Two Parsing Algorithms by Means of Finite State Transducers

Emmanuel Roche*
Mitsubishi Electric Research Laboratories
201, Broadway, Cambridge, MA 02139, roche@merl.com

Abstract

We present a new approach, illustrated by two algorithms, for parsing not only Finite State Grammars but also Context Free Grammars and their extension, by means of finite state machines. The basis is the computation of a fixed point of a finite-state function, i.e. a finite-state transducer. Using these techniques, we have built a program that parses French sentences with a grammar of more than 200,000 lexical rules with a typical response time of less than a second. The first algorithm computes a fixed point of a non-deterministic finite-state transducer and the second computes a fixed point of a deterministic bidirectional device called a bimachine. These two algorithms point out a new connection between the theory of parsing and the theory of representation of rational transductions.

INTRODUCTION

Finite state devices have recently attracted a lot of interest in computational linguistics. Computational efficiency has been drastically improved for morphological analysis by representing large dictionaries with Finite State Automata (FSA) and by representing two-level rules and lexical information with finite-state transducers [8, 4] More recently, [11] has achieved parsing with low level lexical sensitivity by means of finite state automata. Finite state approximation of context-free grammars also proved both useful and efficient for certain application [9].

One common motivation of all this work is to improve efficiency dramatically, both in terms of time and space. These results often provide programs orders of magnitude faster than more traditional implementations. Moreover, FSAs are a natural way to express lexical sensitivity, which has always been a requirement in morphology and which has proved crucial in syntax. The grammar we used for French, called Lexicon-Grammar (see [6] [7] [2] [3] [10] for instance), pushes the lexicalization very far and it is our belief that this lexicalization trend will amplify itself and that it will result in grammars several orders of magnitude larger than today's representations. This uncovers the need for new methods that will be able to handle such large scale grammars.

However, a main drawback of the finite state approach to syntax is the difficulty of representing hierarchical data; this partly explains why FSA-based programs only do incomplete parsing. This paper presents a new parsing approach based on finite-state transducers, a device that has been used already in morphology [8] but not yet in syntax, that provides both hierarchical representations and efficiency in a simple and natural way. The representation is very compact, this allows to implement large lexical grammars.

Two new parsing algorithms illustrate the approach presented here. The first one uses a finite state transducer and computes a fixed point. But finite state transducers, unlike FSAs, cannot be made deterministic; however, a bidirectional device called a bimachine [1] can indirectly make them deterministic. This leads to the second algorithm presented here. The very high efficiency of this approach can be seen in the experiments on French. Sentences can be parsed with a grammar containing more than 200,000 lexical rules[1]; this grammar is, we think, the largest grammar ever implemented.

PRINCIPLES

The concept of Finite-State Transducer

The basic concept here, since we not only match but also add markers, is the concept of finite-state transducer. This device has already proved very efficient in morphological analysis [8]. It can deal with very large amount of data, namely morphological dictionaries containing more than 500,000 entries.

A finite state transducer is simply an FSA except that, while following a path, symbols are emitted. A finite state transducer can also simply be seen as a graph where the vertices, called states, are linked through oriented arrows, called transitions. The transitions are labeled by pairs (*input_label*, *output_label*) [2].

*Supported by a DRET-Ecole Polytechnique contract, this work was done at the Institut Gaspard Monge and at the LADL.

[1] By lexical rule we basically mean a sentence structure, as for example *Nhum say to Nhum that S*, where *Nhum* and *S* respectively stand for human nominal and sentence. Thus the rules we deal with can roughly be seen as sentence structures where at least one element is lexical. This will be developed in section .

[2] An extensive description of this concept can be found in [1].

