Generating Hebrew verb morphology by default inheritance hierarchies

Raphael Finkel
Department of Computer Science
University of Kentucky
raphael@cs.uky.edu

Gregory Stump
Department of English
University of Kentucky
gstump@uky.edu

Abstract

We apply default inheritance hierarchies to generating the morphology of Hebrew verbs. Instead of lexically listing each of a word form's various parts, this strategy represents inflectional exponents as markings associated with the application of rules by which complex word forms are deduced from simpler roots or stems. The high degree of similarity among verbs of different *binyanim* allows us to formulate general rules; these general rules are, however, sometimes overridden by *binyan*-specific rules. Similarly, a verb's form within a particular *binyan* is determined both by default rules and by overriding rules specific to individual verbs. Our result is a concise set of rules defining the morphology of all strong verbs in all *binyanim*. We express these rules in KATR, both a formalism for default inheritance hierarchies and associated software for computing the forms specified by those rules. As we describe the rules, we point out general strategies for expressing morphology in KATR and we discuss KATR's advantages over ordinary DATR for the representation of morphological systems.

1 Introduction

Recent research into the nature of morphology suggests that the best definitions of a natural language's inflectional system are inferential and realizational (Stump, 2001). A definition is **inferential** if it represents inflectional exponents as markings associated with the application of rules by which complex word forms are deduced from simpler roots and stems; an inferential definition of this sort contrasts with a **lexical** definition, according to which an inflectional exponent's association with a particular set of morphosyntactic properties is simply stated in the lexicon, in exactly the way that the association between a lexeme's formal and contentive properties is stipulated. In addition, a definition of a language's inflectional system is **realizational** if it deduces a word's inflectional exponents from its grammatical properties; a realizational definition contrasts with an **incremental** definition, according to which words acquire morphosyntactic properties only by acquiring the morphology which expresses those properties.

The conclusion that inflectional systems should be defined realizationally rather than incrementally is favored by a range of evidence, such as the widespread incidence of extended exponence in inflectional morphology and the fact that a word's inflectional exponents often under-determine its morphosyntactic content (Stump, 2001). Moreover, inferential-realizational definitions can avoid certain theoretically unmotivated distinctions upon which lexical or incremental definitions often depend. For instance, inferential-realizational definitions do not entail that concatenative and nonconcatenative morphology are fundamentally different in their grammatical status; they do not necessitate the postulation of any relation between inflectional markings

and morphosyntactic properties other than the relation of simple exponence; and they are compatible with the assumption that a word form's morphological representation is not distinct from its phonological representation.

Various means of defining a language's inflectional morphology in inferential-realizational terms are imaginable. In an important series of articles (Corbett and Fraser, 1993; Fraser and Corbett, 1995; Fraser and Corbett, 1997), Greville Corbett and Norman Fraser proposed Network Morphology, an inferential-realizational morphological framework that makes extensive use of nonmonotonic inheritance hierarchies to represent the information constituting a language's inflectional system. Analyses in Network Morphology are implemented in DATR, a formal language for representing lexical knowledge designed and implemented by Roger Evans and Gerald Gazdar (Evans and Gazdar, 1989). In recent work, we have extended DATR, creating KATR, which is both a formal language and a computer program that generates desired forms by interpreting that language.

In this paper, we show how KATR can be used to provide an inferential-realizational definition of Hebrew verb morphology. Our objectives are twofold. First, we propose some general strategies for exploiting the capabilities of nonmonotonic inheritance hierarchies in accounting for the properties of "root-and-pattern" verb inflection in Hebrew; second, we discuss some specific capabilities that distinguish KATR from DATR and show why these added capabilities are helpful to account for the Hebrew facts.

2 The *pi'el* verb דבר

The purpose of the KATR theory described here is to generate perfect and imperfect forms of strong verbs belonging to various *binyanim* in Hebrew. In particular, given a verbal lexeme L and a sequence σ of morphosyntactic properties appropriate for verbs, the theory evaluates the pairing of L with σ as an inflected verb form. For instance, it evaluates the pairing of the lexeme דבר "speak" with the property sequence `<perfect 3 sg masc>` as the verb form דִּבֶּר "he spoke".

A theory in KATR is a network of **nodes**; the network of nodes constituting our verb morphology theory is represented in Figure 1. The overarching organizational principle in this network is hierarchical: The tree structure's terminal nodes represent individual verbal lexemes, and each of the nonterminal nodes in the tree defines default properties shared by the lexemes that it dominates. The status of the boxed nodes is taken up below.

