Oxford Research Encyclopedia of Linguistics

Computer-Based Tools for Word and Paradigm Computational Morphology

Raphael Finkel

Subject: Computational Linguistics Online Publication Date: Apr 2016
DOI: 10.1093/acrefore/9780199384655.013.162

Summary and Keywords

The Word and Paradigm approach to morphology associates lexemes with tables of surface forms for different morphosyntactic property sets. Researchers express their realizational theories, which show how to derive these surface forms, using formalisms such as Network Morphology and Paradigm Function Morphology. The tables of surface forms also lend themselves to a study of the implicative theories, which infer the realizations in some cells of the inflectional system from the realizations of other cells.

There is an art to building realizational theories. First, the theories should be correct, that is, they should generate the right surface forms. Second, they should be elegant, which is much harder to capture, but includes the desiderata of simplicity and expressiveness. Without software to test a realizational theory, it is easy to sacrifice correctness for elegance. Therefore, software that takes a realizational theory and generates surface forms is an essential part of any theorist's toolbox.

Discovering implicative rules that connect the cells in an inflectional system is often quite difficult. Some rules are immediately apparent, but others can be subtle. Software that automatically analyzes an entire table of surface forms for many lexemes can help automate the discovery process.

Researchers can use Web-based computerized tools to test their realizational theories and to discover implicative rules.

Keywords: Network Morphology, Paradigm Function Morphology, DATR, KATR, principal parts

1. Background

This article presents several Web-accessible tools the author has created (in collaboration with Gregory Stump) to allow morphologists to experiment with their theoretical understanding of specific languages and to analyze the complexity of specific languages. The tools are publicly available at http://www.cs.uky.edu/~raphael/linguistics/claw.html. This site also includes sample inputs for each of the tools.

The theoretical underpinnings of these tools are Network Morphology, Paradigm Function Morphology, and Principal Part Analysis. All three are in the general category of *Word and Paradigm*, in which "a lemma [or lexeme] is associated with a table, or paradigm, that associates a morphological variant of the lemma with a morphosyntactic property set" (Hippisley, 2010). More formally,

> The *inflectional paradigm* of a lexeme L is the set of realizations of L in cells corresponding to morphosyntactic property sets (MPSs).
> The *inflectional system* of a language is the set of inflectional paradigms of the lexemes of the language.

For example, in English, the lexeme WALK has forms {*walk, walks, walking*, *walked*}, corresponding to the MPSs (among others) {{1 SG PRES}, {3 SG PRES}, {PRES PART}, {PAST PART}}.

One can chart the inflectional system of collections of lexemes, such as the chart for some English verbs, as shown in in Table 1. We notice that at least for some verbs (they are called *weak verbs* in Germanic languages like English), we can infer the PAST surface form from the 1 SG PRES surface form. We can always infer the PRES PART surface form as well (perhaps with small adjustments for spelling).

Table 1 Fragment of the Inflectional System of English Verbs

Lexeme	1 SG PRES	3 SG PRES	PRES PART	PAST	PAST PART
WALK	*walk*	*walks*	*walking*	*walked*	*walked*
HOPE	*hope*	*hopes*	*hoping*	*hoped*	*hoped*
SEE	*see*	*sees*	*seeing*	*saw*	*seen*
SPEAK	*speak*	*speaks*	*speaking*	*spoke*	*spoken*

2. Linguistic theories of inflectional morphology

The panoply of theoretical approaches to inflectional morphology is well summarized by Boyé and Schalchli (IN PRESS), who suggest these categories: syntactic frameworks, syntagmatic frameworks, and paradigmatic frameworks.

Syntactic frameworks generally are not interested in paradigms. Among the syntactic frameworks, the best-known approach is *Item and Arrangement*, in which information in a surface form is composed of the meaning of the stem morpheme and the meanings of the affixes attached to it. Researchers often use finite-state transducers to model morphological theories that follow the Item and Arrangement approach. Since 1983, the principal modeling technique has been Koskenniemi's two-level rules, which are able both to parse and to generate morphological forms Koskenniemi (1983). Many compilers for two-level rules are available, such as MMORPH (Petitpierre & Russel, 1995) and Finite-State Transducer Technology (HFST) (HFST, 2008). Researchers have used such tools to create morphological analyzers for many languages, including English, Spanish, and Dutch. Such analyzers are in heavy use in natural-language processing (NLP) tools.

Finite-state techniques are elegant in the sense that their generative capacity is limited to regular sets, as opposed to more complex sets such as those generated by context-free grammars or Turing machines. After all, the set of morphological forms of a finite set of lexemes is finite, so it should not be necessary to employ techniques that reach beyond regular sets. However, models that use techniques with larger generative capacity can be elegant in their own way, as we will see.

Syntagmatic frameworks include Paradigm Function Morphology and Network Morphology, which are the principal approaches we deal with in this article.

Paradigmatic frameworks try to find implications from cell to cell in paradigms. The Principal Part Analysis program we describe in this araticle belongs in this category of approaches.

Stump presents a different typology of linguistic theories for inflectional morphology (2001). Based on his typology, the *realizational* approach (Anderson, 1992; Corbett & Fraser, 1993; Matthews, 1972; Stump & Finkel, 2013; Zwicky, 1985) applies morphological rules to deduce the realization in each cell of the inflectional system. The tools we describe here for Network Morphology and Paradigm Function Morphology are intended for theorists following this approach to express their theories for specific languages and to verify that those theories produce the expected inflectional system.

The *implicative* approach to analyzing an inflectional system (Blevins, 2005, 2006; Finkel & Stump, 2009A, 2009B) infers some the realizations in some cells of the inflectional system from the realizations of other cells. We have already hinted at such inference in describing the {PAST} form of weak verbs in English. The tool we describe here for Principal Part Analysis provides various implicative analyses of inflectional systems.

Other than the software available for finite-state transducers for the Item and Arrangement approach, the only computational tools for generative morphology that the author is aware of are those presented in this article.

3. Network Morphology

Network Morphology, first presented by Corbett and Fraser (1993), is described by Boyé and Schalchli (IN PRESS):

> An NM analysis starts from a content paradigm [an MPS] and a lexicon and states generalizations about the relations between the content paradigm and the form paradigm [the surface forms] for lexical entries through a network of hierarchies.

In the simplest arrangement, the hierarchy is comprised of nodes arranged in a tree. The leaves of the tree represent lexemes. Lexemes that have similar morphological properties, usually because they belong to the same inflection class, share the same parent. The path from the root of the tree to a lexeme passes through multiple nodes. Each node on that path, from the root to the leaf, can introduce or replace information about the lexeme.

