A Finite-State Approach to Schwa/Zero Alternations in French

Miriam Käshammer
Wankheimer Täle 1/003
72072 Tübingen
miriam.kaeshammer@gmx.de

Thesis submitted in fulfilment of the requirements of the degree
Bachelor of Arts in Computational Linguistics
at Seminar für Sprachwissenschaft, Eberhard-Karls-Universität Tübingen

Course: Finite State Methods in Natural Language Processing
Supervisor: Dale Gerdemann

September 7, 2009

Hiermit versichere ich, dass ich die vorgelegte Arbeit selbstständig und nur mit den angegebenen Quellen und Hilfsmitteln (einschließlich des WWW und anderer elektronischer Quellen) angefertigt habe. Alle Stellen der Arbeit, die ich anderen Werken dem Wortlaut oder dem Sinne nach entnommen habe, sind kenntlich gemacht.

(Miriam Käshammer)

Abstract

Tranel (1999, 2000) presents an approach to interconsonantal schwa/zero alternations in French within the framework of Optimality Theory. The paper at hand extends this approach by providing general syllable constraints, and formalizes the analysis using the Foma finite-state calculus. Each constraint is represented as a finite-state transducer that inserts violation markers into candidate strings. For the minimization of violations in order to find optimal candidates, the Matching Approach (Gerdemann and van Noord, 2000) is adopted.

Contents

1 Introduction

Schwa/zero alternation refers to the optional deletion of interconsonantal schwa as in [sœpano]/[spano] for *ce panneau* ("this panel"). This phenomenon is one of the main domains of variation in French phonology. This paper formalizes the optimality-theoretic approach to schwa/zero alternations in French presented in Tranel (1999, 2000). As no complete analysis is provided there, the approach is augmented with general syllable constraints. Various other approaches to the phenomenon have been given in the context of different linguistic theories. (See Dell (1973), Generative Phonology, and Jetchev (1997), Harmonic Phonology.) The strength of Tranel's (1999, 2000) approach is that it accounts for the alternations with just one *Syllable Economy* constraint, whose variable ranking within the hierarchy of syllable constraints corresponds to register and rate of speech.

For the formalization of the analysis, the Foma finite-state calculus is used. Finite-state methods have been successfully used to efficiently implement generative phonology thanks to the finite-state calculus established in Kaplan and Kay (1994). Since the emergence of Optimality Theory (Prince and Smolensky, 1993), several approaches have been presented to model a constraint-based analysis within finite-state power (Karttunen, 1998; Gerdemann and van Noord, 2000). Such an account is not only compact as it results in a single transducer, it is also based on well-studied ground, and it can be perfectly combined with other finite-state natural language tools, such as a morphological analyzer.

In the implementation presented in this paper, optimal candidates are computed according to the Matching Approach (Gerdemann and van Noord, 2000). The GEN function and each constraint are implemented as finite-state transducers. By matching constraint violations against violations in worsened candidates, non-optimal candidates are filtered out. The implementation has instructive character, demonstrating how an optimality-theoretic analysis is formulated in a finite-state system. It is furthermore a useful component in French text-to-speech systems that aim at producing outputs in different styles and speech rates. The syllabifier, that has been implemented along the way, can be used independently.

The paper is organized as follows: Section 2 serves as a general introduction to the topics covered throughout the paper: Optimality Theory, finite-state methods and how the two relate to each other. In section 3, the data on schwa/zero alternations in French is presented. The corresponding optimality-theoretic analysis is provided in section 4, comprising a general syllabification account and Tranel's *Syllable Economy* constraint. The Foma implementation is outlined and discussed in section 5.

2 Background

2.1 Optimality Theory

Optimality Theory (OT) is a linguistic model proposed by Prince and Smolensky in 1993. It assumes that the output forms of a language are the optimal members of a set of candidate forms. A hierarchy of constraints is applied to the candidates in order to find the most optimal one by comparison. This section introduces the basic architecture of OT.

OT has four main components (McCarthy, 2002):

GEN is the candidate generator. It constructs a possibly infinite set of output candidates from the given input, which, for instance, might be an underlying representation for some phonological application. The candidates emitted by GEN are language independent because GEN is meant to be universal.

CON is the set of constraints. All constraints in OT are violable and universal. This means that even the winning candidate does not have to satisfsy all constraints and that CON is the same for the grammars of all languages. There are two basic groups of constraints: faithfulness and markedness. Faithfulness constraints require that the input and the candidate output are identical; markedness constraints evaluate the structural well-formedness of the output.

H is the language-specific constraint hierarchy that explains why languages differ from one another. Given two conflicting constraints A and B, if A dominates B in the hierarchy (A $\gg$ B), the language is different from the language which exhibits B $\gg$ A (B dominates A).

EVAL is the function that returns the most optimal candidate(s). Given two candidates M and N, M is better than N on a constraint A if M violates A fewer times than N. Considering the entire hierarchy H, the ranking of the constraints is strictly respected and the number of violations is not globally evaluated. This *strictness of strict domination* says that a candidate M that just violates one high-ranked constraint is worse than a candidate N that does not, even if N violates many lower-ranked constraints.

The comparison of candidates is usually done in a table. The rows give different candidates, the columns contain the constraints in domination order from left to right. Constraint violations are indicated by some mark, e.g. a star, that is shown in the corresponding cell. A pointing hand usually indicates the optimal candidate. For illustration (Example inspired by Tranel (2000)), we consider two constraints, FAITHFULNESS, which requires that input and output are identical, and *VV, a markedness constraint which prohibits sequences of two vowels. Tables 1 and 2 show the possible rankings of the two constraints to each other resulting in two different languages. The leftmost upper cell contains the input to GEN. Candidates a and b are two members of the output candidate set. In language X, candidate a is optimal; in language Y, candidate b is optimal. The example illustrates again the strict character of the dominance relation.

/ai/	FAITHFULNESS	*VV
a. ☞ ai		*
b. aCi	*	

Table 1: Language X: FAITHFULNESS $\gg$ *VV

/ai/	*VV	FAITHFULNESS
a. ai	*	
b. ☞ aCi		*

Table 2: Language Y: *VV $\gg$ FAITHFULNESS

The universal constraints unite all languages, while the specific grammar of one language is nothing more than the hierarchy of these constraints. It is through the language-specific constraint hierarchy H that OT can explain why and how languages differ and also resemble each other. OT furthermore provides an explanation for why languages have a phonology at all (Tranel, 2000). The traditional generative theory in which the phonological rules are language-specific could not answer this question. According to the OT approach, the sound material is controlled by universal constraints, and languages therefore have a phonology.

In fact, OT has mainly been used within the area of phonology, with constraints concerning syllabification as the most prominent example. The model is however also applied to other fields of linguistics such as morphology and syntax.

2.2 Finite-State Technology

Finite-state technology is well known in the field of computer science and the properties of finite-state machines are well studied. This section gives an introduction to the basic notions of finite-state technology since it is used in this paper to model an OT analysis for syllabification and schwa/zero alternations in French. For more information please refer to Karttunen (2003) or Roche and Schabes (1997).

A *finite-state automaton* (FSA) consists of a finite set of states and a finite set of edges between the states. Each edge is labeled with a symbol a of the alphabet Σ, on which the FSA is defined, or the empty string ϵ. One of the states is defined to be the initial state and any number of states to be final states.

When depicting a finite-state automaton as a graph, the following conventions are usually used: the leftmost state of the graph is the initial state; final states are indicated by a double circle.

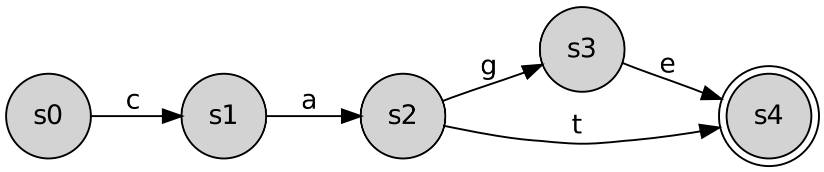

Figure 1: A finite-state automaton accepting the strings *cat* and *cage*

A *path* is a sequence of edges that leads from the initial state to one of the final states. An automaton is said to *accept* or *recognize* the string that results from the concatination of the symbols of such a path. The set of all the paths in the automaton denotes a *regular language.* The automaton in figure 1, for example, encodes the regular language $\{cage, cat\}$.

Regular expressions are a means to exactly describe regular languages without listing all elements. Given an alphabet Σ, this family of languages contains the empty set $\emptyset$, atomic expressions $a \in \Sigma$ and all the expressions than can be created from that by means of Kleene star, concatination and union. The equivalence of finite-state automata, regular languages and regular expressions goes back to Kleene (1956). In relation to his theorem it has also been shown that regular languages are closed under union, Kleene star, concatenation, intersection and complementation.

A *finite-state transducer* (FST) can be considered as a finite-state automaton, in which the edges are not labeled with single symbols but with pairs of strings of symbols. It does not encode a regular language, but a *regular relation*, a set of ordered pairs. This relation maps two languages, the upper and the lower language, to each other. The transducer translates strings from one language into the other language. Regular relations are closed under composition, but, unlike regular languages, regular relations are not closed under intersection and complementation (see Roche and Schabes (1997, p. 18)).

Since a FSA can be seen as an identity transducer, i.e. a finite-state transducer that maps every symbol to itself, automata and transducers are often considered as equal mathematical constructs. This assumption has also been made within the finite-state compiler Foma (see section 5).

Having introduced the basic concepts of finite-state methods, the next section relates them to computational phonology.

2.3 Finite-State Phonology

Although phonological rewriting rules in SPE-tradition superficially resemble general rewriting systems beyond regular languages, it has been shown that the way the rules are used requires only finite-state power Johnson (1972). Kaplan and Kay (1994) establish a high-level finite-state calculus that allows to directly relate phonological rewriting grammars to finite-state transducers: Each rule corresponds to a regular relation and can therefore be compiled into a transducer. Composing the individual transducers according to the rule ordering results in one machine that represents the complete grammar and ensures efficient generation and analysis. Various other finite-state operators have been defined on top of the finite-state calculus that are especially useful for natural language applications, for example conditional replacement (Karttunen, 1995) and directed replacement (Karttunen, 1996).

The finite-state calculus has been implemented in various systems, among them the Xerox finite-state toolkit XFST[1], and the finite-state compiler and library Foma[2] (see Hulden (2009)), that is used for the purpose of this paper. The syntax of the most important regular expression operators in Foma is given in appendix A.

Apart from rewriting rules, the finite-state calculus has also been used to implement optimality-theoretic approaches to phonology (and OT models in general). It is obvious that the comparison method within a table illustrated in section 2.1 is not very practical and efficient as the tables would be huge or even infinite when they contain all candidates generated by GEN. Efficient computation is therefore needed for the application of the constraints and the finding of the optimal candidate by EVAL.

It has been claimed that if GEN encodes a regular relation, and if the constraints can be implemented as filtering identity transducers, they can be treated just as the transducers for rewriting

[1]http://www.stanford.edu/~laurik/fsmbook/home.html
[2]http://foma.sourceforge.net/

rules (Ellison, 1994). The composition of the GEN-relation with the filtering transducers according to the order in the constraint hierarchy results in an efficient and compact machine. This machine, however, is only a simplistic approach, sometimes being called a merciless cascade, that does not exactly represent the OT model of section 2.1. The simplistic implementation lets only perfect candidates pass, namely those that do not violate any of the constraints, whereas OT constraints are usually violable.

The following two sections present two ways to implement the OT model within the finite-state calculus, the Counting Approach and the Matching Approach.

2.3.1 The Counting Approach

To overcome the problems of the merciless cascade, Karttunen (1998) introduces the *lenient composition* operator on the basis of ordinary composition and priority union. With lenient composition between the filters, a filter is applied if at least one candidate meets the constraint. Otherwise all candidates are passed on.

Lenient composition cannot, however, account for the fact that in OT multiple violations of one constraint by one candidate are possible and that the candidate that incurs the fewest violations should be preferred. Counting and comparing the number of violations in order to find the optimal candidates goes beyond finite-state power. Since other tasks of natural language processing (such as morphological analysis) are typically done with finite-state tools, having to go off-line for comparing OT-candidates for the phonology part is an immense drawback.

Karttunen (1998) therefore presents a method to *approximate* the OT analysis, the so-called Counting Approach. Every constraint is associated with a certain predetermined upper bound on counting violations. This can be implemented as a sequence of leniently composed constraints, the first allowing for no violation, the second allowing for one violation etc. until the predetermined bound. The counting technique exactly models the OT analysis for strings up to a certain length, but it is not able to correctly treat input of arbitrary length. It is, however, argued in Karttunen (1998) that having to set a bound is not considered to be a real limitation since the number of violations is usually small. Furthermore one can argue that the length of the input string, for instance a word, is often limited in natural language applications.

