DOI: 10.17469/O2101AISV000030

ANTONIO ORIGLIA, FRANCESCO CANGEMI, FRANCO CUTUGNO

WORDEN: a PYTHON interface to automatic (non)word generation

WORDEN is a PYTHON implementation of a knowledge engine containing data on the phonological structure of Italian words and on their frequency. These data are imported from the PHONITALIA database and used with a rules base to provide a formal description of the operations needed to generate words, nonsense words and phonological neighbourhoods for speech production and perception studies. The knowledge engine supports a wide number of queries from the command line but can also be used via a specifically designed graphical user interface. In this paper, we report the formal rules included in WORDEN and provide usage examples to illustrate the capabilities of the tool.

1. *Introduction*

A knowledge engine, sometimes called an *expert system*, is a software module that, given a set of known *facts* (knowledge base) and a set of inference *rules* (rules base), is able to:

- provide yes/no answers to user *queries* (e.g. "Is <pane> a word?");
- find all the solutions that satisfy a pattern described in the rules (e.g. "Which words have a CVCV structure?").

Logic programming is not new in the field of Natural Language Processing. Nugues (2014) provides a deep overview of the application of PROLOG to NLP in general. In computational phonology, logic programming has been applied, for example, to build computational models of Autosegmental Phonology (Bird, Klein, 1990; Bird, Ellison, 1994) but its usage in the field is documented in the '80s, too (see Grishman, 1986). In the last years, logic programming has found applications in the field of computational semantics (Sowa, 2014; Stone et al., 2014) for the exploration of complex semantic networks. Usage of logic programming is typically considered as opposed to nowadays popular machine learning approaches, where supervised analysis of large corpora is used to obtain a set of probabilistic models to classify objects. The two approaches are, in truth, complementary. Logic programming allows the researcher to define and easily test formal theories but it does not support data analysis during the definition of such theories, which can be very complex in the case of partially known phenomena. Machine learning, on the other hand, allows to easily analyse and detect regularities in large datasets to obtain statistical models. These, however, are often hard to interpret and based on probabilities, thus being less suitable for the definition of formal theories.

In the following, we present WORDEN, a logic programming-based engine that assists linguists in finding suitable stimuli for production and perception experiments. In Section 2, we provide some examples of the research areas in linguistics and psycholinguistics that can benefit from such an engine. The details of the engine (knowledge base, rules base, graphical user interface) are presented in Section 3. Section 4 offers concluding remarks and sketches avenues to future expansion.

2. *Applications*

Much experimental work in linguistics and psycholinguistics requires subjects to utter words or sentences (production) or to react to stimuli presented orthographically or auditorily (perception). Both lines of work require a careful selection of the stimuli to be used.

In intonation research, for example, the experimenter is often concerned with the analysis of fundamental frequency contours extracted from speakers' recordings. However, f0 contours are highly sensitive to segmental perturbations (Lehiste, 1970; Di Cristo, 1985). At best, such microprosodic effects make the collected material highly dependent on the stimuli used in the elicitation task (e.g. vowels have different intrinsic pitch); at worst, they obscure the phenomena under scrutiny (e.g. voiceless segments do not allow for f0 estimation). For these reasons, many intonologists working with utterances elicited from scripted material devote great attention to the selection of experimental stimuli, e.g. by using sonorant segments only, or by restricting stressed vowels to one single type. The following test sentences for data elicitation, drawn from three studies on Neapolitan Italian, exemplify such practices:

1. Vedrai la bella mano di Mammola domani? (D'Imperio, 2001)
   Will you see Mammola's beautiful hand tomorrow?
2. Il benevolo manovale beveva vino (Petrone, 2008)
   The benevolent labourer drank wine

3a. Danilo vola da Roma (Cangemi, D'Imperio, 2013)
   Danilo takes the flight from Rome

3b. Serena vive da Lara
   Serena lives at Lara's

3c. $[\text{CV.C\`V.CV}]_{S}\,[\text{C\`V.CV}]_{V}\,[\text{CV}][\text{C\`V.CV}]_{IO}$

The examples show that it is indeed possible to minimize microprosodic perturbations by design (i.e., by choosing appropriate target words). However, they also show that this outcome has a cost – either in terms of the semantic and pragmatic plausibility of the stimuli (example 1), of the familiarity of the lexemes employed (example 2) or of occasional lapses in constraint satisfaction (example 3b, with its initial voiceless fricative). Examples 3a-b illustrate the case in which the corpus to elicit features various sentences sharing phonological and syntactic make-up (3c), i.e. a trisyllabic paroxitone noun as subject, followed a bisyllabic paroxitone verb, and by a trisyllabic paroxitone prepositional phrase as indirect object. In this

case, the researcher has to find actual words in the lexicon to fill in the slots in the sentence template. Such a template-based stimulus search is particularly costly for the researcher, if performed manually. WORDEN is a first step towards the automatization of such searches.

