that is, outputs generated by the morphological analysis. For instance, the transducer T_{kept} (second transducer of Table 2) will be replaced by the transducer T'_{keep} of Figure 7. In this transducer, the transition *kept/kept* is replaced by a sequence of transitions labeled *#/# v/v ?/? keep/keep ?/?* that will take any string of the shape[5]

$$\# \cdot v{\cdot}? \cdot keep{\cdot}?$$

Such strings are the morphological interpretation of any form of the verb *to keep*. Note that without a morphological analysis, the grammar should contain, in addition to T_{kept}, two similar transducers T_{keep} and T_{keeps}. By contrast, only T'_{keep} will be used now.

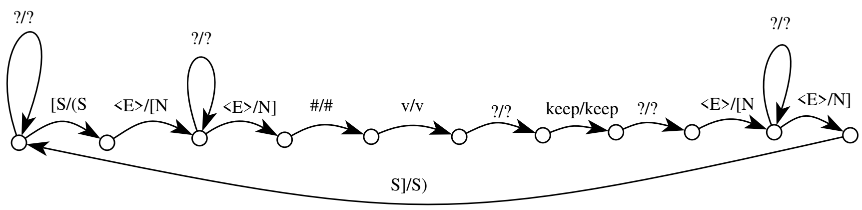


Figure 7: Syntactic transducer to be combined with a morphological analysis.

As a by-product of such decomposition, it becomes straight-forward to transform the syntactic analysis into a part-of-speech tagger (i.e. a program that disambiguates each word). In fact, if $morpho^{-1}$ is replaced by a mapping $gram^{-1}$ that removes the syntactic markers instead of the morphological symbols. The final result is a string representing the exact morphological interpretation of each word. In other words, the process defined by the following diagram:

$$\begin{array}{ccc} \Sigma_w^* & & 2^{(\{\#\}\cdot\Sigma_m^*\cdot\Sigma_w^*)^*} \\ \downarrow_{morpho} & & \uparrow_{gram^{-1}} \\ 2^{(\{\#\}\cdot\Sigma_m^*\cdot\Sigma_w^*)^*} & \xrightarrow{syntax} & 2^{(\Sigma_g^*\cdot\{\#\}\cdot\Sigma_m^*\cdot\Sigma_w^*\cdot\Sigma_g^*)^*} \end{array}$$

will take sentence (8) and process as described on Figure 8.

Another natural consequence of this approach is that a local grammar disambiguation can take place between the dictionary lookup and the syntactic analysis. The introduction chapter showed how it is possible to state negative constraints on morphological interpretations of sentences. By doing that, the number of ambiguities decreases significantly. Formally, a set of constraints

[5] Recall that ? stands for any string.

John left this morning.

$\downarrow$ ($_{morpho}$)

⋮

\# pn *John* # v pt *leave left* # det *this* # nn *morning* # .

\# pn *John* # v adj *left* # det *this* # nn *morning* # .

⋮

$\downarrow$ ($_{syntax}$)

(S (N # pn *John* N) (V # v pt *leave left* V) (N # det *this* # nn *morning* N) S) # .

$\downarrow$ ($_{gram^{-1}}$)

\# pn *John* # v pt *leave left* # det *this* # nn *morning* # .

Figure 8: Part-of-speech tagging through syntactic analysis

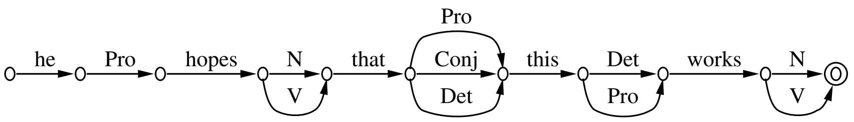


Figure 9: Automaton representing the morphological analysis of *he hopes that this works*

C_1, .. C_n, represented each by a finite-state automaton, is applied on the sentence by the following operation:

$$S' = S - \Sigma^* \cdot (C_1 \cup \ldots \cup C_n) \cdot \Sigma^*$$

Recall that for instance, if S is the sentence represented by the automaton of Figure 9 and if the constraints are represented by the two automata of Figure 10 then S' is given by the automaton of Figure 11. Note that this representation is less ambiguous than S.

Using local constraints can improve the speed of the whole process. Such an analyzer can be summarized by the decomposition diagram of Figure 12:

Figure 10: Automata representing two negative rules.

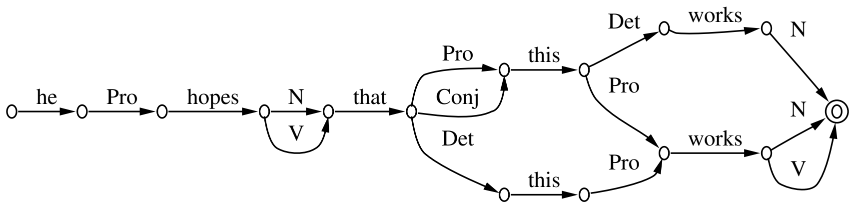


Figure 11: Result of the application of the negative rule of Figure 10 to Figure 9.

$$\begin{array}{ccc} \Sigma_w^* & & 2^{(\Sigma_g^*\cdot\Sigma_w^*\cdot\Sigma_g^*)^*} \\ \downarrow{\scriptstyle morpho} & & \uparrow{\scriptstyle morpho^{-1}} \\ 2^{(\{\#\}\cdot\Sigma_m^*\cdot\Sigma_w^*)^*} & & \\ \downarrow{\scriptstyle local} & & \\ 2^{(\{\#\}\cdot\Sigma_m^*\cdot\Sigma_w^*)^*} & \xrightarrow{syntax} & 2^{(\Sigma_g^*\cdot\{\#\}\cdot\Sigma_m^*\cdot\Sigma_w^*\cdot\Sigma_g^*)^*} \end{array}$$

Figure 12: Parsing Diagram (Local Grammar included)

5 A Parser for Transformation Grammars

In this section, we will illustrate the flexibility of the finite-state transducer formalism through a few examples of increasing linguistic complexity.

5.1 Modal Verbs

Since input sentences, analyzed sentences, and partially analyzed sentences can be written in the same formalism, namely as strings in Σ^*, we can generalize the transducer associated to each structure such that it handles partially analyzed sentences. This extension will make the analysis of apparently very diverse linguistic phenomena homogeneous.

The structure

(11) N <V *read* V> N

is handled by the transducer $T_{N\_read\_N}$, defined by

[S *a read b* S]
↓
(S [N *a* N] <V *read* V> [N *b* N] S)

We now extend it such that it recognizes an already partially analyzed sentence. For instance, $T_{N\_read\_N}$ is extended to realize the additional following mappings:

(12) [S [N *a* N] *read b* S] → (S [N *a* N] <V *read* V> [N *b* N] S)
(13) [S [N *a* N] <V *read* V> [N *b* N] S] → (S [N *a* N] <V *read* V> [N *b* N] S)
(14) [S N *read* N S] → (S N <V *read* V> N S)

In (12), *a* had already been detected as a noun phrase. In a similar way, in (13), the whole structure has already been recognized and in this case the transduction acts as a recognizer. In (14), the transducer just checks whether the proposed structure is correct with respect to the verb.

In addition to these generalizations, the transducer should be able to handle sentences in which sequences, such as adverbials, have been inserted, and therefor perform the following mappings:

(15) [S *a read b* <OP *c* OP> S] → (S [N *a* N] *read* [N *b* N] <OP *c* OP> S)
(16) [S *a* <OP *d* OP> *read b* S] → (S [N *a* N] <OP *d* OP> <V *read* V> [N *b* N] S)

in which the <OP and OP> markers[6] define a sequence that can be ignored during parsing. In practice, the sequence of words *c* could be an adverbial like *yesterday* and *d* could be a modal verb such as *could*.