The parser in term of rational transduction

In our parser, the grammar is a rational transduction f, represented by a transducer T. The input of the parser is the set s_0 containing as only element the input sentence bounded by the phrase marker $[P]$, i.e. $s_0 = \{[P]\ sentence\ [P]\}$. The analysis consists in computing $s_1 = f(s_0)$, $s_2 = f(s_1)$ until a fixed point is reached, i.e. $s_p = f(s_p)$. The set s_p contains trees represented by bracketed strings, this set is the set of grammatical analysis of the sentence, it contains more than one element in the case of syntactically ambiguous inputs. Each set s_i is represented by a Directed Acyclic Graph (DAG) A_i, thus the computation consists in applying the transducer T on the DAGs A_i. We shall write it $A_{i+1} = T(A_i)$.

In the next section we give two complete examples of that.

TWO SIMPLE EXAMPLES

An example of a Top-Down analysis

The graph on figure 1 describes the analysis of the sentence:

$$s_1 = \textit{John said that Mary left}$$

The graph on this figure has to be read in the following way: the input sentence is represented by the DAG A_1on the upper left corner; the subset of the grammar required for the analysis of this sentence is the transducer $f$on the right hand side of the figure 1.

The analysis is then computed in the following way: we apply the transducer f to A_1, that is we compute $A_2= f(A_1)$, this represents one step of a Top-Down analysis of the sentence. The box with a star inside represents this operation, namely applying a transducer to a DAG. If we then apply f to this result (i.e. A_2), we obtain $A_3=f(A_2)= f^2(A_1)$ represented under A_2. If this operation is applied once more, one gets $A_4=f(A_3)= f^3(A_1)$. This last result, A_4, is a fixed point of the transducer f, i.e. $f(A_4)=A_4$. A_4 is a DAG that represents a finite set $Set(A_4)$ of strings. Here, this set only contains one element, namely $Set(A_4) = \{(John)N0(said)V0(that(Mary)N0(left)V0)ThatS\}$. Each element is a bracketed representation of an analysis. Here the analysis is unique.

An example of a simultaneous Top-Down Bottom-Up analysis

The previous example might give the impression that computing a fixed point of a transducer automatically leads to simulating a top-down context free analysis. However, we shall now see that using the flexibility of manipulating transducers, namely being able to compute the composition and the union of two transducers, allows a context sensitive parsing which is simultaneously Top-Down and Bottom-up with the possibility of choosing which kind of rule should be parsed Bottom-Up.

Suppose one wants to analyze the sentence s_2 =*Max bought a little bit more than five hundred share certificates.* Suppose one has the following small functions, each one being specialized in the analysis of an atomic fact (i.e. each function is a lexical rule):

- f_1 : w a little bit more than $w' \longrightarrow w$ (**pred** a little bit more than **pred**) w'; $w, w' \in A^*$
- f_2[3] : w five hundred $w' \longrightarrow w$ (**num** five hundred **num**) w'

  where $w \in A^*$ and $w' \in A^* - \{NUMERAL\}$
- f_3 : w share certificates $w' \longrightarrow w$ (**cn** share certificates **cn**) w' where $w, w' \in A^*$
- f_4 : [P] w bought w'[P] $\longrightarrow$ [N w N] bought [N w' N] where $w, w' \in A^*$
- f_5 : w [N Max N] $w' \longrightarrow w$ Max w' ; $w, w' \in A^*$
- f_6 : w_1 [N (**pred** w_2 **pred**) (**num** w_3 **num**) (**cn** w_4 **cn**) N] $w_5 \longrightarrow w_1$ (N w_2 w_3 w_4 N) w_5

  where $w_1, w_2, w_3, w_4, w_5 \in A^*$
- $f_7 : w \longrightarrow w;\ w \in A^* - (Dom(f_1 \cup f_2 \cup f_3 \cup f_4 \cup f_5)$[4]

If we precompute the transducer representing the rational transduction $f = (f_4 \circ f_3 \circ f_2 \circ f_1) \cup (f_5 \circ f_6) \cup f_7$ then the analysis of the sentence is a two-step application of f, namely

> f([P] Max bought a little bit more than five hundred share certificates [P]) =
>
> [N Max N] bought [N (**pred** a little bit more than **pred**) (**num** five hundred **num**) (**cn** share certificates **cn**) N]

and

> f^2([P]s[P]) =
>
> (N Max N) bought (N a little bit more than five hundred share certificates N)

which is the analysis of the sentence[5].