2.3.2 The Matching Approach

As an alternative to counting, Gerdemann and van Noord (2000) present an exact approach based on matching. The Matching Approach will be described in this section in some more detail as it is the model that has been chosen for the implementation of schwa/zero alternations in French.

For the Matching Approach, constraints are implemented as regular relations introducing one mark, i.e. a star, per violation into the string. Marked candidates could for example be *ab#*cd** (a) and *#ab*cd#* (b) where *abcd* is the input, * the violation marker and # some markup that has been introduced by GEN. Candidate (a) contains two stars while candidate (b) contains only one star, so (b) should be the winner. The overall task is thus to minimize the number of stars.

The general idea is to worsen all candidates by a so-called *worsening transducer*. This transducer inserts at least one additional star into every string, thus turning it into a worse candidate.[3] When subtracting the set of worse candidates from the set of all candidates, only the best candidates remain.

```
define opt(X) [X - worsen(X)];
```
[4]

`worsen(X)` will be derived step by step in the following. When the original set is compared with the set of worsened candidates, the stars in the strings are matched up, and the additional markup (#) does not play a role. For this reason, when constructing the worsened candidates, all additional markup is eliminated at first, leaving only the original input and the stars (*ab*cd** (a) and *ab*cd* (b)).

```
define star {*};
define markup {#};
define deleteMarkup [Σ-markup | markup:ϵ]*;
```

Adding at least one star to every candidate results in an infinite set of worsened candidates. In the example, every string in the set of worsened candidates has at least two stars, and the string *ab*cd**, a worsened version of candidate (b), is a member of it.

```
define addStar [[Σ* ϵ:star]+ Σ*];
define deleteMarkupAndAddStar deleteMarkup .o. addStar;
```

Since the worsened set is supposed to act like a filter that eliminates the bad candidates, the markup has to be reinserted. This can be done randomly as the markup does not play a role in the comparison process. The example worsened set now contains, amongst others, the string *ab#*cd**, a worsened version of candidate (b) with reinserted markup. This string finally eliminates candidate (a).

```
define addMarkup [Σ-markup | ϵ:markup]*;
define changeMarkupAndAddStar deleteMarkup .o. addStar .o. addMarkup;
define worsen(X) [X .o. changeMarkupAndAddStar]₂;
```

The three steps – deleting the markup, inserting at least one star and reinserting the markup – can be implemented as transducers. As the output of one transducer serves as the input for the next step, they can be composed into one transducer. It is clear that the markup has to be deleted before reinserting it, but whether the stars are added before or after deleting the markup or in the very end does not play a role conceptually. There are, however, differences in performance depending on the system.[5]

A crucial point of the Matching Approach is that bad candidates are filtered out by other candidates from the same input. GEN is therefore only allowed to insert *markup symbols* but not

[3]This is star-based worsening. Gerdemann (2009b) also mentions general worsening which will not be considered in this paper.

[4]Foma syntax is used for the notation of regular expression and relations. See appendix A.

[5]personal communication, Dale Gerdemann (2009)

to change the input in any other way. Input symbols and markup symbols have to be clearly distinguished.

As a last step, the stars in the surviving candidates have to be deleted before marking the violations of the following constraint. This can be incorporated into `opt(X)`.

```
define Pardon(X) [X .o. star -> ε]₂;
define opt0(X) Pardon(X - worsen(X));
```

In the approach sketched so far, the constraint violations have to line up somehow to eliminate all but the optimal candidates. Lined up stars are, however, not generally the case: Considering the candidates *a#b#*cd** (a) and *abc#*d* (b), one observes immediately that adding stars would never yield a worsenend candidate that eliminates (a). Gerdemann and van Noord (2000) therefore introduce transducers to permute the stars, in order to create strings in which the stars match up.

```
define permuteLeft [ε:star [Σ-star]* star:ε];
define permuteRight [star:ε [Σ-star]* ε:star];
define permuteStars [Σ* [permuteLeft | permuteRight]]* Σ*;
define permute1(X) [X .o. permuteStars]₂;
define permute2(X) [X .o. permuteStars .o. permuteStars]₂;
...
define worsen1(X) permute1(worsen(X));
define worsen2(X) permute2(worsen(X));
...
define opt0(X) Pardon(X - worsen(X));
define opt1(X) Pardon(X - worsen1(X));
define opt2(X) Pardon(X - worsen2(X));
...
```

As Gerdemann (2009b) states, this technique only approximates the permutation of stars, and the number of permutations needed to filter out all but the optimal candidates might be unbounded. It turns out, however, that in practical applications only `opt0` and `opt1` are needed to obtain an exact implementation. The exactness test described in Gerdemann and van Noord (2000) can be used to determine how much permutation is needed. `Starred` is an automaton representing candidates that have been marked up for a given constraint. Please refer to section 5.3 for the function `Unflatten(X)`.

```
## Exactness test
define isOptimized(Starred) _isfunctional([Unflatten(Starred) .o. [Σ-star -> ε]]);
```

Gerdemann and van Noord (2000) furthermore report that the resulting transducer implementing the Matching Approach is considerably smaller than the transducer one obtains from the Counting Approach. Additionally, constructing the transducer is usually much faster.

This section has provided an introduction to OT and finite-state methods. It has furthermore been shown how the two relate to each other. The following section concentrates on the linguistic data for schwa/zero alternations in French.

3 Schwa/Zero Alternations in French

French exhibits an interconsonantal schwa/zero alternation as in the examples (1) (Tranel, 1999) and (2) (Jetchev, 1997).

(1) *ce panneau*
"this panel"
  a. [sœpano]
  b. [spano]

(2) *Jean secoue (la branche)*
"John is shaking (the branch)"
  a. [ʒɑ̃sœku]
  b. [ʒɑ̃sku]

The vowel schwa that sometimes disappears in spoken language is one of the main areas of phonological variations in French. The dropped schwa is tranditionally referred to as 'E muet', 'E caduc' or 'E instable'. Whether the optional elision of schwa in a certain environment is allowed at all, is mainly determined by syllable structure. It furthermore depends on the dialect and personal preference of the speaker, and on the speed and register of the speech, whether the schwa is actually pronounced or not.

The following sections provide some remarks on the language that is taken as reference in this paper, present schwa in the French sound system and give the data which the approach wants to account for.

3.1 Remarks on the language of reference

As Dell (1973) points out, the behavior of schwa highly varies from one speaker to another, even if their pronunciation is otherwise very similiar. It might therefore be the case that two people, even though they have about the same geographical, cultural, linguistic and educational background, do not agree in their pronunciation concerning schwa. As a consequence, linguistic analysis of schwa/zero alternations is often not based on homogeneous data. (See Jetchev (1997) for an overview.) Dell (1973, and other puplications) uses his own variety of French (the 'Parisian standard' French) as basis for his linguistic descriptions and analyses. His data has also been taken as a point of reference by other phonologists and is considered to be close to the 'social norm'.

The paper at hand also takes Dell's 'Parisian standard' French as a point of reference, but the analysis given in section 4 could easily be modified and expanded to account for other regional dialects such as Canadian French or southern French (Midi French). The data does not have to be restricted to a certain stylistic variant or speech tempo, since the strength of Tranel's (1999, 2000) analysis is precisely to explain all those variations.

3.2 Schwa in the French vowel system

French has two high front vowels ([i], [y]), four mid front vowels ([e], [ø], [ɛ], [œ]), one low front vowel ([a]), one high back vowel ([u]), two mid back vowels ([o], [ɔ]), one low back vowel ([ɑ]), and four nasal vowels ([ɛ̃], [œ̃], [ɑ̃], [õ]) (Tranel, 1987b).

The alternating vowel *schwa* usually does not surface as [ə], but as a sound close to the mid front rounded vowel [œ]. It is, however, not the case that any [œ] can possibly be deleted, as the examples (3) and (4) (Tranel, 1987b, p. 87) show. Both phrases can be pronounced [dɑ̃lœretablismɑ̃], but only in *dans le rétablissement* (3) the pronunciation without [œ] is possible.

(3) *dans le rétablissement*
"in the re-establishment"
a. [dɑ̃lœretablismɑ̃]
b. [dɑ̃lretablismɑ̃]

(4) *dans leur établissement*
"in their shop"
a. [dɑ̃lœretablismɑ̃]
b. * [dɑ̃lretablismɑ̃]

The surfacing [œ]s as in (3) and (4) are therefore usually grouped into two different categories: unstable [œ] which alternates with zero in certain environments and stable/non-alternating [œ]. To account for the data, different underlying representations are assumed (Dell, 1973; Jetchev, 1997): stable [œ] is underlyingly /œ/, whereas in this paper the symbol /ɶ/ is used for the alternating vowel schwa. When /ɶ/ is realized, it is generally pronounced as [œ], or as a sound between [œ] and [ø] (Dell, 1973). Throughout this paper, [ɶ] is used as the transcription for schwa, indicating that the underlying form is /ɶ/ and that the actual realization varies from speaker to speaker. This also helps to visually distinguish alternating schwa and stable [œ] in the transcriptions provided. As a consequence, (3 a.) should be transcribed as [dɑ̃lɶretablismɑ̃].

3.3 Data: Alternating schwa [ɶ]

The alternating vowel schwa can always be dropped in certain environments. In other occurences, elision of schwa is only possible for some speakers or in fast speech. This section gives an overview of instances of schwa/zero alternations.[6]

Schwa can always be dropped after one consonant if this consonant can assume the coda position of the preceding syllable. This is the case in examples (2), (3), (5) (Jetchev, 1997) and (7) (Tranel, 1999). If the preceding coda is already occupied, deletion of schwa is mostly not possible, since this would result in a complex coda. It is generally argued that codas in French contain at most one consonant. (See for example Dell (1995).) Examples for the non-deletion of schwa because of complex codas [kl]/[ks] are provided in (6) and (8) (Jetchev, 1997).

[6]Alternating schwa at the word end in order to break up heavy consonant clusters ([ɛ̃.kõ.tak.t**ɶ**.po.sibl]/ [ɛ̃.kõ.ta**kt**.po.sibl]) is a different type of phenomenon and not treated in this paper.

(5) *Henri le soutient*
"Henry supports him"
a. [ɑ̃.ri.lœ.su.tjɛ̃]
b. [ɑ̃.ril.su.tjɛ̃]

(6) *Jacques le soutient*
"Jack supports him"
a. [ʒak.lœ.su.tjɛ̃]/[ʒa.klœ.su.tjɛ̃][7]
b. * [ʒakl.su.tjɛ̃]

(7) *dans le panneau*
"in the panel"
a. [dɑ̃.lœ.pa.no]
b. [dɑ̃l.pa.no]

(8) *Jacques secoue (la branche)*
"Jack is shaking (the branch)"
a. [ʒak.sœ.ku]
b. * [ʒaks.ku]

Some speakers, however, allow the deletion of schwa in some special cases even if it results in a complex coda. This can be shown with example (9) taken from Tranel (1999) and example (10)[8]. In these cases the preceding coda is already occupied by /r/. The complex coda [rd]/[rt] has falling sonority and therefore obeys well-known syllable sonority principles. At the word end, where complex codas are allowed in French (Dell, 1995), [rd] and [rt] are well attested (*sourde* [surd], *carte* [kart]).

(9) *sur de personne*
"sure of no one"
a. [syr.dœ.pɛr.sɔn]
b. ? [sy**rd**.pɛr.sɔn]

(10) *pour te parler*
"to speak to you"
a. [pur.tœ.par.le]
b. ? [pu**rt**.par.le]

Schwa can also be deleted if the preceding consonant can be syllabified into the following syllable as a complex onset. Example (1) illustrates this case of schwa/zero alternation: The schwa can be dropped easily because [sp] is a well-formed French onset (*sport* [spɔr]). In example (11) (Tranel, 1999), as a contrast, deletion of schwa yields the complex onset cluster [lp], which is less well formed than [sp]. Some speakers, nevertheless, accept forms like [lpa.no]. Then they also permit [spa.no].

Coming back to example (8), this means that the deletion of schwa should be possible, namely with the syllabification [ʒak.sku], because [sk] is a well-formed French onset (*ski* [ski]).

In (12) (Jetchev, 1997), the consonant cluster [dkr], that is created by the deletion of schwa, is not an admissible onset in French (and nor is [kd] an admissible coda).