Each of the nodes in a theory houses a set of **rules**. We represent the verb דבר by a node:

```
Speak:
    <root> = ר ב ד % 1
    <> = PIEL % 2
    .
```

The node is named `Speak`, and it has two rules, terminated by a single dot. Our convention is to name the node for a verb by a capitalized English word representing its meaning. We use KATR-style comments (starting with `%` and continuing to the end of the line) to number the rules so we can refer to them easily.

Rule 1 says that a query asking for the root of this verb should produce a three-atom result containing ד, ב, and ר. Our rules assemble Hebrew words in logical order, which appears in this document as left-to-right. We accomplish reversal by rules in a `REVERSE` node, not shown in this paper.

Rule 2 says that all other queries are to be referred to the `PIEL` node, which we introduce below.

A query is a list of atoms, such as `<root>` or `<vowel2 perfect 3 sg masc>`; in our theory, the atoms generally represent form categories (such as `root`, `binyanprefix`, `vowel1`, `cons2`), morphosyntactic properties (such as `perfect`, `sg`, `fem`) or specific Hebrew characters. Queries are directed to a particular node. The query directed to a given node is matched against all the rules housed at that node. A rule **matches** if all the atoms on its left-hand side match the atoms in the query. A rule can match even if its atoms do not exhaust the entire query. In the case of `Speak`, a query `<root perfect>` would match both rules, but not a rule begining with `<spelling>`. When several rules match, KATR picks the best match, that is, the one whose left-hand side "uses up" the most of the query. This algorithm means that Rule 2 of `Speak` is only used when Rule 1 does not apply,

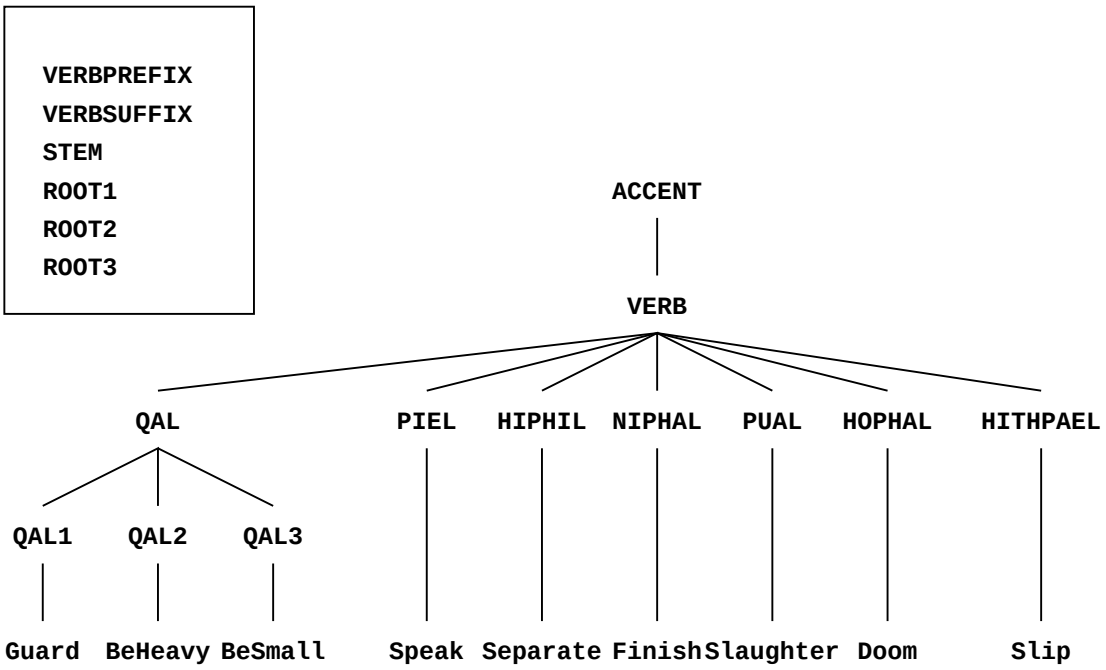

Figure 1: A network of nodes for generating forms of strong verbs in seven *binyanim.*

because Rule 1 is always a better match if it applies at all. Rule 2 is called a **default rule**, because it applies by default if no other rule applies. Default rules define a hierarchical relation among some of the nodes in a KATR theory; thus, in the tree structure depicted in Figure 1, node X dominates node Y iff Y houses a default rule that refers queries to X.

KATR generates output based on queries directed to nodes representing individual words. Since these nodes, such as `Speak`, are not referred to by other nodes, they are called **leaves**, as opposed to nodes like `PIEL`, which are called **internal nodes**.

