Using ILP to learn local linguistic structures
Konstantopoulos, Stasinos Themistokleous

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version
Publisher's PDF, also known as Version of record

Publication date:
2003

Link to publication in University of Groningen/UMCG research database

Citation for published version (APA):

Copyright
Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

Take-down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from the University of Groningen/UMCG research database (Pure): http://www.rug.nl/research/portal. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.
Chapter 5

Phonotactics

The Phonotactics of a given language is the set of rules that identifies what sequences of phonemes constitute a possible word in that language. The problem can be broken down to the syllable structure (i.e. what sequences of phonemes constitute a possible syllable) and the processes that take place at the syllable boundaries (e.g. assimilation).

Previous work on the syllable structure of Dutch includes hand-crafted models, like the ones described by van der Hulst [87] and Booij [4], but also machine-learning approaches: abduction by Tjong Kim Sang and Nerbonne [83] and neural networks by Stoianov and Nerbonne [81] and Stoianov [80, ch. 4]. The work presented here was originally presented in the Student Session of ESSLLI 2001 [34] and subsequently published as special issue of WEB-SLS [35].

This chapter describes experiments on the task of constructing from examples a model of Dutch monosyllabic words. The reason for restricting the domain is to avoid the added complexity of handling syllable boundary phonological processes. Furthermore by not using polysyllables no prior commitment is made to any one particular syllabification (and thus syllable structure) theory.

5.1 Extracting the Data

As a starting point, a rough template matching all syllables is assumed. If $\mathcal{C}$ is the set of all consonant segments and $\mathcal{V}$ the set of all vowels and diphthongs that appear in Dutch, this template is $C_3V C_5$, where $C_n \in \mathcal{C}^n \cup \ldots \cup \mathcal{C} \cup \emptyset$ represents any consonant cluster of length up to $n$ and $V \in \mathcal{V}$ any vowel or diphthong. The problem can now be reformulated as two single-predicate learning tasks where the target theory is one of acceptable affixes to a given.
vowel and partial consonant cluster. The rules for prevocalic and postvocalic affixing are induced in two separate learning sessions.

The training data is derived from the 5692 monosyllabic words found in the Dutch section of the CELEX Lexical Database [59]. The positive examples are constructed by breaking the phonetic transcriptions down to three parts: a prevocalic and a postvocalic consonant cluster (consisting of zero or more consonants) and a vowel or diphthong. The consonant clusters are treated as ‘affixes’ to the vowel, so that syllables are constructed by repeatedly affixing consonants, if the context (the vowel and the pre- or post-vocalic material that has been already affixed) allows it. So, for example, from the word /maːkt/ (Dutch ‘maakt’) the following positives would be generated:

```
prefix(m, [], [a,:]).  suffix(k, [], [:,a]).
prefix(\^, [m], [a,:]). suffix(t, [k], [:,a]).
suffix(\^, [tk], [:,a]).
```

where the context lists in suffix rules is reversed, so that the two processes are exactly symmetrical and can use the same background predicates.

What needs to be noted at this point is the representation of long vowels and diphthongs as lists of two symbols, instead of having unique symbols for each possible syllable nucleus. This should not be read as being phonologically or linguistically motivated, but rather as a computational convenience, since this representation allows for concepts like ‘long vowel’, ‘diphthong’ or ‘short vowel’ to be accessible by the ILP algorithm through the generic list manipulation predicates (head/2 and rest/2, see also section 5.2 below) alleviating the need to bloat the background with explicit vowel length classification predicates.

The caret, \^, is used to mark the beginning and end of a word. The reason that the affix termination needs to be explicitly licensed is so that it is not assumed by the experiment’s setup that all partial sub-affixes of a valid affix are necessarily valid as well.

In Dutch, for example, a monosyllable with a short vowel has to be closed, which means that the null suffix is not valid. The end-of-word mark allows for this to be expressible as a theory that does not have the following clause: `suffix(\^, [], [\w]).`

The positives are, then, all the prefixes and suffixes that must be allowed in context, so that all the monosyllables in the training data can be constructed: 11067 and 10969 instances of 1428 and 1653 unique examples, respectively.

Negative data is constructed from randomly generated words that match the $C_3V C_5$ template and do not appear in the corpus. These are considered to
be non-words that should be accepted by the constructed theory. The random
generator is biased so that the number of examples at each affix length is
balanced, in order to avoid having the large numbers of long, uninteresting
sequences overwhelm the shorter, more interesting ones.