(11) *le panneau*
"the panel"
a. [lœ.pa.no]
b. ?? [lpa.no]

(12) *un bac de crapauds*
"a tub of toads"
a. [ɛ̃.bak.dœ.kra.po]
b. * [ɛ̃.bak.dkra.po]

[7]Both syllabifications are conceivable. Which one is chosen depends on the syllabification algorithm.

[8]The syllabifications [syr.dpɛr.sɔn] and [pur.tpar.le] should also be considered. The complex onsets have, however, no rising sonority, and are hardly possible.

In sequences of syllables with a schwa nucleus, the patterns correspond to those of a single schwa. (The examples are taken from Jetchev (1997).) In (13), the second schwa can be deleted without controversy as the preceding /r/ assumes the coda position of the preceding syllable (b). The elision of the first schwa can be explained in the same way (c). One could also argue that [dr] builds a complex onset which conforms to the syllabification given in Dell (1995). It is, however, not possible to delete both schwas at the same time, as this results in an unpronouncable consonant cluster, no matter how the phrase is syllabified (d).

Phrases (14 b) and (14 d) can be explained in parallel to (13). But dropping the first schwa in [pat.dœ.rœ.nar] is different since the preceding coda is already occupied by /t/, and [td] is not an admissible coda. Jetchev (1997) therefore judges the sequence [patdrœnar] as ungrammatical. One might, however, assume that /d/ creates the complex onset [dr] as in *dromadaire* [drɔmadɛr] which yields the syllabification shown in (14 c). Although this is unattested in the literature, the author of this paper accepts this pronunciation.

(13) *(une) queue de renard*
"(a) fox's tail"

a. [kø.dœ.rœ.nar]
b. [kø.dœr.nar]
c. [kød.rœ.nar]/[kø.drœ.nar]
d. * [kødrnar]

(14) *(une) patte de renard*
"(a) fox's paw"

a. [pat.dœ.rœ.nar]
b. [pat.dœr.nar]
c. ? [pat.drœ.nar] (?)
d. * [patdrnar]

The phrase *envie de te le demander* with four schwas has the following eight possible realizations according to Jetchev (1997). They are all based on the fact that the consonant in the onset is syllabified into the coda of the preceding syllable when schwa is deleted.

(15) *(j'ai) envie de te le demander*
"(I) feel like asking you"

a. [ɑ̃.vi.dœ.tœ.lœ.dœ.mɑ̃.de]
b. [ɑ̃.vid.tœl.dœ.mɑ̃.de]
c. [ɑ̃.vi.dœt.lœd.mɑ̃.de]
d. [ɑ̃.vid.tœ.lœd.mɑ̃.de]
e. [ɑ̃.vid.tœ.lœ.dœ.mɑ̃.de]
f. [ɑ̃.vi.dœt.lœ.dœ.mɑ̃.de]
g. [ɑ̃.vi.dœ.tœl.dœ.mɑ̃.de]
h. [ɑ̃.vi.dœ.tœ.lœd.mɑ̃.de]

4 OT Analysis

The analysis for schwa/zero alternations in French provided here is given within the framework of OT. Tranel (1999) and Tranel (2000, in French, provides the same content but more background) present an approach which is entirely based on syllable structure constraints. Moreover, it is the ranking of a constraint concerning syllable economy (SE) that is responsible for whether schwa is pronounced or not.

Tranel (1999, 2000) assumes that the alternating schwa as presented in section 3.3 has phonological status. This corresponds to the general account given in the literature. (See Dell (1973).) Examples such as (16) and (17) show that the phenomenon cannot be explained by epenthesis. When assuming the underlying forms /sɛtpluz/ and /sɛtplas/, the insertion of /œ/ in the same phonological environment incorrectly predicts (17 a).

(16) *cette pelouse*
"this lawn"

a. [sɛtpœluz]

b. ? [sɛtpluz]

(17) *cette place*
"this place"

a. * [sɛtpœlas]

b. [sɛtplas]

Unfortunately, Tranel (1999, 2000) does not give a full OT analysis for schwa/zero alternations, but just presents the main points illustrated by examples. It is essentially the syllable analysis that is not treated in the two papers. Since no OT account for syllabification of French has been found in the literature, the following section (4.1) provides the constraints for syllable segmentation in French. Building up on the syllabification constraints, Tranel's (1999, 2000) approach is presented in section 4.2.

4.1 OT Syllabification of French

Dividing a word/phrase into phonological syllables is a non-trivial problem. In particular for French, various syllabification theories have been put forward (Goslin and Frauenfelder (2000) for an overview). There is no general agreement about how French utterances are supposed to be segmented into syllables. The paper, including the implementation, follows Dell (1995, pp. 14-16):

1. A prevocalic consonant is tautosyllabic with the following vowel.
2. In an Obli cluster the two consonants are tautosyllabic.
3. A postvocalic consonant is tautosyllabic with the preceding vowel, provided no conflict arises with (1.) and (2.).
4. A coda contains at most one consonant.

Obli stands for a sequence of obstruent and liquid from the following set /pl, pr, bl, br, fl, fr, vl, vr, tr, dr, kl, kr, gl, gr/, and the set of consonants does not include glides. To account for glides, it is furthermore assumed that intervocalic clusters consisting of one or two consonants followed by a glide are not broken up (Féry, 2001). Word-final consonant clusters are usually an exception to (4.), e.g. *peste* "plague" [pɛ**st**]. How exactly one would structure these complex rhymes is only of marginal interest for this application. A flat representation, in which the syllable boundaries are marked, provides sufficient information. The foregoing assumptions yield the following illustrative syllabifications: [ɛ̃s.tryk.tœr], [at.la.ab.strɛ], [lœ.ʒar.dɛ̃], [y.na.ni.ma.sjõ].

As follows from Dell (1995), in French each vowel is parsed as the nucleus of a syllable. Onset and coda are optional, but onsets are preferred over codas. In OT, this is covered by the universal markedness constraints ONSET and NOCODA as given in Kager (1999, Ch. 3).

ONSET : Syllables must have onsets.

NOCODA : Syllables are open.

In French, ONSET and NOCODA seem to have no specific ordering with respect to each other. No matter which of the two constraints would be ranked first, they yield the same most optimal result, see tables 3 and 4.

Table 3

		/pano/	ONSET	NOCODA
a.	☞	pa.no		
b.		pan.o	*	*

Table 4

		/arɔ̃di/	ONSET	NOCODA
a.	☞	a.rɔ̃.di	*	
b.		ar.ɔ̃.di	**	*
c.		ar.ɔ̃d.i	***	**

The faithfulness constraints MAX prevent deletion of input segments.[9] The partial ranking MAX(V) ≫ MAX(Schwa) is taken from Tranel (1999, 2000) and further elaborated on in section 4.2. For completeness, the constraint MAX(C) is introduced. I assume that MAX(C) and MAX(V) have no specific ordering with respect to each other.[10] {MAX(C), MAX(V)} must be the highest ranked constraint so far, since the deletion of other sounds than schwa is not a possible strategy in French syllabification. MAX(Schwa) has to be ranked lower than the syllabification constraints because schwa may be deleted to obtain well-formed syllables. Table 5 illustrates the effects of the MAX constraints. The low-ranked MAX(Schwa) eliminates candidate b. We will see later that MAX(Schwa) is a violable constraint and that a higher ranked constraint may rule out candidate a, leaving b as the optimal candidate.

MAX(X) : Input segments of the type X have output correspondents.

MAX(V) : Input vowels, except schwa, have output correspondents.

MAX(Schwa) : Input schwas have output correspondents.

***COMPLEXONSET** (Kager, 1999, Ch. 3) prohibits more than one consonant in an onset. In French, this constraint has to be ranked above {ONSET, NOCODA}, as can be seen in table 6.

[9]A complete analysis would also have to include the counterpart constraint DEP(X) which ensures that output segments have input correspondents, thereby preventing insertion. Since the analysis at hand is only concerned with the deletion of sounds, this constraint is left out.

[10]This might be a simplification of the facts which does not affect the analysis of schwa/zero alternations. However, for other phenomena such as ‘liaison’, one might have to reconsider this ordering.

Table 5

		/sœpano/	MAX(C)	MAX(V)	ONSET	NOCODA	MAX(Schwa)
a.	☞	sœ.pa.no					
b.		spa.no					*
c.		œ.pa.no	*		*		
d.		sœp.no		*		*	
e.		sœ.no	*	*			

Table 6

		/atla/	*COMPLEXONSET	ONSET	NOCODA
a.		a.tla	*	*	
b.	☞	at.la		*	*
c.		atl.a		**	*

Tranel (1999, 2000) suggests that *COMPLEXONSET actually covers a whole hierarchy of constraints that incorporate well-formedness conditions on syllable onsets, such as the sonority sequencing principle. For French, this means ruling out onsets of more than three consonants, prohibiting onsets of three consonants that do not respect the sonority hierarchy (e.g. /dsw/), prohibiting onsets of two consonants that do not respect the sonority hierarchy (e.g. /td, lp/), and furthermore prohibiting admissible three-consonant (e.g. /str, trw/) and two-consonant onsets (e.g. /tw, sp, dr/). The constraints must furthermore be ranked according to their admissibility. Obviously, most of them do not interact with each other, so no strict dominance relation can be established between all of them. Nevertheless, the interaction with other constraints, such as NOCODA and the syllable economy constraint SE, provides some more evidence for the ordering. The onset constraints are accordingly grouped, and the groups are ranked. This hierarchy could for example be the following:
{*CCCC(ONSET), *dsw(ONSET), ...}$_1$ ≫ {*td(ONSET), ...}$_2$ ≫ {*lp(ONSET), ...}$_3$ ≫
{*str(ONSET), ...}$_4$ ≫ {*sp(ONSET), ...}$_5$ ≫ {*trw(ONSET), *tw(ONSET), ... , *OBLI(ONSET)}$_6$.

***CCCC(ONSET)** : Onsets must not have four or more consonants.

***xyz(ONSET)** : Onsets must not be /xyz/.

***xy(ONSET)** : Onsets must not be /xy/.

In the following, the constraints of group n are taken to be unordered with respect to each other and called *COMPLONSET$_n$ as a group. Group 1 prohibits onsets of four consonants, and onsets of three consonants that do not correspond to well-established onset conditions (sonority principle, special behavior of the fricative /s/). Group 2 and 3 rule out onsets of two consonants that do not have rising sonority or that are not considered as admissible onsets. However, the clusters in group 2 are considered “worse” than those in group 3. Group 4 rules out admissible CCC onsets that do not end in a glide, group 5 admissible CC onsets that are not OBLI and that do not end in a glide, and group 6 OBLI onsets and onsets of two or three consonants that end in a glide. The lowest-ranked group 6 thus contains well-formed onsets that are always clustered together, so that the first consonant is never syllabified as the preceding coda. Ranking NOCODA between group 5 and 6 yields this syllabification. This is illustrated in tables 7 and 8: The Obli cluster /kr/ builds an onset, while /bs/, which is not matched by group 6, is broken into coda and onset. Within this hierarchy, *ComplOnset$_6$ is not active. It is either not violated or violated by the optimal candidate.

Grouping and ranking of the onset constraints is not only governed by general syllable conditions, but also by the dialect and the speaker's individual preferences. Group 2 and 3 for instance might often be merged into one group, but for some speakers it makes sense to keep them apart as will be shown in section 4.2.1.

Table 7

/ɔbsɛn/	*ComplOnset$_2$	Onset	NoCoda
a. ɔ.bsɛn	*	*	
b. ☞ ɔb.sɛn		*	*
c. ɔbs.ɛn		**	*

Table 8

/akro/	Onset	NoCoda	*ComplOnset$_6$
a. ☞ a.kro	*		*
b. ak.ro	*	*	
c. akr.o	**	*	

The ***ComplexCoda** constraint (Kager, 1999, Ch. 3) can be given an analysis in parallel to *ComplexOnset: an internal hierarchy of constraints describing admissibility of codas. For the many speakers that do not allow for complex codas at all, the whole coda hierarchy is just ranked directly before (or after) *ComplOnset$_1$. However, other speakers accept complex codas of two consonants in general ([a.vɛ**kl**.pa.no]) or that have falling sonority ([sy**rd**.pɛr.sɔn]), which assumes the following coda hierarchy: $\{$*CCC(Coda), …$\}_1 \gg \{$*kl(Coda), …$\}_2 \gg \{$*rd(Coda), …$\}_3$. Group 1 rules out codas of three or more consonants, group 2 rules out codas of two consonants that do not have falling sonority and group 3 rules out codas of two consonants that respect the sonority hierarchy. The constraints in group n will be called *ComplCoda$_n$. It is assumed that the *ComplCoda$_n$ constraints do not apply at the word end as French exhibits complex word-final onsets.