Here is the output that KATR generates for the `Speak` node and various queries.

```
Speak:<perfect sg 3 masc> → דִּבֶּר
Speak:<perfect sg 3 fem> → דִּבְּרָה
Speak:<perfect sg 2 masc> → דִּבַּרְתָּ
Speak:<perfect sg 2 fem> → דִּבַּרְתְּ
Speak:<perfect sg 1 masc> → דִּבַּרְתִּי
Speak:<perfect sg 1 fem> → דִּבַּרְתִּי
Speak:<perfect pl 3 masc> → דִּבְּרוּ
Speak:<perfect pl 3 fem> → דִּבְּרוּ
Speak:<perfect pl 2 masc> → דִּבַּרְתֶּם
Speak:<perfect pl 2 fem> → דִּבַּרְתֶּן
Speak:<perfect pl 1 masc> → דִּבַּרְנוּ
Speak:<perfect pl 1 fem> → דִּבַּרְנוּ
```

```
Speak:<imperfect sg 3 masc> → יְדַבֵּר
Speak:<imperfect sg 3 fem> → תְּדַבֵּר
Speak:<imperfect sg 2 masc> → תְּדַבֵּר
Speak:<imperfect sg 2 fem> → תְּדַבְּרִי
Speak:<imperfect sg 1 masc> → אֲדַבֵּר
Speak:<imperfect sg 1 fem> → אֲדַבֵּר
Speak:<imperfect pl 3 masc> → יְדַבְּרוּ
Speak:<imperfect pl 3 fem> → תְּדַבֵּרְנָה
Speak:<imperfect pl 2 masc> → תְּדַבְּרוּ
Speak:<imperfect pl 2 fem> → תְּדַבֵּרְנָה
Speak:<imperfect pl 1 masc> → נְדַבֵּר
Speak:<imperfect pl 1 fem> → נְדַבֵּר
```

Our theory represents Hebrew characters and vowels in Unicode characters (Daniels, 1993). We use ´ to indicate the accented syllable if it is not the ultima, and we mark *shewa na* by ˚.

The rule for `Speak` illustrates one of the strategies upon which we build KATR theories: A node representing a category (here, a particular verb) may provide information (here, the letters of the verb's root) needed by more general nodes (here, `PIEL` and the nodes to which it, in turn, refers). We refer to this strategy as **priming**. As we see below, rules in the more general nodes refer to primed information by means of quoted queries.

3 The PIEL node

We now turn to the PIEL node, to which the Speak node refers.

```
PIEL:
    <> = VERB % 1
    <cons2> = ROOT2:<"<root>"> ּ % 2
    {binyanprefix perfect} = % 3
    {binyanprefix imperfect} = ְ % 4
    {binyanprefix imperfect 1 sg} = ֲ % 5
    {vowel1 perfect} = ַ % 6
    {vowel1 imperfect} = ָ % 7
    {vowel2 perfect 3 sg masc} = ֻ % 8
    {vowel2 perfect} = ָ % 9
    {vowel2 imperfect} = ֻ % 10
    .
```

As with the Speak node, PIEL defers most queries to its parent, in this case the node called VERB, as Rule 1 indicates.

Rule 2 modifies a default that VERB will use, namely, the nature of the second consonant of the root. *Pi'el* verbs double their second consonant by applying a *dagesh*. This rule exemplifies a second strategy of KATR theories: A node representing a specific category (here, *pi'el* verbs) may override information (here, the nature of the second consonant) that is assumed by more general nodes (here, VERB and the nodes to which it, in turn, refers). We refer to this strategy as **overriding**. Rule 2 is an overriding rule because the value it assigns to the sequence <cons2> is distinct from the value assigned at the VERB node to which PIEL refers queries by default. We momentarily defer discussing the strange right-hand side of this rule.

The other rules in PIEL are all priming rules. Instead of using angle brackets ("<" and ">") to match queries, they use braces ("{" and "}"). This syntax causes the left-hand side of a rule to be treated as a set instead of an ordered list. The rule whose left-hand side is {binyanprefix perfect} matches any query containing both the atom binyanprefix and the atom perfect, in any order. As before, more than one rule can match a given query, and the rule with the most comprehensive match is chosen. If there are equally good best rules, the KATR theory is considered malformed.

In formulating Rules 3–5, we assume a distinction between *binyan* prefixes (specific to particular *binyanim*) and the personal prefixes (which cross-cut the various *binyanim*); thus, the form נְדַבֵּר "we will speak" contains the *binyan* prefix ְ and the personal prefix נ.

An empty right-hand side in a rule means that the result of a matching query is the empty string. In particular, Rule 3,

```
{binyanprefix perfect} =
```

indicates that there is no *binyan* prefix for *pi'el* verbs in the perfect form, in contrast to, for instance, *hif'il* verbs. The next two rules indicate the *binyan* prefix for a *pi'el* verb's imperfect forms. By Rule 4, this prefix is generally *shewa* (ְ); but because the personal prefix א cannot co-occur with the *binyan* prefix *shewa*, Rule 5 specifies a different *binyan* prefix for a *pi'el* verb's first-person singular imperfect form. (We can adjust the combination אְ to אֲ as a postprocessing step instead, as we show later when we treat guttural letters.)

