Natural Language Processing & Applications

Speech Synthesis and Recognition

1 Introduction

Now that we have looked at some essential linguistic concepts, we can return to NLP. Computerized processing of speech comprises

- speech synthesis
- speech recognition.

One particular form of each involves written text at one end of the process and speech at the other, i.e.

- text-to-speech or TTS
- speech-to-text or STT.

There are also applications where text is not directly involved, such as the reproduction or synthesis of fixed text, as in the 'speaking clock', or recognition which involves selecting from audio menus via single word responses.

2 Text-to-Speech (TTS)

TTS requires the conversion of text into semi-continuous sounds. The most obvious approach seems to be to store digitized sound sequences representing complete words or phrases, and then simply select and output these sounds. For those who travel by train, a good example of this approach is the automated announcement system used at many UK railways stations, including New Street Station.[1] This approach has the advantage of simplicity and can also produces very natural-sounding speech; however it also has several serious drawbacks.

- In a limited domain (e.g. speaking weather forecasts or train announcements), it is feasible to construct a complete dictionary giving the digitized pronunciations of all relevant words or phrases. However, this is impossible for general text, since natural languages contain a very large number of words, many of which occur only rarely, so that a huge dictionary would be required. Furthermore, natural languages are constantly changing, with new words being introduced (especially in the names of commercial products). Such words could not be handled by a system relying on pre-stored sounds.

- Natural speech does not consist of separate words with pauses between them. 'Blending' words together is essential for natural sounding speech. In most languages, the correct choice of allophone at a word boundary depends on the neighbouring word. For example, in SEE, /r/ is not pronounced unless immediately before a vowel. Thus in the sentence *tear the paper in half*, the /r/ in *tear* is not pronounced. However in the phrase *tear it in half*, it is. This phenomenon means that combining pre-recorded words or phrases into new combinations can seriously lower the quality of the output.

- Many other features of speech, such as stress or intonation (see Section 5), do not involve individual words alone, but operate on longer units such as phrases or sentences.

The last two drawbacks can be overcome in part by storing longer phrases, as the train announcement system does. However, it is still possible to tell when out-of-context words or phrases are inserted into the 'canned text'.

An alternative approach, used for example in the MacinTalk speech synthesizers available on the MacOS platform, involves two stages:

- Conversion of graphemes into phonemes (plus other information, such as syllables and stress indicators, not discussed so far)

- Conversion of phonemes to phones, i.e. the choice of appropriate allophones.

---

[1]   There's a link to some information about this system on the module web site.

Graphemes are the units of a language's spelling conventions. In a language which is written alphabetically, the graphemes are made up of letters and combinations of letters. For example, in English, the sequence *ph* is (normally) a single grapheme, although composed of two letters. Thus the word *Phillip* is composed of five graphemes: *ph, i, ll, i, p*. Graphemes in English can be even more complex; in particular they overlap. Consider for exampe the word *mice*. Its spelling is perfectly regular in modern English, but involves four rules:

- The letter *m* represents the phoneme /m/.

- The pattern *iCV*, where C is any single consonant letter and V is any single vowel letter, means that the letter *i* represents the phoneme /aɪ/.[2]

- The sequence *ce* means that the *c* represents the phoneme /s/.

- A terminal *e* after a consonant letter is 'silent', i.e. does not represent any phoneme.

Thus *mice* represents /maɪs/. (As an exercise, consider how *mica* represents /maɪkə/ via regular rules.)

The extremes in grapheme to phoneme conversion are:

- rules relating graphemes to phonemes
- table/dictionary lookup.

Real languages require different approaches along this spectrum, depending on the writing system and its regularity. In some languages, converting graphemes to phonemes is relatively easy using a rule-based approach, since the spelling system is regular; good examples are Spanish and Modern Greek. For other languages this is more difficult, English being a clear example (as is Irish Gaelic, with French a good runner-up), so that at best a mixed approach is needed. Some writing systems use mixed systems with some phonemic graphemes and some non-phonemic graphemes (e.g. *kana* and *kanji* in Japanese). For other writing systems (e.g. Chinese), a rule-based approach is impossible since characters have no clear phonemic relationship.