***CCC(Coda)** : Codas must not have three or more consonants.

***xy(Coda)** : Codas must not be /xy/.

As already indicated for speakers that do not accept complex codas, the hierarchy of complex onset constraints and the hierarchy of complex coda constraints are interspersed. How exactly one hierarchy is interspersed with the other, depends on the speaker, and probably also on style and speech rate. For the purpose of illustration, let us assume the following partial hierarchy which correctly predicts the examples given in Tranel (1999, 2000): $\{$*ComplOnset$_1$, *ComplCoda$_1\} \gg$ *ComplOnset$_2 \gg$ *ComplCoda$_2 \gg$ *ComplOnset$_3 \gg$ *ComplCoda$_3 \gg$ *ComplOnset$_4 \gg$ *ComplOnset$_5 \gg$ *ComplOnset$_6$. The constraints might well be more unordered in reality, thus more research is needed here. This hierarchy, however, yields the correct syllabification of the input phrases tested (see appendix C), and no evidence against it could be found. For implementation purposes, the constraints need to be strictly ranked anyway.

Table 9 illustrates how the intervocalic cluster /str/ is syllabified. Here again, it is crucial that *ComplOnset$_6$ is ranked below and *ComplOnset$_4$ above NoCoda to obtain the correct syllabification of /str/ according to Dell (1995). Table 10 shows the syllabification of the intervocalic cluster /rdp/ which is created by elision of schwa. From the depicted candidates, candidate c is most optimal since this speaker prefers an [rd]-coda over an [dp]-onset. However, with the constraints considered so far, [syr.dœ.pɔl] is even more optimal as it does not have a complex coda at all. The syllable economy constraint SE, introduced in the next section, provides this missing piece.

Table 9

	/ɛ̃stryktœr/	*ComplCoda$_1$	*ComplCoda$_2$	*ComplOnset$_4$	Onset	NoCoda	*ComplOnset$_6$
a.	ɛ̃.stryk.tœr			*	*	**	
b. ☞	ɛ̃s.tryk.tœr				*	***	*
c.	ɛ̃st.ryk.tœr		*		*	***	
d.	ɛ̃str.yk.tœr	*			**	***	

Table 10

	/syrdœpɔl/	*ComplOnset$_1$	*ComplCoda$_1$	*ComplOnset$_2$	*ComplCoda$_3$	Onset	NoCoda
a.	sy.rdpɔl	*					*
b.	syr.dpɔl			*			**
c. (☞)	syrd.pɔl				*		**
d.	syrdp.ɔl		*			*	**

4.2 SE: Syllable Economy (Tranel, 1999, 2000)

Tranel (1999, 2000) accounts for the interconsonantal schwa/zero alternations as presented in section 3.3 with one single constraint that is called *Syllable Economy* (SE). SE belongs to the family of *Struc constraints (Prince and Smolensky, 1993) that ensure that the candidates are of minimal structure. As every syllable in a candidate is counted as a violation of SE, SE is basically violated by every candidate.

Concerning the optional deletion of schwa, if one considers only the SE constraint, a candidate without schwa will always be more optimal than a candidate with schwa because the non-realization of schwa yields one syllable less. Whether an alternating schwa is pronounced or not, depends therefore on constraints that are ranked higher than SE which are violated if a schwa is deleted. Those constraints are markedness constraints concerning syllable structure and admissible consonsant sequences.

The variable ranking of the SE constraint within the constraint hierarchy H accounts for the optional character of the elision of schwa. Its position correlates with the style of the speaker and the speech rate, and furthermore explains the variations concerning schwa/zero alternations between different speakers. Generally, the more informal and faster the speech, the higher ranked SE is in the constraint hierarchy. If SE is ranked relatively low, all schwas are pronounced, while they are more likely to be deleted when SE is ranked high.

Tranel (1999, 2000) explains the fact that the alternations described in section 3.3 in standard French only concern schwa and not the other vowels with the hierarchy Max(V) $\gg$ Max(Schwa).

This reflects the “weak” character of schwa that is often analysed as being “floating” (without an anchor) in contrast to the other anchored vowels (Jetchev, 1997; Tranel, 1987a). In standard French, SE may dominate MAX(Schwa), but despite its variable character, it is obligatorily dominated by MAX(V): MAX(V) $\gg$ SE $\gg$ MAX(Schwa). In other languages or dialects of French, SE might well climb higher in the hierarchy that is incorporated internally in MAX(V): MAX(lowV) $\gg$ MAX(midV) $\gg$ MAX(highV) $\gg$ MAX(Schwa) (Tranel, 1999).

The following section illustrates how SE interacts with the syllabification constraints, providing an analysis for schwa/zero alternations.

4.2.1 Illustrative SE rankings

(18) *ce panneau*
“this panel”
a. [sœpano]
b. [spano]

(19) *le panneau*
“the panel”
a. [lœ.pa.no]
b. ?? [lpa.no]

Example (1), reproduced here as (18), shows a phrase for which the pronunciation with schwa (a) as well as without schwa (b) is allowed. Tranel (1999, 2000) explains the two options with the variable ranking of SE. If SE is placed below the constraint *sp(ONSET), a rather low-ranked constraint within our *COMPLONSET$_n$ hierarchy, /œ/ surfaces because a candidate with rather unmarked syllables (a) is preferred over a candidate with a complex onset (b) (Table 11). Tranel (1999, 2000) furthermore states that SE climbs up in the hierarchy H according to speech style and rate, so that it dominates *sp(ONSET) (Table 12). Having a minimal number of syllables is then the deciding factor for the optimal candidate.

There is, however, an individual upper limit for the ranking of SE. For speakers that do not accept the deletion of schwa in /lœpano/ (example (11), reproduced as (19)), SE does not dominate *lp(ONSET) as shown in table 14 (Tranel, 1999, 2000). As a result, schwa is always realized to avoid the complex onset /lp/. Generally speaking, SE is ranked below *COMPLONSET$_3$ for this speaker.

If a speaker allows SE to dominate *lp(ONSET), and therefore accepts [lpa.no], this speaker necessarily also accepts [spa.no] according to Tranel (1999, 2000), as the established hierarchy is *lp(ONSET) $\gg$ *sp(ONSET) This corresponds to the syllabification analysis given in section 4.1 where the hierarchy is *COMPLONSET$_3$ $\gg$ *COMPLONSET$_5$.

Table 11

	/sœpano/	*lp(ONSET)	*sp(ONSET)	SE
a.	☞ sœ.pa.no			***
b.	spa.no		*	**

Table 12

	/sœpano/	*lp(ONSET)	SE	*sp(ONSET)
a.	sœ.pa.no		***	
b.	☞ spa.no		**	*

The syllable economy constraint SE also interacts with coda constraints. Tranel (1999, 2000) accounts for the two possible pronunciations of *dans le panneau* in example (7) with the variable ranking of SE with respect to NOCODA: The pronunciation with schwa is produced if SE is

Table 13

/lœpano/	*lp(Onset)	*sp(Onset)	SE
a. ☞ lœ.pa.no			***
b. lpa.no	*		**

Table 14

/lœpano/	*lp(Onset)	SE	*sp(Onset)
a. ☞ lœ.pa.no		***	
b. lpa.no	*	**	

ranked lower than NoCoda as shown in table 15. Ranking SE higher than NoCoda yields the pronunciation without schwa, which has fewer syllables (Table 16).

Table 15

/dɑ̃lœpano/	NoCoda	SE
a. ☞ dɑ̃.lœ.pa.no		****
b. dɑ̃l.pa.no	*	***

Table 16

/dɑ̃lœpano/	SE	NoCoda
a. dɑ̃.lœ.pa.no	****	
b. ☞ dɑ̃l.pa.no	***	*

Tranel (1999, 2000) further explains the data in example (6) [ʒak.lœ.su.tjɛ̃]/ *[ʒakl.su.tjɛ̃] in contrast to example (9) [syr.dœ.pɛr.sɔn]/ ?[syrd.pɛr.sɔn] by the interaction of the SE constraint with the hierarchy of *ComplCoda constraints. This hierarchy includes *kl(Coda) ≫ *rd(Coda), or more generally, as established in section 4.1, *ComplCoda$_2$ ≫ *ComplCoda$_3$. Intercalating SE between the two constraints yields a speaker that allows [syrd.pɛr.sɔn] but not [ʒakl.su.tjɛ̃] in a certain style and speech rate. The corresponding tables are 17 (taken from Tranel (1999)) and 18. This ranking clarifies why it makes sense to devide two-consonant codas into different groups according to their admissibility (see 4.1). If neither [ʒakl.su.tjɛ̃] nor [syrd.pɛr.sɔn] is accepted, SE is ranked below *rd(Coda), probably lower than the complete *ComplCoda$_n$ hierarchy. If both are permitted, SE's position is higher than *kl(Coda), probably between *ComplCoda$_1$ and *ComplCoda$_2$.

Table 17

/syrdœpɛrsɔn/	*kl(Coda)	SE	*rd(Coda)
a. syr.dœ.pɛr.sɔn		****	
b. ☞ syrd.pɛr.sɔn		***	*

Table 18

/ʒaklœsutjɛ̃/	*kl(Coda)	SE	*rd(Coda)
a. ☞ ʒak.lœ.su.tjɛ̃		****	
b. ʒakl.su.tjɛ̃	*	***	

Tranel (1999, 2000) mentions one additional constraint that prohibits sequences of consonant + liquid + glide. This is however not a syllable constraint, and it is treated as a simplification of the facts. It is therefore neglected in the approach at hand. Data in which schwa is followed by an 'h-aspiré' requires more than the presented constraints (Tranel and Gobbo, 2002). These cases are not considered here.

As it has been shown, Tranel (1999, 2000) can account for schwa/zero alternations by pointing out only one additional constraint. The strength of this constraint-based analysis is furthermore that it manages to explain the variation between different speakers, establishing specifically that speech rate and style correspond to the position of SE in the constraint hierarchy H. This elegant analysis, however, does not cover all the data presented in section 3.3 as will be shown in the following section.

4.2.2 Problems with Tranel's analysis

Tranel's (1999, 2000) analysis for schwa/zero alternations has a considerable weak point when it comes to input strings with more than one schwa. If the deletion of several schwas depends on the same constraint, either the candidate preserving all schwas is optimal (SE is ranked low), or the candidates with the highest number of deleted schwas are optimal (SE is ranked higher than the constraint that rules out these candidates). As Tranel (1999, 2000) presents his approach, there is no way to explain output forms in which some but not all possibly deletable schwas are elided. Considering example (15) *envie de te le demander*, the candidate in which all four schwas are realized (a) is optimal as long as NoCoda dominates SE. If SE climbs up in the hierarchy, candidates (b)-(d) with two of four surfacing schwas are optimal, as they have the fewest syllables. Candidates (e)-(h) with three of four surfacing schwas, however, will never be optimal as they either lose against (a) applying the NoCoda constraint, or against (b)-(d) applying the SE constraint, although they are well-formed and attested output forms.

There is another, slightly different problem, that is additionally related to the given syllable constraints. The forms (13 a) [kø.dœ.rœ.nar] and (14 a) [pat.dœ.rœ.nar] are correctly predicted if SE is ranked sufficiently low. However, when SE moves up and dominates *ComplOnset$_6$ (*dr(Onset)), the candidates [kø.drœ.nar] and [pat.drœ.nar] are the most optimal, while the forms in which the second schwa is deleted never surface. The reason for this is the low ranking of *ComplOnset$_6$ comprising Obli onsets. It is dominated by NoCoda to formulate Dell's (1995) syllabification that preferes [kø.**drœ**.nar] over [kø**d.rœ**.nar]. In the example at hand [kø.**drœ**.nar] competes with [kø.**dœr**.nar], which violates the same constraints as [kø**d.rœ**.nar] (but in different locations) and is therefore eliminated from the set of candidates. In example (14), the situation is about the same. [pat.dœr.nar] (three violations of NoCoda) loses against [pat.drœ.nar] (two violations of NoCoda, one violation of *ComplOnset$_6$). The ranking of SE does not even play a role here, as both candidates have the same number of syllables. One could argue that the deletion of the first schwa should not be possible at all because the preceding coda is already occupied. However, as Tranel's (1999, 2000) analysis allows [spa.no], it should also allow [drœ.nar].