Every form of a verb separates the three letters of the root by two vowels, which we call vowel1 and vowel2. The *pi'el* is characterized by the fact that in the imperfect, these vowels are the *pataḥ* (by Rule 7) and the *tseyre* (by Rule 10), as in נְדַבֵּר "we will speak"; in the perfect, they are instead generally the *ḥiriq* (by Rule 6) and the *pataḥ* (by Rule 9), as in דִּבַּרְנוּ "we spoke". There is an exception in the perfect third singular masculine (דִּבֵּר), as specified in Rule 8.

Rules 5 and 8 are examples of a third strategy for building KATR theories: A rule may show an exception to a more general pattern introduced by another rule housed at the same node. For instance, Rule 8 establishes a special value for vowel2 for one combination of person, number, and gender, supplanting the more typical value for vowel2 established for imperfect forms by Rule 9. We refer to this strategy as **specializing**.

We now revisit the strange right-hand side of Rule 2. The term on its right-hand side is a node name (ROOT2), a colon, and new query to present to that node. The new query involves a quoted path, "<root>". KATR treats quoted paths in this context as queries on the node from which we started, that is, Speak. In our case, the right-hand side of this rule is equivalent to ROOT2:<ר ב ד>, because of the first rule in the Speak node.

ROOT2 is one of a family of three nodes each

of which isolates a particular consonant in a verb's triliteral root.

```
#vars $consonant:  א ב ג ד ה ו ז ח ט י כ ך
    ל מ ם נ ן ס ע פ ף צ ץ ק ר ש שׁ ת .
ROOT1:  <$consonant#1 $consonant#2
    $consonant#3> = $consonant#1 .
ROOT2:  <$consonant#1 $consonant#2
    $consonant#3> = $consonant#2 .
ROOT3:  <$consonant#1 $consonant#2
    $consonant#3> = $consonant#3 .
```

The #vars declaration introduces a class of atoms: Hebrew consonant characters. Each of the three ROOT nodes has a single rule that matches a three-consonant sequence, assigning each member of the sequence a local number. The rule selects one of those consonants as the result.

These three nodes follow a fourth strategy for writing KATR theories: A node may be invoked solely to provide information (here, a particular consonant in a verb's root) needed by other rules. We refer to this strategy as **lookup**. Lookup nodes (such as the boxed nodes in Figure 1) do not participate in the hierarchical relationships defined by the network's default rules.

To demonstrate that the PIEL node characterizes its *binyan*, we present the somewhat simpler HOPHAL node as a point of comparison.

```
HOPHAL:
    <> = VERB % 1
    {binyanprefix perfect} = הָ % 2
    {binyanprefix imperfect} = ָ % 3
    {vowel1} = ְ % 4
    {vowel2} = ָ % 5
    .
```

4 The VERB node

Queries on Speak are generally reflected to its parent, PIEL, which then reflects them further to VERB.

```
VERB:
    <cons1> = ROOT1:<"<root>"> % 1
    <cons2> = ROOT2:<"<root>"> % 2
    <cons3> = ROOT3:<"<root>"> % 3
    {shortvowel2} = ְ % 4
    <> = ACCENT:<VERBPREFIX STEM
        VERBSUFFIX endofword> % 5
    .
```

Rules 1–3 of VERB determine the three consonants of the root if they have not already been determined by earlier processing. In the case of *pi'el* verbs, <cons2> has been determined (by Rule 2 at the *pi'el* node), but the other consonants have not. That is, if we pose the query Speak:<cons2>, the Speak node reflects it to the PIEL node, which resolves it. But the query Speak:<cons3> is not resolved by PIEL; it is reflected to VERB, which resolves it now by means of lookup.

Rule 4 introduces a priming that is needed by the lookup node STEM: Usually, the shortened version of <vowel2> is the *shewa*. In one *binyan*, namely *hif'il*, the shortened version of <vowel2> is special and overrides this priming.

Rule 5 is the most complicated. It exemplifies two more strategies of programming KATR theories: (1) **Combining:** It combines various pieces of morphology, namely those represented by the nodes VERBPREFIX, STEM, and VERBSUFFIX, each of which is referred to by VERB, and (2) **Postprocessing:** It presents the entire result of that combination to a postprocessing step represented by the node ACCENT.

Combining works by invoking each of the nodes VERBPREFIX, STEM, and VERBSUFFIX with the query presented originally to Speak; such a query might be, for example, Speak:<imperfect sg 3 masc>. (The fact that no query list is explicitly presented to those nodes implies that KATR should use the original query.)