Turning again to English verbs, we can construct a tree with nodes as shown in Figure 1.

Here, information about HOPE begins at the root, VERB, which provides the morphology for {1 SG PRES}, {3 SG PRES}, and {PRES PART}, all based on *stem*. VERB has three children, WEAK, See, and Speak. HOPE is a child of WEAK, which provides the morphology for {PAST} and {PAST PART}, also based on *stem*. Finally, the leaf Hope

Click to view larger

Figure 1. A tree of nodes for some English verbs. Children are shown within their parent.

provides the *stem* information. Another way to look at this picture is to start at a leaf such as Hope with a query such as {3 SG PRES} and to travel up the tree until that question is answered.

This example motivates the DATR formalism, as well as its successor, KATR.

3.1. DATR

DATR is a formal language for realizing the cells of an inflectional system following the Network Morphology approach. DATR was designed and implemented by Roger Evans and Gerald Gazdar (1996). DATR has the generative capacity of Turing machines. With this (perhaps excessive) capacity comes an ability to present remarkably clear morphological theories.

In DATR, the theorist describes the morphology of a language as a collection of *nodes* containing *rules*. Some of those nodes, primarily leaf nodes, represent individual lexemes of the language. Each rule is composed of a *guard* and a *result*. (Usually people just refer to the "left-hand side" and the "right-hand side" of the rule.) Computation begins by addressing a *query* to a lexeme node; the query typically represents an MPS, that is, a content form. The guards of all the rules in the node are checked; we say that a guard that matches the query (as described below) is *open*; otherwise, it is *closed*. Of all the open guards, DATR selects the rule with the most restrictive guard. It then returns the value of the result of that rule. That result may be a surface form, but often it specifies further computation in the form of new queries to be directed to nodes.

To provide a specific example, Figure 2 displays some nodes in a theory of Lingala (Niger-Congo, Congo River). The rule numbers are not part of the theory; we only include them for easy reference.

The node Hit is a leaf node representing the lexeme ***bet*** "hit." A query to that node might be subj 1 sg obj 2 pl historical, representing the MPS associated with "I once hit you (all)." Computation matches the query against the guards on the two Rules 24 and 25. The guard <stem> requires that the query begin with the word stem, which it doesn't, so this guard is closed. The guard <> imposes no requirement on the query, so it is open. If multiple guards are open, computation chooses the most restrictive, that is, the longest one, in accordance with Pāṇini's principle.

In this case, Rule 25 reflects the query to the VERB node. All lexeme nodes for verbs in the theory have the same pattern as Hit: they reflect queries other than <stem> to VERB. That is, the nodes are organized into a tree, with verbal lexemes at the leaves and VERB as their common parent.

```
AGR: % generic personal prefix
1    <2 pl> = b o
2    <gen12 pl> = b a
3    <gen34 sg> = m o
4    <gen34 pl> = m i
5    <gen56 sg> = l i
6    <gen56 pl> = m a
7    <gen78 sg> = e
8    <gen78 pl> = b i
9    <gen910 sg> = e
10   <gen910 pl> = i
11   <gen11 sg> = l o
12   <gen14 sg> = b o

SUBJAGR: % subject personal prefix
13   <1 sg> = n a
14   <1 pl> = t o
15   <2 sg> = o
16   <gen12 sg> = a
17   <> = AGR

OBJAGR: % object personal prefix
18   <1 sg> = n
19   <1 pl> = l o
20   <2 sg> = k o
21   <gen12 sg> = m o
22   <> = AGR

VERB:
23   % details omitted; invokes both SUBJAGR and OBJAGR

Hit:
24   <stem> = bat
25   <> = VERB
```

Click to view larger

Figure 2. DATR rules from a Lingala theory.

The VERB node has a rule too complex to show here, but it reflects a subquery 1 sg to SUBJ_AGR and a subquery 2 pl to OBJ_AGR. SUBJ_AGR has only one rule with an open guard: Rule 13, which returns the surface form *n a*. OBJ_AGR also has only one matching rule, number 22, which reflects the query to the more general AGR node. Rule 1 provides the surface form *b o*. Rule 23 combines these results, along with tense information (not shown) to derive the surface form *nabobetakí*.

3.2. KATR

Raphael Finkel and Gregory Stump created KATR, an extension of DATR, to accommodate more analyses (Finkel, Shen, Stump, & Thesayi, 2002). The extensions let the theorist represent morphosyntactic properties as unordered sets, easily express nonlocal sandhi phenomena, and override Pāṇini's principle.

KATR descriptions of a language's morphology are called *theories*. Theories are presented in Unicode.

3.2.1. Node names and atoms

Node names start with a capital letter. Surface-form fragments (we call them *atoms*) start with a lower-case letter. The theorist can declare specific atoms to start with an upper-case letter.

3.2.2. Variables

Often a theory must produce or match queries that have generic components, such as tense, polarity, person, number, voice, and mood. To that end, one can declare variables in directives like these:

> #vars $tense: past continual present future imaginary.
> #vars $polarity: negative positive conjectural.
> #vars $person: 1 2 3 4.
> #vars $number: sg du pl.

A guard such as <subj $person $number> matches the query subj 1 du. The result of a rule with that guard may compute subqueries that mention $person and $number, which become 1 and du for the current computation.

3.2.3. Paths and sets

The guard of a rule may be either a *path*, surrounded by < and >, or a *set* (KATR only), surrounded by "{" and "}." Between the delimiters, a guard lists components, each of which is an atom or variable. A path guard matches a query if each of the guard's components matches the respective element in the query in order. A set guard matches a query if each of the guard's components matches some element in the query, in any order. A single element of the query may only be matched by one component of the guard. In both cases, excess elements of the query are ignored for the purpose of matching.

3.2.4. Generating queries

A KATR theory specifies what queries to generate by one or more #show directives. For Lingala, as an example, these directives might be:

```
#show <subj 1 sg obj 1 sg habitual>.
#show <subj 1 sg obj 1 pl habitual>.
#show <subj 1 sg obj 2 sg habitual>.
...
```

The theory may combine such directives (KATR only):

```
#show <subj:: $number:: $person:: obj::
   $number:: $person:: $tense>.
```

KATR generates all possible substitutions of the variables, with the last one (here $tense) varying the most quickly, and the first one (here $number) varying the least quickly.

KATR then directs each query to each node. Usually, one only wants to direct queries to leaf nodes, which represent lexemes, and let the queries percolate, if necessary, on the path to the root. KATR provides directives to restrict the set of nodes that accept queries.