I will not discuss rule-based grapheme to phoneme conversion in detail here, but it is clear that mapping spelling to phonemes is likely to be easier than mapping spellings to phones. Spelling systems are at best phonemic; they are not phonetic since those who devised the system had no interest in attempting to represent sound differences which don't affect meaning. We don't have different symbols for [p] and [pʰ] in English, because these phones correspond to the same phoneme and hence don't affect meaning. There are different symbols for [p] and [pʰ] in the Devanagari script used for writing Hindi because in this language the phones do correspond to different phonemes and hence do affect meaning.

Early grapheme-phoneme systems for English relied largely on rule-based approaches, supplemented by a smallish dictionary of exceptions. Now that memory and storage are much cheaper, the tendency has been to reverse this and use a large dictionary of phonemic pronunciations, supplemented by rules to deal with words not in the dictionary (e.g. the names of people or companies and new words).

The second stage in this approach to speech synthesis (graphemes-phonemes-phones) also involves a combination of rules and lookup. Phonological rules, as described in the previous handout, relate phonemes to their allophones. A rule-based approach enables rare or new words to be handled. For example, *uncool* is unlikely to be included in a reasonably-sized dictionary, but can be generated from the components *un* stored phonemically as /ʌn/ and *cool* stored as /kul/. It can then be predicted that in slow speech it will be pronounced [ʌnkul], whereas in fast speech it may be pronounced [ʌŋkul]. There are always likely to be exceptions even to phonological rules, which can only be handled by a dictionary look-up. As with grapheme to phoneme conversion, there are various practical issues to be considered:

---

2   Note that other rules, which must be applied earlier, may over-ride this rule, e.g. if *C* is the letter *r*. English spelling reform is a subject often discussed, not always in an informed way. English spelling is not phonemic for two main reasons. Firstly, there aren't enough letters in the Latin alphabet to easily represent the 40-odd phonemes of English, and in particular its many vowel phonemes. Secondly, English spelling has not kept up with changes in the language; sometimes it was originally phonemic but now isn't. For example, there was once a /x/ phoneme in English, often spelt *gh*, so that *light* was once pronounced closer to the German *licht*. This phoneme has now disappeared but the spelling remains.

- The relative cost of writing and coding rules as opposed to constructing look-up tables and dictionaries.

- Speed of processing versus storage requirements, since more complex rules allow a smaller dictionary of exceptions, but are likely to cause processing to be slower.

The choice of allophone also has to be such that words 'blend' both internally and across word boundaries. Producing very natural sounding speech requires hundreds of allophones. Little seems to have been published about this process (at least in a form suitable for non-specialists) and the inner details of commercial speech synthesis software are usually not available. What is clear is that this approach requires complex rules to select exactly the right allophone to produce well blended speech.

One way of minimizing 'blending' problems works by storing **biphones**. (The term is a slight misnomer, because in fact only parts of phones are stored.) The basic idea is to take samples of speech and then cut them up (digitally) in the MIDDLE of a phone, to yield a sound composed of the end of one phone and the beginning of the next. This relies on the idea that the middle part of a phone is less influenced by its neighbouring phones than the beginning or the end. In principle it is necessary to store a digital sound for every possible sequence of two phones which can occur in the language. With only one allophone per phoneme, this would seem to require at least $40^2 = 1600$ biphones in English. However, many of these combinations don't occur, so the number is more manageable, even allowing for multiple allophones for each phoneme.

To see how this approach might work, consider the word *uncool* introduced above. A grapheme → phoneme converter could generate /ʌnkul/, then a phoneme → phone converter could produce [ʌŋkul]. Now the word can be output as the sequence of previously stored biphones. If we represent a biphone as [X+Y] meaing the second half of the phone X followed by the first half of the phone Y, then [ʌŋkul] will be generated as [?+ʌ], [ʌ+ŋ], [ŋ+k], [k+u], [u+l] and [l+?], where the ?s represent the need to blend with neighbouring words. Note that some initial conversion of phonemes to phones is still required (to choose [ʌ+ŋ] rather than [ʌ+n] for example.) However, simpler phonological rules are needed with this approach since blending is handled within the biphones. Some longer but still common phoneme sequences (e.g. *str* in English) can be stored as **triphones**; even longer sequences may be useful in some languages.