5 Nodes for stems and affixes

Verbs in the imperfect take personal prefixes.

```
VERBPREFIX:
    {perfect} = % 1
    {imperfect 1 sg} = א % 2
    {imperfect 2 sg} = ת % 3
    {imperfect 3 sg masc} = י % 4
    {imperfect 3 sg fem} = ת % 5
    {imperfect 1 pl} = נ % 6
    {imperfect 2 pl} = ת % 7
    {imperfect 3 pl masc} = י % 8
    {imperfect 3 pl fem} = ת % 9
    .
```

We choose not to include the vowel following the prefix as part of this node, but rather as part of STEM.

Such decisions are common in cases of combining; it often makes little difference whether such "boundary" markers are placed at the end of one combining formative or the start of the next one.

Rule 1 indicates that for all queries containing the atom `perfect`, there is no verb prefix. This single rule concisely covers many cases, which are implicitly included because the atoms pertaining to number, person, and gender are omitted. The other rules all apply to the imperfect tense. In the first and second person, the prefix is independent of gender, so the rules there are shorter, again concisely covering multiple cases with only a few rules.

Suffixes have a similar node; here we choose to include the vowel that separates the suffix from the stem.

```
VERBSUFFIX:
    {perfect 1 sg} = ְ תִ @ % 1
    {perfect 2 sg masc} = ְ תָ @ % 2
    {perfect 2 sg fem} = ְ תְ % 3
    {perfect 3 sg masc} = ְ % 4
    {perfect 3 sg fem} = ָה % 5
    {perfect 1 pl} = ְ נוּ @ % 6
    {perfect 2 pl masc} = ְ תֶם % 8
    {perfect 2 pl fem} = ְ תֶן % 9
    {perfect 3 pl} = וּ % 10
    {imperfect sg} = ְ % 11
    {imperfect 2 sg fem} = ִי % 12
    {imperfect 1 pl ++} = ְ % 13
    {imperfect pl masc} = וּ % 14
    {imperfect pl fem} = ְ הָ נָ @ % 15
    .
```

Rules 1, 2, 6, and 15 include the @ character, which we use to indicate that the given syllable should not be accented. Hebrew words are generally accented on the ultima; we place @ on the ultima to force the accent to the penultima. Placing of accents is one of the jobs relegated to the postprocessing step.

The left-hand side of rule 13 includes the symbol ++. This symbol tells KATR that even if another, seemingly better rule matches a query, this rule should take precedence if it matches. The situation arises for the query `<imperfect pl 1 masc>`, for instance. Both rules 13 and 14 match, but the former is preferred. The other way we could have represented this situation is by restricting rule 14 to 2nd or 3rd person, either by explicitly indicating these morphosyntactic properties or by adding the atom `!1`, which means "not first person". We choose to use the disambiguator ++ in Rule 13 instead; in the terminology of (Stump, 2001), the ++ symbol identifies rules that apply in "expanded mode".

The most complex node defines the stem part of a verb.

```
STEM:
    <> = "<binyanprefix>" "<cons1>"
        "<vowel1>" "<cons2>" <anyvowel2>
        "<cons3>" % 1
    <anyvowel2> = "<vowel2>" % 2
    {anyvowel2 perfect 3 sg fem} =
        "<shortvowel2>" % 3
    {anyvowel2 perfect 3 pl} =
        "<shortvowel2>" % 4
    {anyvowel2 imperfect 2 sg fem} =
        "<shortvowel2>" % 5
    {anyvowel2 imperfect !1 pl masc} =
        "<shortvowel2>" % 6
    .
```

Rule 1 uses combining to assemble the parts of the stem, starting with the *binyan* prefix, then alternating all the consonants and vowels. Most of these parts are surrounded in quote marks, meaning that these elements are queries to be reflected back to the starting node, in our case, `Speak`. These queries percolate through `Speak`, `PIEL`, and `VERB` until a priming rule satisfies them.

The only exception is that instead of `<vowel2>`, this rule queries `<anyvowel2>` without quote marks. The absence of quote marks directs this query to the current node, that is, `STEM`; the remaining rules determine what vowel is appropriate.

Rule 2 indicates that unless another rule is better, `anyvowel2` is just `vowel2`. However, in four cases, `vowel2` must be replaced by `shortvowel2`, typically *shewa* (primed by the `VERB` node), but occasionally something else (overridden by *hif'il* verbs).

6 Postprocessing

Many languages have rules of euphony. These rules are often called *sandhi* operations, based on a term used in Sanskrit morphology. We use the node `ACCENT` to introduce sandhi operations. Its name comes from the fact that the first operation we

needed was to place the accent on the penultima, but we use it for other purposes as well.