Let us can now consider the following sentence:

(17) *John should* v *read this book*

in which *v* represents a morphological tag that we suppose results from a morphological analysis. *should* is considered to be an operator that applies on a sentence [12]. It takes a sentence of structure, for instance,

N V W

in which *W* stands for any sequence of complements, and it produces the same sentence with *should* inserted:

N *should* V W

[6]OP stands for operator. In fact, [8] shows that adverbials are often, but not always, analyzed as transformational operators on sentences. In other words, an adverbial is an operator that takes one argument: a sentence.

.

From the analysis point of view, if a sequence contains *should* between what could be a subject and a verb, then the same sequence, minus *should*, should be analyzed as a sentence. In terms of string transformation, the analysis of *John should read this book* is done as follows:

[S *John should read* v *this book* S]
↓
[S [N *John* N] <OP *should* OP> v *read this book* S]
↓
(S [N *John* N] <OP *should* OP> <V v *read* V> [N *this book* N] S)
↓
(S (N *John* N) <OP *should* OP> <V v *read* V> (N *this book* N) S)

The second and third transformations are already handled by T_{dic} as described before. The first transformation is a translation of the fact that if *John should read this book* has to be analyzed as a sentence, then *John read this book* should be analyzed as a sentence whose subject is *John* and whose main verb is *read*. This is taken care of by $T_{N\_should\_W}$, the transducer associated to the structure *N should V W* in the syntactic dictionary. $T_{N\_should\_W}$ computes the following mappings:

[S *a should* v *c* S]
↓
[S [N *a* N] <OP *should* OP> v *c* S]

[S *a should* v <OP *d* OP> *c* S]
↓
[S [N *a* N] <OP *should* <OP *d* OP> OP> v *c* S]

The first mapping was the one used in the example above, it keeps the markers [S and S] meaning that the sequence still have to be analyzed as a sentence. The second mapping handles the case in which another operator had already been inserted.

$T_{N\_should\_W}$ should therefore be added to T_{dic}.

The analysis of sentences with the auxiliary verb *to have* can be handled in a similar way. A further addition to T_{dic}, namely $T_{N\_have\_W}$, will include the following mapping[7]:

(18) [S *a have* v *c* S] → [S [N *a* N] <OP *have* OP> v *c* S]

It is possible to analyze sentences in which several operators are used. For instance, a sentence like:

[7]Whether a verb can take *to have* as an auxiliary can be added in this transducer.

(19) [S *John should have read this book* S]

will be analyzed as follows:

[S *John should have* v *this book* S]
↓
[S [N *John* N] *should* <OP *have* OP> v *read this book* S]
↓
[S [N *John* N] <OP *should* <OP *have* OP> OP> v *read this book* S]
↓
(S [N *John* N] <OP *should* <OP *have* OP> OP> <V v *read* V> [N *this book* N] S)
↓
(S (N *John* N) <OP *should* <OP *have* OP> OP> <V v *read* V> (N *this book* N) S)

in which, as above, sequences enclosed in <OP and OP> markers, once inserted, are simply ignored at latter stage of the analysis. The result of the parsing, namely the string

(S (N *John* N) <OP *should* <OP *have* OP> OP> <V v *read* V> (N *this book* N) S)

should be interpreted as follows: *John read this book* is the main sentence with *John* as subject, *read* as verb and *this book* as direct object. On this sentence, the operator *have* is first applied which leads to the more complex sentence *John has read this book*. Finally, the operator *should* is applied on the previous sentence which leads to the actual sentence *John should have read this book*.

The same strategy can be used to handle discontinuous operators such as *not ... so much*. Consider the following sentence:

(20) *John should not read this book so much.*

The first step of the analysis recognizes both *not* and *so much* as being one discontinuous operator. The analysis goes as follows:

[S *John should not* v *this book that much* S]
↓
[S [N *John* N] *should* <OP *not* OP> v *read this book* <OP ←not *that much* OP> S]
↓
[S [N *John* N] <OP *should* <OP *not* OP> OP> v *read this book* <OP ←not *that much* OP> S]
↓
(S [N *John* N] <OP *should* <OP *not* OP> OP> <V v *read* V> [N *this book* N] <OP ←not *that much* OP> S)
↓
S (N *John* N) <OP *should* <OP *not* OP> OP> <V v *read* V> (N *this book* N) <OP ←not *that much* OP> S)

The symbol $\leftarrow$*not* is inserted such that the final analysis of the sentence clearly identifies *not ... so much* as one single operator. To achieve the previous analysis, a transducer $T_{not\_so\_much}$ needs to be added to the whole grammar. $T_{not\_so\_much}$ is defined functionally by:

[*S a should not v b so much S*]

↓

[*S* [*N a N*] *should* <OP *not* OP> v *b* <OP ←not *so much* OP> *S*]

5.2 Operators on a sentential complements

Another typical situation can be found in sentences like

(21) *John expected Mary to come.*

in which the operator *expected* takes three arguments: a noun, namely *John*, a second noun, namely *Mary*, and a sentence, namely *Mary came*. The subject of the second argument is used as first complement and it is deleted from the argument sentence *Mary came*. We want to have a finite-state transducer representing *expected*, or more precisely the structure[8]

N *expected* N *to* $\epsilon N\ V\ W$

such that nothing has to be changed in the transducer associated to *N come*. In other words, the analysis of *Mary came* should remain as it would be if it where a simple isolated sentence. We achieve this goal with a transducer representing N *expected* N *to* $\epsilon N\ V\ W$ that, not only puts boundary markers around N, but also explicitly adds an N symbol that stands for the deleted noun phrase. For the sentence (21), the transformation is the following:

[S *John expected Mary to come* S]

↓

(S [N *John* N] <V *expected* V> [N *Mary* N] [S N <OP *to* OP> *come* S] S)

and the analysis continues as follows:

(S [N *John* N] <V *expected* V> [N *Mary* N] [S N <OP *to* OP> *come* S] S)

↓

(S (N *John* N) <V *expected* V> (N *Mary* N) (S N <OP *to* OP> <V *come* V> S) S)

[8] The structure N *expected* N *to* $\epsilon N\ V\ W$ could also be written N *expected* N *to Vinf* W. The fist notation emphasizes the fact that the subject of the second verb is deleted. The second structure, on the other hand, emphasizes the fact that the surface form of the second argument is an infinitive clause.

This last step is possible because T_{N_come} as been generalized as described at the beginning of this section. Namely, the symbol N can be used in place of an actual noun phrase and the sequence enclosed by <OP and OP> is ignored.

For the same sentence, with an additional adverbial:

(22) *John expected Mary to come yesterday.*

the analysis goes as follows:

[S *John expected Mary to come yesterday* S]
↓
(S [N *John* N] <V *expected* V> [N *Mary* N] [S N <OP *to* OP> *come yesterday* S] S)
↓
(S (N *John* N) <V *expected* V> (N *Mary* N) (S N <OP *to* OP> *come* <OP *yesterday* OP> S) S)
↓
(S (N *John* N) <V *expected* V> (N *Mary* N) (S N <OP *to* OP> <V *come* V> <OP *yesterday* OP> S) S)

Note that, simultaneously to this analysis, the parser performs the analysis in which the adverbial *yesterday* is attached to the verb *expected* rather than to the verb *came*. In fact, the sentence is ambiguous and the correct result of a syntactic analysis should provide these two analyses.

5.3 Support Verb Construction

[9] demonstrates that three categories of sentences have to be considered to build complete grammars. The first category consists of free sentences like *John eats potatoes*, the second one consists of sentences like *John makes concessions*, with complements with a smaller degree of variability. They are called *support verb constructions*. Finally, the third category consists of *frozen sentences*, or idiomatic expressions, such as *John kicks the bucket*, in which one or several arguments, e.g. complements, are fixed. The rest of this section presents a few typical problems encountered while handling sentences of one of the last two categories.