The two stages of grapheme → phoneme and phoneme → phone conversion are not totally independent, and many systems incorporate what linguists would call phonological rules into grapheme to phoneme conversion.

3 Coding Phonological Rules

In text-to-speech (TTS), phonemes need to be converted to (allo)phones, so rules are used in the forwards direction. One version of the algorithm needed for English is:

```
Ensure that the string of phonemes has 'word-boundary' markers at the
start and end of every word (these and the phonemes are the 'units' of
analysis);
for (i = 1; i <= number of units - 2; i++)
{ Set the current substring to units i to i+2;
  for (each valid phonological rule in turn)
  { Apply the rule to the current substring to generate a new
      current substring;
  }
}
```
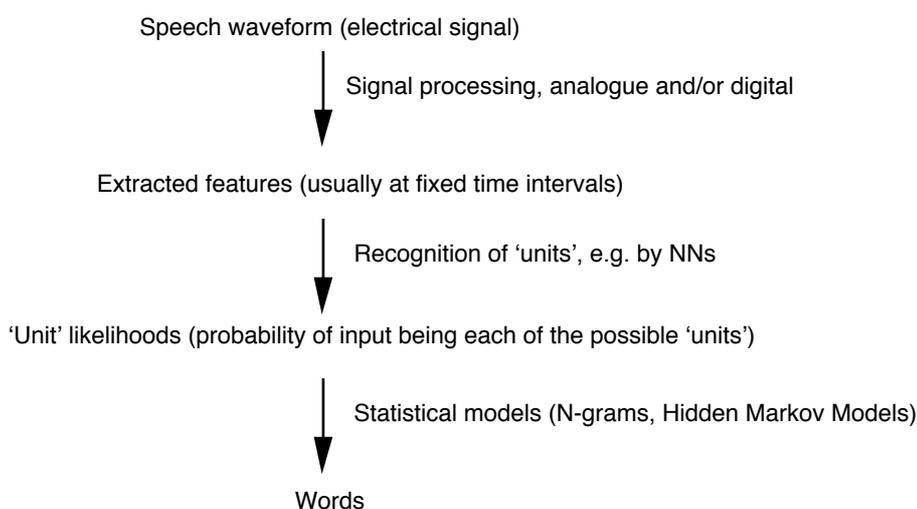
Note that all valid rules must be applied before moving on to the next input triple. The ordering of the phonological rules is important (and of considerable linguistic interest).

(Appendix 2 shows how this can be coded in Prolog, but it is straightforward to implement in any programming language. Appendix 1 suggests some ASCII equivalents to the IPA characters used in this handout, since ASCII is easier to work with in a program.)

4 Speech-to-Text (STT)

Speech-to-text (STT) is considerably more difficult than the reverse. The simplest approach is to require the user to speak in discrete words and then attempt to recognise them 'whole' (perhaps by digitizing and matching with a dictionary of stored sounds). With sufficiently slow speech, variation in pronunciation caused by phonological rules operating across word boundaries should be minimal. A major problem of course is that speaking in this way is highly unnatural.

Whole word recognition can also be applied to continuous speech. It requires considerably more processing, since the position at which a match should be attempted is unknown, so that multiple matches will be needed. Furthermore to take into account variations in pronunciation caused by neighbouring words, some 'looseness' will be required in the matching process.

Speech waveform (electrical signal)

Signal processing, analogue and/or digital

Extracted features (usually at fixed time intervals)

Recognition of 'units', e.g. by NNs

'Unit' likelihoods (probability of input being each of the possible 'units')

Statistical models (N-grams, Hidden Markov Models)

Words

Realistic current systems rely heavily on statistical prediction. To do justice to this approach would require more time than is available in this module (as well as a significant amount of mathematics). See Chapter 7 of Jurafsky & Martin (2000) for a reasonably up-to-date summary. The broad stages are shown in the diagram.