3.2.5. Guard precedence

When a query reaches a node, KATR evaluates the guards on all its rules. Pāṇini's principle then requires that KATR choose the most specific open rule. It is an error if this choice is ambiguous. If no guard is open, the query has no result. To find the most specific open rule, KATR computes the *precedence* of the open guards by counting the number of elements in those guards; it chooses the rule with the highest precedence. Atoms have slightly higher precedence than variables. In KATR (only), it is also possible to artificially increment the precedence of a guard by an integer amount or an infinite amount.

3.2.6. The result of a rule

Once KATR has chosen a rule, it evaluates the rule's result (its right-hand side), which is a list of zero or more items. The value of the result is the concatenation of the values of all those items.

Items can be of several varieties.

- **!** (KATR only) This rule produces no result. KATR abandons this query.
- ***atom*** The value is its string of characters.

• ***<path>*** The value is the result of presenting an *enhanced query* to the same node. The enhanced query is the original query, minus all atoms that match the rule's guard, plus (at the end) all atoms in the given path. If the guard and the rule are separated by =+= instead of = (KATR only), the enhanced query does not lack the atoms that match the rule's guard. The path may include variables, which must reference identical variables in the guard and which evaluate to the query atom matched by the guard. A guard may use the same variable several times; the path can refer to each occurrence separately.

• ***Node*** The value is the result of presenting the query to the given node.

• ***Node***: ***<path>*** The value is the result of presenting an enhanced query (as described above) to the given node.

• "***<path>***" The value is the result of presenting the enhanced query to the leaf node to which the original query was presented.

3.2.7. Sandhi

KATR (only) represents rules of sandhi with a set of directives that it applies to the final result of each query. A simple example that reduces all doubled *o* vowels to a single one looks like this:

```
#sandhi o o => o.
```

Often rules of sandhi are more generic. To remove *o* after any vowel, one can introduce a class of surface forms with a variable declaration and refer to it in the sandhi rule:

```
#vars $vowel: a e i o u.
#sandhi $vowel o => $1.
```

In this rule, $1 refers to the first variable on the left-hand side. Slightly more complex is a sandhi rule from Latin that reduces *ū* to *u* before two stops:

```
#vars $stop: t d n m p b k g.
#sandhi ū$stop $stop => u $1 $2.
```

KATR implements sandhi rules by applying a finite-state transducer as a postprocessing step to the final result of each query.

3.3. Standard practices

Writing DATR and KATR theories is an art. Theorists must often choose among multiple ways to achieve an effect. Standard practices can often guide the theorist.

Lexemes are represented by leaves of the network. The rules in lexeme nodes typically have guards for nonmorphological information (such as part of speech and gloss), stems, and exceptional forms, and they contain one catch-all rule to reflects queries to the parent node. An example for Latin verbs:

```
Lead:
1 <gloss> = lead % non-morphological
2 <root> = d ū c % stem
3 {sg imperative} = <root> % exceptional situation
```

4 <> = VerbE % reflection to parent

The order of the rules is immaterial. Rule 1 responds to a nonmorphological query. Rule 2 handles stem requests. Latin verbs can have several stems, but a single stem often implies the others. This information is needed by nodes further up the tree; rules that provide such information are said to perform *priming* (Finkel & Stump, 2007A). This particular verb has an exceptional singular imperative form; one would expect **dūce*, but the actual form is *dūc*, as specified in Rule 3. Rule 4 is a *default rule*; it applies when no other rule applies, and it reflects queries to the parent node, which deals with all verbs in the third (E) conjugation:

VerbE:
1 <themeVowel> = I
2 {themeVowel indicative} = ē
3 <stemImperfective> = "<root>" <themeVowel>
4 {stemImperfective present} = "<root>"
5 <stemPerfective> = "<root>" s
6 <stemParticiple> = "<root>" t
7 <> = Verb:<conj3>

VerbE introduces the concept of a *theme vowel*, which its parent, Verb, uses to construct a surface form. Theme vowels distinguish conjugations in Latin. For the third conjugation, the theme vowel is sometimes *ē*, in situations covered by Rule 2, sometimes a "weak *i*." We use a morphophoneme *I* to represent the latter case. By convention, morphophonemes are single capital letters declared as atoms. The realization of *I* is governed later by sandhi rules:

#sandhi $unroundedVowel I => $1.
#sandhi I r => e r.
#sandhi I => i.

Rule 2 specializes the default theme vowel specified in Rule 1. In general, *specialization* refers to competing rules within the same node where Pāṇini's principle causes KATR to choose the most restrictive one (Finkel & Stump, 2007A). The VerbE node produces all the stems based on the root in Rules 3–6. In each case, "<root>" in the result causes the query to be reflected to the original leaf node, for instance, Lead. VerbE reflects all other queries to the Verb node, but it adds conj3 to the query so that Verb can apply this knowledge if necessary.

We next show the Verb node:

Verb:
1 <> = StemAspect Tense1 Tense2 PersonVoice
2 {perfective passive $conj} = % details omitted
3 {imperative sg} = "<stemImperfective imperative>"
4 {imperative pl} = "<stemImperfective imperative>" t e
5 {active infinitive} = "<stemImperfective infinitive>" r e
6 {passive infinitive} =+= "<stemImperfective>" r ī
7 {passive infinitive conj3} =+= "<stemImperfective>" ī
8 {future subjunctive} = !

Rule 1 is a default rule that combines the results of applying the query to four other nodes. This rule demonstrates two strategies: *combining* (putting together morphological segments) and *lookup* (consulting other nodes for details) (Finkel & Stump, 2007A). Rules 2–7 provide forms for the imperative and infinitive. Rule 8 precludes any result from a query for future subjunctive; Latin has no such form.

Here is a small subset of the information in the four nodes that Verb consults:

StemAspect:
1 {imperfective} =+= "<stemImperfective>"
2 {perfective} =+= "<stemPerfective>"
Tense1:
3 {conj1 present imperfective subjunctive} = ē
4 {present imperfective subjunctive} = ā
Tense2:
5 {past indicative} = ā
6 {present perfective subjunctive} = ī
PersonVoice:
7 {2 pl passive} = I m i n ī

Rules 1 and 2 refer back to values primed by lower nodes in the tree. The guard of Rule 3 checks for conj1, which is introduced by the default rule of the VerbA node.