The following sentences are example of *support* (or *light*) verb constructions.

(23) *John makes concessions to his friend.*
(24) *John makes a right turn*
(25) * *John makes a right turn to his friend.*
(26) *John's concessions to his friend were unexpected.*

If *make* is analyzed as a verb such as *read* then sentences (23) and (24) should be analyzed with the structures

(27) N_0 *makes* N_1 to N_2

(28) N_0 *makes* N_1

This analysis takes the verb as the head of the sentence but fails to explain why sentence (25) is forbidden.

Furthermore, (23) is clearly present in sentence (26). In fact, the noun phrase construction *N's N to N* is not general and

(29) * *John's turn to his friend.*

is clearly forbidden too. These two observations, among others (see [5]), lead to analyze sentences such as (23) with the noun (called *predicative noun*) as the real head and the verb, called *support verb* (or *light verb*), as support for the tense. (23) will therefore be described by the structure

(30) N_0 (V_{sup} make V_{sup}) (N_{pred} *concessions* N_{pred}) *to* N_1

The diversity of the following examples, with the support verb *take*, further illustrates that the predicative noun, and not the verb, governs the number and the nature of the arguments.

(31) *John takes a decision.*
(32) *John takes a look at this.*
(33) *John takes advantage of his position.*
(34) *John takes credible steps toward solving this problem.*
(35) *The party is not likely to take a backseat.*

To be able to parse such sentences each of the following transducers[9], corresponding respectively to (23), (31), (32) and (33) should be added to the global syntactic dictionary

[*S a make concessions to b S*] $\rightarrow$
(S [N *a* N] $< V_{sup}$ *make* $V_{sup} > < N_{pred}$ *concessions* $N_{pred} >$ *to* [N *b* N] S)

[*S a take a decision S*] $\rightarrow$
(*S* [N *a* N] $<V_{sup}$ *take* $V_{sup}> <N_{pred}$ *a decision* $N_{pred}>$ *S*)

[*S a take a look at b S*] $\rightarrow$
(*S* [N *a* N] $<V_{sup}$ *take* $V_{sup}> < N_{pred}$ *a look* $N_{pred}>$ *at* [N *b* N] *S*)

[*S a take advantage of b S*] $\rightarrow$
(*S* [N *a* N] $< V_{sup}$ *take* $V_{sup} > < N_{pred}$ *advantage* $N_{pred} >$ *of* [N *b* N] *S*)

Figure 13 gives the trace of the analysis of sentence (23).

[9]Here, the transductions are defined functionally but in practice they are defined by their graph representation.

$$[S\ \textit{John makes concessions to his friend}\ S]$$
$$\downarrow$$
$$(S\ [N\ \textit{John}\ N]\ <V_{sup}\ \textit{makes}\ V_{sup}>\ N_{pred}<\ \textit{concessions}\ N_{pred}>\ \textit{to}\ [N\ \textit{his friend}\ N]\ S)$$
$$\downarrow$$
$$(S\ (N\ \textit{John}\ N)\ <V_{sup}\ \textit{makes}\ V_{sup}>\ N_{pred}<\ \textit{concessions}\ N_{pred}>\ \textit{to}\ (N\ \textit{his friend}\ N)\ S)$$

Figure 13: Trace of a Support Verb Construction Analysis

5.4 Support Verb and Sentential Clause

In sentence (34), the construction *John takes credible steps toward* is followed by a sentence clause whose verb's subject is also the subject of the main clause, that is, *John*. The analysis works as follows: the input sequence for the parsing transducer is

(36) $[S\ \textit{John takes credible steps toward}\ V_{ing}\ \textit{solving the problem}\ S]$

in which V_{ing} is a marker generated by the morphological analysis[10].

We build the grammar such that the first application of the transduction to (36) leads to the following sequence:

(37) $(S\ [N\ \textit{John}\ N]\ <V_{sup}\ \textit{takes}\ V_{sup}>\ <N_{pred}\ \textit{credible steps}\ N_{pred}>\ \textit{toward}\ OP_{N_0 \rightarrow NS}\ [S\ N\ V_{(ing)}\ \textit{solving the problem}\ S]\ S)$

This performs simultaneously the following eight actions:

- the subject is enclosed into $[N$ and $N]$ brackets,
- the sequence *take credible steps toward* is recognized,
- *take* is marked as a support verb with $<V_{sup}$ and $V_{sup}>$,
- *credible steps* is marked as a predicative noun with $<N_{pred}$ and $N_{pred}>$,
- *solving* is recognized as a verb in *ing* form,
- *solving this problem* is marked as a sentence to be recognized while a subject marker N is added.
- The morphological marker V_{ing} is transformed into the marker $V_{(ing)}$ to signify that although the verb was originally in the *ing* form, it has to be considered as a simple conjugated verb during the rest of the analysis.

[10]Recall that, in practice, the transduction doesn't apply on the text directly but to the sequence of words and symbols representing the result of the morphological analysis.

- A marker $OP_{N_0 \to NS}$ is inserted to link the subject of the sentence with the subject of the *ing* clause, that is the subject of *solving*.

This should be interpreted in the context of transformation grammars in which the sentence is analyzed as the operator *John takes credible steps toward* applying to the sentence *John solves the problem*. The subject of the second sentence is deleted within this operation.

The second step of the analysis consists of analyzing *John* as a nominal and N $V_{(ing)}$ *solving the problem* as a sentence. The sentence structure *N solve N* should therefore be compiled into a transduction that takes into account both the possibility for the sentence to appear by itself, e.g. *John solves the problem*, or, as in our example, to appear as an *ing* clause. In order to handle both situations, the grammar should perform the following two mappings:

[*S a solve b S*] $\rightarrow$ (*S* [*N a N*] <V *solve* V> [*N b N*] *S*)
[*S N* $V_{(ing)}$ *solving b S*] $\rightarrow$ (*S N* $V_{(ing)}$ <V *solving* V> [*N b N*] *S*)

The application of the grammar to the sequence of (37), i.e. the second step of the analysis of (36), leads to the following sequence:

(38) (*S* (*N John N*) <V_{sup} *takes* V_{sup}> <N_{pred} *credible steps* N_{pred}> *toward* $OP_{N_0 \to NS}$ (*S N* $V_{(ing)}$ <V *solving* V> [*N the problem N*] *S*) *S*)

and finally to the analysis

(39) (*S* (*N John N*) <V_{sup} *takes* V_{sup}> <N_{pred} *credible steps* N_{pred}> *toward* $OP_{N_0 \to NS}$ (*S N* $V_{(ing)}$ <V *solving* V> (*N the problem N*) *S*) *S*)

5.5 Support Verb Recovery in Noun Clauses

Let us now consider a different sentence:

(40) [*S John's concessions to his friend were unexpected S*]

The difficulty is to analyze correctly the nominal *John's concessions to his friend*. Recall that the grammar should not contain a rule that says that the structure $N's N to N$ can always form a nominal, this would generate many incorrect analyses. In fact, this type of nominal is made possible here by the underlying support verb construction

(41) *John makes concessions to his friend*

and the analysis should therefore reduce the problem of analyzing the nominal into the analyzing this sentence. The first application of the transducer representing the grammar will transform the original sentence (40) into the following one:

(42) (*S* [*N John's concessions to his friend N*] <*V were V*> [ADJ *unexpected* ADJ] *S*)

Linguistically, a sentence, such as (41), of the shape[11]

(43) $N\ V_{sup}\ N_{pred}\ W$

can often be transformed into a nominal with the following shape:

(44) N's $N_{pred}\ W$

in which the support verb disappears. To cover this phenomenon, the following mapping, corresponding to the nominal structure (44) should be added to the grammar

[*N a* 's *concessions to b N*] $\rightarrow$ (*N* [*S a* V_{sup} ? *concessions to b S*] *N*)

The transduction representing the structure

(45) *N make concessions to N*

should also perform the two mappings

[*S a make concessions to b S*] $\rightarrow$
[N *a* N] $<V_{sup}$ *make* $V_{sup}> <N_{pred}$ concessions $N_{pred}>$ to [N b N] S)

[*S a* V_{sup} *make concessions to b S*] $\rightarrow$
(S [N *a* N] $V_{sup} <V_{sup}$ *make* $V_{sup}> <N_{pred}$ concessions $N_{pred}>$ to [N b N] S)

The first one handles sentences such as (41) and the second one validates the support verb construction hypothesis.

