KATR: A Set-Based Extension of DATR*

Raphael Finkel, Lei Shen, Gregory Stump and Suresh Thesayi
University of Kentucky

(raphael@cs.uky.edu, lshen@rswsoftware.com, gstump@uky.edu, suresh_t_r@hotmail.com)

Technical Report No. 346-02

ABSTRACT

In the framework of Network Morphology (Corbett & Fraser 1993; Fraser & Corbett 1995, 1997), realizational models of natural-language morphology have customarily been defined in DATR, a language for lexical knowledge representation designed and implemented by Roger Evans and Gerald Gazdar (Evans & Gazdar 1989a,b, 1996). We show that certain kinds of morphological analyses that are wholly consonant with the general program of Network Morphology are not directly expressible in existing forms of DATR; we therefore propose and exemplify KATR, an extension of DATR whose motivation is to accommodate these desired kinds of morphological analyses. The proposed modifications are motivated by the need to represent a word's morphosyntactic property as essentially unordered; to account for the incidence of nonlocal sandhi phenomena; to define common patterns of inflectional syncretism; and to allow certain morphological rules to apply in "expanded mode" (Stump 2001). Although KATR affords morphological definitions that are more streamlined than those afforded by DATR, its generative capacity is no greater than that of DATR, since both languages are capable of emulating a Turing machine.

1 Introduction

In an important series of articles (Corbett & Fraser 1993; Fraser & Corbett 1995, 1997), Greville Corbett and Norman Fraser elaborated an approach to morphology that makes extensive use of default inheritance hierarchies in the analysis of complex inflectional systems. This approach, dubbed Network Morphology, is squarely in the tradition of inferential-realizational theories of morphology: It is inferential (as opposed to lexical) in the sense that it represents inflectional exponents not as lexically listed elements, but as markings associated with the application of rules by which complex word forms are deduced from simpler roots and stems; and it is realizational (as opposed to incremental) in the sense that it entails that a word's association with a particular set of morphosyntactic properties is a precondition for--rather than a consequence of--the application of the rule introducing the inflectional exponents of those properties. What distinguishes Network Morphology from certain other inferential-realizational theories is its systematic use of non-monotonic inheritance hierarchies to structure the information constituting a language's morphology. Analyses in Network Morphology are formally implemented in DATR, a formal language for lexical knowledge representation designed and implemented by Roger Evans and Gerald Gazdar (Evans & Gazdar 1989a,b, 1996).

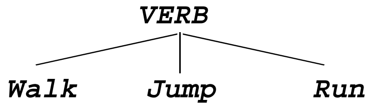
In this paper, we show that certain kinds of morphological analyses that are wholly consonant with the general program of Network Morphology are not directly expressible in existing forms of DATR; we therefore propose and exemplify KATR, an extension of DATR whose motivation is to accommodate these desired kinds of morphological analyses. Our discussion proceeds as follows. In Section 2, we review the fundamental characteristics of DATR proper. In Section 3, we demonstrate that under the assumptions of Network Morphology, these characteristics entail undesirable complications for the analysis of certain kinds of morphological phenomena. These complications stem from the essential lack of linear ordering among the members of a morphosyntactic property set; from the incidence of nonlocal sandhi phenomena; and from common patterns of inflectional syncretism. In Section 4, we introduce KATR and show how it resolves the issues raised in Section 3. In Section 5, we discuss an additional empirical challenge for KATR and show how KATR meets this challenge. In Section 6, we consider the relative power of DATR and KATR; we show that KATR is no more powerful than DATR, since both are capable of emulating a Turing machine. Section 7 summarizes our conclusions.

2 Basic characteristics of DATR

Before introducing the formal properties of KATR and the linguistic phenomena that have motivated its development, we outline the basic characteristics of DATR.

DATR is a formal language for defining nonmonotonic inheritance networks. It has been applied with considerable success in modeling morphological systems (in addition to the cited works by Corbett and Fraser, see Brown 1996, 1998a,b,c; Brown & Hippisley 1994; Brown *et al.* 1996; Cahill & Gazdar 1997; Hippisley 1996, 1997, 1998). The central notion in DATR is that of a **THEORY**: a network of nodes such that (i) each node houses some body of information and (ii) this information is shared among nodes in a deterministic fashion. A simple example of a DATR theory is the network of nodes in (1), which affords a compact description of some of the morphological and syntactic characteristics of three English verbs. Each **NODE** in a theory is a location at which facts are situated; for instance, the nodes in (1) are locations at which the facts in (2) are situated.

(1)



(2)
```
VERB:
      <syntactic category> == verb
      <present participle> == "<root>" ing
      <past> == "<root>" ed.
Walk:
      <> == VERB
      <root> == walk.
Jump:
      <> == VERB
      <root> == jump.
Run:
      <> == VERB
      <root> == run
      <past> == ran
      <past participle> == <root>.
```

A **FACT** pairs an atom-path with a value and has the format *atom_path* == *value*; thus, the first fact located at the `VERB` node in (2) pairs the atom-path `<syntactic category>` with the value `verb`. An **ATOM-PATH** is a sequence of atoms enclosed in angle brackets; thus, the atom-path `<syntactic category>` in (2) consists of the atoms `syntactic` and `category`.[1] We refer to the atom-path as the fact's **LEFTHAND SIDE** (LHS) and to the value as the fact's **RIGHTHAND SIDE** (RHS).