To illustrate this last point, consider words where, for example, /v/ is
the inner-most prefix to a vowel V: at this point there are some single con-
sonants that may be prefixed (as in, for example, /kvark/, ‘kwark’) but no
two consonants. This makes negative examples like \texttt{prefix(r, [v], [V])} and
\texttt{prefix(f, [v], [V])} more useful that examples like \texttt{prefix(k, [r, v], [V])}
and \texttt{prefix(l, [r, v], [V])}, because the former are helping identify a more
complex and difficult to learn boundary. There are, however, \(|C| = 22
possible ways to prefix one consonant but \(|C|^2 = 484 \) ways to prefix two, meaning
that in a uniformly selected sample there will be 22 times as many examples
from the latter (less interesting) location than from the former one.

These non-words are then split into two parts: one will be used for deriv-
ing the negative data and the other for evaluation. The negative examples
are derived by the following deductive algorithm:

1. For each example, find the maximal substring that is provable by the
positive \texttt{prefix/3} and \texttt{suffix/3} clauses in training data. So, for ex-
ample, for /mtratk/ it would be \texttt{trat} and for /mlat/, \texttt{lat^*}.

2. Choose the clause that should be a negative example, so that this word
is not accepted by the target theory. Pick the inner-most one on each
side, i.e. the one immediately applicable to the maximal substring com-
puted above. For /mlat/ that would be \texttt{suffix(m, [l], [a])}. /mtratk/,
however, could be negative because either \texttt{prefix(m, [tr], [a])} or
\texttt{suffix(k, [t], [a])} are unacceptable affixations. In such cases, pick
one at random. This is bound to introduce false negatives, but no alter-
native could be devised that does not presuppose at least part of the
solution.

3. Iterate, until enough negative examples have been generated to disprove
all the words in the negative training data.

The underlying assumption is that the space of monosyllables is very
‘dense’ or ‘saturated’ in the sense that almost all of the monosyllables allowed
by Dutch phonology and syllable structure are actual words and the randomly
generated words are much more likely to be missing from the corpus because
they fall into a systematic gap rather than due to an accidental gap.

The simplest way to generate negative examples would have been to gen-
erate random \texttt{prefix} and \texttt{suffix} clauses that do not appear as positive ex-
amples. It would, however, \textit{still} be necessary to generate non-words for
the purposes of evaluating the accuracy of the theory at the word level (as opposed to the level of each application of the affix predicates). For this reason it was deemed more consistent to generate negatives only once, for both training at the individual affixation level and the final evaluation at the word level.

5.2 The Background Knowledge

The background knowledge plays, as seen in section 2, a decisive role in the quality of the constructed theory, by implementing the theoretic framework to which the search for a solution will be confined. In more concrete terms, the background predicates are the building blocks that will be used for the construction of the hypothesis’ clauses and they must be defining all the relations necessary to discover an interesting hypothesis.

Since the problem is, in effect, that of identifying the sets of consonants that may be prefixed or suffixed to a partially constructed monosyllable, the clauses of the target predicate must have a means of referring to various subsets of \( \mathcal{C} \) and \( \mathcal{V} \) in a meaningful and intuitive way. This is achieved by defining a (possibly hierarchical,) linguistically motivated partitioning of \( \mathcal{C} \) and \( \mathcal{V} \). Each partition can then be referred to as a feature-value pair, for example \( \text{LAB}^+ \) to denote the set of the labials or \( \text{VOIC}^+ \) for the set of voiced consonants. Intersections of these basic sets can then be easily referred to by feature-value vectors; the intersection, for example, of the labials and the voiced consonants (i.e. the voiced labials) is the feature-value vector \( [\text{VOIC}^+, \text{LAB}^+] \).

For the purposes of this task, they have been defined as relations between individual phones and feature values, e.g. \( \text{labial}(m,+ \) or \( \text{voiced}(m,+ \). Feature-value vectors can then be expressed as conjunctions like, for example, \( \text{labial}(C,+ \wedge \text{voiced}(C,+ \) to mean the voiced labials.

Except for the linguistic features predicates, the background knowledge also contained the \text{head}/2 and \text{rest}/2 list access predicates. These are the standard Prolog library predicates that match a list against its first element and against its tail (everything but the head), respectively. The second element \( E \) of a list \( L \) would then by accessed by \( \text{rest}(L,L_1), \text{head}(L_1,E) \) and so on. This approach was chosen over direct list access with the \text{nth}/3 predicate, as bias towards rules with more local context dependencies.

Special note should be made here to the nucleus, where (as noted in the previous section) long vowels and diphthongs are represented as lists of length 2 (e.g. \( [a,i] \) or \( [a,u] \)) instead of having a separate symbol for each vowel, long vowel and diphthong. This means that concepts like ‘is a simple,
5.3. The Baseline theory

long vowel’ or ‘is a diphthong or long vowel’ can be expressed through head/2 and rest/2 without having to introduce further background predicates that capture these relations:

rest(N, :). % N is simple, long vowel
rest(N, N1), rest(N1, _). % N is diphthong or long vowel

The background knowledge described in sections 5.4, 5.5 and 5.6 below, encodes increasingly more information about Dutch phonology as well as Dutch phonotactics: for the experiment in 5.4 the learner has access to the way the various symbols are arranged in the IPA table, whereas for the experiment in 5.5 a classification that is sensitive to Dutch phonological processes was chosen. And, finally, in section 5.6 the sonority level feature is implemented, which has been proposed with the explicit purpose of solving the problem of Dutch syllable structure.