It is not immediately clear how these problems, that are supported by the output of the implemented system (section 5), can be solved. Different syllable constraints that do not follow Dell (1995) probably lead to the desired output forms in cases like examples (13) and (14). However, the problem of constraint violations in different domains might also occur in other examples. The forms of *envie de te le demander* in which only one schwa is deleted, might furthermore arise from the interaction with other constraints that are not mentioned in Tranel (1999, 2000).

5 Foma Implementation

Having established the necessary constraints, the OT analysis of schwa/zero alternations in French including syllabification is formulated within finite-state power in this section. The Matching

Approach will be adopted since the input might theoretically be arbitrarily long. The foregoing analysis in section 4 will be formalized using the finite-state compiler and library Foma.

The general idea is to provide a regular relation Gen that maps the input to a set of candidates with freely introduced syllable boundaries and deleted sounds. The constraints are implemented as replacement rules that insert violation markers (stars) into the candidates. Composing the Gen-transducer with a constraint transducer yields a transducer representing the starred candidates. The `opt0(X)/opt1(X)/...` functions given in section 2.3.2 turn it into a transducer representing only the optimal candidates with respect to the constraint. The following sections provide an outline of the implementation of the Gen function, of the constraints and how everything is put together.

First of all, some basic definitions are provided in order to simplify defining the following regular expressions: sound segments that are useful for phonological analysis, and a word boundary symbol, which is needed in the process of syllabification. All these symbols are *input symbols*, meaning that the input string is made of sound segments and word boundaries.

```
## Vowels
define Schwa Œ;
define VWithoutSchwa[11] [ u | o | ɔ | ɑ | i | y | e | ø | %ɛ | a | œ ]; [12]
define V [ VWithoutSchwa | Schwa];
## Consonants
define Glide [ɥ | w | j];
define Obstruent [ p | b | f | v | k | g | t | d | s | z | ʃ | ʒ ];
define Liquid [ l | r ];
define Nasal [ m | n | ɲ | ŋ ];
define Con [Obstruent | Liquid | Nasal ];
define C [Glide | Con];
## Sound segments
define Seg [V | C];
## Word boundary
define WB %#; [12]
```

As can be seen from the definitions, no general treatment for floating segments is implemented. Schwa is defined as a regular vowel. Its actual pronunciation (see section 3.2) is determined elsewhere. Schwa can be deleted as any other vowel. Its unstable character is accounted for by the ranking of the Max constraints (see section 4.1).

[11]The four nasal vowels ([ɛ̃], [œ̃], [ɑ̃], [õ]) are left out as they just act like normal vowels and do not contribute anything special to syllabification or schwa/zero alternations. In the implementation, their non-nasal counterparts ([ɛ], [œ], [ɑ], [o]) are used. They could however easily be added to VWithoutSchwa.

[12]Some symbols have to be escaped because they have a special meaning in Foma.

5.1 The Gen function

The Gen function must fulfill two goals: (1) optionally delete sound segments from the input[13] and (2) parse the input into syllables. As Gen has to be general and productive, every possible candidate has to be generated, no matter how weird it may be.

Deletion of segments. A real deletion of input segments in Gen is not possible within the Matching Approach as it requires that the input symbols are not changed. Deletion can, however, be simulated by inserting editing instructions, i.e. deletion markers. As Gerdemann (2009b) suggests, a (phonetic) module is then responsible for undertaking the real changes at the very end. Consequently, a deleted segment is defined as the sequence of a deletion marker and a segment. In Gen, a deletion marker is optionally inserted before any segment of the input string. This is formalized in the `DeleteSeg` transducer.

```
define Del %%D;                                 # Deletion marker
define DeletedSeg [Del Seg];                    # Deleted segment
## Insert deletion markers
define DeleteSeg [ [..] (->) Del || _ Seg ];
```

Insertion of syllable markers. The basic idea is to insert syllable markers everywhere into the input string. The syllabification constraints are then in charge to filter out the correctly syllabified candidates. Nevertheless, some language-specific obligatory constraints are incorporated in Gen to get rid of constraints that do not play a role in the approach to schwa/zero alternations at hand: every vowel in French provides the nucleus of exactly one syllable. The nucleus is put in parentheses, which is not strictly necessary with the above restriction but makes it easier to refer to onset and coda in the constraints. The word boundary marker and deleted segments need a special treatment in the syllable definitions because a syllable boundary before or after those elements yields the same syllabification. Deleted segments are therefore always grouped with the following segment, as can be seen in the definitions of `SOnset`, `SCoda` and `SNucleus`. In the definition of `Syllable`, word boundaries are ignored in the first step.

```
define SB %.;                                   # Syllable boundary
define rNuc %);                                 # Nucleus: right bracket
define lNuc %(;                                 # Nucleus: left bracket
define Nuc [ rNuc | lNuc ];                     # Nucleus markers
define Syll [ Nuc | SB ];                       # Syllable markers

## Syllable definitions
define SOnset [DeletedSeg* C]*;
define SCoda [DeletedSeg* C ]*;
define SNucleus [lNuc DeletedSeg* V rNuc];
```

[13]In order to have a universal Gen function, it should also optionally insert sound segments into the input. The corresponding constraint Dep(X) that ensures that output segments have corresponding input segments was not considered however. Insertion of sounds in Gen is consequently also neglected.

```
define Syllable [SOnset SNucleus SCoda]/WB;
```

The transducer `InsertSyll` that maps the input to every possible syllabification, in which each vowel provides the nucleus of one syllable, has three steps. First, the nuclei are marked up, second, syllable boundaries are inserted, and third, `[SB WB]` sequences are ruled out, as these candidates have the same syllabification as candidates with `[WB SB]` sequences.

```
define InsertSyll [
    DeletedSeg* V @-> lNuc ... rNuc || [.#.|\Del] _   # Insert nucleus markers
    .o.
    Syllable -> ... SB || _ Syllable                  # Insert syllable boundaries
    .o.
    ~$[SB WB]                                         # Word boundary correction
];
```

The Gen function. Gen is then the composition of the transducer for deletion and the transducer for syllabification. It is furthermore ensured that the input contains neither stars, which are used as violation markers (see section 2.3.2), nor markup symbols introduced by Gen.

```
define markup [ Del | Syll ];
define Gen0 [ ~$star .o. ~$markup .o. DeleteSeg .o. InsertSyll ];
```

Gerdemann (2009a) describes a series of tests to check whether `Gen0` is well-formed. They are given in appendix B, and all yield a positive result for `Gen0`. For illustration of Gen, consider the input /ty#va#/ (*tu vas* "you go"). The composition `{ty#va#} .o. Gen0` yields the following 18 candidates[14] from which [t(y)#.v(a)#] is the correct one:

t(y)##, t(y)#v#, t(y)#v.(a)#, t(y)#.(a)#, **t(y)#.v(a)**#, t#(a)#, t##, t#v(a)#, t#v#, (y)##, (y)#v#, (y)#v.(a)#, (y)#.(a)#, (y)#.v(a)#, #(a)#, ##, #v(a)#, #v#

5.2 The Constraints

The constraints in the implementation all have the same format `Z -> ... star || L _ R`, where `Z` is a regular expression that describes the violation. The rule transducer inserts the violation marker `star` after every violation `Z` that occurs in the specified context. The left (`L`) and right (`R`) contexts are optional.

The syllable economy and maximization constraints have a straight forward implementation. Every syllable boundary counts as a violation of **SE**. Max(X) is violated by a segment X preceded by the deletion marker. The violation patterns are not restricted by context. **Max(C)**, **Max(V)** and **Max(Schwa)** are defined using the function `MAX(X)`.

```
define SE [ SB -> ... star ];
define MAX(X) [[ Del X ] -> ... star ];
```

[14]For readability, application of the phonetic module is assumed.

The **NoCoda** constraint is violated if a coda is occupied by one or more consonants. Deleted segments and word boundaries that also happen to be in the coda have to be ignored. A coda in the implementation at hand is defined as having the right parenthesis of the nucleus at its left, and a syllable boundary or the end of the string at its right. With the basic definitions given above and the end-of-string symbol provided by Foma, these delimiters are defined as left and right contexts.

```
define NoCoda [[ C+/DeletedSeg ]/WB -> ... star || rNuc _ [ SB | .#. ]];
```

An empty onset is counted as a violation of **Onset**. The Foma implementation of this constraint is therefore a special case of the general format. The violation pattern is the empty string, and it does not have to be referenced on the right-hand side of the replacement. Deleted segments and word boundaries that might occupy the onset are accounted for in the context. An onset is generally delimited in our implementation by a syllable boundary or the beginning of the string on the left, and by the left parenthesis of the nucleus on the right.

```
define Onset [[..] -> star || [ SB | .#. ] DeletedSeg*/WB _ lNuc ];
```

Implementing the ***ComplexOnset** and ***ComplexCoda** hierarchy is more involved as every possible and impossible onset and coda has to be described. The hierarchies furthermore vary among speakers. In the implementation, admissible clusters are explicitely defined, and also used to describe non-admissible onsets and codas. The constraints in the groups *ComplOnset$_n$ and *ComplCoda$_n$ are not explicitely spelled out, but generalized in order to implement them as one constraint per group. This strategy might over-generalize a bit since, other than the Obli onsets, no exceptions in the general patterns are defined. To have a perfect description, more work is needed here.

Admissible onsets are defined as Obli clusters, and two-consonant and three-consonant clusters that end in a glide or start with /s/. Other clusters that have rising sonority are sequences of an obstruent and a liquid that are not Obli, and an obstruent followed by a nasal. The set `ExOnset` comprises onsets that are only exceptionally accepted by some speakers (in fast and informal speech).

```
define OBLI [[[ Obstruent & ~[ s | z | ʃ | ʒ ]] Liquid ]  & ~[ tl | dl ]];
define CGOnset [ Con Glide ];
define sCOnset [ s [[ Obstruent-s ] | Nasal ]];
define CCGOnset [ Obstruent Liquid Glide ];
define sCCOnset [ s [Obstruent-s ] Liquid ];

define notOBLI [ Obstruent Liquid ] & ~OBLI;
define ObNOnset [ Obstruent Nasal ];
define ExOnset [ {lp} ];
```

With the above given onset definitions, six *ComplexOnset constraints are defined that correspond to the six ***ComplOnset**$_n$ groups in section 4.1. The all have the form

```
define MComplOnsetN [[ X/DeletedSeg ]/WB -> ... star || [ SB | .#. ] _ lNuc ];
```

with N and X having the following values:

```
N    X
1    [[ C^4 C*] | [ C^3 & ~[ sCCOnset | CCGOnset ]]]
2    [ C^2 & ~[ ExOnset | ObNOnset | notOBLI | sCOnset | CGOnset | OBLI ]]
3    [ notOBLI | ObNOnset | ExOnset ]
4    sCCOnset
5    sCOnset
6    [ CCGOnset | CGOnset | OBLI ]
```

A similiar strategy is adopted for the *ComplexCoda constraints. They are generalized in three constraints, that correspond to the three ***ComplCoda**$_n$ groups, which have the following format:

```
define MComplCodaN [ Y -> ... star || rNuc _ [ SB | .#. ]];
```

Deleted segments and word boundaries cannot simply be ignored as in `MComplOnsetN`, as word-final codas are in fact often complex in French. They should not violate the *ComplexCoda constraints. When describing the violation patterns `Y`, the clusters at word's end have to be carefully left out for this reason.

```
N    Y
1    [[[[ C C C+ ]/DeletedSeg ]/WB ] & ~[ C C C+ WB ]]
2    [[[ C C ]/DeletedSeg ]/WB ] & ~[C C WB ] & ~CCCoda
3    CCCoda
```

`CCCoda` is the set of two-consonant codas that are widely accepted if a speaker admits complex codas. The first consonant typically is a liquid or /s/. Here again, there is room for finer descriptions of codas.

```
define CCCoda [[[ Liquid [ Obstruent | Nasal | Liquid ]] |
                 [ s [ Obstruent-s ]]]/DeletedSeg ]/WB;
```

Gerdemann (2009a) gives a test that can be used to check whether a constraint is well-formed. It makes sure that any string is accepted as input, and that all the constraint transducer does is to insert stars into the input. All the constraints given in this sections yield a positive result when applying `ConstraintWellFormed` to them.

```
define ConstraintWellFormed(C)
       and(equal(C₁, Σ*), _isidentity(~$star .o. C .o. [star -> 0]));
```