We begin by defining character classes similar to the $consonant class introduced earlier.

```
#vars $vowel:  ַ ָ ֵ ִ ֹ ֻ ְ ֱ ֲ ֳ .
#vars $accent:  ֽ .
#vars $unaccentableVowel:  ְ .
#vars $accentableVowel:  $vowel -
    $unaccentableVowel .
#vars $letter:  $vowel + $consonant +
    $accent .
#vars $noAccent:  $letter -
    $accentableVowel .
```

Each class contains a subset of the Hebrew characters. We treat some combinations as single characters for this purpose, in particular, the vowels הַ and יִ. The first three classes are defined by enumeration. The fourth class, $accentableVowel, is defined in terms of previously defined classes, specifically, all vowels except those that are unaccentable. Similarly, the $letter class includes all vowels, consonants, and accents, and the $noAccent class contains all letters except for accentable vowels. These classes are used in the ACCENT node.

```
ACCENT:
    <$letter> = $letter <> % 1
    <endofword> = % 2
    <$accentableVowel#1 $noAccent*
        $accentableVowel#2 @> =
        $accentableVowel#1 ֽ $noAccent*
        $accentableVowel#2 <> % 3
    <ְ endofword> = % 4
    <ךְ endofword> = ך ְ % 5
    <ְ $consonant ְ endofword> =
        $consonant ְ % 6
    .
```

A query to ACCENT is a fully formed Hebrew word ready for postprocessing, with the endofword tag placed at the end. The first rule is a default that often is overridden by later rules; it says that whatever letter the query starts with, that letter can be removed from the query, and placed as a result. Furthermore, the unmatched portion of the query, indicated by <> on the right-hand side, is to be directed to the ACCENT node for further processing. Rule 2 says that if a resulting query has only endofword, that tag should be removed, and no further processing is needed.

Rule 3 places accents in words that contain the @ sign, which we use to indicate "do not accent this syllable." The left-hand side matches queries that contain an accentable vowel, followed by any number (zero or more, indicated by the Kleene star *) of letters that cannot be accented, followed by a second accentable vowel, followed by the @ mark. Such words must have the @ removed and an accent mark placed after the first accentable vowel matched, as indicated in the right-hand side. The empty <> at the end of the right-hand side directs unused portions of the query to ACCENT for further processing.

Rules 4, 5, and 6 deal with *shewa* near the end of a word. Generally, *shewa* is deleted at the very end (rule 4), but not if it follows ך (rule 5) or if the previous vowel is also a *shewa* (rule 6).

7 Accommodating guttural letters

Our current efforts involve accommodating verb roots containing guttural letters. We have found that new rules in the postprocessing step, that is, the ACCENT node, cover many of the cases.

We first introduce postprocessing rules that convert *shewa naḥ* (which we continue to represent as ְ) to *shewa na* (which we represent as ְ°).

```
#vars $longVowel:  ָ ו  ִ ְ א .
ACCENT:
     ...  % other rules as before
    <startofword $consonant $dagesh?  ְ/ְ°>
        =+= <> % 8
    <ְ/ְ°$consonant#1 $dagesh?  ְ
        $consonant#2> =+= <> % 9
    <$longVowel $consonant ְ/ְ°> =+= <>
        % 10
    <$consonant $dagesh ְ/ְ°> =+= <> % 11
    <$consonant#1 $dagesh?
        ְ/ְ°$consonant#1> =+= <> % 12
```

Rule 8 converts *shewa naḥ* to *shewa na* on the first consonant of the word. We introduce the atom startofword in order to detect this situation, and we modify the reference to the ACCENT node in the VERB node to include this new atom. This rule uses =+= instead of = to separate the two sides. This notation indicates a non-subtractive rule; the right-hand side path encompasses the entire query, including that part matched by the left-hand side, except

that the *shewa naḥ* has been replaced by *shewa na*. After this replacement, KATR continues to process the new query at the same node. The left-hand side uses the ? operator, which means "zero or one instances." This notation allows a single rule to match situations both with and without a *dagesh*.

The other rules use similar notation. Rule 9 converts the first of two *shewa*s in a row to a *shewa na*, except at the end of the word. Rule 10 converts a *shewa naḥ* following a long vowel. Rule 11 converts a *shewa naḥ* on a consonant with a *dagesh*. Rule 12 converts the *shewa naḥ* on the first of two identical consonants.

