Morphological Blocking

in a Computational Context

Christopher Graham

Morphology LIN121

Raúl Aranovich

University of California, Davis

June 9, 2008
The problem of morphological blocking and its inner-workings are a common issue in modern morphological debate. Computational linguistics, a growing field, may offer some insights into how morphological blocking works, why it happens, and what the problems are with current theories on the matter. In this paper, I will first address the issue of blocking by defining it, explaining it, and then offering forth a theory which attempts to elucidate the phenomenon. I will then offer a small computational simulation of blocking, and describe what the results of that simulation imply about the theory.

**A Definition of Blocking**

There are not many concepts in morphology considered as puzzling or mysterious as that of blocking. Blocking is described by linguist Mark Aronoff as “the nonoccurrence of one form due to the simple existence of another” (1976, p. 43). By that definition, it may seem as if morphological blocking is a trivial matter; upon further investigation, however, there is no triviality.

A native speaker of English can easily observe the results of morphological blocking by use of their own linguistic intuition. Consider the following two words of English: *ox* and *fox*. These two words rhyme, but other than that, have no overt specialty. They are regular nouns, and neither is derived. They are both monomorphemic. There is nothing separating these two nouns, typologically speaking. Consider, however, the plural forms (1).

<table>
<thead>
<tr>
<th>Noun</th>
<th>Plural</th>
</tr>
</thead>
<tbody>
<tr>
<td>fox</td>
<td>fox + es</td>
</tr>
<tr>
<td>ox</td>
<td>ox + en</td>
</tr>
</tbody>
</table>
As previously stated, there is nothing overtly different about *fox and *ox, and yet they each utilize a different form of the English plural morpheme. This phenomenon is described as blocking.

The theory, as may be gathered by Aronoff’s definition of the term above, is that the form *oxen blocks a hypothetical form *oxes by its very existence. To the contrary, there is no special form *foxen. Due to the nonexistence of that hypothetical form, another plural, foxes, is generated from the standard rules of pluralization in English and used instead. The exceptional forms, of course, are marked, and forms generated by the addition of a standard plural morpheme are unmarked by comparison.

This generalization may be represented as in (2), below, in which the existence of marked, exceptional forms in column 2 blocks any regular forms in column 3.

(2)

<table>
<thead>
<tr>
<th>Noun</th>
<th>Marked Exceptions</th>
<th>Unmarked Forms</th>
</tr>
</thead>
<tbody>
<tr>
<td>fox</td>
<td>———</td>
<td>fox + es</td>
</tr>
<tr>
<td>ox</td>
<td>ox + en</td>
<td>*ox + es</td>
</tr>
</tbody>
</table>

**A Phonological Explanation of Blocking**

Kiparsky, in 1982, further refined this concept of blocking with a possible explanation of the phenomenon by using phonology (in Iverson & Wheeler, 1988). According to the revised theory, the absence of unmarked forms is the result of the existence of marked exceptions, as was the case with the previous definition (1988). However, the existence of marked exceptions is governed by certain special phonological rules, from which the forms are derived (1988). Kiparsky posited the blocking among some English verbs and their irregular past tense forms as evidence for his claim (1988). Some of these forms are listed below (3).
Verbs with similarities | Marked Exceptions | Unmarked Forms
--- | --- | ---
keep | kep + t | *keep + ed
weep | wep + t | *weep + ed
meet | me + t | *meet + ed

Such verbs (and other forms subject to blocking), according to Kiparsky, have roots which share some phonological environment that triggers the derivation of the marked exceptions (1988). Thus, it follows that the unmarked forms are blocked from the lexicon, based on the definition provided by Aronoff (1976).

This phonological theory, however, is weak at best at explaining morphological blocking. It is easy to come up with an example form that should retain the elusive phonological environment that is said to govern the derivation of marked exceptions, but does not itself have an exceptional form. Consider the English verb *seep*, which would seem extremely similar to *keep* phonologically, and yet has a regular past-tense form *seeped* in the lexicon (and therefore no irregular exception). The level of detail required to separate *keep* from *seep* with a phonological rule would seem to paint this phonologically-conditioned theory as highly unlikely.

**Lexical Phonology and Blocking**

Iverson and Wheeler (1988) detail another explanation for blocking termed “Lexical Phonology” (also detailed by Kiparsky). This theory involves the existence of hypothetical strata which govern the ordering of morphological and phonological processes (1988). Certain processes are covered by stratum 1, whereas certain processes are handled under stratum 2 (1988). Morphological and phonological processes occur in parallel; there is no ordering...
preference between the two within a stratum, and both are accepted to take place simultaneously (1988). A graphical representation of such a stratum model is shown below (4).