With these mappings, the analysis of (40) is performed as indicated on Figure 14. Here again, the underlying support verb sentence is explicitly recovered during parsing and can be extracted from the resulting analysis.

[11] W represents any number of arguments; the type and number of arguments depend on the predicative noun N_{pred}.

$$[S\ \textit{John's concessions to his friend were unexpected}\ S]$$
$$\downarrow$$
$$(S\ [N\ \textit{John's concessions to his friend}\ N]\ <V\ \textit{were}\ V>\ [\text{ADJ}\ \textit{unexpected}\ \text{ADJ}]S)$$
$$\downarrow$$
$$(S\ (N\ [S\ \textit{John}\ V_{sup}\ ?\ \textit{concessions to his friend}\ S]\ N)\ <V\ \textit{were}\ V>$$
$$(\text{ADJ}\ \textit{unexpected}\ \text{ADJ})\ S)$$
$$\downarrow$$
$$(S\ (N\ (S\ [N\ \textit{John}\ N]\ V_{sup}\ <V_{sup}\ \textit{make}\ V_{sup}>\ <N_{pred}\ \textit{concessions}\ N_{pred}>$$
$$\textit{to}\ [N\ \textit{his friend}\ N]\ S)\ N)\ <V\ \textit{were}\ V>\ (\text{ADJ}\ \textit{unexpected}\ \text{ADJ})\ S)$$
$$\downarrow$$
$$(S\ (N\ (S\ (NN\ \textit{John}\ N)\ V_{sup}\ <V_{sup}\ \textit{make}\ V_{sup}>\ <N_{pred}\ \textit{concessions}\ N_{pred}>$$
$$\textit{to}\ (N\ \textit{his friend}\ N)\ S)\ N)\ <V\ \textit{were}\ V>\ (\text{ADJ}\ \textit{unexpected}\ \text{ADJ})\ S)$$

Figure 14: Analysis of the sentence *John's concessions to his friend were unexpected*

5.6 Hidden Support Verb Constructions

Let us now consider the following more complex sentences:

(46) *John asked Peter for an immediate decision*
(47) *John asked Peter for his belongings*
(48) *John asked Peter for a ride*

which share the common surface sentence structure:

(49) N_0 *ask* N_1 *for* N_2

However, describing such sentences only with the surface structure is not sufficient. In fact, such description would not explain why the following sentence

(50) * *They asked Peter for their immediate decision*

is not accepted, contrary to

(51) *They asked Peter for his immediate decision*
(52) *They asked Peter for his belongings*
(53) *They asked Peter for their belongings*

The linguistic explanation is that sentences (46) and (51) are analyzed as transformations of

(54) *They asked Peter to make (a/his) decision immediately*

whereas (52) and (53) are analyzed as transformations of

(55) *They asked Peter to give them (his/their) belongings*

[20] shows that, in French, complex verbs such as *to ask*, should be described in conjunction with a list of support verbs. The support verbs such as *take* and *give* are then deleted by the constraints they impose remain. The situation is almost identical in English. Here for instance, the infinitive of (54) contains the sentence

(56) *Peter makes a decision immediately*

or equivalently[12],

(57) *Peter makes an immediate decision*

The verb *makes* disappears within the transformation. However it is still necessary to know which construction is really used. In fact, the support verb construction allows sentences

(58) *Peter makes his decision*

in which the word *his* has to refer to the subject of the construction. This explains why (50) is incorrect.

In order to take these properties into account the parsing of (46) works as indicated in the following trace:

$$[S\ \textit{John asks Peter for an immediate decision}\ S]$$
$$\downarrow$$
$$(S\ [N\ \textit{John}\ N]\ <\mathrm{V\ asks\ V}>\ [N\ \textit{Peter}\ N]\ \textit{for}\ [S\ N\ <V_{sup}\ (\textit{make/give})\ V_{sup}>$$
$$\textit{an immediate decision}\ S]\ S)$$
$$\downarrow$$
$$S\ (N\ \textit{John}\ N)\ <\mathrm{V\ asks\ V}>\ (N\ \textit{Peter}\ N)\ \textit{for}\ (S\ N\ <V_{sup}\ \textit{make}\ V_{sup}>$$
$$<N_{pred}\ \textit{an immediate decision}\ N_{pred}>\ S)\ S)$$

Here again, it is possible, within the parsing program, to introduce hypotheses about several support verb constructions. Some of these hypotheses are invalidated at a later stage of the analysis and the correct support verb is recovered.

[12]This transformation, first described by Z.S. Harris [11], is very common an is even on the criterion to identify a support verb construction (see also [5, 19]).

5.7 Frozen Expressions

A recurrent problem when recognizing support verb constructions or frozen expressions is that these sentences also have the surface form of a free expression. For instance, the sentence

(59) $[S\ John\ take\ (a/his)\ seat\ S]$

is going to be parsed in two different ways. First, it will be parsed as a free sentence (by resemblance to $[S\ John\ take\ a\ seat\ S]$) and the resulting analysis will be

(60) $(S\ (\mathrm{N}\ John\ \mathrm{N})\ <\mathrm{V}\ take\ \mathrm{V}>\ (\mathrm{N}\ a\ seat\ \mathrm{N})$

It will also be parsed as a support verb construction and, in that case, the analysis will be

(61) $(S\ (\mathrm{N}\ John\ \mathrm{N})\ <V_{sup}\ take\ V_{sup}>\ <N_{pred}\ a\ seat\ N_{pred}>$

Whereas both analysis are possible in principle, the frozen interpretation is usually the correct one. For the analysis to contain the frozen interpretation only, the parser not only has to produce the correct analysis, but it has to delete the incorrect one. In other words the transducer representing the structure *N take a seat* should perform the following two mappings:

$$[S\ a\ take\ a\ seat\ S] \rightarrow$$
$$(S\ [N\ a\ N]\ <V_{sup}\ take\ V_{sup}>\ <N_{pred}\ a\ seat\ N_{pred}>\ S)$$

$$(S\ (N\ a\ N)\ (V\ take\ V)\ (N\ a\ seat\ N)\ S) \rightarrow \emptyset$$

The first mapping handles the analysis per see whereas the second one deletes the improper analysis resulting from the free structure.