FORMAL DESCRIPTION

The algorithm

Formally, a transducer T is defined by a 6-uplet (A, Q, i, F, d, δ) where A is a finite alphabet, Q is a finite set of states, $i \in Q$ is the initial state, $F \subset Q$ is the set of terminal states, d the transition function maps $Q \times A$ to the set of subsets of Q and δ the emission function maps $Q \times A \times Q$ to A.

The core of the procedure consists in applying a transducer to a FSA, the algorithm is well known, we give it here for the sake of readability.

is_fixed_point=ApplyTransducer(A, T_1, A_2)
$i = 0; P[0] = (i_1, i_2); n = 1; q = 0; is\_fixed\_point = YES;$
do {
  $(x_1, x_2) = P[q]$)
  if $x_1 \neq i_1$ then $is\_fixed\_point = NO;$[6]
  if $x_1 \in F_1$ and $x_2 \in F_2$ then $x \in F$;

[3] Here f_2 simulates a context sensitive analysis because of $w' \in A^* - \{NUMERAL\}$

[4] $Dom(f)$ stands for the domain of f.

[5] Note that it is always possible to keep more information along the analysis and to keep track, for instance, of the position of the determiners.

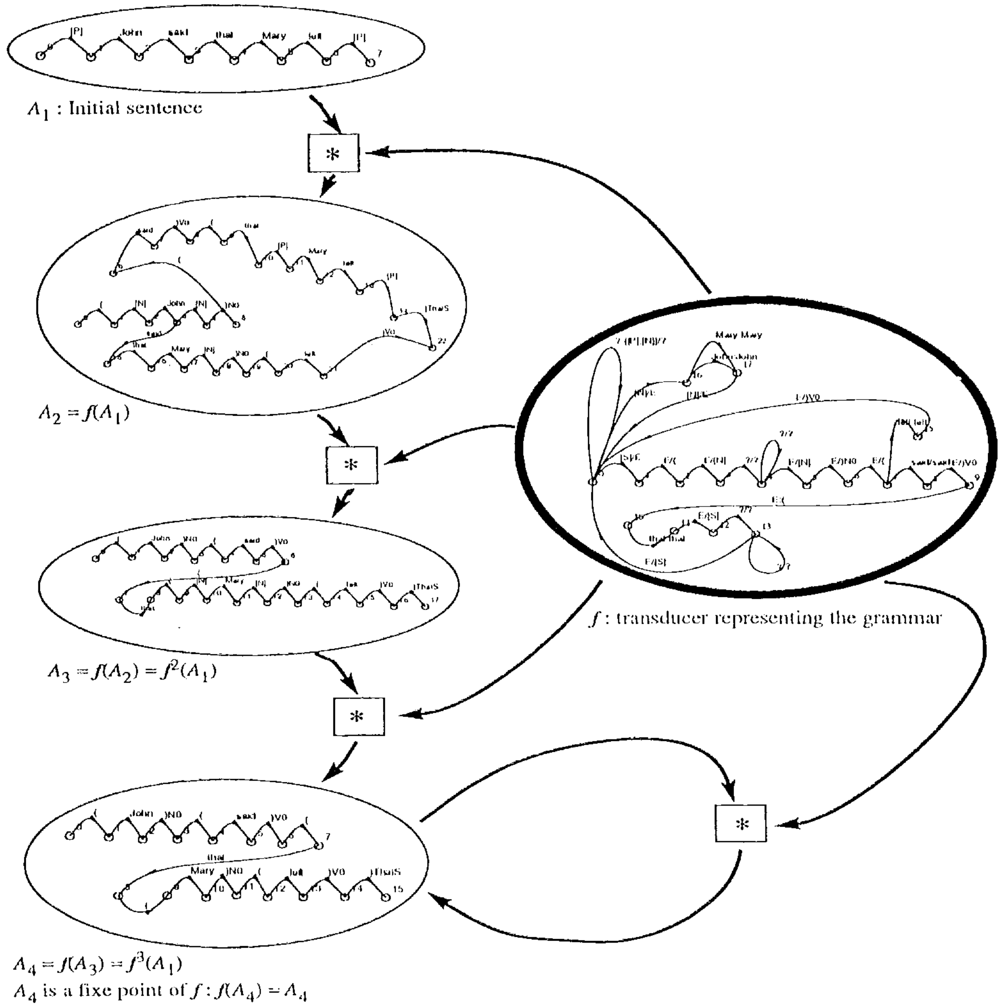

Figure 1: Overview of the analysis of the sample

```
foreach s ∈ Alph | d1(x1, s) ≠ ∅ ,d2(x2, s) ≠ ∅
  foreach y1 ∈ d1(x1, s) and y2 ∈ d2(x2, s)
    if∃ p < n such that P[p] == (y1, y2) then
        e = p;
    elseP[e = n + +] = (y1, y2);
    add e to d(q, δ1(x1, s, x2));
q++;
} while (q < n);
PRUNE(A); (this line is optional)
return is_fixed_point;
```

The analysis algorithm is then the following one:

```
ANALYSE_1(A, T)
    fin = NO;
    while fin ≠ YES do
        fin = ApplyTransducer(A, T, A);
```

Transducers v.s. Context Free Grammars

It should be pointed out that, given a Context-Free Grammar, it is always possible to build a transducer such that this method applies. In other words, any context free grammar can be translated into a transducer such that the algorithm parse the language described by this grammar. Moreover, the operation that transforms a CFG into its related transducer is itself a rational transduction. Although this cannot be developped here due to the lack of place, this result comes naturally when looking at the example of section 3.1.