Lexical Phonology can provide a fairly complete explanation of morphological blocking. Irregular formations are handled in stratum 1, whereas regular forms are handled in stratum 2 (1988). This would seem to be congruent with Aronoff’s definition of morphology, wherein exceptional forms blocked regular forms (1976). Once an irregular, exceptional form has been derived in stratum 1, a regular derivation (handled in stratum 2) cannot take place (Iverson & Wheeler, 1988).

Note that there is an implication in the lexical phonology model: in order for a lexical entry to be irregularly derived, it must be marked as an exception before it reaches the strata. If that wasn’t so, it would logically follow that each and every case would be derived in a marked fashion (since irregular inflection is handled in stratum 1), and no surface regularities would exist. Thus, the stratum theory does well at explaining why marked exceptions block unmarked regularities, but does little to explain why such exceptions are called for on some lexical entries and not on others.
The Elsewhere Condition

A greater generalization of morphological blocking is described by Ackema and Neeleman, who relate blocking to the so-called “elsewhere condition” (2004). The elsewhere condition states simply that “a general rule is blocked where a more specific rule can apply” (2004, p. 48). The elsewhere condition may be demonstrated across linguistic subfields (2004).

For example, there is a morphological suffix -er in English which is comparative; it can be added to forms like big to make forms like bigger, which denotes a relation between two entities in a clause (2004). Note, however, that a syntactic formation like *more big appears to be blocked by the existence of bigger, much like the regular form *oxes is blocked by the irregular oxen (2004). Thus, the elsewhere condition is a decent description of blocking, and one that is not restricted to morphology.

Morphological blocking, as a whole, is generally considered to be a result of the application of the elsewhere condition (Andrews, 1990). The elsewhere condition is congruent with the aforementioned stratum theory of lexical phonology as well, albeit a modified version; one could easily consider an expanded theory in which morphology and syntax are each handled in their respective sets of strata, thus explaining not only morphological blocking, but morpho-syntactic blocking of forms like bigger over forms like *more big. This expanded model, however, would almost certainly not be free from its own set of problems and exceptions.

Computational Implementation of Morphological Blocking

Computational linguistics offers a unique methodology for examining linguistic phenomena. Language is a complex entity, and computers are complex devices. In order for a computer to process language, however, every minute detail must be described. If a computer program designed to simulate morphological blocking produces an invalid form, for example, the
programmer knows that a greater level of detail is required to fully explain the phenomenon.

Understanding of that phenomenon, in turn, is increased by necessity; a computer cannot use an incomplete theory to produce complete results. Using computers to attempt to simulate blocking, then, can provide interesting insights as to the methodology the human brain might use to process it.

There are clear differences, of course, between computers and the human brain. Computers are procedural, meaning they process information in the order in which it is received, which might seem to go against intuitions relative to how the brain processes information. The artificial intelligence movement of the late 1960s and early 1970s saw the invention of a programming language specifically designed to operate more like the brain (Bratko, 2001). PROLOG (short for Programming in Logic) was developed by Robert Kawalski, Maartin van Emden and others (2001), and is currently commonly used to simulate natural language.¹

As previously mentioned, the stratum model does not explain why certain forms are marked as exceptions in the lexicon, only the relative ordering of irregular versus regular inflection. As a result, one can use this as a basis for a basic blocking simulation written in PROLOG, which will simulate one irregular inflection and one regular inflection, and provide a starting point for adding other inflections later.

To begin, a list of lexical entries (English verbs, in this case) is populated with forms that are marked for both regular and irregular inflections in the past tense (5).

<table>
<thead>
<tr>
<th>Verb</th>
<th>Type of inflection</th>
</tr>
</thead>
<tbody>
<tr>
<td>keep</td>
<td>irregular</td>
</tr>
<tr>
<td>weep</td>
<td>irregular</td>
</tr>
</tbody>
</table>

¹ The PROLOG code used in this paper is designed for use with SWI-PROLOG, version 5.1.
These words will then be transcribed into an IPA substitute compatible with PROLOG syntax (6). Note that IPA \(i\) in PROLOG is represented as \(ii\), and IPA \(ɛ\) is represented as \(e\).