The quantitative evaluation given in the following four sections was done by 10-fold cross-validation. The 5692 monosyllables were randomly split in 10 sections and the example generation-training-evaluation cycle was repeated 10 times with a different section reserved for testing at each iteration. The recall and precision figures reported are the mean (and standard deviation) of the 10 recall and precision figures calculated from the 10 different datasets.

5.3 The Baseline theory

The training examples themselves can be regarded as a theory consisting of fully instantiated clauses only. When 10-fold cross-validated, the ‘theory’ had a mean recall of 47.07% ($\sigma^2 = 2.23$) with 98.88% ($\sigma^2 = 91E-4$) mean precision. This recall will be considered the performance baseline for the recall of subsequent experiments. The precision is given only for completeness’ sake, but it cannot be considered a baseline in any sense, since such an over-specific theory is expected to reject almost all negatives.

5.4 The IPA Chart

The International Phonetic Association’s International Phonetic Alphabet [2] is collecting all possible phones in a chart organised per place and manner of articulation. Furthermore, each position in the chart hold two phones; consonants can be voiced or not and vowels can be rounded or not.

The organisation of the IPA chart can be seen as two, disjoint, spaces, one of consonants and one of vowels, with a feature vector for each phone
Table 5.1: IPA Chart, Consonants. For each position, the voiced consonant is on the right.

<table>
<thead>
<tr>
<th>Plosive</th>
<th>Bilabial</th>
<th>Labiodental</th>
<th>Alveolar</th>
<th>Post-alveolar</th>
<th>Palatal</th>
<th>Velar</th>
<th>Glottal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nasal</td>
<td>p</td>
<td>b</td>
<td>t</td>
<td>d</td>
<td>n</td>
<td>r</td>
<td>k</td>
</tr>
<tr>
<td>Trill</td>
<td>m</td>
<td>f</td>
<td>v</td>
<td>s</td>
<td>z</td>
<td>j</td>
<td>g</td>
</tr>
<tr>
<td>Fricat.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>η</td>
</tr>
<tr>
<td>Approx.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>χ</td>
</tr>
<tr>
<td>Lat. Appr.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>y</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>h</td>
</tr>
</tbody>
</table>

Table 5.2: IPA Chart, Vowels. For each position, the rounded consonant is on the right. /i/ and /i/ can only be distinguished by explicitly referring to one or the other.

<table>
<thead>
<tr>
<th>Front</th>
<th>Central</th>
<th>Back</th>
</tr>
</thead>
<tbody>
<tr>
<td>Close</td>
<td>i, y</td>
<td>u</td>
</tr>
<tr>
<td>Close-mid</td>
<td>e, ø</td>
<td></td>
</tr>
<tr>
<td>Mid</td>
<td></td>
<td>ø</td>
</tr>
<tr>
<td>Open-mid</td>
<td>e, œ</td>
<td>A</td>
</tr>
<tr>
<td>Open</td>
<td>a</td>
<td>a</td>
</tr>
</tbody>
</table>

specifying the position in this space. This provides us with a way to break down the phonetic inventory of Dutch into various subsets depending on its purely phonetic properties and without taking into account any peculiarities of Dutch phonology. Tables 5.1 and 5.2 show the parts of the IPA consonant and vowel tables that are pertinent to Dutch.

5.4.1 Design

The background predicates for this experiment are placing each phone in its position on the IPA chart along the dimension (feature) that each predicate is encoding. With the relevant parts of the IPA chart reproduced in tables 5.1 and 5.2, it can be seen that five such predicates must be defined: place/2 and manner/2 referring to both consonants and vowels, voiced/2 for consonants, and rounded/2 for vowels.

These predicates are extensionally defined to relate each value of the feature they implement with all the phones that carry that value. So, for example, some of the clauses of the manner/2 predicate are:
5.4. The IPA Chart

manner(plosive, p). manner(plosive, b).
manner(nasal, m). manner(nasal, n).
manner(open, a). manner(open, o). manner(open, u).

and so on.

Note that the end-of-word mark has no phonological features whatsoever and it does not belong to any of the partitions of either $C$ or $V$.

5.4.2 Results

The evaluation function used was the Laplace estimated accuracy (see Section 2.2.3). Since the randomly generated negatives must also contain false negatives, it cannot be expected that even a good theory will fit it perfectly. In order to avoid over-fitting, the learning algorithm was set to only require an accuracy of 85% over the training data.