The parsing trace of the sentence is then the following:

$$[S\ John\ takes\ a\ seat\ S]$$
$$\downarrow$$
$$(S\ [N\ John\ N]\ <V\ takes\ V>\ [N\ a\ seat\ N]\ S)$$
$$(S\ [N\ John\ N]\ <V_{sup}\ takes\ V_{sup}>\ <N_{pred}\ a\ seat\ N_{pred}>\ S)$$
$$\downarrow$$
$$(S\ (N\ John\ N)\ <V\ takes\ V>\ (N\ a\ seat\ N)\ S)$$
$$(S\ (N\ John\ N)\ <V_{sup}\ takes\ V_{sup}>\ <N_{pred}\ a\ seat\ N_{pred}>\ S)$$
$$\downarrow$$
$$(S\ (N\ John\ N)\ <V_{sup}\ takes\ V_{sup}>\ <N_{pred}\ a\ seat\ N_{pred}>\ S)$$

In the gradation from free to frozen for the verb *take*, we should also consider sentences such as *The elected candidate takes his seat in the House.* This sentence should be interpreted neither as a free construction with an object complement *his seat* and an adverbial *in the House*, nor as the support verb construction *to take (a/his) seat* with the adverbial *in the House* but rather as the frozen expression *N take POSS seat in the House* in which only the subject may vary. For the parser to achieve this analysis, it should perform three mappings:

$$\begin{array}{c} [S\ \ a\ \ take\ \ his\ \ seat\ \ in\ \ the\ \ House\ \ S] \\ \downarrow \\ (S\ \ [N\ \ a\ \ N]\ <\mathrm{F}\ \ take\ \ his\ \ seat\ \ in\ \ the\ \ House\ \ \mathrm{F}>\ S) \end{array}$$

$$\begin{array}{c} (S\ \ (N\ \ a\ \ N)\ \ (V\ \ take\ \ V)\ \ (N\ \ a\ \ seat\ \ N)\ \ (\mathrm{ADV\ in\ the\ House\ ADV})\ \ S) \\ \downarrow \\ \emptyset \end{array}$$

$$\begin{array}{c} (S\ \ (N\ \ a\ \ N)\ <V_{sup}\ \ take\ \ V_{sup}>\ <N_{pred}\ \ a\ \ seat\ \ N_{pred}>\ (\mathrm{ADV\ in\ the\ House\ ADV})\ \ S) \\ \downarrow \\ \emptyset \end{array}$$

The first mapping performs the analysis per see, the second one deletes the free sentence analysis while the third one deletes the support verb construction analysis.

6 Finite-State Acceleration

In section 5, a sentence with a modal verb such as

(62) *John should read this book*

was analyzed with two different steps for *should* and *read.* It is also possible to think about *should read* as one single compound verb. This is not linguistically motivated but it might improve parsing efficiency. Such a composite verb could take two arguments: a noun phrase as subject and a noun phrase as direct object. From this point of view, the first of the analysis would be a mapping from

(63) [S *John should read this book* S]

to

(64) [S [N *John* N] <V *should read* V> [N *this book* N] S]

The advantage of such an approach obviously is that it shortens the number of analysis phases by one, at the cost of more space requirement for T_{dic}.

What this also realizes is that longer sequences, such as *should read* here, are recognized through a simple finite-state mechanism. This also follows the intuition that some sequences within natural language, such as auxiliary verb sequences, are very well modeled by a simple finite-state automaton. For instance, the set of auxiliary verb sequences could be modeled by the finite-state automaton of Figure 15. In this automaton, sequences of auxiliary verbs and verbs are represented with a slightly new convention. Sequences are elements of $(\Sigma_w \cdot \Sigma_{pps})^*$ in which Σ_w is the set of words in the language and Σ_{pps} is a finite set of part-of-speech tags. The tags used here are derived from those used in the Brown Corpus [4]. *vb* stands for *infinitive form of the verb*, *vbd* stands for the past tense, *vbg* stands for the progressive form, *vbn* stands for the passive form and *vbz* stands for the third person singular of the verb. The symbol *???* matches any other symbol. For instance the sequence

$$???\cdot vbz$$

stands for a verb conjugated at the third person singular.

From a computational efficiency point of view, it seems important to recognize these sequences through a pure finite-state mechanism rather than through more complex parsing procedures. In this section we will see that, by combining some of finite-state transducers derived from the syntactic dictionary, it is possible to obtain a pure finite-state recognition process for these specific sequences while continuing the whole analysis of the sentence. In some sense, this addresses the following remark: many sequences within natural language sentences seems to be finite-state, however, a pure finite-state modeling cannot model the whole syntax and more powerful, yet less efficient, formalism, such as context-free grammars, are required.

We now suppose that we have each finite-state transducer associated to each structure of the syntactic dictionary. In particular, we have $T_{N\_read\_N}$ that performs the following matching, among others:

$$\begin{array}{c}\text{[S [S } a \text{ N] <OP } c \text{ OP> v } \textit{read}\ b \text{ S]}\\ \downarrow \\ \text{(S [N } a \text{ N] <OP } c \text{ OP> <V a } \textit{read} \text{ V> [N } b \text{ N] S]}\end{array}$$

and $T_{N\_should\_V\_W}$ that performs the following matching:

$$\begin{array}{c}\text{[S } a\ \textit{should} \text{ v } \textit{read}\ b\ S]\\ \downarrow \\ \text{[S [N } a \text{ N] <OP } \textit{should} \text{ OP> v } b \text{ S]}\end{array}$$

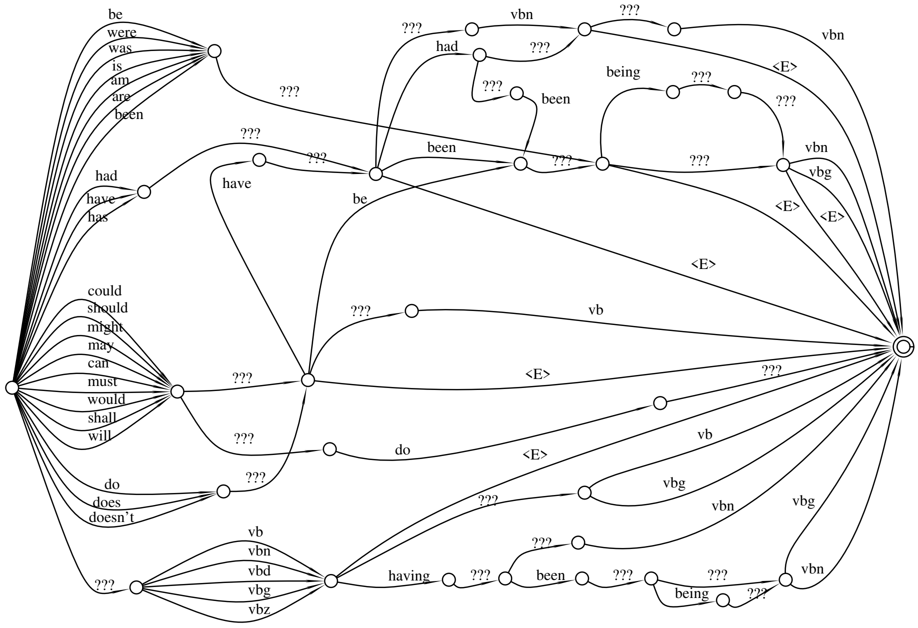


Figure 15: Finite-state automaton representing auxiliary verb sequences

and we saw in section 5 how these two mappings are combined together to analyze (63). Let us now compose $T_{N\_should\_V\_W}$ and $T_{N\_read\_N}$ into one transducer that we denote $T_{M\_should\_read\_N}$. This transducer performs the following mapping:

[S *a* *should* v *read* *b* S]
↓
(S [N *a* N] <OP *should* OP> <V v *read* V> [N *b* N] S)

This maps the input sentence (63) to the following sequence:

(65) (S [N *John* N] <OP *should* OP> <V v *read* V> [N *this book* N] S)

on which the analysis is continued as in section 5.

Formally, if $T_{dic_1} = T_{N\_should\_V\_W} \cup T_2$[13] then we can build a new finite-state transducer $T_{dic_2} = T_2 \cup (T_{N\_should\_V\_W} \circ T_2)$ for which the analysis of (63) goes as follows:

[13] T_2 represents the union of the finite-state transducers associated to each structure, expect the transducer associated to *should*

[S *a should* v *read b* S]
$\downarrow (T_{dic_2})$
(S [N *John* N] <OP *should* OP> <V v *read* V> [N *this book* N] S)
$\downarrow (T_{dic_2})$
(S (N *John* N) <OP *should* OP> <V v *read* V> (N *this book* N) S)

By using T_{dic_2} instead of T_{dic}, we achieve our goal of recognizing the sequence *should read* through a pure finite-state process while obtaining the exact same analysis: *should* is still analyzed as an modal operator on the whole sentence and *read* is also still analyzed as the main verb. Formally, T_{dic_2} has the property that:

$$T^{\infty}_{Dic_2} = T^{\infty}_{dic}$$

which guaranties that the language recognized and the analysis are identical for T_{dic} and T_{dic_2}. The cost of such speed improvement however, is that T_{dic_2} will take more space than T_{dic}.