Careful stimulus selection is also required for psycholinguistic studies investigating spoken-word recognition (e.g. Luce, Pisoni, 1998) and the structure of the mental lexicon. In the past twenty years, research in this vein has documented the importance of phonological neighbourhood effects in speech production and perception. Neighbourhoods can be understood as "conglomeration of words that are highly similar to one another along a critical characteristic" (Marian et al., 2012: 1). Phonological neighbours are thus the words which can be constructed by substituting, inserting or deleting one or more phonemes from a source word (or similar revisions of Coltheart et al., 1977 *N* metric). For example, close phonological neighbours of Eng. <cat> /kæt/ include <bat> /bæt/, <scat> /skæt/ and <at> /æt/.

The individuation of phonological neighbours for words in any given language is a task that can only be performed using computerized tools. Such tools exist for Dutch, English, French, German and Spanish words (CLEARPOND, see Marian et al., 2012), and for Basque, Dutch, English, French, German, Serbian, Spanish, and Vietnamese nonwords (WUGGY, see Keuleers, Brysbaert, 2010) but, to our knowledge, not for Italian words or nonwords. This state of affairs greatly limits the possibility of exploring the structure of the mental lexicon in Italian through the prism of phonological neighbourhood.

A recent example of such limitations is provided by the study on phonological analogy in Neapolitan Italian by Cangemi & Barbato (2014). The study tested the hypothesis that mid-vowel selection in nonce words is guided by the existence of phonologically similar actual word in the lexicon. For example, given the existence of the Neapolitan Italian words /ɛrba/ ('grass') and /aʃɛrba/ ('unripe', fem.), subjects were expected to utter /tɛrba/ (rather than /terba/) when asked to read aloud the nonword <TERBA>. Crucially, in the absence of a tool for the calculation of phonological neighbours, the notion of *phonologically similar words* had to be operationalized as *words with the same (poetical) rhyme*. While this choice made it possible to use an invert dictionary to find the target words, it also motivated some unexpected results which had to be reinterpreted post-hoc. This is the case of the nonword <PERMO>, which was expected to be uttered as /permo/ under the influence of /fermo/ 'closed', and /ermo/ 'lonely, remote' (which is however a relatively infrequent word). <PERMO> was however most often uttered as /pɛrmo/, probably under the influence of /pɛrno/ 'pivot', /pɛrdo/ 'to lose', and the very frequent /pɛrso/ 'lost' – words which do not have the same (poetical) rhyme as the stimulus, but are among its phonological neighbours.

By using a syllabic template to generate (non)words, and by listing their phonological neighbours along with their frequency of usage, WORDEN provide linguists and psycholinguists with a powerful tool to select stimuli for their experiments.

3. *WORDEN*

WORDEN is a library of functions to generate words and non-sense words based on a PYTHON module for knowledge engines development called *Python Knowledge Engine* (PYKE) (Frederiksen, 2008). We chose an declarative programming solution integrated in Python, rather than relying on external engines, to improve portability. It is also capable of providing insight about their phonological neighbours and the associated frequency information. In this section, we describe how the knowledge and the rules bases used in WORDEN were built. Examples on the queries the system can answer are provided: for each set of facts and rules, its formal definition is reported together with examples of queries and the system's output. We provide here a short summary of how logic programming works and how the examples are provided. Assume the following situation:

– Bill is Tom's parent;
– Bill is Will's sibling;
– Will is Richard's parent.

This is represented by a set of *facts* in the knowledge base. Genealogy states that the children of two siblings are cousins and that if one person is sibling of another the opposite is also true. This is represented by *rules* that can be used by a knowledge engine to infer the fact that Tom and Richard are cousins. We then query the system to ask it if

– Will is Bill's sibling (answer = yes);
– Who are Tom's cousins, if any (answer = Richard);
– Which parents are known by the system (answer = Bill and Will).