The resulting hypothesis consisted of 199 prefix and 147 suffix clauses and achieved a recall rate of 99.3% with 89.4% precision.

All the false negatives were rejected because they couldn’t get their onset licensed, typically because it only appears in a handful of loan words. The /d/ onset necessary to accept ‘jeep’ and ‘junk’, for example, was not permitted and so these two words were rejected.

The most generic rules found were:

```
prefix(A,B,C) :- A= '^^'.
prefix(A,[],C).

affix(A,B,C) :- A= '^^'.
affix(A, [], C).
```

meaning that (a) the inner-most consonant can be anything, and (b) all sub-prefixes (-suffixes) of a valid prefix (suffix) are also valid.

There is also a few pairs of rules that only differ in a couple of literals. This suggests that if a richer feature system would include features that effectively disjoin the features of this system, these pairs could be collapsed to one rule each. To give an example, consider these two rules:

```
prefix(A,B,C) :-
    head(B,D), manner(approx,D), head(C,E), length(short,E),
    place(front,E), voiced(minus,A).
prefix(A,B,C) :-
    head(B,D), manner(approx,D), head(C,E), length(short,E),
    place(front,E), manner(plosive,A), place(alveolar,A).
```
The first rule prefixes devoiced consonants to \(<\text{approximant}>\)<short front vowel> sequences, for example allowing the /k/ in /kwilt/ ‘quilt’. The second rule prefixes alveolar plosives to the same sequences, allowing words like /dwin/ ‘dwing’. Devoiced alveolar plosives in words like /twist/ ‘twist’ are licensed by both rules.

These two rules could have been collapsed to one if a feature like ‘devoiced consonant or alveolar plosive’ was available. This particular disjunction might be unintuitive or impossible to independently motivate, but it suggests that a redundant feature set might allow for more compact theories than the minimal, orthogonal one used for this experiment. This is particularly true for a system like Aleph, that performs no predicate invention or background theory revision.

5.5 Feature Classes

For this experiment a richer (but more language-specific) background knowledge was made available to the inductive algorithm, by implementing the feature hierarchy suggested by Booij [4, ch. 2].

5.5.1 Design

The various phones are, again, accessed by means of feature vectors, but in this case the features are not orthogonal dimensions, but are forming the feature hierarchy shown in figure 5.1.

In this figure features are given in [brackets], with the remaining symbols being feature classes. All features are binary, and not all segments carry all features. Bearing a feature, however, makes it obligatory to also bear all its ancestors all the way to the root.

It follows that the major class features of the root node are always present. These are the features CONSONANT and SONORANT that divide the segment space into vowels [CONSONANT→, SONORANT[,]], obstruents [CONSONANT+, SONORANT→] and sonorant consonants [CONSONANT+, SONORANT+]. Since all vowels are sonorous, [CONSONANT→, SONORANT→] is an invalid combination, a restriction which is encoded in the background knowledge.

The features specifying the continuants, nasals and the lateral /l/ are positioned directly under the root node, with the rest of the features bundled together under two feature classes, those of the laryngeal and the place features.

Features that are bundled together in feature classes have to be all present or all missing from the specification of a segment, and their presence also
marks the segment as belonging to a certain class. So, for example, all (and only) laryngeals bear the voiced-voiceless distinction, with ASPIRATION separating /h/ from the rest. Feature classes are chosen so that they collect together features that behave as a unit in phonological processes of Dutch. In other words, the laryngeal features are bundled together because (a) it is useful to be able to collectively refer to them when talking about Dutch phonology, and (b) there are some segments for which all of these features are pertinent, but no segments for which only some of these features are pertinent.

It has to be noted that when it was mentioned that all features of a class have to be either present or missing, we had in mind features appearing in a segment’s definition. When employing the feature hierarchy to refer to sets of features, under-specified feature vectors such as [CONSONANT+, BACK+] are both meaningful and useful.

Furthermore some derived or redundant features such as GLIDE, APPROXIMANT and LIQUID are defined. The vowels do not include the schwa, which is set apart and only specified as SCHWA+.

### 5.5.2 Results

The Prolog implementation of model described above consists of a database of segments where the values for all the features of each segment are stored. For example:

```
seg(k ,+, -, larynx(-, -), -, -, - , [dorsal(+,+,+)] ).
```
specifies /k/ as being [CONSONANT+, SONORANT—, larynx(VOICED—, ASPIRATED—), CONTINUANT—, NASAL—, LATERAL—, Place(dorsal(BACK+, HIGH+, MID—))].