7 A Transducer Parser for Tree-Adjoining Grammars

In this section we will see on a formal example that the finite-state transducer framework can be used to parse tree-adjoining grammars. Tree-adjoining grammars (TAGs) [14, 13] is a formalism in which elementary trees are combined through an operation of adjunction. This formalism allows many linguistic properties to be encoded in a natural way [16]. We will not develop here linguistic examples but rather illustrate the transducer approach on a formal language typical of tree adjoining grammars.

Let us consider the following tree adjoining grammar:

$$G_1 = (\{a, b, c, d, e\}, \{S\}, \{\alpha_1\}, \{\alpha_2\}, S)$$

in which $\{a, b, c, d, e\}$ is the alphabet of terminal symbols, $\{S\}$ is the alphabet of non terminal symbols, $\{\alpha_1\}$ is the set of initial trees (α_1 is represented on Figure 16, left), $\{\alpha_2\}$ is the set of auxiliary trees (α_2 is represented on Figure 16, right) and S is the root. This grammar generates the language

$$L_1 = \{a^n b^n e c^n d^n | n \geq 1\}$$

[25]. This language is not context-free, that is, there exists no context-free grammar G such that $L(G) = L_1$. L_1 is generated through a mechanism we now describe informally.

The main composition operation of TAGs is called *adjoining* or *adjunction*; it builds a new tree from an auxiliary tree, α_2 here, and any other tree. For instance, because α_1 contains a node labeled S and the root node of α_2 is also labeled S, it is possible to make the adjunction of α_2 on α_1. The resulting tree, β_1 of Figure 17, obtained by adjoining α_2 to α_1 at the root node root_{α_1} of α_1 is built as follows:

- the sub-tree of α_1 dominated by root_{α_1}, called t, is excised, leaving a copy of root_{α_1},
- the auxiliary tree α_2 is attached at the copy of root_{α_1} and its root node is identified with the copy of root_{α_1},
- the sub-tree t is attached to the foot node of α_2 and the root node of t, i.e. root_{α_1}, is identified with the foot node of α_2

In addition, when the special symbol *NA* is used to label a node, no adjunction is allowed on this node. For instance, for β_1, the adjunction can take place only at the medium node S and therefore β_2 is the only tree that can be derived from β_1 through a single adjunction. The language defined by a set of trees is built by taking the sequences generated by the leaves of each tree. For instance α_1, β_1 and β_2 respectively generate the strings e, *abecd* and *aabbeccdd*.

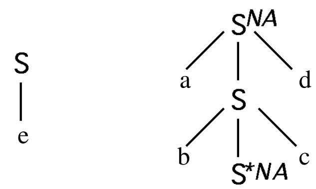


Figure 16: *left:* α_1, *right:* α_2

We can consider, as we did for context-free grammars in Section 3, that the grammar, is a syntactic dictionary. Each entry in this dictionary is a tree of the grammar and each of these trees will be converted into a transducer. So, as for context-free grammars, the core of transducer parsing for tree adjoining grammars consists of building a dictionary of finite-state transducers. The transducer representing the union of each individual transducer will then be used to parse input sentences.

Formally, we can define two general mappings[14] TAG-FST$_{ini}$ and TAG-FST$_{aux}$. TAG-FST$_{ini}$ will take any initial tree as input and will build a finite-state transducer associated to this tree; if γ is an initial tree, TAG-FST$_{ini}(\gamma)$ will denote

[14]Not detailed here.

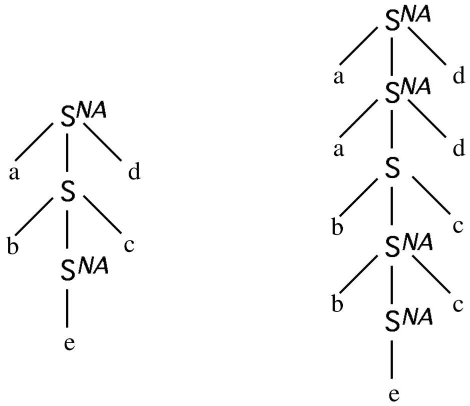


Figure 17: *left:* β_1, *right:* β_2

the transducer associated to γ. Similarly, TAG-FST$_{aux}$ will convert any auxiliary tree into a finite-state transducer; if ω is an auxiliary tree, we denote by TAG-FST$_{aux}(\omega)$ the finite-state transducer associated to ω. A tree adjoining grammar being given, the transducer representing the whole grammar will be built by taking each initial tree and each auxiliary tree and by converting each one into a finite-state transducer. The final transducer is then the union of all individual transducers derived that way. In other words, the transducer T_G associated to a grammar G is defined by:

$$T_G = \bigcup_{\text{tree}_i \in ini(G)} \text{TAG-FST}_{ini}(\text{tree}_i) \cup \bigcup_{\text{tree}_i \in aux(G)} \text{TAG-FST}_{aux}(\text{tree}_i)$$

in which $ini(G)$ and $aux(G)$ respectively denote the set of initial and auxiliary tree of G. At this point the parser of a tree adjoining grammar G can be defined by

$$\text{parser} = T_G^{\infty}$$

meaning, as for context-free grammars, that the syntactic analysis of a sequence w of the language $L(G)$ generated by G is the fixed-point of $(T_G^n(w))_{n\geq 0}$. Therefore, in practice, the parsing process consists of applying the transducer T_G to the input sequence, then to the result of the first application and continue until the automaton representing the set of sequences stays unmodified.

To illustrate this, consider again the grammar G_1. In this grammar, only two transducers have to be built: (1) the transducer $T_{\alpha_1} = \text{TAG-FST}_{ini}(\alpha_1)$ representing the initial tree α_1 and (2) the transducer $T_{\alpha_2} = \text{TAG-FST}_{aux}(\alpha_2)$ representing the auxiliary tree α_2.

Figure 18 illustrates the parsing of the input string *aabbeccdd*. As for context-free grammars, the input of the transducer parsing will be the string to be parsed enclosed into the markers $[S$ and $S]$. Here, this will be the sequence $[S$ *aabbeccdd* $S]$ and it will be represented by a finite-state automaton (represented at the top of Figure 18) for more efficiency.

The transducer T_{α_1} representing the tree α_1 is defined functionally as follows:

$$T_{\alpha_1}\colon\ w_1\ [\mathrm{S}\ w_2\ e\ w_3\ \mathrm{S}]\ w_4 \longrightarrow \begin{array}{l} (1)\ w_1\ [[\mathrm{S}\ w_2\ \{\mathrm{S}\ e\ \mathrm{S}\}\ w_3\ \mathrm{S}]]\ w_4 \text{ if } w_2 \neq \epsilon \text{ and } w_3 \neq \epsilon \\ (2)\ w_1\ (\mathrm{S}\ e\ \mathrm{S})\ w_4 \text{ if } w_2 = w_3 = \epsilon. \\ (3)\ \emptyset \text{ otherwise} \end{array}$$

in which w_1, w_2, w_3 and w_4 are elements of Σ^* in which $\Sigma = \{a, b, c, d, e\}$ is a finite alphabet. This transducer takes sequences in which the markers $[S$ and $S]$ are used, it has three different types of output depending on the shape of the input string: (1) if the string enclosed between the markers $[S$ and $S]$ is of the shape $w_2 e w_3$ in which both w_2 and w_3 are not the empty string then the markers $[S$ and $S]$ are transformed into $[[S$ and $S]]$ while two markers $\{S$ and $S\}$ are inserted around e. This transformation indicates that the string should be regarded as the tree α_1 on which some adjunction has been made. The markers $[[S$, $S]]$, $\{S$ and $S\}$ indicate the position of the possible adjunction within the input string. (2) If the string is such that the sequence enclosed within $[S$ and $S]$ is the simple character e then the $[S$ and $S]$ are respectively transformed into $(S$ and $S)$ to indicate that the input sequence was recognized as the sequence of leaves of α_1, i.e. the symbol e. (3) If the input string is not of the previous two shapes then the input sequence cannot be parsed with the tree α_1.