The first stage is to use analogue and/or digital signal processing to extract a set of basic 'features' from the speech waveform usually at fixed time intervals. It is important to note that these 'features' are SOUND-based rather than LANGUAGE-based. For example, the presence or absence of a particular frequency ('formant') may be recorded.

The second stage is to convert the sets of features to language 'units', which may be phones, but are often smaller entities. For example, in a stop, the closure period and the release period may be treated separately. Since there are considerable similarities between some phones, this stage results in a set of probabilities rather than a single identification. For example, at a particular time point the system might decide that the unit involved is the release part of [d] with a probability of 0·3 or the release part of [t] with a probability of 0·25, plus other less probable identifications making up a total probability of 1.

The third stage is to use 'language models' to refine the initial identification. The input units are matched with a language model and the best fit (or set of best fits) returned. Language models generally involve knowledge of the likely frequency of N length sequences (N-grams) of the input units and of words in the language. The simplest approaches use bigrams: for example, storing information on the likelihood of each phone occurring after a given phone. As an example of how this might work, suppose that the units are phones and the system has decided that a particular phone is [s] with a probability of 0·4 and the following phone either [d] with a probability of 0·3 or [t] with a probability of 0·25. On its own, this evidence suggests that the sequence is [sd] with a probability of $0·4 * 0·3 = 0·12$ or [st] with a probability of $0·4 * 0·25 = 0·10$; neither very convincing. However, phone bigram statistics will show that in English after [s], [t] is MUCH more likely than [d], so that if the choice is between [sd] and [st], [st] is actually much more probable.

A major problem with all approaches to speech recognition is variation among speakers. Age, gender, health (e.g. a blocked nose) and the quality of the transmission and recording system will all change the sounds available for recognition. Phonological rules operate both within words and between words and differ according to the speed of speech and the speaker's dialect (e.g. speakers of SAE do not pronounce [r] before consonants, speakers of Scottish English do). Speaker-independent recognition systems can only use statistical models based on averages over speakers. It is not surprising therefore that recognition is difficult and of limited accuracy. State-of-the-art systems applied to randomly chosen realistic English speech samples are reported to have word error rates exceeding 20% (and possibly as high as 40%).

In some contexts, knowledge of the domain allows more specific language models to be used. For example, a system intended for use in dictating business letters can use statistics on word pair frequencies in business letters rather than text generally, which can significantly improve recognition.

Dialect and speaker variation can be dealt with by using **adaptive systems** which incorporate 'training'. The user reads a standard passage a number of times and the system adapts (by parameter adjustment, etc.) until a sufficient degree of accuracy is reached. Coupled with good language models, such systems have much higher claimed accuracy, with word error rates below 5%.

All approaches to speech recognition which rely heavily on models have the same problem with rare or new words as the whole word approach to speech synthesis. The language models will not contain rare or newly introduced words. In these circumstances, the ideal would seem to be to reverse the two stages used by MacinTalk and similar speech synthesisers, i.e. to use phonemes as an intermediate unit, and then use these to generate graphemes (i.e. text).[3] Phonological rules (normally represented only implicitly via statistical data on allophone sequencing) would then need to be explicitly coded. An important advantage of including explicit phonological rules in speech recognition would be that by varying the rules, it should be possible to adapt to different accents, speakers, etc. Such a system could recognize, for example, that [ʌnkul] and [ʌŋkul] represent the same word since the underlying phonemes are /ʌnkul/. (Adaptive systems effectively do this by altering their pattern-recognizers and statistical models.) The weakness of any purely sound-based (i.e. phonetic) speech recognition approach is that it cannot readily generalize in this way. However, although there may be sound theoretical reasons for preferring an approach which uses explicit phonological rules, the evidence at present is that more 'brute force' statistically-based approaches work better.

My personal view is that there is an upper limit beyond which purely statistical approaches will not progress, and that only a system which operates in a much more human-like fashion, with meaning playing an important role in recognition, will achieve human level performance. Time will tell whether this view is correct.