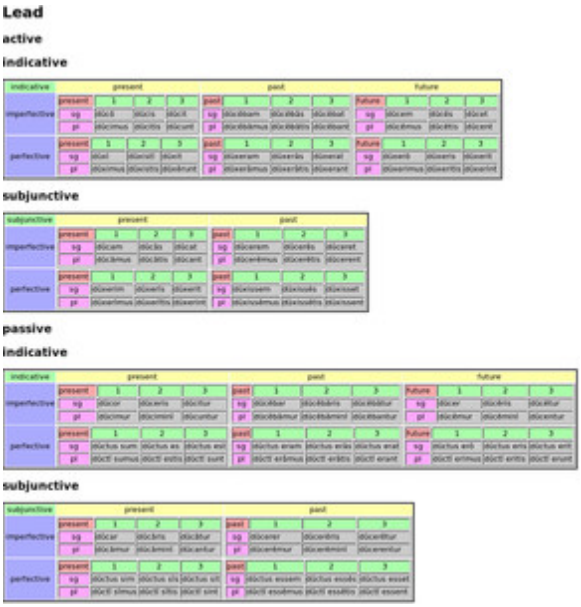
3.4. Outputs

KATR computes surface forms for all queries applied to all appropriate nodes. It omits spaces between fragments and converts commas to spaces to create such surface forms as *futurus sum*.

The output form is a list of nodes, queries, and surface forms:

Lead active,indicative,imperfective,present,sg,1 dūcō
Lead active,indicative,imperfective,present,sg,2 dūcis
Lead active,indicative,imperfective,present,sg,3 dūcit
Lead active,indicative,imperfective,present,pl,1 dūcimus
Lead active,indicative,imperfective,present,pl,2 dūcitis
Lead active,indicative,imperfective,present,pl,3 dūcunt
Lead active,indicative,imperfective,past,sg,1 dūcēbam
...

KATR can also generate a chart of outputs, such as the Latin verb chart for LEAD shown in Figure 3.



Click to view larger

Figure 3. KATR chart output for Latin verbs.

4. Paradigm Function Morphology

Gregory Stump introduced Paradigm Function Morphology (PFM) as a way to express the morphology of natural languages. (Stump, 2001, 2015). PFM is loosely based on the earlier *A-Morphous Morphology* (Anderson, 1992). *PFME* is a Web-based engine that generates word forms from language theories expressed in PFM.

4.1. Overview

PFM computes a lexeme's paradigm by evaluating a *paradigm function* on queries. Each query is a pair: an MPS and stem of the lexeme. The collection of MPSs is generated by the *content-paradigm schema*. The paradigm function is a series of transforming steps applied to the lexeme's stem. Each step is represented by a *block* of *word-realization rules*. Each rule is composed of a *guard* and a *result*. The guard can match both MPS-based and lexeme-based information. PFM chooses the open rule with the most restrictive guard and returns the value of the result.

As an example from a PFM theory for Turkish nouns, the content-paradigm schema (format discussed later) is:

```
Content-paradigm schema(N) = {
  CASE:{nom/acc/dat/abl/loc/gen}
  NUM:{sg/pl}
  POSS:{1/2/3 sg/pl}
}
```

One of the MPSs that this schema specifies is CASE:{acc} NUM:{pl} POSS:{1 pl}. The paradigm function has four blocks. A query formed from that MPS and the stem *adam* MAN requests the surface form for OUR MEN in the accusative case. The first block affixes *-l2r* (*2* is a morphophoneme referring to twofold vowel harmony) to account for NUM:{pl}. The second block affixes *-4m* to account for POSS:{1} (*4* is a morphophoneme referring to four-fold vowel harmony). The third block affixes *-4z* to account for POSS:{pl}. The fourth block affixes *-4ı* to account for CASE:{acc}. The result is *adaml2r4m4z4ı*, which sandhi rules convert to *adamlarımızı*.

A complete PFME theory specifies the content-paradigm schema, the lexemes and their stems, the paradigm function (itself composed of the word-realization rules arranged in blocks), sandhi transformations, and a list of known results against which PFME can verify its outputs.

4.2. Notation

Many components of a PFM theory are expressed by a shorthand that we call an *expandable*. For example, sg/pl 1/2/3 is an expandable representing a list with the following contents:

> sg 1
> sg 2
> sg 3
> pl 1
> pl 2
> pl 3

An expandable is a list of elements. Each element is a solidus ("/")-separated list of alternatives. Each alternative is either a parenthesized expandable, an atom (such as masc or 3 in the example above), or a supertoken. Here are some sample supertokens:

> AGR(subject):{1/2/3 masc/fem}
> TENSE:{past/present/future}

Supertokens must have a name (conventionally in upper case), a colon, and braces surrounding an expandable; they may include a parenthesized subname.

A *bracketed expandable* is an expandable surrounded by curly or square brackets, as in {1/2 fem pl} and [noun aStem].

PFME uses curly-bracketed expandables to refer to MPSs that are supersets of any member of the expanded list. So {1/2 fem pl} matches any MPS that includes, for instance, {1 fem pl}. PFME uses square-bracketed expandables to refer to combinations of syntactic category and inflection class. So [noun aStem] matches any noun that is in inflection class aStem.

A *named expandable* is formed by a name, a colon, and a curly-bracketed expandable, such as S:{1 sg} or σ:{dat pl}. The name can be any word, but S and σ are the most common names.

A *context* contains information about a particular lexeme and a morphosyntactic property set (MPS). For instance, in English we might be interested in forming the lexeme EAT in the third-person singular present. PFM theories represent a context by a pair in pointy brackets, such as <L, σ:{1 pl}>, where L represents the lexeme and σ:{1 pl}> represents the MPS. The MPS may be a name, a curly-bracketed expandable, or a named expandable.

PFME often needs to match components of contexts against patterns. A *component* could be a lexeme's syntactic category and inflection class, or it could be the MPS. A *pattern* is an expandable. We say the pattern and the component *match* if at least one of the alternatives generated by the pattern has elements all of which appear in the component. The *strength* of the match is the number of such elements that appear. When PFME must choose among

alternative patterns that match some context, it uses Pāṇini's principle, selecting the most restrictive alternative: that with the highest-strength match.

4.3. PFM sections

A PFM theory should specify the language it represents by a line like this:

> Language: *Name*

The language name contains everything on the line following Language: .

It should also specify the author of the theory:

> Author: *Name*

The author name contains everything on the line following Author: .

Each lexeme must be described by a *lexical entry* like the following:

> Lexeme: EAT
> Meaning: eat
> Syntactic category: V
> Inflection class: strong n

The lexeme name should be in upper case, and the syntactic category should be V, N, and A for verbs, nouns, and adjectives. The Lexeme and Syntactic category must be a single word. The Meaning may be several words. The Inflection class may be multiple words.

Each lexeme must have one or more associated *roots*. Roots are defined by syntax like the following.