From the syntactic point of view, a *free variable* (an object the system can assign values to provide solutions to queries) is represented by a name starting with an uppercase character (e.g. X, Prefix, etc.). The underscore character represents a "don't care" variable, to which any possible value can be assigned during the solution of the query. The graphical representation of the example described above is organized as follows. In the left column, we display both facts (top) and rules (bottom). On the right side, lines starting with "?" indicate a user's query. Lines starting with ">>" indicate the system's feedback.

parent(Bill, Tom)	*? sibling(Will, Bill)*
parent(Will, Richard)	>> yes
sibling(Bill, Will)	
	? cousin(Tom, X)
sibling (X, Y) : - sibling(Y, X)	>> X= Richard
cousin (X, Y) : - parent(Z, X),	
parent(K, Y),	*? parent(X, _)*
sibling(Z, K)	>> *X= Bill*
	>> *X= Will*

Normally, an expert system is queried by command line to allow wide expressivity to the user. To help users access a closed set of functionalities related to words and nonsense words generation without using the command line, WORDEN also comes with a Graphical User Interface.

3.1 Knowledge base

The current version of WORDEN contains a knowledge base describing Italian only. This has been automatically generated starting from the PHONITALIA database (Goslin et al., 2013, http://www.phonitalia.org/). For each term in PHONITALIA, the grammatical categories it is assigned to are imported as part of the *wordGramCat* fact together with the frequency of occurrence of the word with that particular usage. The symbols used to represent grammatical categories are the same used in PHONITALIA and the syllabic structure of the word is imported as part of the *word* fact together with its phonological representation. For example, the word *abate* (Eng. 'abbot') is represented in WORDEN as follows (see the PHONITALIA guidelines for the meaning of the labels describing the grammatical categories). When asked, the system knows *abate* is a word.

wordGramCat(abate, S, 8)	*? word(abate, _, _)*
wordGramCat(abate, E, 4)	>> yes
wordGramCat(abate, E_IN_Ea, 2)	
wordGramCat(abate, S_IN_Ea, 1)	*? word(argine, _, _)*
word(abate, (a,ba,te), (a,b,a,t,e))	*>> Cannot prove*

The reader should note that, if asked about an atom being a word, if the knowledge base does not contain any information about that particular atom being a word, the system's output specifies that, with the current knowledge, it is not possible to prove that the atom is a word, implying that the answer may still be *yes,* should more knowledge be provided. In WORDEN, asking the system if an atom is a word actually means asking if the word is present in PHONITALIA. Thus, given the knowledge base represented in the left column of the example, the query on whether *argine* is a word yields the feedback *Cannot prove*.

The knowledge base of WORDEN also imports data concerning lemmas from PHONITALIA. However, in order to reduce the amount of data to be loaded, we only imported the words that are not lemmas of themselves. This way, it is later possible to introduce a rule stating that, if a word does not have a lemma in WORDEN's knowledge base, its lemma is the word itself.

	? lemma(X, abati)
lemma(abate, abati)	*>> X= abate*
lemma(X, X) :- not(lemma(_, X))	*? lemma(X, abate)*
	>> X= abate

In order to manage queries related to the words' CV structures, WORDEN's knowledge base contains information differentiating the symbols that can form a word in PHONITALIA between consonants and vowels.

vowel(a)	*? vowel(a)*
...	>> yes
consonant(t)	
...	*? consonant(9)*
	>> Cannot prove

3.2 Rules base

Rules are the most important part of WORDEN. A small set of rules can give the system the capability of inferring a large set of facts. Rules are often defined inductively: first, a simple base case is defined and, then, an induction step is introduced attempting to subdivide a complex problem into smaller ones while checking if it is possible to reach the base case.

In WORDEN's knowledge base we have imported the phonological representation of PHONITALIA words and we have introduced knowledge about PHONITALIA symbols representing consonants or vowels. This allows WORDEN to infer the CV structure of lists of PHONITALIA symbols using the following rules:

– The CV structure of an empty sequence is empty, too (base case)
– The CV structure of a sequence of symbols is composed by the symbol representing whether the first symbol of the list is a consonant or a vowel followed by the CV structure of the rest of the list.

In the following examples, we will indicate a list as [*Head, *Rest*], where *Head* is a variable containing the first symbol in the list while **Rest* is a list containing the same symbols of the whole list minus the first one. [] indicates an empty list.

cvStructure	*([], [])*	
cvStructure	*([Head, *Rest],*	
	*[CvHead, *CvRest]) :-*	
	CvHead = 'V'	*? cvStructure([l, a], X)*
	vowel(Head)	*>> X= [C, V]*
	*cvStructure(*Rest, *CvRest)*	
cvStructure	*([Head, *Rest],*	*? cvStructure(X, [C, V])*
	*[CvHead, *CvRest]) :-*	*>>X=[ba, da, ..., bi, di...]*
	CvHead = 'C'	
	consonant(Head)	
	*cvStructure(*Rest, *CvRest)*	

To better illustrate the example, let's consider the list of symbols representing the article *la*.