Given the distinction between the two *shewa*s, we now add postprocessing rules that convert a guttural with a *shewa na* to an appropriate alternative.

```
#vars $guttural:  א ח ה ע .
ACCENT:
     ...  % other rules as before
    <$guttural ְ> = $guttural ַ <> % 13
    <$guttural ְ> = ַ $guttural ַ <> % 14
    <$guttural ְ> = ָ $guttural ָ <> % 15
    <א $letter> = אְ <$letter> % 16
    <אְ א> = אְ <> % 17
```

Rule 13 corrects, for example, שָׁחְטוּ to שָׁחֲטוּ; Rule 14 corrects תִּעְמֹד to תַּעֲמֹד, and Rule 15 corrects אֶעְמֹד to אֶעֱמֹד. Rules 16 and 17 correct the initial א in פ"א verbs in the *qal*.

We add other rules, such as the following Rule 18, to correct situations where a guttural letter would otherwise acquire a *dagesh*.

```
<$guttural ּ> = <$guttural> % 18
```

We have not begun work on weak verbs containing ה, ו, and י, which might require different approaches.

8 Further work

We continue to develop our Hebrew KATR theory. Our goal is to cover all forms, including the *waw* consecutive, infinitive, *makor*, and predicate suffixes, for both strong and weak verbs. We will then turn to nouns, including personal suffixes. Our success so far indicates that KATR is capable of representing Hebrew morphology in a concise yet readable form.

Our larger goal is to host a library of KATR theories for various languages as a resource for linguists. Such a library will provide interested researchers with morphological descriptions that can be directly converted into actual word forms and will serve as a substitute, to some extent, for voluminous natural-language and table-based descriptions. In the case of endangered languages, it will act as a repository for linguistic data that may be essential for preservation.

9 DATR and KATR

We discuss KATR and its relation to DATR extensively elsewhere (Finkel et al., 2002); here we only summarize the differences. The DATR formalism is quite powerful; we have demonstrated that it is capable of emulating a Turing machine. The KATR enhancements are therefore aimed at usability, not theoretical power. The principal innovations of KATR are:

- Set notation. The left-hand sides of DATR rules may only use list notation. KATR allows set notation as well, which allows us to deal with morphosyntactic properties in any order.

  Hebrew verb morphology provides abundant motivation for this enhancement. In the VERB-SUFFIX node, Rule 15 identifies נָה as an exponent of number and gender but not of person; Rule 10 identifies ו as an exponent of person and number but not of gender. Both rules are indifferent to the order in which properties of person, number, and gender are listed in any matching query. If a rule's left-hand side were required to be a list (as in ordinary DATR), then one of these two rules would have to be complicated by the inclusion of either a variable over properties of person (Rule 15) or a variable over properties of gender (Rule 10); moreover, all queries would have to adhere to a fixed (but otherwise unmotivated) ordering among properties of person, number, and gender.

- Regular expressions. KATR allows limited regular expressions in lists in left-hand sides of rules; DATR has no such expressions. We use this facility in the ACCENT node in the Hebrew theory, both for the Kleene star * and for the ? operator. More generally, we often find regular

expressions valuable in representing non-local sandhi phenomena, such as the Sanskrit rule of n-retroflexion.

- Non-subtractive rules. DATR rules have a subtractive quality: The atoms of the query matched by the left-hand side are removed from the query used for subsequent evaluation in the right-hand side. The KATR `=+=` operator allows us to represent rules that preserve the atoms matched by the left-hand side, substituting new atoms where necessary. We generally use this facility for rules of referral. For example, Latin neuter nouns share the same nominative and accusative plural; we capture this fact by a rule that converts accusative to nominative in the context of neuter plural. In the Hebrew theory, we use non-subtractive rules to convert *shewa naḥ* to *shewa na*.

- Enhanced matching length. In some cases, competing rules have left-hand sides of the same length, but one of the rules should always be chosen when both apply. KATR includes the `++` syntax for explicitly enhancing the effective length of the preferred left-hand side; we use this facility in the VERBSUFFIX node. DATR does not have this syntax.

- Syntax. KATR has several minor syntax enhancements. It allows special characters to be used as atoms if escaped by the \ character. The atom `$$` can be used to match the end of the query. Variables can be computed instead of being enumerated; we use this facility in defining the `$letter` variable. KATR allows greater control over which nodes are to be displayed under default queries. The interactive KATR program has new facilities for rapid testing and debugging of theories.

KATR is entirely coded in Java, making it quite portable to a variety of platforms. It runs as an interactive program, with commands for compiling theories, executing queries, and performing various debugging functions. The KATR algorithm is based on evaluating a query at a node within a context. First, KATR identifies the rule within the node with the best matching left-hand side. The result of the query involves evaluating the associated right-hand side, which might require further evaluations of new queries at a variety of nodes and contexts; KATR recursively undertakes these evaluations. The algorithm is completely deterministic and reasonably fast: Compiling the entire Hebrew theory and evaluating all the forms of a verb takes about 2 seconds on an 863MHz Linux machine.