> Root(<EAT, σ:{past}>) = ate
> Root(<EAT, {perfect/futPerf}>) = eaten
> Root(<EAT, σ:{}>) = eat
> Root(<CLIMB, σ>) = climb
> Root(PERFORM) = perform

This example demonstrates a variety of acceptable formats. The first two lines show the most general format, where the left-hand side is in full context notation. The other examples use various acceptable shorthands. Root formats obey these rules:

- The MPS, if present, may be either a curly-bracketed expandable or a named expandable. The name is ignored.
- The lexeme component on the right-hand side must be a single word.

PFME selects the root whose MPS pattern has the strongest match to the MPS in the context.

If all stems are simply roots, one may omit any direct mention of stems. In some languages, however, stems are formed from roots by morphophonological operations. An example comes from Hua (dialect of Yagaria, Trans-New Guinea):

```
Stem(L:front) = front(Root(L))
Stem(L:back) = back(Root(L))
Stem(L:diag) = low(Root(L))
Morphophonological operations = {
  front(Pu) = Pi
  front(Po) = Pe
  low(Po) = Pa
  low(Pu) = Pa
  back(Pi) = Pu
  back(Pe) = Po
}
```

In this example, front, back, and diag are *lexeme modifiers*. They are also names of morphophonological operations. Their definitions use P to represent arbitrary phonemes. The two rules for front say that a root ending with *u* should have that ending changed to *i*, whereas a root ending with *o* should have that ending changed to *e*. A root that satisfies neither of these situations remains unchanged.

A PFM theory must have at least one *content paradigm* schema; it may have several such schemata. A simple content-paradigm schema looks like this:

```
Content paradigm schema(V) = {
  present/past/perfect/future/futPerf sg/pl 1/2/3
}
```

The first line may have a nonempty parenthesis-bracketed expandable pattern (here, (V)) that matches syntactic categories and inflection classes. The rest of the schema is a curly-bracketed expandable that generates a list of MPSs.

A complex-paradigm schema may expand to several schemata. Here is an example taken from a theory of nouns and adjectives in Noon (Niger-Congo, Senegal):

```
Content paradigm schema(N <\d>A) = {
  CLASS:{$1}
  NUM:{sg/pl}
  DEF:{plus/minus}
  LOC:{1/2/3/noLoc}
  POSS:{(1 sg)/(1 pl incl/excl)/(2/3 sg/pl)/noPoss}
}
```

The presence of an expression bracketed by < and > in the pattern indicates expansion. The special characters \d represent any number 0. . .9. The later use of $1 in the right-hand side refers back to that bracketed expression. In this language, nouns have inflection classes 1A . . . 6A. This content-paradigm schema allows each noun to gain an MPS supertoken called CLASS containing a number in 1. . .6.

The MPS list that PFME produces from the content-paradigm schemata may include some unwanted combinations. For instance, in Noon nouns, possession requires definiteness, so we don't want to generate MPSs that contain a POSS other than noPoss if we have DEF:{minus}. We indicate unacceptable combinations by *disallow schemata*, which have the same form as content-paradigm schemata. For instance, we can have:

```
Disallow(N) = {
  (POSS:{sg/pl} DEF:{minus})/
  (LOC:{1/2/3} DEF:{minus})
}
```

This particular schema enforces the fact that possession and location both require definiteness.

PFME must check every generated MPS against the list of disallowed MPSs, so where possible, it is better to use restrictive expandables in the content-paradigm schema instead of listing disallowed entries.

For each lexeme L, PFME finds all content-paradigm schemata whose pattern matches L's syntactic category and inflection class. PFME generates all MPSs from those matching schemata, removing those that are disallowed. The resulting MPSs, along with lexeme stems, constitute the inputs to the paradigm function.

In most theories, the *form paradigm* is the same as the content paradigm. In these cases, there is no need to specify a correspondence. When the form paradigm differs from the content paradigm, we express their correspondence by a Corr function. Here is an example from Noon.

```
Corr(<L[like], σ>) = <Stem(L), objPos(σ)>
Corr(<L[balaa], σ>) = <Stem(L), objRel(σ)>
Corr(L) = <Stem(L), σ> % default rule; unnecessary
Property mapping objPos = {
  (INFL:{obj}) → (INFL:{poss})
}
Property mapping objRel = {
  (INFL:{obj}) → (INFL:{rel})
}
```

The left-hand side of each Corr rule specifies a pattern, including (optionally, in brackets) the syntactic category and inflection class and (optionally) an MPS, to be matched against the content paradigm. There are two acceptable right-hand side forms.

1. A context specifying both the stem and the form paradigm. If the left-hand side MPS is named, the same name must appear on the right-hand side; if the MPS is not named, it is taken to be σ. This form paradigm may indicate a modification of the MPS by naming a *property mapping*. Each property mapping must be defined with one or more rules. Each rule has a left-hand side pattern to match the MPS and a right-hand side replacement for that part of the MPS. The two sides are separated by →. The best property mapping is chosen based on Pāṇini precedence of matches with the MPS. If no property mapping applies, the mapping is the identity function.
2. A referral to another Corr rule, such as Corr(<L, pm(σ)>)), which passes the lexeme (and its syntactic category and inflection class) along with an MPS modified by a property mapping.

If a Corr rule is chosen, it computes the form paradigm. If not, the form paradigm is the same as the content paradigm.

A PFM theory specifies a single *paradigm function* that PFME should apply to all lexemes. Here is a sample paradigm function:

Paradigm function
PF(<X,σ>) = [person: [tense: [I: <X,σ>]]]]

By convention, <X,σ> refers to the context composed of the form paradigm (which may be a modification of the content paradigm) and the stem (which may be a modification of the root).

This particular function says that the way to generate a surface form from the context <X,σ> is to apply word-realization rules (described below), first choosing an appropriate rule from block I, then a rule from block tense, then a rule from block person. Any word may name a block, although it is conventional to name blocks either by Roman numerals (like I) or by names of morphosyntactic properties (like person).

A *block* of word-realization rules has the following form:

Block I
I, X,S:{3 sg present} → Xs
I, X[weak],S:{perfect/past/futPerf} → Xed

Every block implicitly contains the *default rule*:

blockName, X[],S:{} → X

Each rule starts with a guard: the block name, comma, the letter X (referring to the input to the rule, typically a partial surface form), an optional *classifier* (a square-bracketed expandable), a comma, and an MPS. The MPS may be either a curly-bracketed expandable or a named expandable; the name, if present, is immaterial. The contents of supertokens in the MPS may be abbreviated by a single Greek letter. For instance, the MPS component may look like this:

{transitive AGR(SUBJ):{τ} AGR(OBJ):{σ}}

The result of the rule, its right-hand side, is composed of arbitrary characters and may contain:

- The letter X, standing for the input to the rule.
- An embedded expression, such as [Negator:[Mood:<X,S>]], which refers to a subordinate paradigm function, in this case, invoking first the block Mood and then the block Negator. Embedded expressions must refer to X by a full context, such as <X,S>. The context may be modified from the input context, and it may refer to abbreviations from the left-hand side:

  [II:<X, τ>] [IV:<X, σ>]

In this example, the right-hand side invokes two blocks, each in a new context. Block II only uses the τ part of the MPS component from the left-hand side; block IV only uses its σ part.