In addition feature-access methods like the following are provided:

sonorant(Seg, Son) :- seg(Seg, _, Son, _, _, _, _).
larynx(Seg, Lar) :- seg(Seg, _, _, Lar, _, _, _, _).

\[
\text{voiced(larynx(Val,\_), Val).}
\]
\[
\text{aspirated(larynx(\_\_,Val), Val).}
\]

to access the data in the \text{seg/8} terms. The relevant semantic information is also provided in the background, so that the variables used as input in \text{voiced/2} and \text{aspirated/2}, will always be \text{larynx/2} terms:

\[
\begin{align*}
\text{:- mode(1, larynx(+seg,-lar) ).} \\
\text{:- mode(1, voiced(+lar,-voic)).}
\end{align*}
\]

The fact that the, say, larynx-related features are placed together under one term, means that they can be referred to individually but also collectively; whereas, for example, CONTINUANT and NASAL cannot. So it is possible to express the concept ‘same laryngeal features’ with fewer body literals that the concept ‘same CONTINUANT and NASAL features’:

\[
\begin{align*}
\text{same_laryngeal(Seg1, Seg2) :-} \\
\text{larynx(Seg1, L), larynx(Seg2, L).}
\end{align*}
\]
\[
\begin{align*}
\text{same_cont_nasal(Seg1, Seg2) :-} \\
\text{cont(Seg1, C), cont(Seg2, C),} \\
\text{nasal(Seg1,N), nasal(Seg2,N).}
\end{align*}
\]

which constitutes bias towards trying first the relation which the phonological model is considering as more common in Dutch phonology, or somehow more interesting for Dutch phonological descriptions.

Using the Laplace estimated accuracy as evaluation function and the background described above, the constructed theory consisted of 13 prefix and 93 suffix rules, accepting 94.2\% of the test positives and under 7.4\% of the test negatives.

Among the rejected positives are loan words (‘jeep’ and ‘junk’ once again), but also all the words starting with perfectly Dutch /s/ - obstruct - liquid clusters like the onset of /straat/ ‘straat’. The prefix rule with the widest coverage is:

\[
\begin{align*}
\text{prefix(A,B,C) :-} \\
\text{head(C,D), sonorant(D,plu), rest(B,[]).}
\end{align*}
\]
or, in other words, ‘prefix anything before a single consonant before a nucleus other than the schwa.’

The suffix rules were less strict, with only 3 rejected positives, ‘branche’, ‘dumps’ and ‘krimpst’ that failed to suffix /j/, /s/ and /s/ respectively. Note that the first two of the false negatives are loan words and the remaining one is a superlative. Dutch superlatives are notoriously difficult to handle, and hand-crafted models of Dutch phonotactics also have to make special rules to treat them.

Some rules achieve wide coverage (although never quite as wide as that of the prefix rules,) but some make reference to individual phonemes and are of more restricted application. For example:

\[
\text{suffix}(A,B,C) :-
\text{rest}(C,D), \text{head}(D,E), \text{rest}(E,[]), A=t.
\]

or, ‘suffix a /t/ after exactly one consonant, if the nucleus is a long vowel or a diphthong.’

Of some interest are also the end-of-word marking rules (see in section 5.1 above about the ^ mark), because of the fact that open, short monosyllables are very rare in Dutch (there are four in CELEX: ‘schwa’, ‘ba’, ‘he’, and ‘joh’). This would suggest that the best way to treat those is as exceptions, and have the general rule disallow open, short monosyllables. What was learnt instead was a whole set of 29 rules for suffixing ^, the most general of which is:

\[
\text{postvoc}(A,B,C) :-
\text{head}(B,t), \text{larynx}(t,E), \text{rest}(B,F),
\text{head}(F,G), \text{larynx}(G,E), A= ‘^’.
\]

or ‘suffix an end-of-word mark after at least two consonants, if the outer-most one is a /t/ and has the same values for all the features in the LARYNGEAL feature class as the consonant immediately preceding it.’ In fact, suffix suffix obstruent (non-sonorant consonant) clusters in Dutch are always devoiced, and thus share their laryngeal features. This rule displays this phenomenon, although fails to capture it in its complete generality.

A final note that needs to be made regarding this experiment, is one regarding its computational complexity. Overlapping and redundant features might offer the opportunity for more interesting hypotheses, but are also making the search space bigger. The reason is that overlapping features are diminishing the effectiveness of the inverse resolution operator at keeping uninteresting predicates out of the bottom clause: the more background predicates can be used to prove the positive example on which the bottom clause is seeded, the longer the latter will get.
<table>
<thead>
<tr>
<th>phoneme</th>
<th>obstruents</th>
<th>m</th>
<th>n</th>
<th>l</th>
<th>r</th>
<th>glides</th>
<th>vowels</th>
</tr>
</thead>
<tbody>
<tr>
<td>sonority</td>
<td>1</td>
<td>2</td>
<td>2.25</td>
<td>2.5</td>
<td>2.75</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 5.3: The Sonority Scale

5.6 Sonority Scale

The baseline theory of section 5.3 serves as a lower limit for the machine-learnt theories, in terms of size as well as predictive power: it simply stores the examples (thus performing no compression) and makes no generalization from the original data (thus performing very poorly on unseen data). To better appreciate the performance and compression rates of the two experiments described above, a hand-crafted model of Dutch syllabic structure is implemented as an upper limit of how well it is possible for any theory to perform. The model implemented is the one suggested by van der Hulst [87, ch. 3].