Moreover the method has much more expressive power than CFG, in fact computing a fixed point of a rational transduction has the same power as applying a Turing Machine (althought there might not be any practical interest for that).

THE SECOND ALGORITHM : A DETERMINISTIC DEVICE

Given a transducer representing the grammar there are two different ways of obtaining new parsing programs. The first solution is to build a transducer T' equivalent to T from the view point of their fixed points,

$T \sim_{fixed-point} T'$. Namely $T \sim_{fixed-point} T'$ iff for each $x \in A^*$, $T(x) = x \Leftrightarrow T'(x) = x$. For instance, if T is such that for each $x \in A^*$, $T^n(x)$ converges then $T^2 \sim_{fixed-point} T$. The second approach is to try using a different representation of T or to apply it differently. In this section, we shall give an algorithm illustrating this second approach. The basic idea is to transform the finite-state transducer into a deterministic device called bimachine [1]. We will detail that latter but, basically, a bimachine stands for a left sequential function (i.e deterministic from left to right) composed to a right sequential function (i.e. deterministic from right to left). Such a decomposition always exists.

The interest of this concept appears when one looks at how the algorithm ApplyTransducer performs. In fact the output DAG of this algorithm has a lot of states that lead to nothing, i.e. states that are not coaccessible, thus the PRUNE function (called on line 14 of the ApplyTransducer function) has to remove most of the states (around 90% in our parser of French).

Let us for instance consider the following example: suppose the transducer T_a is the one represented figure 2 and that we want to compute $T_a(A)$ where A is the DAG given figure 2.

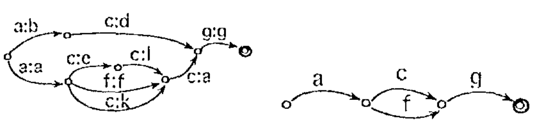

Figure 2: *left*: initial transducer; *right*: initial DAG

Following the algorithm described in ApplyTransducer up to line 14 excluded provides the DAG A' of figure 3.