5 Postscript

In the preceding sections in this and the previous handout, I have chosen to focus on phonemes and allophones in order to try to provide a deeper insight into the kinds of issue involved in the processing of speech. However, to handle realistic speech, both in synthesis and in recognition, MANY other issues would need to be considered. Three such are length, syllable stress and intonation.

I have ignored the **length** of phones, i.e. the actual time spent making the sound.[4] However, this can be important. In English, for example, *beat* and *bead* differ in the length of the [i] as

---

[3] Even writing systems which are non-phonemic, such as Chinese, have phoneme-based methods for handling foreign words or brand names.

[4] Be careful not to confuse the common description of different vowel phonemes in English as being long or short with the length of a particular phone. Thus the pronunciation of *bath* in southern England as [bɑθ] and in northern England as [bæθ] is commonly said to involve a 'long a' [ɑ] or a 'short a' [æ]. However, these are different phonemes as well as being of different lengths: the words *Bart* and *bat*, for example, are

well as in the voicing of the stop. Using the IPA symbol ː to show that the preceding phone is sounded for longer than normal, *beat* and *bead* are pronounced [bit] and [biːd] respectively. However, phone length is not phonemic in English: [bit] and [biːt] are not different words; the change of [t] to [d] is what changes *beat* to *bead*.

In some languages, phone length is phonemic: in Italian, [nonɔ] is 'ninth', [nonːɔ] 'grandfather'. Thus in generating English we need to consider phone length in order to produce realistic speech, but we can ignore it when recognizing English since it is not phonemic. In Italian, phone length will be important both in generation and recognition.

Words need to be divided into syllables, partly because some phonological rules are affected by syllable boundaries, but also because **syllable stress** is an important component of languages like English, where some words are distinguished only by stress differences. Consider:

This process is called assimilation.
I usually process at the degree ceremony.

Stressing the first syllable produces the noun [ˈprou sɛs] as in the first sentence; stressing the second syllable produces the verb [prou ˈsɛs] as in the second sentence. The IPA symbol ˈ marks the following syllable as being particularly stressed.

Intonation (changing sound frequency or pitch) is also important. Intonation alone can distinguish statements from questions:

You want some coffee.   (Statement: tone falls at the end of *coffee*.)
You want some coffee?   (Question: tone rises at the end of *coffee*.)

However, intonation differences are not phonemic in English. In some languages ('tone languages'), such as the Chinese languages and many West African languages, sequences containing the same phones but with different intonation patterns correspond to different words. For example in Thai, the phones [naː] correspond to 5 different words depending on the tone in which they are spoken: [naː] with a falling tone means 'face', [naː] with a rising tone 'thick', and so on.

In writing, the two sentences:

The lion killed yesterday afternoon in the open bush and was seen today.
The lion killed yesterday afternoon in the open bush was seen today.

cannot be distinguished until the presence or absence of the word *and* is known (which also makes parsing difficult as we shall see later). In speech, however, differences in intonation, stress and pausing immediately identify *killed* as either the main verb or part of a subordinate clause.

In addition, syntax and semantics need to be taken into account. For example the written form *lives* corresponds to two differently pronounced words in the sentences *Several lives were lost in the accident* and *He lives in Birmingham.* The plural noun is /laɪvz/, the third singular verb is /lɪvz/. I will return to the interaction between levels in natural language processing later in the module.

Taking into account such extra aspects of speech processing means that, for example, a realistic TTS system for English is likely to have at least the following stages.

1. Expand symbols (e.g. digits) and abbreviations into words.

2. Analyse the syntax of the sentence sufficiently to disambiguate words written with the same graphemes but having different phonemes, as well as to identify 'intonation units' and heavily stressed words.

3. Complete the conversion to phonemes, adding stress and intonation markers.

4. Select the correct allophones, together with their length (duration), stress (volume) and intonation (frequency).

5. Combine the allophones by adjusting their edges to blend smoothly.

---

distinguished in SEE only by the change from [ɑ] to [æ]. Prolonging the vowel phone in either dialect does not change the word *bath* into a different word.