5.6.1 Design

The Dutch syllable is analysed by van der Hulst as having three prevocalic and 5 postvocalic positions, (some of which may be empty) and constraints are placed on the set of consonants that can occupy each.

The most prominent constraint is the one stipulating a high-to-low sonority progression from the nucleus outwards. Each phoneme is assigned a sonority value (table 5.3) based not only on language-independent features such as it being a SONORANT or an OBSTRUENT, but also because of syllable structure of Dutch itself. Especially the fine tuning done with respect to the sonority values of the nasals and the liquids is explicitly justified by the need to filter out impossible consonant clusters that would otherwise be predicted by the simpler model. It must, therefore, be noted that the one of the ‘background predicates’ is used is not only language-specific, but is also directly aimed at solving the very problem that is being investigated.

In addition to the high-to-low sonority-level progression from the nucleus outwards, there are both filters and explicit licensing rules. The former are restrictions referring to sonority (e.g. ‘the sonority of the three left-most positions must be smaller than 4’) or other phonological features (e.g. the ‘no voiced obstruents in coda’ filter in p. 92) and are applicable in conjunction with the sonority rule. The latter are typically restricted in scope rules that take precedence over the sonority-related constraints mentioned so far. The left-most position, for example, may be /s/ or empty, regardless of the
5.6. Sonority Scale

contents of the rest of the onset.

5.6.2 Implementation and Results

The core clauses in the implementation of the model were the ones enforcing the sonority-level progression:

\[
\text{suffix}(A, [B|\_], \_):- \\
\quad \text{sonority}(A, SA), \text{sonority}(B, SB), SA < SB.
\]

The additional filters required by the model are implemented as predicates like the following one:

\[
\text{filter1}(A) :- \text{larynx}(A, \text{Lar}), \text{voiced}(	ext{Lar}, -).
\]
\[
\text{filter1}(A) :- \text{sonorant}(A, +).
\]
\[
\text{filter1}(\_).
\]

which directly implements the ‘no voiced obstruents in coda’ filter. As it can seen from the example, whenever it was necessary to refer to phone classes, the ‘background predicates’ used were the ones from Booij’s feature-classes model of Dutch phonology (see section 5.5).

These filter predicates are then added to the body of the licensing rules to enforce the restrictions, so that the suffix/3 clause given above would actually be:

\[
\text{suffix}(A, [B|\_], \_):- \text{filter1}(A), \\
\quad \text{sonority}(A,SA), \text{sonority}(B,SB), SA < SB.
\]

In total, the implementation required 11 clauses (three prefix and eight suffix clauses and filter-predicate clauses), but it must be noted again that one of the predicates referred to (the sonority/2 relation) is extremely informed with respect to the problem at hand, so this lower size limit is not tight if one restricts the background predicates to not be informed about the actual solution of the problem.

This basic sonority progression rule as well as the most widely-applicable filters and rules\(^1\) yielded an impressive compression rate and it also performed comparably to the two previous experiments, at 93.1% recall and 83.2% precision.

\(^1\)Some were left out because they were too lengthy when translated from their fixed-position framework to the affix licensing one used here, and were very specifically fine tuning the theory to individual onsets or codas.
5.7 The Search Space

As mentioned in section 5.2 above, the background predicates should be providing meaningful and intuitive ways of accessing subsets of $\mathcal{C}$ and $\mathcal{V}$. One important aspect of the background theory is how finely grained it is and how many different ways or overlapping viewpoints it is providing for accessing these subsets. The most useful background theories will then be the ones that are most informed about the target concept; that is, the ones that most tightly circumscribe the minimum background necessary to express the target concept: at the limit all and only those background predicates will be present that are to be found in the bodies of clauses of the (ideal) target predicate.

The reason for this, as explained below, is that richer background theories might allow for more interesting hypotheses to be constructed, but they also imply an increase in the size of the bottom clause and, subsequently, the space within which the search for a hypothesis has to be performed.

5.7.1 Bottom Clause Size

In the experiments described in sections 5.4 and 5.5 the ‘grain’ of the background predicates was the same (that is, all individual phones were addressable), but the feature-class background of sections 5.5 was, nevertheless, richer and allowed more ways to access subsets of $\mathcal{C}$.