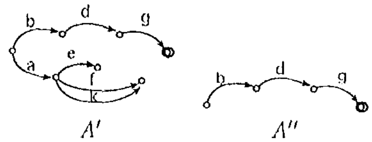

Figure 3: *left*: before pruning; *right*:after pruning

The PRUNE function has then to remove 3 of the six states to give the DAG A'' of figure 3

A way to avoid the overhead of computing unnecessary states is to first apply a left sequential transducer T_{aa}, (that is a transducer deterministic in term of its input when read from left to right) given figure 4 and then apply a right sequential transducer T_{ab} (i.e. deterministic in term of its input when read from right to left) given figure 4. We shall call the pair $B_a = (T_{aa}, T_{ab})$ the bimachine functionally equivalent to T_a, i.e. $B_a \sim_{function} T_a$. With the same input A we first obtain $A_a = T_{aa}(A)$ of figure 5 and then $A_b = A'' = reverse(T_{ab}(reverse(A_a))) = T(A) = B_a(A)$.

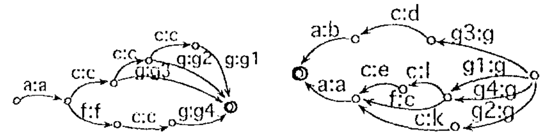

Figure 4: *left*:left sequential function; *right*:right sequential function

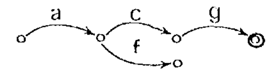

Figure 5: A_a

It should be pointed out that both T_{aa} and T_{ab} are deterministic in term of their input, i.e. their left labels, which was not the case to T_a. Just like for FSA, the fact that it is deterministic implies that it can be applied faster (and sometime much faster) than non-deterministic devices, on the other hand the size of the bimachine might be, in the worst case, exponential in term of the original transducer. The following algorithm formalizes the analysis by mean of a bimachine[7].

```
ANALYSE_2(A, B = (T1, T2))
        fin = NO;
        while fin ≠ YES do {
                fin = ApplyTransducer(A, T1, A);
                if fin≠ YES{
                        reverse(A);
                        ApplyTransducer(A, T2, A);
                        reverse(A);
                }
        }
```

IMPLEMENTATION AND RESULTS

The main motivation for this work comes from the linguistic claim that the syntactic rules, roughly the sentence structures, are mostly lexical. The grammar of French we had at our disposal was so large that none of the available parsers could handle it.

Although the implemented part of the grammar is still incomplete, it already describes 2,878 sentential verbs (coming from [6]), that is verbs that can take a sentence as argument, leading to 201,723 lexical rules[8]; 1,359 intransitive verbs [2] leading to 3,153 lexical rules; 2,109 transitive verbs [3] leading to 9,785 lexical rules; 2,920 frozen expression (coming from [7]) leading to 9,342 lexical rules and 1,213 partly frozen adverbials leading to 5,032 lexical rules. Thus, the grammar describes 10,479 entries and 229,035 lexical rules. This

[7] The FSA $reverse(A)$ is A where the transitions have been reversed and the initial and final states exchanged.

[8] For a verb like *étonner* the set of rules include $N hum_0$ *étonner* $N hum_1$ as well as $N hum_0$ *avoir étonné* $N hum_1$, $N hum_0$ *être étonné par* $N hum_1$ or $N hum_0$ *s'étonne auprès de* $N hum_1$ *de ce* QuP_2 which gives an idea of how these complexe verbs generate an average of 100 rules, or sentence structures, even if no embbeding is involved at this stage.

grammar is represented by one transducer of 13,408 states and 47,119 transitions stored in 908KB.

The following input :

> **Jean est agacé par le fait que son ami , dans la crainte d'être puni par ses parents, ne leur ait pas avoué ses mauvaises notes.**

is parsed in the following way in 0.95s [9] with a program implementing the algorithm ANALYSE_1.

> **(N Jean)N est &Vpp0 agacé par le_fait_QuP le fait (QuP que (N son u ami ami)N , (ADV dans la crainte de (V0W N0 être &Vpp0 puni par (N ses n parent parents)N V0W) ADV) , leur #Nhum2 avoir ait (op #ne -pas op) &Vpp0 avoué (N ses mauvaises n note notes)N QuP)**

Typical time spending varies from 0.05 second for a ten words sentence to 5 seconds for a hundred words sentence under the current implementation. A key point about this method is that the time spending is quite insensitive to the size of the grammar, this is crucial for scaling up the program to much larger grammars. For instance the preceding example is analyzed in 0.93s (instead of 0.95s) for a grammar of half its size (around 100,000 lexical rules).