Exercises

Past examination papers will be found on the NLPA web site (http://www.cs.bham.ac.uk/~pxc/nlpa/). Apart from being able to describe, explain and compare approaches to both TTS and STT, you should be able to apply your knowledge to examples such as the following.

1. The handout "Phones and Phonemes" describes the relationships between the (allo)phones [p] and [pʰ] and their underlying phonemes in English, Hindi and French. What are the consequences of these relationships for TTS and STT in each of these languages?

2. The word *garage* is pronounced in English in at least two ways: [gærɑʒ] and [gærɪʤ]. What problems could these variant pronunciations of *garage* cause a speech-to-text (STT) system? How might they be overcome?

3. How might a TTS system based on a two stage approach (graphemes to phonemes followed by phonemes to allophones, as in the MacinTalk synthesiser) convert the text *line which* into the output [laɪmwɪtʃ]?

4. Exercise 6 in the handout "Phones and Phonemes" illustrates a phonological rule routinely applied to German by native speakers of this language. Such speakers often incorrectly apply the same rule when speaking English. What problems will this cause in attempting to recognize English spoken by Germans?

5. Given English text, a film company wants to generate English speech for use in dubbing so that it sounds to a native English speaker as if it were spoken by a German. How might this be done?

Appendix 1

It's useful to have an ASCII alternative to the IPA for use in computer processing. The symbols in the second column of the Appendix to Phones and Phonemes (1) can be used for this purpose.

If a phonetic transcription is written as words separated by spaces, then the symbols can be used exactly as given. Note that 'digraphs' are always written in capitals. This enables TH (θ in the IPA) to be distinguished from the sequence of sounds th (th in the IPA). Ignoring aspiration, the IPA version of my pronunciation of the first line of Lewis Carroll's *The Walrus and the Carpenter* is:

> ðə taɪm hæz kʌm ðə wɔlrʌs sɛd

This can be written as:

> DHAX tAYm hAEz kUXm DHAX wAOlrUXs sEHd

Note that punctuation symbols can be used in this notation, but words must not have initial capitals.

An alternative is represent phonetic text as a comma-separated list of ASCII symbols (this is useful when coding in Prolog for example). In this case, the digraphs can be kept in lower case, and a special symbol used as a word separator. I generally use !. Transcribing the same line of the poem in this format gives:

> [!,dh,ax,!,t,ay,m,!,h,ae,z,!,k,ux,m,!,dh,ax,!,w,ao,l,r,ux,s,!,s,eh,d,!]

The first and last ! are not strictly necessary but make some algorithms easier to write.

Appendix 2

A simple Prolog program can be written to handle context-sensitive phonological rules. Words will be represented as lists of phonemes or phones. The symbol ! will be used as a 'word delimiter'. It must be present at the start and end of every word to provide start-of-word and end-of-word context. We need to invent ASCII names for the IPA symbols. (See Appendices 1 and 2.) Thus *pit stop* with the phonemic representation /pɪt stɒp/ might be represented in Prolog as [!,p,ih,t,!,s,t,oh,p,!].

To code phonological rules in Prolog, we first need to store information about phones (and the corresponding phonemes). One approach is shown below; it allows for a fixed number of features for each phone (three).

```
phone(ih,vowel,_,voiced).              % hIt [ɪ]
phone(eh,vowel,_,voiced).              % mEt [ɛ]
phone(ax,vowel,_,voiced).              % About, aftER, fERn [ə]
phone(ux,vowel,_,voiced).              % fUn [ʌ]
...
phone(p,stop,bilabial,voiceless).
phone(t,stop,alveolar,voiceless).
phone(k,stop,velar,voiceless).
phone(m,nasal,bilabial,voiced).
phone(n,nasal,alveolar,voiced).
...
```

For stops, I have chosen to store the position of articulation and whether voiced or voiceless a. Different information can be stored for nasals or vowels; in the present version nothing is stored is one 'slot' of the phone/4 predicate.