- A reference to a stem based on an updated context:

  [Stem: <X,S:{prefixed}>]

In this example, the MPS of the context is enhanced by adding the element prefixed.

- A reference to a morphophonological operation, such as (!back(X)). The parentheses and exclamation point are required.

Parentheses, brackets, and the letter X are not allowed in the right-hand side except as described here.

In evaluating a block, PFME selects a single rule:

1. Select only rules with open guards.
2. Of those, retain only those rules with the strongest match between their classifiers with the syntactic category and inflection class of the lexeme.
3. Of those, retain the rule with the strongest match between its MPS component and the query. There must be exactly one such rule (possibly the implicit default rule) or the PFM theory is erroneous.

A PFM theory may include *rules of sandhi*, including shorthands for phonological classes. For example:

```
PhonologicalClass voiceless = f k p t
Sandhi {
  z → s/[voiceless]_
}
```

The theory may contain any number of PhonologicalClass specifications. The Sandhi section contains a braced set of rules. Each rule is of the form

original → *replacement*/*when*

The rule replaces the string indicated by *original* by the string indicated by *replacement* under the situation indicated by *when*. The *when* string has a single underscore (_) to represent the original string; to its left and right may be indicators specifying the environment surrounding the original string that enables the rule. These indicators are *enhanced strings*, which are strings that may contain phonological class shorthands, which are enclosed in square brackets. In the example above, the rule specifies that z converts to s if it is preceded by a voiceless letter: any of f, k, p, or t.

The *replacement* may be ∅ to indicate that PFME should simply delete the *original* in the given environment.

If the Sandhi section includes multiple rules, PFME applies them in the order shown; later rules can therefore further modify forms that earlier rules have produced. Whenever a rule applies, all the rules are applied again, starting from the first sandhi rule.

A PFM theory may include a *truth section* showing known forms for some lexemes and MPSs. For instance, we can say:

```
Truth = {
  PF(<EAT, σ:{1 sg present}>) = I eat
  PF(<EAT, σ:{3 sg present}>) = he eats
  PF(<EAT, σ:{3 sg past}>) = he ate
  PF(<EAT, σ:{3 sg perfect}>) = he has eaten
  PF(<EAT, σ:{1 pl perfect}>) = we have eaten
  PF(<EAT, σ:{3 sg futPerf}>) = he will have eaten
```

}

The MPS name should be σ or S. The MPS itself is a curly-bracketed expandable or a named expandable. PFME verifies all results against the list in the Truth section.

If the theory contains the line

ShowOnlyTruth

then instead of following the content-paradigm schema, PFME only applies the paradigm function to the lexemes and MPSs indicated by the Truth section.

A PFM theory may include a *randomization request*:

Random 10

to indicate that PFME is to randomly generate a limited number (in this case, 10) MPSs for each lexeme from the content-paradigm schema. This facility is especially useful if the number of possible MPSs is very large.

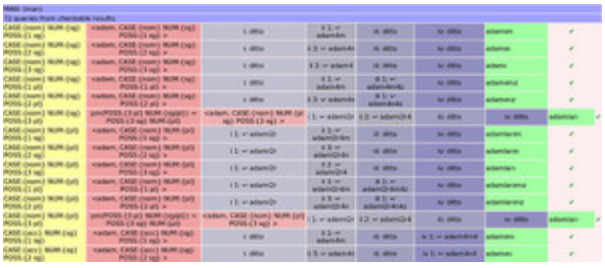
PFME ignores a randomization request if the theory also requests ShowOnlyTruth.

4.4. How PFME works

For each lexeme, PFME consults the content-paradigm and disallow schemata (or, if the theory specifies ShowOnlyTruth, the Truth set) to produce content-paradigm MPSs (either all possible MPSs or a random selection, if the theory specifies randomization), which it packages with the lexeme into *queries*. For each query, PFME then finds the appropriate root for the lexeme, from which it builds the stem. It applies the most applicable Corr rule to convert the content MPS to a form MPS. PFME then applies the paradigm function, which invokes stem-selection rules and blocks. For each block that it applies, PFME selects the single best rule. If the block is ambiguous, that is, there are several best rules, but they all agree on their right-hand side, the ambiguity is innocuous; otherwise, PFME arbitrarily picks a best rule but flags the error. After it has finished applying the paradigm function to the context, PFME applies all sandhi rules to the result.

The output of PFME is a Web page with one section per lexeme expressed as a table, with one row per query. The row's cells indicate the query, the chosen stem, and each block consulted. For each block, PFME either indicates ditto if it uses the default rule or the rule number it chooses and its result. PFME applies any sandhi rule to the result of the last block.

A sample of the output of the Turkish theory is shown in Figure 4.



Click to view larger

Figure 4. Partial PFME output for Turkish.

5. Principal part analysis

Gregory Stump and Raphael Finkel have used principal parts to perform implicative analyses of languages (Finkel & Stump, 2007B, 2009A, 2009B; Stump & Finkel, 2013). The Principal Part Analyzer (PPA) assists in these analyses.

A chart showing the full inflectional system of English verbs would need to be quite large to encompass all lexemes and all MPSs. Several simplifications can reduce the size of this representation:

1. Group lexemes into *inflection classes* that follow identical patterns. For example, weak verbs in English constitute an inflection class.
2. Represent the realizations in a phonetic, rather than a spelled, form. Phonetic representation shows that *walk* and *hope* follow identical rules, although spelling rules have changed **hopeing* to *hoping*.
3. Replace the realizations in the cells by a description of how to generate those realizations from one or more lexeme-specific stems. For example, realizations in the {PRES PART} column are formed by suffixing /ing/to the lexeme's stem.
4. Introduce rules of sandhi. For English, the ending for {3 SG PRES} is arguably /z/, which devoices to /s/ after an unvoiced consonant. Therefore, the {1 SG PRES} and the {3 SG PRES} follow the same pattern precisely: the latter is constructed from the former by the suffixation of /z/.