The coverage of this grammar still has to be extended, not all data we had at our disposal are yet encoded in the transducer (around 50% remain). Thus, given an arbitrary text, whereas most of the simple short sentences (five to fifteen words) are analyzed, the probability of having all lexical descriptions for longer sentences decreases rapidly. However, since all the lexical rules have been checked by hand one by one, the accuracy of the analysis is higher than what can be expected with less lexicalized grammars. This means two things:

- whenever the analysis is found and unless the sentence is syntactically ambiguous, the analysis is unique,
- incorrect sentences are systematically rejected. Thus the set of sentence defined by the parser is a subset of the set of correct sentences. This property is very difficult to achieve through non or less lexicalized grammars.

CONCLUSION

We have introduced two different parsing algorithms based on Finite-State Transducers illustrating a method capable of handling extremely large grammars very efficiently. We have shown how Finite-State Transducers can handle not only finite state grammars but also hierarchical descriptions expressed by context-free based formalisms.

The method has been successfully implemented for a French Lexicon-Grammar consisting of 200,000 lexical rules. The use of Finite-State Transducers yields a typical response time of a fractions of a second.

We have also introduced a bidirectional parsing algorithm which further improves response time.

These investigations have, we believe, important implications for the implementation of large grammars. Moreover, it should be possible to improve these results appreciably by exploring different representations and different decompositions of the grammar transducer with tools from the theory of Finite-State Transducers.

[9] On an HP720, this is the unique parsing, in other words the input is found not to be ambiguous. The time spending includes a morphological analysis by mean of a dictionary look-up. This inflected form dictionary contains 600,000 entries [5].

References

[1] Berstel, Jean, 1979. *Transductions and Context-Free Languages.* Stuttgart, B.G. Teubner 277p.

[2] Boons, Jean-Paul; Alain Guillet; Christian Leclère 1976. *La structure des phrases simples en français. I Constructions intransitives.* Genève: Droz:377p.

[3] Boons, Jean-Paul; Alain Guillet; Christian Leclère 1976. *La structure des phrases simples en français. II Constructions transitives.* Technical Report LADL. Université Paris 7. Paris.

[4] Clemenceau, David; Emmanuel Roche, 1993. *Enhancing a morphological dictionary with two-level rules.* EACL'93, Proceedings of the Conference. Utrecht.

[5] Courtois, Blandine, 1989. *DELAS : Dictionnaire Electronique du LADL pour les mots simples du français.* Rapport technique du LADL, Paris: Université Paris 7.

[6] Gross, Maurice, 1975. *Méthodes en syntaxe, régime des constructions complétives.* Paris: Herman, 415p.

[7] Gross, Maurice, 1986. *Grammaire transformationnelle du français : 3) Syntaxe de l'adverbe.* Paris: Cantilène, 669p.

[8] Karttunen, Lauri; Ronald M. Kaplan; Annie Zaenen 1992. *Two-Level Morphology with Composition.* COLING'92, Proceedings of the conference. Nantes.

[9] Peireira, Fernando C. N., Rebecca N. Wright, 1991. *Finite-State Approximation of Phrase Structure Grammars.* 29th Annual Meeting of the ACL, Proceedings of the conference. University of California, Berkeley.

[10] Roche, Emmanuel, 1993. *Analyse Syntaxique Transformationnelle du Français par Transducteurs et Lexique-Grammaire.* PhD dissertation, Université Paris 7, Paris.

[11] Tapanainen, Pasi; Atro Voutilainen, 1993. *Ambiguity resolution in a reductionistic parser.* Sixth Conference of the European Chapter of the ACL, Proceedings of the Conference. Utrecht, April 1993.