1. Of course, [l, a] does not represent an empty list so the first rule cannot be applied and the answer cannot be [];
2. The second rule checks if the first element of the list is a vowel: /l/ is not a vowel so it not possible to say that the answer is [V] followed by the structure of the list [a];
3. The third rule checks if the first element of the list is a consonant: /l/ is a consonant so it is possible to say that the answer is [C] followed by the structure of the list [a];
4. To complete the solution at point 3, the same set of rules is applied to the sublist [a]: since the first element is a vowel, the answer to the subproblem is [V] followed by the structure of an empty list [];
5. The CV structure of [] is [] by the first rule, so the solution to the subproblem of establishing the structure of [a] is [V] prepended to [], thus [V];
6. Since the solution to the problem of establishing the structure of [l, a] is [C] prepended to the structure of [a], the answer to the original query is [C, V] and is provided as output.

This way, in WORDEN a formal definition of the concept of *CV structure* is present. This provides the following advantages:
– It is not necessary to specify the structure of every word;
– It is possible to get the structure of any list of symbols (not only words in the knowledge base);
– Given an inductive definition of the concept, the knowledge engine is able to use the same rules to find all the possible solutions that lead to a specific structure.

In the example shown above, we query the system to get all the possible sequences of consonants followed by vowels. It is also possible to use the CV rules to get all the words or all the nonsense words with a specific CV structure. This is accomplished by adding two rules covering the different situations.

```
    cvNonce([CvHead, *CvRest],
[Head, *Rest]) :-
cvStructure([Head, *Rest],                  ? cvNonce([C, V], X)
[CvHead, *CvRest])                          >> X= [[k, u], [w, o], [z, u]...]
    not(word(_, _, [Head, *Rest]))

    cvWord([CvHead, *CvRest],               ? cvWord([C, V], X)
[Head, *Rest]) :-                           >> X= [[k, i], [s, e], [k, e]...]
cvStructure([Head, *Rest],
[CvHead, *CvRest])
    word(_, _, [Head, *Rest])
```

An additional set of rules allows WORDEN to consider also optional symbols in the CV structure during generation so that, by considering the sequence [C, (C), V], the system outputs all the possible CV and CCV sequences of symbols.

In order to be able to manage phonological neighbourhoods, it is necessary to introduce in the rules base a formal description of the operations of *insertion* (a symbol is introduced in the starting sequence), *deletion* (a symbol is removed from the starting sequence) and *substitution* (a symbol becomes another one in the sequence). As these rules should be used in inductive procedures to generate phonological neighbourhoods, it is sufficient to define the three operations in terms of the first element, as we did for the first example in this section. The three rules should describe that:

- A sequence is obtained from another sequence by substituting its first element if all the symbols in the two sequences are the same but the first. In both cases the head symbols must be PHONITALIA symbols;
- A sequence is obtained by inserting a specific symbol to the head of another sequence if that symbol is prepended to the original sequence;
- A sequence obtained by deleting the head symbol of another sequence is the original one minus its head, regardless of what it was.

```
substitution([Start, *Rest1],
[Altstart, Rest2]) :-
    symbol(Altstart)
    symbol(Start)
    Start != Altstart

insertion([Start, *Rest], Ins, [Ins,
Start, *Rest]) :-
    symbol(Ins)
    symbol(Start)

deletion([_, *Rest], Rest)
```

```
? substitution([o,r,a], [a,r,a])
>> yes

? substitution(X, [a,r,a])
>> X= [[t,r,a], [f,r,a],...]

? insertion([m,a,r,i], a, [a,m,a,r,i])
>> yes

? substitution([p,a,r,c,o], [p,a,r,t,o])
>> yes
```

At this point, it is possible to introduce a set of rules to verify if a sequence can be obtained by applying one of the possible operations to another sequence. We will refer to the transformation of a string to another by insertion, deletion or substitution of a symbol as *perturbation*. During the exploration of the possible strategies to obtain a target sequence starting from an initial one it is necessary to consider that skipping a symbol is an additional operation. The rules should describe that:

1. A sequence is a perturbation of itself;
2. If two sequences share the same head symbol, this can be skipped;
3. If two sequences have different heads, it is possible to consider a substitution, deletion or insertion operation and check if the rest of the sequence is the target one.