Input to PPA is a chart that may take advantage of all these simplifications. We call the input form a *plat*. Figure 5 shows a small plat for Finnish verbs, corresponding to the chart of the inflectional system in Table 2. There are many ways to build such a plat; this example shows only one way (Stump & Finkel, 2013).

```
% Finnish i-stem and e-stem nouns (Buchholz, 2004)

ABBR 1 1S1C
ABBR 2 1S1C2S

IC       nomSg genSg partSg partPl inessPl
TEMPLATE 1A    1An   2A     2A     2A
door     i     e     e      i      iss
language i     e     t      i      iss
water    i     de    t      i      iss
glass    i     i     i      ej     eiss
teddy    Ø     Ø     Ø      j      iss
letter   Ø     e     tt     it     iss

CLASS sibilant s z
SANDHI [:sibilant:] d => d   % vesde
SANDHI [:sibilant:] t => t t % vettä

LEXEME door     door     1:ov      2:a
LEXEME language language 1:kiel    2:ä
LEXEME water    water    1:ves     2:ä
LEXEME glass    glass    1:las     2:a
LEXEME teddy    teddy    1:nalle   2:a
LEXEME letter   letter   1:kirje   2:ä

KEYS door     SHORT FREQ=4
KEYS language LONG  FREQ=10
KEYS water    SHORT FREQ=423
KEYS glass    SHORT FREQ=1
KEYS teddy    LONG  FREQ=22
KEYS letter   LONG  FREQ=6

ANALYZE ANY=patteja inessPl=patteissa
```

Click to view larger

Figure 5. PPA plat for Finnish.

Table 2 Chart of Part of the Finnish Verb Inflectional System

Lexeme	nomSg	genSg	partSg	partPl	inessPl
Door	*ovi*	*oven*	*ovea*	*ovia*	*ovissa*
Language	*kieli*	*kielen*	*kieltä*	*kieliä*	*kielissä*
Water	*vesi*	*veden*	*vettä*	*vesiä*	*vesissä*
Glass	*lasi*	*lasin*	*lasia*	*laseja*	*laseissa*
Teddy	*nalle*	*nallen*	*nallea*	*nalleja*	*nalleissa*
Letter	*kirje*	*kirjeen*	*kirjettä*	*kirjeitä*	*kirjeissä*

5.1. Plat format

The file has several sections, which conventionally appear in the order described here.

The lines in Figure 5 starting with ABBR are *template abbreviations*, which are used later in TEMPLATE lines.

The main part of the plat is a table with columns representing MPSs and rows representing inflection classes (ICs). The table starts with a header row. Its first column is IC; subsequent columns name MPSs. Each name must be a single word without spaces; columns are separated by any number of spaces. After the header row and an optional TEMPLATE row, each row represents a single IC. Its IC representation can be any word. For our example, we name the IC by a gloss, such as door, which exemplifies a lexeme of that IC. If the number of MPSs is large, the table can be organized in multiple sections, separated by blank lines. Each section must have the same IC column (in the same order), but it has its own set of MPS columns.

Within the table are *cells*. Each cell describes the exponence for its IC and MPS. Exponences can be simple or complex.

- The plat in Figure 5 has a single string for each cell. If an exponence comprises discontinuous components, separate them by hyphens. The components are represented in the plat's template as 1C, 2C, and so forth (C means "component"). This feature is useful for languages like Hebrew with multiple components in their stems.
- An empty exponence is represented by the null character ∅ (Unicode \u2205). If an exponence has more than one component, you can omit empty components: -mo— is equivalent ∅-mo-∅-∅.
- The TEMPLATE line shows how to expand exponences into word forms. For instance, the genitive singular realization of ***vesi*** "water" is determined by the exponence de in accordance with the template 1An. Here, 1A refers to the abbreviation ABBR 1, which expands to 1S1C, so the template 1An is equivalent to 1S1Cn: the first stem *ves*, the first cell component *de*, and the suffix *n*, together forming *vesden*, which is then subject to sandhi refinement.

• If an exponence starts with !, it is an exceptional form not subject to template expansion. For instance, if the genitive singular of KIRJE were *kirjek*, not ending in *-n* and therefore violating the template for this MPS, we could put !kirjek in the cell. This exponence would apply to every noun in the IC KIRJE, so the ! notation is only helpful for ICs with a single member.

• If an exponence starts with with @, the first component is replaced by the string following @, and ordinary template expansion occurs. If the genitive singular of KIRJE were *korjeen*, using a variant stem, we could write its exponence as @korje. Again, this exponence would apply to every noun in the IC KIRJE.

PPA accepts optional sandhi specifications. These specifications may include character-class definitions, such as sibilant, and replacement rules, such as the ones shown in Figure 5. The right-hand side of a sandhi rule may refer to the class matches on the left hand side by $1, $2, and so forth.

PPA does not require a list of lexemes, but it can generate a paradigm for each listed lexeme, which is useful as an accuracy check. Each lexeme is listed with a gloss, IC, and stem(s). One may list zero or more lexemes for each IC in any order.

In some languages, lexemes have many stems, and ICs differ with respect to which stems are used for each MPS. The plat can indicate on a per-IC basis that some stems are syncretic with others. For instance, one analysis of French verbs has 15 stems, but most conjugations need only a few of them. A French verb plat might have:

REFER absoudre 2, 11, 15 -> 1; 10 -> 7; 4, 5, 8 -14 -> 3

This line tells us, for instance, that stem 2 for the IC typified by ***absoudre*** “exonerate” is the same as stem 1. One should not specify stem 2 for any lexeme in this IC.

The user may restrict any analysis to a subset of the MPSs and to a subset of the ICs (by referring to their keys), and entropy computations can take into account type frequency, as specified by keys, as demonstrated in Figure 5.

To identify word forms that can be attributed to multiple ICs, the plat may request that PPA analyze a given word form for IC membership. The first ANALYZE line in Figure 5 requests that PPA analyze the hypothetical pre-sandhi word form *patteja*. PPA responds that such a form might be in the glass IC with stem *patt*; it enumerates other possibilities as well. It then narrows down the possibilities based on an innessive plural form *patteissa*.

5.2. PPA output

The user may request that PPA compute various analyses by clicking on checkboxes on the PPA website. The technical terms involved, such as *distillation*, *static* (*adaptive*, *dynamic*) *principal part*, *predictability*, *predictiveness*, *MPS entropy*, and *IC signature*, are the subject of an entire book (Stump & Finkel, 2013), which defines these terms, shows how to compute them, and shows how languages vary with respect to them. Detailed discussions of information-theoretic measures such as conditional entropy and their usefulness for understanding morphological complexity are available in several places. (Ackerman, Blevins & Malouf, 2009; Milin, Kuperman, Kostić, & Baayen, 2009; Moscoso del Prado Martín, Kostić, & Baayen, 2004)

The user may also specify that PPA should generate a chart of surface forms. It does so by converting the plat into KATR and then using the KATR engine to generate the output.