5.3 Finding optimal candidates

The strategy for finding optimal candidates as presented in Gerdemann (2009a) is applying one constraint `C` to Gen yielding starred candidates. They are "optimized" by one of the `opt` functions, which means that the optimal candidates with respect to the constraint are returned (without stars). The FSA representing them is used as Gen to the next lower-ranked constraint, and the template given below is applied for each constraint following the order of the hierarchy H. In this way, a series of more and more restrictive GENs is defined successively. To determine how many permutations are needed in order to find the optimal candidates (i.e. which `opt` function to use), `isOptimized`, as given in section 2.3.2, is applied.

```
define CStarred [ThisGen .o. C]₂;
define NextGen opt0(CStarred);
define NextGenIsOptimized isOptimized([NextGen .o. C]₂);
```

As presented in the above given code, `ThisGen` and `NextGen` are finite-state automata, not transducers. They represent FSTs, however, since, throughout the implementation, markup and violation symbols are always distinguished from input symbols. The FSAs for GEN can therefore be retransformed into FSTs as given in the function `Unflatten(X)`. Furthermore, the original `Gen0` (section 5.1) can also be turned into an FSA that represents an FST since the well-formedness tests in appendix B succeed. Gerdemann (2009a) describes that the final GEN (or any intermediate GEN) can be applied to input phrases with the help of `Unflatten(X)`. `ApplyGen` defines this function. The motivation for using automata instead of transducers is efficiency as minimization algorithms can always be applied to FSAs as Gerdemann (2009a) states. They are furthermore closed under intersection and complementation in contrast to FSTs.

```
define Unflatten(X) [[ markup | star ]  -> ε ]⁻¹ .o. X;
define Gen Gen0₂;                                   # FSA
define ApplyGen(Lex, G) [Lex .o. Unflatten(G)]₂;   # Lex: input; G: Gen as FSA
```

If the constraints are applied one after the other as given above, this implements a strictly ranked hierarchy. However, as we have seen in section 4.1, in linguistic applications this is not always the case, and the constraints often do not have an obvious order with respect to each other. Nevertheless, for implementation purposes, such a strict ranking has to be assumed. The schwa/zero alternations are therefore implemented with the following hierarchy that corresponds to what is given in section 4.1:

1. MAX(C)
2. MAX(V)
3. *COMPLONSET$_1$
4. *COMPLCODA$_1$
5. *COMPLONSET$_2$

SE border

6. *COMPLCODA$_2$
7. *COMPLONSET$_3$
8. *COMPLCODA$_3$
9. *COMPLONSET$_4$
10. *COMPLONSET$_5$
11. NOCODA
12. ONSET
13. *COMPLONSET$_6$
14. MAX(Schwa)

SE's variable ranking within this hierarchy is delimited to the top by *COMPLONSET$_2$. As pointed out before, many speakers might also require *COMPLONSET$_3$ and/or *COMPLCODA$_2$ to dominate SE.

Picking up Tranel's examples from section 4.2.1, the implementation at hand reproduces the theoretical results. Ranking SE lower than *COMPLONSET$_5$, `ApplyGen([{lŒ#pano#}|{sŒ#pano#}],` `Gen7)`, with `Gen7` being the final transducer, yields `l(Œ).p(a).n(o)` and `s(Œ).p(a).n(o)`.[15] With all rankings that include *ComplOnset$_3$ ≫ SE ≫ *ComplOnset$_5$, the result is `l(Œ).p(a).n(o)` and `sp(a).n(o)`, while SE ≫ *ComplOnset$_3$ yields `lp(a).n(o)` and `sp(a).n(o)`.

If SE is placed such that it is dominated by NoCoda, `ApplyGen([{da#lŒ#pano#}], Gen7)` preserves schwa (`d(a).l(Œ).p(a).n(o)`). In contrast, all rankings that incorporate SE ≫ NoCoda yield `d(a)l.p(a).n(o)`.

Concerning complex codas, if *ComplCoda$_2$ ≫ SE ≫ *ComplCoda$_3$, then `syr#dŒ#pɛrson#` is mapped to `s(y)rd.p(ɛ)r.s(o)n`, while the schwa in `ʒak#lŒ#sutjɛ#` is not deleted (`ʒ(a).kl(Œ).s(u).tj(ɛ)`). When ranking SE lower than *ComplCoda$_3$, the schwa in both phrases is retained (`s(y)r.d(Œ).p(ɛ)r.s(o)n`, `ʒ(a).kl(Œ).s(u).tj(ɛ)`). It is deleted in both, when SE is ranked higher than *ComplCoda$_2$ (`s(y)rd.p(ɛ)r.s(o)n`, `ʒ(a)kl.s(u).tj(ɛ)`).

As a side effect, the final transducer `Gen7` can also be used for syllable segmentation, no matter whether the input contains a schwa or not.

5.4 Implementation difficulties

Unfortunately, the generation of syllabified output does not work exactly as described above. When putting together Gen with a series of constraints, the network grows so large in size that Foma fails to handle it and runs out of memory. This typically happens when applying the `opt1` function. When decomposing `opt1` into individual parts and applying them one after the other, one observes that it is the extraction of the lower language after having added a star, changed the markup and permuted the stars that causes Foma to crash. This happens for instance with the Max(Schwa) constraint when SE is ranked directly before it. The network from which the lower-language automaton should be extracted has a size of 6.4 MB, about 11 200 states and 421 000 arcs. As a comparison, the application of `opt1` to SE before works fine; the transducer from which the lower side is taken has a size of 5.8 MB, about 10 000 states and 378 800 arcs. Similiar (or even worse) results are observed when running the same implementation with XFST to make sure that the error is not due to the Foma system.

As a consequence, in order to obtain the results given in section 5.3, `Gen` is first applied to a *toy lexicon*, i.e. a set of selected example phrases, which obviously yields considerably smaller networks that typically shrink in size with every constraint application. Having a finite toy lexicon, however, means that the transducer has to be recompiled every time a new phrase is added, which defeats the purpose of using finite-state methods. Usually, the generation of output from a given input is very efficient as the transducer is already compiled. The implementation of schwa/zero alternations in the framework of OT therefore has a rather illustrative character, showing the strengths and weaknesses in Tranel's (1999, 2000) analysis.

However, as already mentioned, the implementation involves a constraint-based syllabifier that compiles with the general `Gen` if all constraints are left out that are not necessarily needed for syllabification. After having tested it on a large-scale corpus, it can be used for syllable segmentation according to the analysis in Dell (1995).

[15] Appliciation of the (phonetic) module, which effects the deletions and deletes the word boundaries, is assumed for better readability.

The implementation furthermore has the conceptual weakness that a complete, strict hierarchy of constraints is required to compute optimal candidates. As can be seen in section 4.1, establishing such a hierarchy is not always straight forward, sometimes not even possible, and requires a lot of data. The *COMPLONSET and *COMPLCODA hierarchies for example have been assumed to be interspersed with each other in a certain way for the implementation. The assumptions match the examples given by Tranel (1999, 2000), but also fix language characteristics that are unattested in the literature. For instance, if the speaker that exhibits the implemented hierarchy permits /lp/-onsets (SE $\gg$ *COMPLONSET$_3$), he automatically also accepts /rd/-codas (*COMPLONSET$_3$ $\gg$ *COMPLCODA$_3$).

6 Conclusion

The paper at hand has formalized and implemented Tranel's (1999, 2000) OT approach to schwa/zero alternations in French. The syllable constraints given in his approach to illustrate the variable ranking of the central SE constraint have been completed. They now provide an OT model for syllable segmentation in French. Besides ONSET and NOCODA, the hierarchies *COMPLONSET$_n$ and *COMPLCODA$_n$ have been suggested, and they have been shown to correctly syllabify critical input phrases.

The implementation outlined in section 5 has served as a verification and debugging tool for the syllable constraints and the various rankings of SE given in Tranel (1999, 2000). As GEN produces all conceivable outputs for a given input, it is almost impossible to evaluate them at hand, and a computational device comes in handy.

By theoretical analyses and application of the implemented transducer to example phrases, Tranel's (1999, 2000) approach has been shown to correctly predict phrases containing one alternating schwa. Ranking SE low, preserves the schwa, while the higher SE moves, the more likely it is that schwa is deleted, even in a complex consonantal environment. It has furthermore been found that Tranel (1999, 2000) fails to generally account for phrases that contain more than one alternating schwa. In the forms that are optimal according to his approach, either all schwas surface or the deletions of schwa are maximized. Intermediate results are not predicted.

Additionally, using the Foma finite-state calculus, the paper has demonstrated how GEN can be represented as a transducer and how syllable constraints can be formulated as transducers that insert violation markers into input strings. The Matching Approach has been chosen for evaluating the candidates. It has been found that the networks that are created generally grow very large in size, and that a toy lexicon is therefore needed. When seeing the implementation as a debugging tool to verify linguistic analyses, this requirement, however, is not a big shortcoming.

Directions for future work

The presented work can be improved and extended in two main directions. From the linguistic point of view, Tranel's (1999, 2000) OT approach to schwa/zero alternations should be augmented such that it correctly predicts all output forms for phrases with multiple schwas. The presented syllable constraints could be made more accurate, so that they also account for exceptions in the general sonority patterns. Above all, segmentation into syllables should be tested on a large scale. The constraints that were primarily written to account for schwa/zero alternations could then be taken as an OT approach to syllabification of French. The corresponding implementation could be used as a syllabifier.

On the implementation side, more research should be done on why and when the networks grow so large that the Foma system is not able to handle them anymore. If they can be kept smaller, the implemented transducer could serve as an important component in a text-to-speech system that considers style and speech rate. One could furthermore try to find a way to evaluate candidates for unranked constraints.

References

Dell, F. (1973). *Les règles et les sons.* Herman, Paris.

Dell, F. (1995). Consonant clusters and phonological syllables in french. *Lingua 95*, pages 5–26.

Ellison, M. T. (1994). Phonological Derivations in Optimality Theory. In *Proceedings of the 15th International Conference on Computational Linguistics*, pages 1007–1013, Kyoto.

Féry, C. (2001). Markedness, Faithfulness, Vowel Quality and Syllable Structure in French. In *Linguistics in Potsdam No. 15.*

Gerdemann, D. (2009a). FinnOTMatching.fom. unpublished Matching Approach implementation of Karttunen's Finnish Prosody example in Foma.

Gerdemann, D. (2009b). Mix and Match Replacement Rules. to be published.

Gerdemann, D. and van Noord, G. (2000). Approximation and Exactness in Finite State Optimality Theory. In *Proceedings of COLING/Sigphon Workshop on Finite State Phonology.*

Goslin, J. and Frauenfelder, U. H. (2000). A Comparison of Theoretical and Human Syllabification. *Language and Speech*, 44(4):409–36.

Hulden, M. (2009). Foma: a finite-state compiler and library. In *Proceedings of the EACL 2009 Demonstrations Session*, pages 29–32, Athens, Greece. Association for Computational Linguistics.

Jetchev, G. I. (1997). *Ghost Vowels and Syllabification. Evidence from Bulgarian and French.* PhD thesis, Scuola Normale Superiore di Pisa.

Johnson, C. D. (1972). *Formal Aspects of Phonological Descriptions.* Mouton.

Kager, R. (1999). *Optimality Theroy.* Cambridge University Press.

Kaplan, R. M. and Kay, M. (1994). Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–79.

Karttunen, L. (1995). The Replace Operator. In *33th Annual Meeting of the Association for Computational Linguistics.*

Karttunen, L. (1996). Directed Replacement. In *34th Annual Meeting of the Association for Computational Linguistics.*

Karttunen, L. (1998). The Proper Treatment of Optimality in Computational Phonology. In *Finite-State Methods in Natural Language Processing*, pages 1–12. Ankara.

Karttunen, L. (2003). Finite-State Technology. In Mitkov, R., editor, *The Oxford Handbook of Computational Linguistics*, pages 339–57. Oxford University Press.

Kleene, S. C. (1956). Representation of Events in Nerve Nets and Finite Automata. In Shannon, C. E. and McCarty, J., editors, *Automata Studies*, pages 3–42. Princeton University Press.

McCarthy, J. J. (2002). *A Thematic Guide to Optimality Theory.* Cambridge University Press.

Prince, A. and Smolensky, P. (1993). *Optimality Theory: Constraint interaction in generative grammar.* Rutgers Optimality Archive. ROA version 8/2002.

Roche, E. and Schabes, Y. (1997). Introduction. In Roche, E. and Schabes, Y., editors, *Finite-State Language Processing*, pages 1–93. MIT Press.