This set of rules covers the case where one operation is allowed to reach a target sequence from a starting one. In this paper, we will cover this situation only. Future work will extend the system to include more complex planning to include sequences of operations.

```
    perturbation([Head, *Rest], [Head,
*Rest])

    perturbation([Head, *Rest1], [Head,
*Rest2]) :- perturbation(Rest1, Rest2)

    perturbation([Head1, *Rest1], [Head2,
*Rest2]) :-
substitution([Head1, *Rest1], [Head2,
*Rest2])

    perturbation([Head1, *Rest1], [Head2,
*Rest2]) :-
    insertion([Head1, *Rest1], [Head2,
*Rest2])

    perturbation([Head1, *Rest1], [Head2,
*Rest2]) :-
    deletion([Head1, *Rest1], [Head2,
*Rest2])
```

```
    ? perturbation([p,a,r,c,o],
[p,a,r,t,o])
    >> yes

    ? perturbation([p,a,r,c,o],
[p,a,r,c,o])
    >> yes

    ? perturbation([p,a,r,c,o],
[p,a,r, e, t, i])
    >> Cannot prove
```

At this point, it is possible to define a phonological neighbour of a sequence of symbols as another sequence of symbols that is a perturbation of the initial sequence and corresponds to a word. The starting sequence may be either a word or a nonsense word.

```
    phonNeighbour([Head1, *Rest1],
[Head2, *Rest2]) :-
    word(_, _, [Head2, *Rest2])
    perturbation([Head1, *Rest1],
[Head2, *Rest2])
```

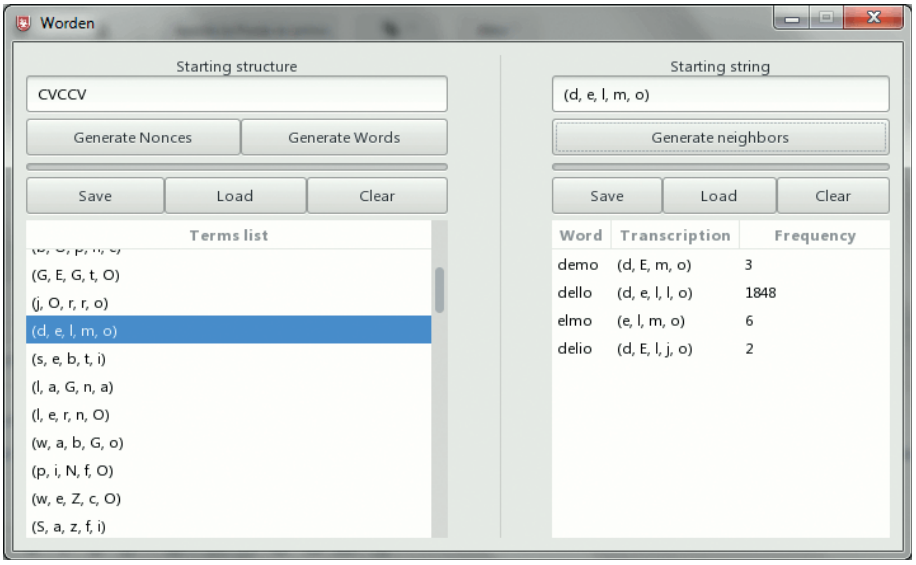
```
    ? phonNeighbour([p,a,r,c,o],
[p,a,r,t,o])
    >> yes

    ? phonNeighbour([p,e,r,m,o],
[f,e,r,m,o])
    >> yes

    ? phonNeighbour([p,e,r,m,o], X)
    >> X= [[f,e,r,m,o], ...]
```

3.3 Graphical User Interface

The WORDEN knowledge engine can be queried from command line to submit a variety of *questions* to the system. However, to support researchers working in the specific field of speech production and perception, we created a Graphical User

Interface to submit specific queries to the system. In Figure 3.1, we show a screenshot of the GUI.

Figure 1 - *Screenshot of the WORDEN Graphical User Interface*



The window is organized in two vertical frames. The frame on the left is dedicated to word and nonsense words generation starting from a given CV structure possibly including optional symbols. The frame on the right is dedicated to the generation of phonological neighbourhoods starting from given sequences of symbols. As phonological neighbourhoods only include words in PHONITALIA, the corresponding frequency is also given as output. It is possible to move a generated sequence to the field for phonological neighbourhood generation by double clicking the generated word/nonsense word. Functions to clear and load/save to text files both lists are provided.