Further Reading

A good introduction to various theories of inflectional morphology is in the first section of Stump (**2001**), distinguishing lexical from inferential theories and distinguishing incremental from realizational theories. Under this taxonomy, Word and Paradigm is inferential and realizational; Item and Arrangement is lexical and incremental.

The handbook chapter by Boyé and Schalchli (**in press**) discusses the principal generative-morphology theories. The handbook chapter by Hippisley (**2010**) is a good introduction to both the Word and Paradigm and the Item and Arrangement approaches of formal morphology. The survey by Sproat (**1992**) is a classic reference for computational methods for Item and Arrangement.

An early reference for the Word and Paradigm approach is Zwicky (**1985**). It heavily influenced the development of Paradigm Function Morphology, for which Stump (**2001**) is a comprehensive reference. Stump (**2015**) updates this earlier work and motivates many of the facilities we describe for PFME.

The theory and practice of DATR are well described by Brown and Hippisley (**2012**).

Analysis of morphological paradigms based on the metaphor of principal parts is the subject of the monograph by Stump and Finkel (**2013**).

References

Ackerman, F., Blevins, J. P., & Malouf, R. (2009). Parts and wholes: Patterns of relatedness in complex morphological systems and why they matter. In J. P. Blevins & J. Blevins (Eds.), *Analogy in grammar: Form and acquisition* (pp. 54–82). Oxford: Oxford University Press.

Find this resource:

Anderson, S. R. (1992). *A-morphous morphology*. Cambridge, U.K.: Cambridge University Press.

Find this resource:

Blevins, J. P. (2005). Word-based declensions in Estonian. In G. Booij & J. van Marle (Eds.), *Yearbook of morphology 2005* (pp. 1–25). Dordrecht, Netherlands: Springer.

Find this resource:

Blevins, J. P. (2006). Word-based morphology. *Journal of Linguistics*, *42*, 531–573.

Find this resource:

Boyé, G., & Schalchli, G. (in press). The status of paradigms. In A. Hippisley & G. T. Stump (Eds.), *The Cambridge Handbook of Morphology*. Cambridge, U.K.: Cambridge University Press.

Find this resource:

Brown, D., & Hippisley, A. (2012). *Network morphology: A defaults-based theory of word structure* (Vol. 133). Cambridge, U.K.: Cambridge University Press.

Find this resource:

Buchholz, E. (2004). *Grammatik der finnischen Sprache*. Bremen, Germany: Hempen Verlag.

Find this resource:

Corbett, G. G., & Fraser, N. M. (1993). Network morphology: A DATR account of Russian nominal inflection. *Journal of Linguistics*, *29*, 113–142.

Find this resource:

Evans, R., & Gazdar, G. (1996). DATR: A language for lexical knowledge representation. *Computational Linguistics*, *22*, 167–216.

Find this resource:

Finkel, R., Shen, L., Stump, G., & Thesayi, S. (2002). *KATR: A set-based extension of DATR* (Tech. Rep. No. 346-02). Lexington: University of Kentucky Department of Computer Science. Retrieved from ftp://ftp.cs.uky.edu/cs/techreports/346-02.pdf

Find this resource:

Finkel, R. & Stump, G. (2007a). A default inheritance hierarchy for computing Hebrew verb morphology. *Literary and Linguistic Computing*, *22*(2), 117–136.

Find this resource:

Finkel, R., & Stump, G. (2007b). Principal parts and morphological typology. *Morphology*, *17*(1), 39–75.

Find this resource:

Finkel, R., & Stump, G. (2009a). Principal parts and degrees of paradigmatic transparency. In J. P. Blevins & J. Blevins (Eds.), *Analogy in grammar: Form and acquisition* (pp. 15–53). Oxford: Oxford University Press.

Find this resource:

Finkel, R., & Stump, G. (2009b). What your teacher told you is true: Latin verbs have four principal parts. *Digital Humanities Quarterly*, *3*(1).

Find this resource:

Helsinki Finite-State Transducer Technology (HFST). (2008). Retrieved from **www.ling.helsinki.fi/kieliteknologia/tutkimus/hfst/**

Find this resource:

Hippisley, A. (2010). Lexical analysis. In N. Indurkhya & F. J. Damerau (Eds.), *Handbook of natural language processing* (2nd ed., pp. 31–58). Boca Raton, FL: CRC Press.

Find this resource:

Koskenniemi, K. (1983). *Two-level morphology: A general computational model for word-form recognition and production* (Tech. Rep. No. 11). Helsinki, Finland: University of Helsinki, Department of General Linguistics.

Find this resource:

Matthews, P. H. (1972). *Inflectional Morphology*. Cambridge, U.K.: Cambridge University Press.

Find this resource:

Milin, P., Kuperman, V., Kostić, A., & Baayen, R. H. (2009). Paradigms bit by bit: An information theoretic approach to the processing of paradigmatic structure in inflection and derivation. In J. P. Blevins & J. Blevins (Eds.), *Analogy in grammar: Form and acquisition* (pp. 214–252). Oxford: Oxford University Press.

Find this resource:

Moscoso del Prado Martín, F., Kostić, A., & Baayen, R. H. (2004). Putting the bits together: An information theoretical perspective on morphological processing. *Cognition*, *94*, 1–18.

Find this resource:

Petitpierre, D., & Russel, G. (1995). *MMORPH—the Multitext Morphology Program, Version 2.3* (Tech. Rep.). MULTEXT deliverable report for task 2.3.1, Switzerland: University of Genva.

Find this resource:

Sproat, R. (1992). *Morphology and computation*. Cambridge, MA: MIT Press.

Find this resource:

Stump, G. (2001). *Inflectional morphology—a theory of paradigm structure* (Vol. 93). Cambridge, U.K.: Cambridge University Press.

Find this resource:

Stump, G. (2015). *Inflectional paradigms: Content and form at the syntax-morphology interface*. Cambridge, U.K.: Cambridge University Press.

Find this resource:

Stump, G., & Finkel, R. A. (2013). *Morphological typology: From word to paradigm* (Vol. 138). Cambridge, U.K.: Cambridge University Press.

Find this resource:

Zwicky, A. M. (1985). How to describe inflection. In *Proceedings of the 11th Annual Meeting of the Berkeley Linguistics Society* (pp. 372–386).

Find this resource:

Raphael Finkel

University of Kentucky