The 'nasal assimilation' rule discussed above was:

> {nasal} → {nasal, Position} : 'anything' _ {stop, Position}

In terms of Prolog variables, such rules are equivalent to:

> In → Out : Left _ Right

Hence one way of coding this rule in Prolog is:

```
p_rule([Left,In,Right|Tail],[Left,Out,Right|Tail]):-
    phone(In,    nasal,_,    _),
    phone(Right,stop,  Posn,_),
    phone(Out,   nasal,Posn,_).
```

The logic of a clause of `p_rule/2` is to separate off enough of the start of the input list to provide the required context for the rule, check the 'units' to see if they fit the rule, and then make the necessary change(s) to the central unit. I will ALWAYS provide a left and right context of at least one unit even if not strictly needed. (The reason for including `Tail` in the code for a rule is that it makes later coding of the algorithm which handles phonological rules easier.)

Note that the absence of any restriction on the variable `Left` in the rule above means that it will match anything, as required.

Coding the **algorithm** for handling phonological rules in Prolog requires recursion. The predicate required will be called `p_process`, for 'phonologically process'. It takes two arguments: an input list and an output list. The definition has three clauses:

- The first clause handles the 'base case': an empty input list returns an empty output list.

  ```
  p_process([],[]).
  ```

- The second clause handles the application of a phonological rule. If the input list is not empty, it must have a rule applied to it by `p_rule` to yield an intermediate temporary list, after which phonological processing continues (so that further rules may be applied):

  ```
  p_process(InList,OutList):-
      p_rule(InList,TempList),
      p_process(TempList,OutList).
  ```

  However, this definition is not quite right. If applying `p_rule/2` does not change anything, then the same rule may be applied over and over again when `p_process/2` is called again. One (inefficient) way of stopping this is to check that `InList` and `TempList` are different:

  ```
  p_process(InList,OutList):-
      p_rule(InList,TempList),
      InList \= TempList,              % \= is Prolog for 'doesn't match'
      p_process(TempList,OutList).
  ```

- Finally, if no phonological rule applies, processing continues one step into the list:

  ```
  p_process(InList,OutList):-
      InList = [Head|InTail],
      p_process(InTail,OutTail),
      OutList = [Head|OutTail].
  ```

  This can be written more efficiently, if less clearly, as:

  ```
  p_process([Head|InTail],[Head|OutTail]):-
      p_process(InTail,OutTail).
  ```

The complete definition of `p_process/2` is thus:

```
p_process([],[]).
p_process(InList,OutList):-
    p_rule(InList,TempList), InList \= TempList,
    p_process(TempList,OutList).
p_process([Head|InTail],[Head|OutTail]):-
    p_process(InTail,OutTail).
```

Don't worry if you find this hard to understand! It's only really important to grasp the logic of the underlying rules and algorithm. Phonological rules are applied repeatedly (via `p_rule`) so long as they change the input. Afterwards processing moves on one 'unit'. (There are more efficient ways of implementing the algorithm in Prolog than the one given here.)

Now that `p_process/2` is coded, we can test our coding of the nasal assimilation rule:

```
?- p_process([!,ih,n,k,ux,m,!],Result).      % i.e. "income"
Result = [!,ih,ng,k,ux,m,!]                   % i.e. [ɪŋkʌm]

?- p_process([!,ae,n,k,l,!],Result).          % i.e. "ankle"
Result = [!,ae,ng,k,l,!]                       % i.e. [æŋkl]

?- p_process([!,ae,n,p,l,!],Result).          % i.e. "anple"
Result = [!,ae,m,p,l,!].                       % i.e. "ample"
```

It's interesting that with this Prolog code, processing *input* gives alternatives corresponding to fast and slow speech (or southern and Scottish English dialects):

```
?- p_process([!,ih,n,p,uh,t,!],Result).       % i.e. "input"
Result = [!,ih,m,p,uh,t,!] ;                   % i.e. [ɪmpʊt]
Result = [!,ih,n,p,uh,t,!] ;                   % i.e. [ɪnpʊt]
no
```