The interested reader can acquire KATR and our Hebrew morphology theory from the authors (under the GNU General Public License).

10 Strategies for building KATR theories

We have been applying KATR to generation of natural-language morphology for several years. In addition to Hebrew, we have built a complete morphology of Latin verbs and nouns, large parts of Sanskrit (and other related languages), and smaller studies of Bulgarian, Swahili, Georgian, and Turkish. We have found that KATR allows us to represent morphological rules for these languages with great elegance. It is especially well-suited to cases like Hebrew verbs, where a similar structure applies across the entire spectrum of words, and where that spectrum is partitioned into *binyanim* with distinguishable rules, but where euphony introduces standard vowel shifts based on accent, guttural letters, and weak letters.

As we have gained experience with KATR, we have noted encoding strategies that apply across language families; we used each of these in our Hebrew verb specification.

- **Priming.** A node representing a specific category provides information needed by more general nodes to which it refers queries. Rules in the more general nodes refer to primed information by means of quoted queries.

- **Lookup.** A node is invoked solely to provide information needed by other rules.

- **Overriding.** A node representing a specific category answers a query that is usually answered (with different results) by a more general node to which queries are usually referred.

- **Specializing.** A rule introduces a specific exception to a more general pattern specified by another rule housed at the same node.

The strategies of overriding and specializing both exploit the nonmonotonicity inherent in KATR's semantics.

- **Combining.** A rule concatenates various morphological units by referring queries to multiple nodes.
- **Postprocessing.** The result of combining morphological units is referred to a node that makes local adjustments to account for euphony and other sandhi principles.

We do not want to leave the impression that writing specifications in KATR is easy. The tool is capable of presenting elegant specifications, but arriving at those specifications requires considerable effort. Early choices color the entire structure of the resulting KATR specification, and it happens frequently that the author of a specification must discard code and rethink how to represent the morphological structures that are being specified. Perhaps our experience will eventually lead to a second-generation KATR that better facilitates the linguist's task.

The definition of Hebrew verb inflection that we have sketched here rests on the hypothesis that an inflected word's morphological form is determined by a system of realization rules organized in a default inheritance hierarchy. There are other approaches to defining Hebrew verb inflection; one could, for example, assume that an inflected word's form is determined by a ranked system of violable constraints on morphological structure, as in Optimality Theory (Prince and Smolensky, 1993), or by a finite-state machine (Karttunen, 1993). The facts of Hebrew verb inflection are apparently compatible with any of these approaches. Even so, there are strong theoretical grounds for preferring our approach. It provides a uniform, well-defined architecture for the representation of both morphological rules and lexical information. Moreover, it embodies the assumption that inflectional morphology is inferential and realizational, readily accommodating such phenomena as extended exponence and the frequent underdetermination of morphosyntactic content by inflectional form; in this sense, it effectively excludes a morpheme-based conception of word structure, unlike both the optimality-theoretic and the finite-state approaches.

Acknowledgements

This work was partially supported by the National Science Foundation under Grant 0097278 and by the University of Kentucky Center for Computational Science. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

References

Greville G. Corbett and Norman M. Fraser. 1993. Network Morphology: A DATR account of Russian nominal inflection. *Journal of Linguistics*, 29:113–142.

P. T. Daniels. 1993. The Unicode Consortium: The Unicode standard. *Language: journal of the Linguistic Society of America*, 69(1):225–225, March.

Roger Evans and Gerald Gazdar. 1989. Inference in DATR. In *Proceedings of the Fourth Conference of the European Chapter of the Association for Computational Linguistics*, pages 66–71, Manchester.

Raphael Finkel, Lei Shen, Gregory Stump, and Suresh Thesayi. 2002. KATR: A set-based extension of DATR. *under review*.

Norman M. Fraser and Greville G. Corbett. 1995. Gender, animacy, and declensional class assignment: a unified account for Russian. In G. Booij and J. van Marle, editors, *Yearbook of Morphology 1994*, pages 123–150. Kluwer, Dordrecht.

Norman M. Fraser and Greville G. Corbett. 1997. Defaults in Arapesh. *Lingua*, 103:25–57.

Lauri Karttunen. 1993. Finite-state constraints. In John Goldsmith, editor, *The Last Phonological Rule*. University of Chicago Press, Chicago.

Alan S. Prince and Paul Smolensky. 1993. Optimality theory: Constraint interaction in generative grammar. Technical Report RuCCs Technical Report #2, Rutgers University Center for Cognitive Science, Piscataway, NJ.

Gregory T. Stump. 2001. *Inflectional morphology*. Cambridge University Press, Cambridge, England.