Tranel, B. (1987a). French schwa and nonlinear phonology. *Linguistics*, 25:845–66.

Tranel, B. (1987b). *The Sounds of French.* Cambridge University Press.

Tranel, B. (1999). Optional Schwa Deletion: On Syllable Economy in French. In Authier, J.-M., Bullock, B. E., and Reed, L. A., editors, *Formal Perspectives on Romances Linguistics*, volume 185 of *Amsterdam Studies in the Theory and History of Linguistic Science*, pages 271–88. John Benjamins Publishing Company.

Tranel, B. (2000). Aspects de la phonologie du français et la théorie de l'optimalité. In *Langue française*, volume 126, pages 39–72.

Tranel, B. and Gobbo, F. D. (2002). Local Conjunction in Italian and French Phonology. In Wiltshire, C. R. and Camps, J., editors, *Romance Phonology and Variation*, volume 217 of *Amsterdam Studies in the Theory and History of Linguistic Science*, pages 191–218. John Benjamins Publishing Company.

Appendices

A Foma Operators

Operator	**Function**
[]	grouping parentheses
()	optionality
{}	concatinate symbols
\	term negation
+	Kleene plus
*	Kleene star
^n	n-ary concatinations
$_1$ .u	upper language
$_2$ .l	lower language
$^{-1}$ .i	inverse
$\neg$ $\sim$	complement
\$	containment operator
/	ignore operator
| $\cup$	union
& $\cap$	intersection
-	set minus
.P.	priority union
.o. $\circ$	composition
.O.	lenient composition
->	replacement
(->)	optional replacement
@->	directed replacement: left-to-right, longest match
_	replacement context specifier
||	replacement direction specifier: simultaneous
[..]	single-epsilon control in replacements
...	markup replacement control
Σ ?	'any' symbol
ε 0 []	epsilon symbol
$\emptyset$	empty language symbol
.#.	word boundary in replacement restrictions
_isfunctional	test if FST is functional
_isidentity	test if FST represents identity relations only

B Well-formedness tests for Gen

Basic definitions.

```
define true ε;
define false ∅;

define and(B1, B2) [ B1 B2 ];
define or(B1, B2) [ B1 | B2 ];
define not(B) [ ~B & true ];

define empty(L) not([L:true]₂);
define subset(A1,A2) empty(A1 & ~A2);
define equal(A1, A2) and(subset(A1,A2), subset(A2,A1));
```

Tests. `Gen0` is the Gen transducer that should be tested.

```
define MarkupDoesNotIncludeStar empty(markup & star);
define NoMarkupOrStarInInput empty(Gen0₁ & $[markup | star]);
define NoStarInOutput empty(Gen0₂ & $star);
define ContainsIdentity _isidentity(Gen0 .o. markup -> ε);
define MarkupOnlyInserted _isfunctional(Gen0⁻¹);
```

Taken from Gerdemann (2009a).

C Syllabification examples

Input	**Syllabified output**
ɛstryktœr#	(ɛ)s.tr(y)k.t(œ)r#
atla#	(a)t.l(a)#
abstrɛ#	(a)b.str(ɛ)#
atla#abstrɛ#	(a)t.l(a)#.(a)b.str(ɛ)#
lœ#ʒardɛ#	l(œ)#.ʒ(a)r.d(ɛ)#
yn#animasjo#	(y).n#(a).n(i).m(a).sj(o)#
pano#	p(a).n(o)#
arɔdi#	(a).r(ɔ).d(i)#
ɔbsɛn#	(ɔ)b.s(ɛ)n#
akro#	(a).kr(o)#
mɛrkrœdi#	m(ɛ)r.kr(œ).d(i)#
la#trwa#	l(a)#.trw(a)#
da#lœ#retablisma#	d(a)#.l(œ)#.r(e).t(a).bl(i)s.m(a)#
da#lœr#etablisma#	d(a)#.l(œ).r#(e).t(a).bl(i)s.m(a)#
ɛkstaz#	(ɛ)k.st(a)z#
astral#	(a)s.tr(a)l#
anovrjɛ#	(a).n(o).vrj(ɛ)#