The bottom clause is a minimal generalization of an example, where all ground atoms have been replaced by variables, and a body has been constructed where these variables appear in the body literals in all the ways that are (a) permitted by the semantics defined for the predicates that appear as body literals, and (b) consistent with the requirement that the original example and only that can be derived from the bottom clause. (See section 2.3.2 about the bottom clause in general and 2.4 about the specifics of the Progol algorithm used here.)

In more concrete terms, this means that in the context of a richer, more redundant background theory the saturation (that is, bottom-clause construction) process will yield longer bottom clauses. This is corroborated by experimental data taken from the two experiments of this chapter: when saturating all the available positive examples, the mean bottom clause length is 14.38 literals ($\sigma^2 = 2.58$) for the IPA experiment and 26.35 ($\sigma^2 = 7.95$) for the Feature Classes experiment. The density functions of the bottom clause sizes can be seen in Fig. 5.2, estimated on a Gaussian kernel (bandwidth = 4). What can be seen in that figure in particular, is that the high deviation of the second experiment’s bottom clause size is due to the bi-modality
Figure 5.2: The density function of the Bottom Clause sizes. The dashed line is from the IPA experiment (section 5.4) and the solid line from the Feature Geometry experiment (section 5.5).
that it exhibits, with most of the bottom clauses clustering around length 22 and length 36, whereas the the curve derived from IPA experiment is much smoother.

5.7.2 Search-Space Size

The advantage of not including background predicates that will not be employed by the final theory is that it reduces the space that needs to be searched, since its size is directly related to the number of background predicates that are present in the bottom clause constructed for each example that gets saturated. In the absence of syntactic bias to restrict the ways in which background predicates can be combined, all possible subsets of the bottom clause are valid generalizations of the example and the search space consists of the power-set of the body of bottom clause; it thus grows exponentially as a function of the size of the bottom clause.

In practice, the number of nodes actually visited is smaller that the total size of the search space. Fig. 5.3, for example, plots the number of nodes actually visited during each reduction vs. the size of the bottom clause that seeded that particular reduction (data is taken from the IPA experiment). Although quite sharp, the log-linear curve seems to be sub-exponential.

This sharp increase in the size of the search can be checked by imposing very restrictive syntactic bias, so that the actual search space is much smaller.
than the raw space defined by the bottom clause. It can also be by-passed by employing a high-quality, informed evaluation function that (acting as a heuristic) will quickly guide the search towards a satisfactory clause without having to consider most of the space. Or, finally, the space can be kept small enough to be searched efficiently by keeping the size of the bottom clause small. It can be seen then that in all three cases the key is prior knowledge available regarding syntactic and semantic properties of the target concept itself, as contrasted to prior knowledge in general which includes information about the target as well as the ‘universe’ in which it operates.

In most non-trivial applications this information will be limited, and its discovery the very task of ILP. The usual trade-off is then between the computationally expensive choice of including all relations known to hold in the experiment’s universe and imposing no syntactic restrictions on the one hand, and on the other hand severely restricting the search space and running the risk of missing good solutions. This trade-off can also be seen when comparing the background described in section 5.4 with that of section 5.5 above. Although the grain is equally fine in both experiments (in the sense that it is possible to refer to all possible subsets of consonants and vowels), the former experiment (based on the IPA table) employs a system of orthogonal features that implement a minimal (in terms of features × values) set of methods to access the members of $C$ and $V$. By contrast the feature hierarchy described in section 5.5 is richer in that it offers multiple ways of looking at the data, which amounts to being redundant. This allows for potentially more interesting theories to arise, but also results in longer bottom clauses and a larger search space.

### 5.7.3 Data-Parallelism Vs. Or-Parallelism

Predicates consist of multiple clauses which represent multiple ways for the predicate to be satisfied; individual clauses might also include explicit OR operators in their bodies; finally there might be multiple ways to instantiate the variables in a clause’s literals. All these situations constitute choicepoints for a Prolog engine, where one of many alternative paths through the proof search space has to taken. The Prolog backtracking mechanism will exhaust each path before backtracking to the last choicepoint and trying the next option, and so on until the goal is satisfied or there are no choices left and the goal fails.

An Or-parallel Prolog implementation is one that tries all the clauses at a choicepoint in parallel. Or-parallelism can be used to divide the search space of the ILP algorithm in ‘sectors’ which will be searched in parallel by the nodes of the machine, since the ILP algorithm’s search is actually the search
for a proof: each time a literal is added to the clause under construction it is picked from the ‘pool’ of literals provided by the bottom clause. In (very schematic) Prolog this is implemented along the lines of this code fragment:

\[
\text{add_one_literal}(C, \text{NextC}) :- \\
\text{bottom_clause( (Head:-Body) )}, \\
\text{member(Lit, Body)}, \\
\text{append(Lit, C, NextC)}, \\
\text{is_good_clause(NextC)}.
\]

which will backtrack back to \text{member/2} each time \text{is_good_clause/1} is not satisfied, until all the member of the bottom clause have been tried out. An Or-parallel Prolog implementation would employ the nodes of a parallel machine to try out all the ways to instantiate \text{Lit} in \text{member(Lit,Body)} in parallel.