<table>
<thead>
<tr>
<th>Verb</th>
<th>Transcription</th>
</tr>
</thead>
<tbody>
<tr>
<td>keep</td>
<td>([k, ii, p])</td>
</tr>
<tr>
<td>weep</td>
<td>([w, ii, p])</td>
</tr>
<tr>
<td>seep</td>
<td>([s, ii, p])</td>
</tr>
<tr>
<td>peep</td>
<td>([p, ii, p])</td>
</tr>
</tbody>
</table>

With (6) in mind, several facts may be stated in PROLOG representative of the data. Each verb will be listed phonologically, and as either a regular or irregular verb (7). Note that a simple predicate name “english” will be used to denote each fact.²

```
(7) english([k, ii, p], [verb, irregular]).
english([w, ii, p], [verb, irregular]).
english([s, ii, p], [verb, regular]).
english([p, ii, p], [verb, regular]).
```

With only one regular and one irregular inflection, the simulation here can be very simple, and will contain only two predicates. The first predicate (8) will handle the irregular inflection. This predicate may be thought of as stratum 1 in the lexical phonology model; it will handle the vowel ablaut in the middle of the root, and add a past-tense morpheme \(t\) to the end of the verb. The result of the predicate will be listed in PROLOG’s lexicon as a new class of verb, called final.

---
² Ivan Bratko’s PROLOG Programming for Artificial Intelligence is a good book for learning PROLOG. As an explanation of PROLOG syntax would certainly take many pages of explanation, code snippets are listed with only general descriptions of their function. Intricacies of PROLOG syntax are left to the reader to pursue.
A second predicate will handle the regular inflections, which consist of the addition of the plural morpheme *t* (9).

\[(9)\]
\[
\text{english([A, B, C, t], [verb, final]) :-}
\text{english(Verb, [verb, regular]),}
\text{Verb = [A, B, C].}
\]

Upon executing the program, PROLOG will come up with 4 forms classified as final verbs (10).

\[(10)\]
\[
\text{english([k, e, p, t], [verb, final]).}
\text{english([w, e, p, t], [verb, final]).}
\text{english([s, ii, p, t], [verb, final]).}
\text{english([p, ii, p, t], [verb, final]).}
\]

**Implications of the Computational Model**

The computational model used here (shown in its entirety in Appendix A) handles the four terms in its lexicon correctly, producing proper forms for both regular and irregular inflection. At first glance, it would seem that the computer has processed language in much the same way that the aforementioned stratum model suggested.

While the computational model listed here is congruent with the model of lexical phonology as defined by Iverson and Wheeler (1988), there is a subtle and interesting difference. Forms in lexical phonology are blocked by virtue of the existence of an exception, as defined by Aronoff (1976). This can be explained, as has been mentioned, by way of a stratum model. However, the first predicate (stratum 1) and the second predicate (stratum 2) in the computational simulation do not act as they should within the model; forms marked as regular never pass
through stratum 1. Instead, they skip over it, and go directly to stratum 2. While this might work for the small lexicon included with the program, a real-world simulation would have to deal with this problem. For example, by splitting the lexicon into two classes (regular and irregular), the program would be unable to mate a regular noun with a non-neutral affix, due to the fact that it would never be processed in stratum 1.

This result, stemming from a computational methodology, showcases the shortcomings of the lexical phonology stratum model. How are lexical items marked to be inflected regularly or irregularly? How can they be processed in one stratum for one thing and another stratum for the other? Lexical phonology offers a good insight into the basics of blocking, but leaves many questions about the true inner-workings of the morphological system unanswered. Perhaps a complete, real-world morphological simulation programmed somewhere down the line will offer answers to these ever elusive questions.
Appendix A: Simulation Code

CODE:

english([k, ii, p], [verb, irregular]).
english([w, ii, p], [verb, irregular]).
english([s, ii, p], [verb, regular]).
english([p, ii, p], [verb, regular]).

english([A, e, B, t], [verb, final]) :-
   english(Verb, [verb, irregular]),
   Verb = [A, _, B].

english([A, B, C, t], [verb, final]) :-
   english(Verb, [verb, regular]),
   Verb = [A, B, C].

OUTPUT:

   Prompt: english(X, [verb, final]).

   english([k, e, p, t], [verb, final]).
   english([w, e, p, t], [verb, final]).
   english([s, ii, p, t], [verb, final]).
   english([p, ii, p, t], [verb, final]).
Works Cited