The transducer T_{α_2} representing the tree α_2 is defined functionally as follows:

$$w_1\ [[\mathrm{S}\ a\ w_2\ b\ \{\mathrm{S}\ w_3\ \mathrm{S}\}\ w_4\ d\ \mathrm{S}]]\ w_5 \longrightarrow \begin{array}{l} (1)\ w_1\ (\mathrm{S}\ a\ [[\mathrm{S}\ w_2\ \{\mathrm{S}\ b\ (\mathrm{S}\ w_3\ \mathrm{S})\ c\ \mathrm{S}\}\ w_4\ \mathrm{S}]]\ d\ \mathrm{S})\ w_5 \\ \quad \text{if } w_1, w_5, w_3 \in (\Sigma \cup \{(S, S)\})^* \text{ and} \\ \quad w_2, w_4 \in \Sigma^+ \\ \\ (2)\ w_1\ (\mathrm{S}\ a\ (\mathrm{S}\ \mathrm{b}\ (\mathrm{S}\ w_3\ \mathrm{S})\ c\ \mathrm{S})\ d\ \mathrm{S})\ w_5 \\ \quad \text{if } w_1, w_5, w_3 \in (\Sigma \cup \{(S, S)\})^* \text{ and } w_2 = w_4 = \epsilon \\ \\ (3)\ \emptyset \text{ otherwise} \end{array}$$

in which w_1 to w_5 are strings of Σ^*. The input string of such a transduction contains the markers $[[S$, $S]]$, $\{S$ and $S\}$, meaning that the sequence has been parsed with an adjunction. A sequence of the shape

$$w_1\ [[\mathrm{S}\ a\ w_2\ b\ \{\mathrm{S}\ w_3\ \mathrm{S}\}\ w_4\ d\ \mathrm{S}]]\ w_5$$

indicates that w_1, w_3 and w_5 have been recognized and that the part appearing between $[[S$ and $\{S$, on one hand, and between $S\}$ and $S]]$, on the other hand, should be analyzed as the adjunction of one or several auxiliary trees. The transduction can generate three different types of output depending on the shape of the input. (1) If w_2 and w_4 are not the empty string then the tree α_2 has been adjoined and at least one other adjunction has also taken place, therefore, the transducer converts the symbols $[[S$, $S]]$, $\{S$ and $S\}$ to indicate that an adjunction of α_2 has been recognized. At the same time, new symbols $[[S$, $S]]$, $\{S$ and $S\}$ are introduced to indicate the possibility for another adjunction. (2) If the sequence of symbols to be analyzed with an adjunction has exactly the shape $a \cdot b \cdot c \cdot d$ then the markers $[[S$, $S]]$, $\{S$ and $S\}$ are replaced by the markers $(S$ and $S)$ to indicate that an adjunction of α_2 has been recognized and that the analysis is completed. (3) the transducer outputs the empty set in all other circumstances.

The process is best illustrated by following the analysis of a simple example. The transducer associated to G_1 is defined by:

$$T_{G_1} = T_{\alpha_1} \cup T_{\alpha_2}$$

T_{G_1} is represented on Figure 19. In addition to the convention used previously, a transition labeled by A/A stands for a set of transitions labeled respectively by a/a, b/b, c/c, d/d and e/e. Recall also that if a state q has an outgoing transition labeled ?/?, this transition stands for all the pairs s/s such that there is no other outgoing transition from q whose input label is s.

Let us now come back to the analysis of the sequence *aabbeccdd* illustrated on Figure 18. The first input of T_{G_1} is the singleton

$$\{[S\ aabbeccdd\ S]\}$$

represented by the top finite-state automaton of Figure 18. The markers $[S$ and $S]$ indicate that the enclosed sequence has to be analyzed as a sentence. The first application of T_{G_1} to this input automaton leads to the second automaton from the top, this automaton represents the singleton

$$\{[[S\,aabb\{S\,eS\}\,ccdd\ S]]\}$$

The only part of T_{G_1} which leads to a non empty output comes from T_{α_1}. The resulting automaton is pruned before applying the transducer T_{G_1} a second time. The second application of T_{G_1} leads to the third automaton from the top, this automaton represents the singleton

$$\{(S\,a[[S\,ab\{S\,b(S\,eS)\,cS\}\,cdS]]\,dS)\}$$

The part of T_{G_1} which leads to a non empty string corresponds this time to T_{α_2}. This automaton is pruned and the transducer is applied again which leads to the bottom automaton which represents the singleton

$$\{(S\,a(S\,a(S\,b(S\,b(S\,eS)\,cS)\,cS)\,dS)\,dS)\}$$

Here again, the active part of T_{G_1} corresponds to T_{α_2}. Finally, this last automaton is pruned. Since, another application of T_{G_1} does not modify the input, this last automaton is a fixed-point of T_{G_1} and therefore the result of the analysis of $[S\ aabbeccdd\ S]$. Note that this last automaton represents the tree β_1 of Figure 17.