The data-parallelism described in chapter 3, by comparison, is a parallel implementation of the evaluation function employed by \text{is_good_clause/1} to decide whether to accept the current clause or not. It should, then, be noted that the computation expense discussed here cannot be treated by data-parallelism, since most of the time is consumed in constructing candidate clauses and traversing the search space, which means that the bottleneck is not the large amount of data against which each hypothesis needs to be tested.

### 5.8 Conclusions

The quantitative results from the ILP experiments presented above are collected in Table 5.4, together with those of an abductive approach to the same problem by Tjong Kim Sang and Néron [53, Section 4.3], and the results from the baseline and the sonority scale implementations. Those last ones in particular are listed for comparison’s sake and as the logical end-point of the progression towards more language- and task-specific prior assumptions.

The second and third rows are directly comparable, because they both refer only to phonetic primitives without any phonologically motivated background knowledge. Furthermore the fact that the $C_3VC_5$ template assumed in this work is not taken for granted in the abductive experiment is taken into account in terms of compactness as well as performance, since (a) the 1154 rules of Table 5.4 only refer to affixation to an already formed ‘basic word’, which is described by 41 extra rules not included in the affix-rule count, and (b) in the abductive experiment precision is measured on random strings, whereas here only strings matching the $C_3VC_5$ template are
5.8. Conclusions

<table>
<thead>
<tr>
<th></th>
<th>Recall</th>
<th>Precision</th>
<th>Size</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>47.1%</td>
<td>98.9%</td>
<td>12133</td>
<td>13171</td>
</tr>
<tr>
<td>(Tjong et al., 2000)</td>
<td>99.1%</td>
<td>74.8%</td>
<td>577</td>
<td>577</td>
</tr>
<tr>
<td>(Stoianov, 2001), $\theta = 1.6$</td>
<td>95.0%</td>
<td>95.0%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Stoianov, 2001), $\theta = 1$</td>
<td>99.0%</td>
<td>81.0%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPA</td>
<td>99.3%</td>
<td>79.8%</td>
<td>145</td>
<td>36</td>
</tr>
<tr>
<td>Feat. Classes</td>
<td>94.2%</td>
<td>92.6%</td>
<td>13</td>
<td>93</td>
</tr>
<tr>
<td>Sonority</td>
<td>93.1%</td>
<td>83.2%</td>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 5.4: Results

used. As can be seen, then, the ILP-constructed rules compare favourably (in both performance and hypothesis compactness) with those constructed by abduction.

One should also make a note of the results reported by Stoianov and Nerboune [81] and Stoianov [80]. There, neural networks (and, more precisely, SRN) were trained to predict the following phoneme in a word based on the candidate word so far. When using the network on unseen data, Stoianov [80, p. 90] plots the error rates for different thresholds\(^2\) where the negative data is randomly generated strings fitting a $C_4 V C_3$ template (idem., p. 83).

If we pick the optimal threshold to be the point where the false negatives and the false positives error rates are equal, that is, the intersection of the lines labelled (A) and (I) on the graph (threshold $\theta = 1.6$), we get an error rate of 5%. If we assume equal number of positive and negative examples, then we can interpret the error rates in the graph as precision and recall of 95%, slightly better than the results of the Feature Classes experiment. If we pick the point $\theta = 1$ where the false negatives are comparable to the 99.3% recall of the IPA experiment, the false positives approach 22%, yielding a precision of around 81%. (always under the assumption that the number of positive and negative examples is equal.) Please note the assumptions made in this paragraph and also that these calculations are rather rough, since the figures used are read off a graph and are not the exact measurements. They do, however, suggest similar performance as the one achieved here.

What can be also seen by comparing the two ILP results with each other, is that the drop in recall between the the third and fourth row is compensated by higher precision and compression, suggesting a direct correspondence between the quality of the prior knowledge encoded in the background theory and that of the constructed hypothesis.

\(^2\)The threshold is effectively the point of balance between favouring recall and favouring precision.
Chapter 5. Phonotactics

One interesting follow-up to these experiments would be attempting to expand their domain to that of syllables of multisyllabic words and, eventually, full word-forms. In the interest of keeping the problems of syllabic structure and syllable-boundary phonology apart, a way must be devised to derive from the positive data (i.e. a corpus of Dutch word-forms) examples for a distinct machine learning session for each task.

Given the discussion of Section 5.7 on data-parallelism versus or-parallelism, it would also be interesting to port an ILP system such as Aleph to an or-parallel Prolog compiler. Such a system on which Aleph could be ported is YapOr [71], based on the Yap Prolog compiler.