4. *Conclusions*

We have presented WORDEN, a knowledge engine including a knowledge base extracted from the PHONITALIA database combined with rules providing a formal definition of the operations needed to generate words and nonsense words. WORDEN also contains a formal definition of phonological neighbourhood starting from arbitrary sequences of symbols.

Using knowledge engines allows to represent formal theories that can be automatically proved or disproved by the engine. Moreover, given a formal description of the elements involved in a theory, knowledge engines remove the need to store a large amount of data that can be inferred at runtime. Words, nonsense words and phonological neighbourhoods, in WORDEN, are generated rather than stored.

This allows future extension of the system to include more complex theories building on the top of the already provided concepts. While knowledge engines can answer a wide set of queries on the basis of the provided facts and rules sets, we developed a GUI to let researchers interact more easily with the system. While the GUI limits the possible queries to the ones that are connected to the buttons, it is always possible to submit more complex queries from the command line by using PYKE. Being implemented in PYTHON, the system is cross platform as long as PYTHON is installed. To make it easier for WINDOWS users to access WORDEN, a platform specific binary is provided.

Future work will concentrate on extending the concepts included in the WORDEN knowledge. This can both be done by including more languages from databases other than PHONITALIA and by extending the rules set to allow more complex theories to be represented. Our current orientation, in this sense, is to include the possibility of building perturbation plans involving sequences of operations based on sub- or supra-segmental features and introducing more specific rules on symbols assembling (i.e. the sonority scale principle) to automatically syllabify input sequences of symbols. The tool can be obtained by contacting the authors.

References

Bird, S., Klein, E. (1990). Phonological Events. In *Journal of Linguistics,* 26, 33-56.

Bird, S., Ellison, T.M. (1994). One level phonology autosegmental representations and rules as finite automata. In *Computational Linguistics*, 20, 55-90.

Cangemi, F., Barbato, M. (2014). A laboratory approach to acquisition and change: Italian mid-vowels in nonce words. Talk presented at the *7th Laboratory Approaches to Romance Phonology (LARP) Conference.*

Cangemi, F., D'Imperio, M. (2013). Tempo and the perception of sentence modality. In *Journal of the Association for Laboratory Phonology*, 4 (1), 191-219.

Coltheart, M., Davelaar, E., Jonasson, J.T. & Besner, D. (1977). Access to the internal lexicon. In *Attention and Performance*, VI, 535-555.

D'Imperio, M. (2001). Focus and tonal structure in Neapolitan Italian. In *Speech Communication*, 33 (4), 339-356.

Di Cristo, A. (1985). *De la microprosodie à l'intonosyntaxe*. Aix-en-Provence: Publications Université de Provence.

Frederiksen, B. (2008). *Applying expert system technology to code reuse with Pyke*. In Proc. of PyCon [Online: pyke.sourceforge.net/PyCon2008- paper.html].

Goslin, J., Galluzzi, C. & Romani, C. (2013). Phonitalia: a phonological lexicon for Italian. In *Behavior Research Methods*, 46 (3), 872-886.

Grishman, R. (1986). *Computational Linguistics: An introduction*. Cambridge University Press.

Keuleers, E., Brysbaert, M. (2010). Wuggy: A multilingual pseudoword generator. In *Behavior Research Methods*, 42 (3), 627-633.

Lehiste, I. (1970). *Suprasegmentals*. Cambridge: The MIT Press.

Luce, P.A., Pisoni, D.B. (1998). Recognizing spoken words: The neighborhood activation model. In *Ear and Hearing*, 19, 1-36.

Marian, V., Bartolotti, J., Chabal, S. & Shook, A. (2012). CLEARPOND: Cross-Linguistic Easy-Access Resource for Phonological and Orthographic Neighborhood Densities. In *PLoS ONE* 7 (8).

Nugues, P.M. (2014). *Language Processing with Perl and Prolog: Theories, Implementation and Application*. New York: Springer-Verlag.

Petrone, C. (2008). *Le rôle de la variabilité phonétique dans la représentation des contours intonatifs et de leur sens*, PhD Thesis, Université Aix-Marseille I.

Stone, M. (2014). Semantics and computational semantics. In Aloni, M., Dekker, P. (Eds.), *Cambridge Handbook of Semantics*. Cambridge: Cambridge University Press.