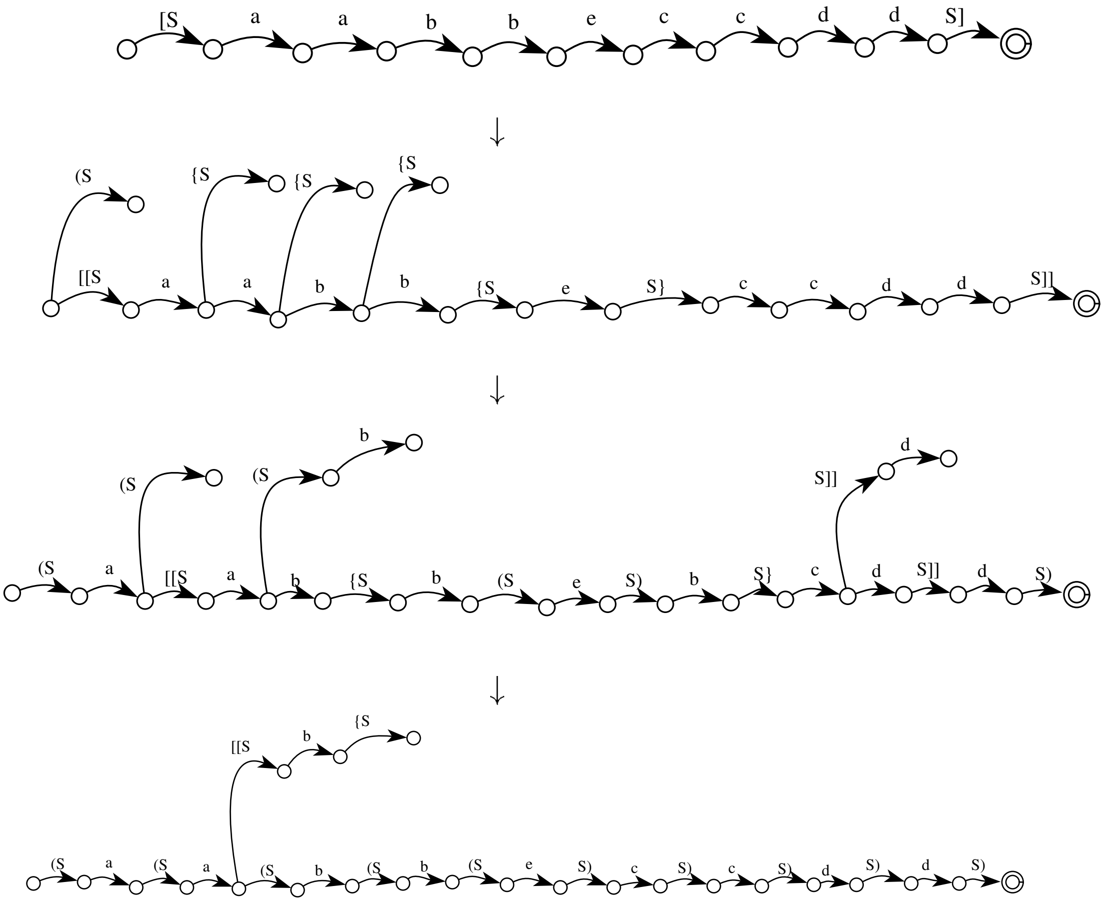


Figure 18: Example of Analysis.

This short example is only an illustration of the flexibility of transducer parsing and the proof of the correctness of the previous example is left to

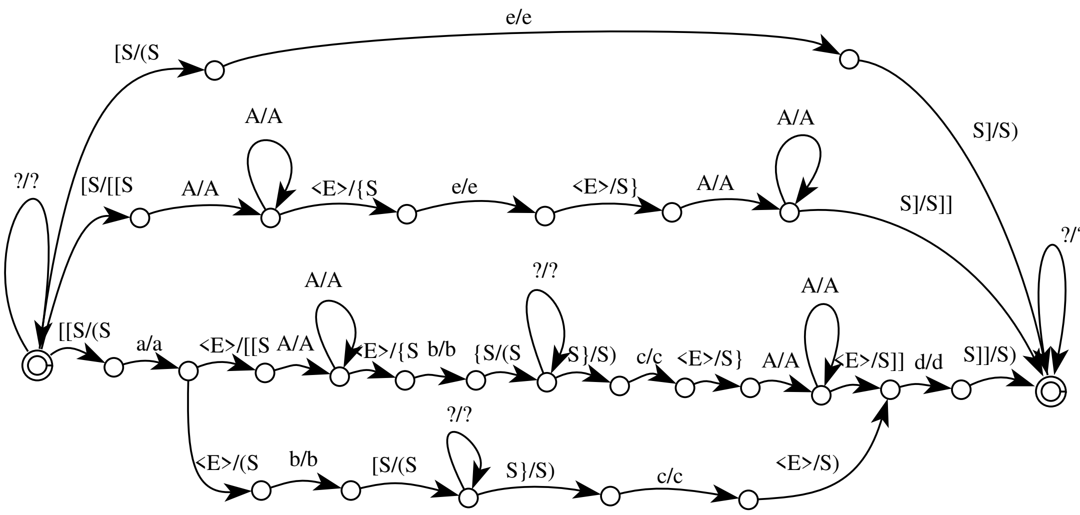


Figure 19: T_{G_1}

the reader[15]. The practical efficiency of transducer parsing for tree adjoining grammar can only be evaluated by comparing this parsing method with others (in particular with the algorithms described in [25]) on realistic data (both input strings and grammars). Such experiments should be done on very large lexical grammars.

[15] We also leave to the reader the interesting exercise of finding the class of tree adjoining grammars that can be analyzed that way.

References

[1] Alain Guillet Boons, Jean-Paul and Christian Leclere. *La structure des phrases simples en français, Constructions Intransitives.* Librairie Droz, Geneve-Paris, 1976.

[2] David Clemenceau and Emmanuel Roche. Enhancing a large scale dictionary with a two-level system. In *EACL-93, proceedings of the conference*, 1993.

[3] Blandine Courtois. Delas: Dictionnaire electronique du ladl pour les mots simples du français. Technical report, Université Paris 7, 1989.

[4] W. Nelson Francis and Henry Kučera. *Frequency Analysis of English Usage.* Houghton Mifflin, Boston, 1982.

[5] Jacqueline Giry-Schneider. *Les preédicats nominaux en français, les phrases simples à verb support.* Droz, Genève, Paris, 1978.

[6] Maurice Gross. *Grammaire transformationnelle du Français,1. Syntaxe du verbe.* Cantilène, 1968.

[7] Maurice Gross. *Méthodes en syntaxe,régime des constructions complétives.* Hermann, 1975.

[8] Maurice Gross. *Grammaire transformationnelle du Français,3. Syntaxe de l'adverbe.* Cantilène, 1986.

[9] Maurice Gross. Les limites de la phrase figée. *Langages*, (90):7–22, June 1988.

[10] Maurice Gross. The construction of local grammars. In *this volume*. 1996.

[11] Zellig Harris. *Notes du cours de syntaxe.* Seuil, Paris, 1976.

[12] Zellig Harris. *Theory of Language and Information.* Oxford University Press, 1991.

[13] Aravind K. Joshi. How muc context-sensitivity is necessary for characterizing structural descriptions - tree adjoining grammars. In D. Dowty, L. Karttunen, and A. Zwicky, editors, *Natural Language Processing - Theortical, Computational and Psychological Perspectives.* Cambridge University Press, 1985.

[14] Aravind K. Joshi, L. S. Levy, and M. Takahashi. Tree adjuct grammars. *Journal of Computer and System Sciences*, 10(1), 1975.

[15] Lauri Karttunen, Ronald M. Kaplan, and Annie Zaenen. Two-level morphology with composition. In *Proceedings of the* 14^{th} *International Conference on Computational Linguistics (COLING'92)*, 1992.

[16] Anthony Kroch and Aravind K. Joshi. Linguistic relevance of tree adjoining grammars. Technical Report MS-CIS-85-18, Department of Computer and Information Science, University of Pennsylvania, April 1985.

[17] Eric Laporte. Phonétique et transducteurs. Technical report, Université Paris 7, 1993.

[18] Eric Laporte. Rational transductions for phonetic conversion and phonolgy. In *this volume*. 1996.

[19] Annie Meunier. *Nominalisations d'adjectifs par verbs supports*. PhD thesis, Université Paris 7, 1981.

[20] Mehryar Mohri. *Analyse et Représentation par Automates de Structures Syntaxiques Composees*. PhD thesis, Université Paris 7, January 1993.

[21] Fernando Pereira and Rebecca N. Wright. Finite state approximatnion of phrase structure grammars. In *this volume*. 1996.

[22] Fernando C. N. Pereira, Michael Riley, and Richard W. Sproat. Weighted rational transductions and their application to human language processsing. In *ARPA Workshop on Human Language Technology*. Morgan Kaufmann, 1994.

[23] Emmanuel Roche. *Analyse Syntaxique Transformationelle du Français par Transducteurs et Lexique-Grammaire*. PhD thesis, Université Paris 7, January 1993.

[24] Emmanuel Roche. Smaller representations for finite-state transducers. In *Lecture Notes in Computer Science, Combinatorial Pattern Matching, Fifth Annual Symposium, Helsinki, Finland, Proceedings*, 1995.

[25] Yves Schabes. *Mathematical and Computatnional Aspects of Lexicalized Grammars, Processing with Lexicalized Tree-Adjoining Grammars*. PhD thesis, University of Pennsylvania, 1991.

[26] Max Silberztein. *Dictionnaires Electroniques et Analyse Lexicale du Français— Le Système INTEX*. Masson, 1993.

[27] Max Silberztein. The lexical analysis of natural languages. In *this volume*. 1996.