D Foma Coda

To include the code, some utf8 symbols had to be replaced because of encoding problems. `O`, `A`, `H`, `Z` and `N` have originally been ɔ, ɑ, ɥ, ʒ and ɲ.

```
## french.fom
## Miriam Käshammer

## Lexicon
define TestWords [
       {da#lŒ#pano#}|
       {la#trwa#}|
       {lŒ#pano#}|
       {sŒ#pano#}|
       {avɛk#lŒ#pano#}|
       {syr#dŒ#pɛrson#}|
       {pa#dŒ#skrypyl#}|
       {la#vɛst#dŒ#pol#}|
       {pa#dŒ#sŒ#kreta#}|
       {mɛrkrœdi#}|
       {abstrɛ#}|
       {kwartz#} |
       {ɛstryktœr#}|
       {atla#}|
       {atla#abstrɛ#}|
       {lŒ#Zardɛ#}|
       {yn#animasjo#}|
       {pano#}|
       {arOdi#}|
       {akro#}|
       {il#tŒ#lŒ#dŒmad#}|
       {la#tɛr#sŒ#vo#bjɛ#}|
       {pur#sŒ#pɛNe#}|
       {pur#tŒ#pɛNe#}|
       {Avi#dŒ#tŒ#lŒ#dŒmAde#} |
       {ZŒ#nŒ#sɛ#pa#}|
       {ZŒ#noz#pa#}|
       {sogrŒny#} |
       {da#lŒ#retablisma#}|
       {da#lœr#etablisma#}|
       {yn#pat#dŒ#rŒnar#}|
       {yn#kø#dŒ#rŒnar#}|
       {Zak#sŒku#la#braʃ#}|
       {Zak#lŒ#sutjɛ#}|
       {ɛl#tŒ#dŒmad#}|
       {Obsɛn#} |
       {ɛkstaz#} |
       {astral#} |
       {anovrjɛ#}|
       {abstrɛ#}
];

###################################################################
######################## Basic definitions ########################
###################################################################

## Vowels
define Schwa Œ;
define stableSchwa œ;
define BackV [u | o | O | A];
define FrontV [i | y | e | ø | %ɛ | a | stableSchwa];
define VWithoutSchwa [BackV | FrontV];
define V [ VWithoutSchwa | Schwa];

## Consonants
define Glide [H | w | j];
define Obstruent [ p | b | f | v | k | g | t | d | s | z | ʃ | Z ];
define Liquid [ l | r ];
define Nasal [ m | n | N | ŋ ];
define Con [Obstruent | Liquid | Nasal ];
define C [Glide | Con];

define Seg [V | C];

define WB %#;                     # Word Boundary

##  Syllable Markers
define rNuc %);                   # Nucleus
define lNuc %(;                   # Nucleus
define SB %.;                     # Syllable boundary
define Nuc [rNuc | lNuc];
define Syll [ Nuc | SB ];

## Edit Markers
define Del %%D;                  # Deletion marker
define Edit [Del];

define DeletedSeg [Del Seg];

define markup [Edit | Syll];

## Violation marker
define star %*;


###################################################################
############################## GEN ################################
###################################################################

# Insert deletion markers
define DeleteSeg [
       [..] (->) Del || _ Seg
];

# Insert syllable markers
define SOnset [[DeletedSeg]* C]*;
define SCoda [DeletedSeg* C ]*;
define SNucleus [lNuc DeletedSeg* V rNuc];
define Syllable [SOnset SNucleus SCoda]/WB;

define InsertSyll [
       DeletedSeg* V @-> lNuc ... rNuc || [.#.|\Del] _
       .o.
       Syllable -> ... SB || _ Syllable
       .o.
       ~$[SB WB]
];


define Gen0 [
       [Seg | WB]*
       .o.
       ~$markup
       .o.
       ~$star
       .o.
       DeleteSeg
```

```
        .o.
        InsertSyll
];

############################ TESTING #############################

## tests taken from Gerdemann, FinnOTMatching.fom

define true ε;
define false ~$true; #empty language

define and(B1, B2) B1 B2;
define or(B1, B2) B1 | B2;
define not(B) ~B & true;

define empty(L) not([L:true]_2);
define subset(A1,A2) empty(A1 & ~ A2);
define equal(A1, A2) and(subset(A1,A2), subset(A2,A1));

## DG: If all these tests succeed, then the FST Gen0 is well-formed and it
## is safe to represent Gen as the FSA Gen0_2. The last test,
## MarkupOnlyInserted, says essentially that Gen_0 should not (even
## optionally) turn some input symbols into markup symbols.
define MarkupDoesNotIncludeStar empty(markup & star);
define NoMarkupOrStarInInput empty(Gen0_1 & $[markup | star]);
define NoStarInOutput empty(Gen0_2 & $star);
define ContainsIdentity _isidentity(Gen0 .o. markup -> ε);
define MarkupOnlyInserted _isfunctional(Gen0^-1);

## DG: Represent Gen0 as an FSA. Since input symbols and markup symbols are
## clearly distinguished, this FSA can represent an FST.
define Gen Gen0_2;

## DG: We turn the FSA Gen back into an FST by removing markup symbols
## from the input side.
define Unflatten(X) [[markup | star] -> ε]^-1 .o. X;

## DG: ApplyGen allows the FSA Gen to be run as if it were an FST.
define ApplyGen(Lex, G) [Lex .o. Unflatten(G)]_2;


##################################################################
```

```
####################### Matching Approach ########################
##################################################################

## taken from Gerdemann, FinnOTMatching.fom

define Pardon(X) [X .o. star -> 0]_2;
define ChangeMarkup(X)  [X .o. [? | ε:markup | markup:ε]*]_2;
define AddStar(X) [X .o. [[?* ε:star]+ ?*]]_2;

## DG: It might be better to combine ChangeMarkup and AddStar so as to
## avoid the intermediate step of taking the lower language:
define msig ? | ε:markup | markup:ε;
define AddStarAndChangeMarkup(X)
   [X .o. [[msig* ε:star]+ msig*]]_2;

define ksig \markup | markup:ε;
define AddStarAndDeleteMarkup(X)
   [X .o. [[ksig* ε:star]+ ksig*]]_2;


## DG: Approximate permution of stars
define permuteStarLeft [ε:star [?-star]* star:ε];
define permuteStarRight [star:ε [?-star]* ε:star];
define permuteStars [?* [permuteStarLeft | permuteStarRight]]* ?*;
define permute0(X) X;
define permute1(X) [X .o. permuteStars]_2;
define permute2(X) [X .o. permuteStars .o. permuteStars]_2;
define permute3(X) [X .o. permuteStars .o. permuteStars .o. permuteStars]_2;
define permute4(X)
   [X .o. permuteStars .o. permuteStars .o. permuteStars .o. permuteStars]_2;

define opt0(Starred)
   Pardon(Starred -  AddStarAndChangeMarkup(Starred));
define opt1(Starred)
   Pardon(Starred -  permute1(AddStarAndChangeMarkup(Starred)));
define opt2(Starred)
   Pardon(Starred -  permute2(AddStarAndChangeMarkup(Starred)));
define opt3(Starred)
   Pardon(Starred -  permute3(AddStarAndChangeMarkup(Starred)));
define opt4(Starred)
   Pardon(Starred -  permute4(AddStarAndChangeMarkup(Starred)));
```

```
## DG: The exactness test as described in Gerdemann & van Noord (2000)
define isOptimized(Starred) _isfunctional([Unflatten(Starred) .o. [\star -> ε]]);


## DG: Note that a star-constraint should be a transducer that accepts any
## string as input and does nothing more than to add stars to this
## input. One might also require that the input to the constraint not
## contain stars, but this is not necessary since each Gen is
## star-free in its output.
define ConstraintWellFormed(C)
   and(equal(C_1, ?*), _isidentity(~$star .o. C .o. [star -> ε]));




####################################################################
###################### Phonetic Module #############################
####################################################################

## This Module effects the deletions and removes the word boundaries.
## It could also remove the syllable markers or determine the
## realization of schwa.

define RemoveDel [ DeletedSeg -> 0 ];
define RemoveWB [WB -> 0];

define PhoneticModule [
       RemoveDel
       .o.
       RemoveWB
];

## new ApplyGen
define ApplyGenMod(Lex,Gen,Mod) [ApplyGen(Lex,Gen) .o. Mod].l;

####################################################################
######################### Constraints ##############################
####################################################################

## Input segments must have output correspondents
define MAX(X) [
       [ Del X ] -> ... star
];

## MaxC: Input Consonants must have output correspondents
define MaxC MAX(C);
define MaxCWellFormed ConstraintWellFormed(MaxC);

## MaxV: Input Vowels except Schwa must have output correspondents
define MaxV MAX(VWithoutSchwa);
define MaxVWellFormed ConstraintWellFormed(MaxV);

## MaxSchwa: Input Schwas must have output correspondents
define MaxSchwa MAX(Schwa);
define MaxSchwaWellFormed ConstraintWellFormed(MaxSchwa);

## Onset: Syllables must have an onset.
define Onset [..] -> star || [SB | .#.] DeletedSeg*/WB _ lNuc;
define OnsetWellFormed ConstraintWellFormed(Onset);

## SE: Syllable Economy (less is better)
define SE [
       SB -> ... star
];
define SEWellFormed ConstraintWellFormed(SE);


## MComplexOnset: Onsets have rising sonority.

define sCCOnset [s [Obstruent-s] Liquid];
define CCGOnset [Obstruent Liquid Glide];
define sCOnset [s [[Obstruent-s] | Nasal]];
define CGOnset [Con Glide];
define ObNOnset [Obstruent Nasal];
define ExOnset [{lp}];

# OBLI: Sequence of obstruent and liquid (Dell 1995, p.7)
define OBLI [[[Obstruent & ~[s | z | ʃ | Z]] Liquid]  & ~[{tl}|{dl}]];
define notOBLI [Obstruent Liquid] & ~OBLI;


define MComplOnset1 [
       [[[C^4 C*] | [C^3 & ~[sCCOnset|CCGOnset]]]/DeletedSeg]/WB
```

```
                    -> ... star || [SB|.#.] _ lNuc
];
define MComplOnset2 [
       [[C^2 & ~[ExOnset|ObNOnset|notOBLI|sCOnset|CGOnset|OBLI]]/DeletedSeg]/WB
                    -> ... star || [SB|.#.] _ lNuc
];
define MComplOnset3 [
       [[notOBLI|ObNOnset|ExOnset]/DeletedSeg]/WB
                    -> ... star || [SB|.#.] _ lNuc
];
define MComplOnset4 [
       [sCCOnset/DeletedSeg]/WB
                    -> ... star || [SB|.#.] _ lNuc
];
define MComplOnset5 [
       [sCOnset/DeletedSeg]/WB
                    -> ... star || [SB|.#.] _ lNuc
];
define MComplOnset6 [
       [[CCGOnset|CGOnset|OBLI]/DeletedSeg]/WB
                    -> ... star || [SB|.#.] _ lNuc
];

define MComplOnset1WellFormed ConstraintWellFormed(MComplOnset1);
define MComplOnset2WellFormed ConstraintWellFormed(MComplOnset2);
define MComplOnset3WellFormed ConstraintWellFormed(MComplOnset3);
define MComplOnset4WellFormed ConstraintWellFormed(MComplOnset4);
define MComplOnset5WellFormed ConstraintWellFormed(MComplOnset5);
define MComplOnset6WellFormed ConstraintWellFormed(MComplOnset6);


## NoCoda: Syllables must not have a coda.
define NoCoda [
       [C+/DeletedSeg]/WB -> ... star || rNuc _ [SB | .#.]
];
define NoCodaWellFormed ConstraintWellFormed(NoCoda);


## MComplexCoda

define MCCCCoda [
       [[[[ C C C+ ]/DeletedSeg ]/WB ] & ~[ C C C+ WB ]]
```

```
];
define CCCoda [
       [[[Liquid [Obstruent|Nasal|Liquid]] | [s [Obstruent-s]]]/DeletedSeg]/WB
];
define MCCCoda [
       [[[C C]/DeletedSeg]/WB] & ~CCCoda & ~[C C WB]
];
define MComplCoda1 [
       MCCCCoda -> ... star || rNuc _ [SB | .#.]
];
define MComplCoda2 [
       MCCCoda -> ... star || rNuc _ [SB | .#.]
];
define MComplCoda3 [
       CCCoda  -> ... star || rNuc _ [SB | .#.]
];

define MComplCoda1WellFormed ConstraintWellFormed(MComplCoda1);
define MComplCoda2WellFormed ConstraintWellFormed(MComplCoda2);
define MComplCoda3WellFormed ConstraintWellFormed(MComplCoda3);

##############################################################################
##############################################################################




## frenchRankings.fom
## Miriam Käshammer

## Import the definitions.
source french.fom


###################################################################
######################### Hierarchy  ############################
###################################################################

## Use Gen for the general implementation, and
## GEN for the toy lexicon.
define GEN ApplyGen(TestWords, Gen);

echo
```

```
echo Gen1: MaxC
define MaxCStarred [GEN .o. MaxC]_2;
define Gen1 opt0(MaxCStarred);
define Gen1IsOptimized isOptimized([Gen1 .o. MaxC]_2);
define Gen1WasAlreadyOptimized isOptimized(MaxCStarred);
regex ApplyGen(TestWords, Gen1);
#print words

echo
echo Gen2: MaxV
define MaxVStarred [Gen1 .o. MaxV]_2;
define Gen2 opt0(MaxVStarred);
define Gen2IsOptimized isOptimized([Gen2 .o. MaxV]_2);
define Gen2WasAlreadyOptimized isOptimized(MaxVStarred);
regex ApplyGen(TestWords, Gen2);
#print words

echo
echo Gen41: MComplexOnset
define MComplOnset1Starred [Gen2 .o. MComplOnset1]_2;
define Gen41 opt0(MComplOnset1Starred);
define Gen41IsOptimized isOptimized([Gen41 .o. MComplOnset1]_2);
define Gen41WasAlreadyOptimized isOptimized(MComplOnset1Starred);
regex ApplyGen(TestWords, Gen41);
#print words

echo
echo Gen61: MComplexCoda
define MComplCoda1Starred [Gen41 .o. MComplCoda1]_2;
define Gen61 opt0(MComplCoda1Starred);
define Gen61IsOptimized isOptimized([Gen61 .o. MComplCoda1]_2);
define Gen61WasAlreadyOptimized isOptimized(MComplCoda1Starred);
regex ApplyGen(TestWords, Gen61);
#print words

echo
echo Gen42: MComplexOnset
define MComplOnset2Starred [Gen61 .o. MComplOnset2]_2;
define Gen42 opt0(MComplOnset2Starred);
define Gen42IsOptimized isOptimized([Gen42 .o. MComplOnset2]_2);
define Gen42WasAlreadyOptimized isOptimized(MComplOnset2Starred);
regex ApplyGen(TestWords, Gen42);
```

```
#print words

echo
echo Gen62: MComplexCoda
define MComplCoda2Starred [Gen42 .o. MComplCoda2]$_2$;
define Gen62 opt0(MComplCoda2Starred);
define Gen62IsOptimized isOptimized([Gen62 .o. MComplCoda2]$_2$);
define Gen62WasAlreadyOptimized isOptimized(MComplCoda2Starred);
regex ApplyGen(TestWords, Gen62);
#print words


######### ----------------- SE border----------------- ###############

############################### 1Gen7 ################################

echo
echo 1Gen7: SE >> Max(Schwa)

define 1MComplOnset3Starred [Gen62 .o. MComplOnset3]$_2$;
define 1Gen43 opt0(1MComplOnset3Starred);
define 1Gen43IsOptimized isOptimized([1Gen43 .o. MComplOnset3]$_2$);

define 1MComplCoda3Starred [1Gen43 .o. MComplCoda3]$_2$;
define 1Gen63 opt0(1MComplCoda3Starred);
define 1Gen63IsOptimized isOptimized([1Gen63 .o. MComplCoda3]$_2$);

define 1MComplOnset4Starred [1Gen63 .o. MComplOnset4]$_2$;
define 1Gen44 opt0(1MComplOnset4Starred);
define 1Gen44IsOptimized isOptimized([1Gen44 .o. MComplOnset4]$_2$);

define 1MComplOnset5Starred [1Gen44 .o. MComplOnset5]$_2$;
define 1Gen45 opt0(1MComplOnset5Starred);
define 1Gen45IsOptimized isOptimized([1Gen45 .o. MComplOnset5]$_2$);

define 1NoCodaStarred [1Gen45 .o. NoCoda]$_2$;
define 1Gen5 opt0(1NoCodaStarred);
define 1Gen5IsOptimized isOptimized([1Gen5 .o. NoCoda]$_2$);

define 1OnsetStarred [1Gen5 .o. Onset]$_2$;
define 1Gen3 opt0(1OnsetStarred);
define 1Gen3IsOptimized isOptimized([1Gen3 .o. Onset]$_2$);
define 1MComplOnset6Starred [1Gen3 .o. MComplOnset6]_2;
define 1Gen46 opt0(1MComplOnset6Starred);
define 1Gen46IsOptimized isOptimized([1Gen46 .o. MComplOnset6]_2);

define 1SEStarred [1Gen46 .o. SE]_2;
define 1Gen100 opt0(1SEStarred);
define 1Gen100IsOptimized isOptimized([1Gen100 .o. SE]_2);

define 1MaxSchwaStarred [1Gen100 .o. MaxSchwa]_2;
define 1Gen7 opt0(1MaxSchwaStarred);
define 1Gen7IsOptimized isOptimized([1Gen7 .o. MaxSchwa]_2);


############################### 2Gen7 ################################

echo
echo 2Gen7: SE >> MComplOnset6

define 2MComplOnset3Starred [Gen62 .o. MComplOnset3]_2;
define 2Gen43 opt0(2MComplOnset3Starred);
define 2Gen43IsOptimized isOptimized([2Gen43 .o. MComplOnset3]_2);

define 2MComplCoda3Starred [2Gen43 .o. MComplCoda3]_2;
define 2Gen63 opt0(2MComplCoda3Starred);
define 2Gen63IsOptimized isOptimized([2Gen63 .o. MComplCoda3]_2);

define 2MComplOnset4Starred [2Gen63 .o. MComplOnset4]_2;
define 2Gen44 opt0(2MComplOnset4Starred);
define 2Gen44IsOptimized isOptimized([2Gen44 .o. MComplOnset4]_2);

define 2MComplOnset5Starred [2Gen44 .o. MComplOnset5]_2;
define 2Gen45 opt0(2MComplOnset5Starred);
define 2Gen45IsOptimized isOptimized([2Gen45 .o. MComplOnset5]_2);

define 2NoCodaStarred [2Gen45 .o. NoCoda]_2;
define 2Gen5 opt0(2NoCodaStarred);
define 2Gen5IsOptimized isOptimized([2Gen5 .o. NoCoda]_2);

define 2OnsetStarred [2Gen5 .o. Onset]_2;
define 2Gen3 opt0(2OnsetStarred);
define 2Gen3IsOptimized isOptimized([2Gen3 .o. Onset]_2);
define 2SEStarred [2Gen3 .o. SE]$_2$;
define 2Gen100 opt0(2SEStarred);
define 2Gen100IsOptimized isOptimized([2Gen100 .o. SE]$_2$);

define 2MComplOnset6Starred [2Gen100 .o. MComplOnset6]$_2$;
define 2Gen46 opt0(2MComplOnset6Starred);
define 2Gen46IsOptimized isOptimized([2Gen46 .o. MComplOnset6]$_2$);

define 2MaxSchwaStarred [2Gen46 .o. MaxSchwa]$_2$;
define 2Gen7 opt0(2MaxSchwaStarred);
define 2Gen7IsOptimized isOptimized([2Gen7 .o. MaxSchwa]$_2$);


##############################################################################
##############################################################################

## 3Gen7 - 9Gen7 are defined following the pattern above.

##############################################################################

## Some tests

echo
echo lowest SE ranking (1)
regex ApplyGenMod(
    {ɛstryktœr#}|
    {atla#abstrɛ#}|
    {lŒ#Zardɛ#}|
    {yn#animasjo#}|
    {yn#pat#dŒ#rŒnar#}|
    {yn#kø#dŒ#rŒnar#}|
    {Avi#dŒ#tŒ#lŒ#dŒmAde#}|
    {da#lŒ#pano#}|
    {lŒ#pano#}|
    {sŒ#pano#}, 1Gen7, PhoneticModule);
print words

echo
echo SE ranking (2)
regex ApplyGenMod(
    {yn#pat#dŒ#rŒnar#}|
    {yn#kø#dŒ#rŒnar#}|
    {Avi#dŒ#tŒ#lŒ#dŒmAde#}|
    {da#lŒ#pano#}|
    {lŒ#pano#}|
    {sŒ#pano#}, 2Gen7, PhoneticModule);
print words
```