2.1 Matching facts to queries in DATR

The DATR engine follows a deterministic algorithm for extracting information from a theory of this sort; in particular, when queries are addressed to a theory such as (2), the DATR engine computes values for them. A **QUERY** takes the form *Node_q:atom_path_q*; we refer to *Node_q* and *atom_path_q* as the **QUERY NODE** and as **QUERY PATH**, respectively.

An example of an appropriate query for the theory in (2) is `Run:<past participle>`; given the facts in (2), the DATR engine computes `run` as the value corresponding to this query.

In (2), two facts have LHS `<past>` and `<past participle>`. We say that the first is a prefix of the second. More formally, if *atom_path_1* is of length *n*, where length is measured as the number of atoms in an atom-path, then *atom_path_1* is the **PREFIX** of *atom_path_2* if and only if for every *i* (where 1 ≤ *i* ≤ *n*), *atom_path_1*'s *i*th member is also *atom_path_2*'s *i*th member. For instance, the atom-paths `<>`, `<past>`, and `<past participle>` are all prefixes of the atom-path `<past participle>`; of these, `<past participle>` is the longest prefix.

Suppose now that we have a query *Node_q:atom_path_q*. In order to compute a value for this query, the DATR engine matches a fact situated at the query node *Node_q* to the query path *atom_path_q*; in particular, the engine identifies that fact at *Node_q* whose LHS is the longest prefix of *atom-path_q*. Thus, when the DATR engine searches the theory in (2) in response to the query `Run:<past participle>`, it employs the fact `<past participle> == <root>` at the `Run` node in evaluating this query. This principle for matching a fact at the query node to a query path can be seen as an expression of Pāṇini's principle (Kiparsky's (1973) "Elsewhere Condition"): The evaluation of a query of the form *Node_q:atom_path_q* proceeds according to that fact at *Node_q* whose LHS *atom_path_f* is a prefix of *atom_path_q* and is such that for any other fact at *Node_q* whose LHS *atom_path_c* is a prefix of *atom_path_q*, *atom_path_f* is more specific (that is, longer) than *atom_path_c*.

2.2 Evaluating queries in DATR

If a query path *atom_path_q* is matched by a fact at the query node *Node_q*, evaluation of the query *Node_q:atom_path_q* is possible. In DATR, the RHS of a fact is a sequence of zero or more terms. A **TERM** may be (A) an atom (as in the fact `<syntactic category> == verb` in (2)); (B) a node (as in the fact `<> == VERB` in (2)); (C) a **TERM-PATH**, that is, a sequence of terms enclosed in angle brackets (as in the fact `<past participle> == <root>` in (2)); (D) a pairing of a node with a term-path, that is *Node_y*`:`*<term_1 ... term_n>*; (E) a quoted term-path, that is `"`*<term_1 ... term_n>*`"`; (F) a quoted node, that is `"`*Node_y*`"`; or (G) a quoted pairing of a node with a term-path, that is `"`*Node_y*`:`*<term_1 ... term_n>*`"`.

Empty case: At *Node_q*, a query path *atom_path_q* is matched by the fact in (3), whose RHS is a sequence of zero terms; in that case, the value computed for the query *Node_q*`:`*atom_path_q* is the empty sequence. Single-term case: At *Node_q*, a query path *atom_path_q* is matched by the fact in (4), whose RHS is a single term *term_0*.

(3) *atom_path_c* **==**
(4) *atom_path_f* **==** *term_0*

In this case, the value computed for the query *Node_q***:***atom_path_q* is the value of *term_0* relative to the query path *atom_path_q* and the LHS *atom_path_f* at *Node_q*. The value of a term relative to a query path and an LHS at some node depends on which of the seven types of terms is involved. If *term_0* is an atom, as in Table 1(A), then no further computation is necessary: The value computed for the query *Node_q:atom_path_q* is simply that atom. But if *term_0* is any other sort of term, as in Table 1(B)-(G), then subsequent computation is necessary. As an example, consider the DATR theory in (5).

Given a query *Node_q:atom_path_q* such that *atom_path_q* is matched by the fact in (4) at *Node_q:*	
if *term_0* is of type	then the value of *term_0* relative to the query path *atom_path_q* and the LHS *atom_path_f* at *Node_q* is
(A) *atom*	*term_0*
(B) *Node_y*	the value computed for the query *Node_y***:***atom_path_q*
(C) *<term_1 ... term_n>*	the value computed for the query *Node_q***:***<X Y>*, where (i) X is the result of concatenating the values of *term_1* ... *term_n* relative to the query path *atom_path_q* and the LHS *atom_path_f* at *Node_q*, and (ii) Y is the (possibly empty) sequence of atoms such that *atom_path_q* is <W Y> and *atom_path_f* is <W>
(D) *Node_y***:***<term_1 ... term_n>*	the value computed for the query *Node_y***:***<X Y>*, where X and Y are as above
(E) *"<term_1 ... term_n>"*	the value computed for a new query *Node_INIT***:***<X Y>*, where *Node_INIT* is the initial query node (defined below) and X and Y are as above
(F) *"Node_y"*	the value computed for a new query *Node_y***:***<Z>*, where <Z> is the initial query path (defined below)
(G) *"Node_y***:***<term_1 ... term_n>"*	the value computed for a new query *Node_y***:***<X Y>*, where X and Y are as above

Table 1. Terms and their values in DATR

(5)
```
A:
      <a b c> == B
      <d e f> == <a C B>
      <g> == B:<d e <h>>
      <h> == k
      <i> == B
      <j> == <k l>
      <k l> == "B"
      <m> == "B:<n>".
B:
      <a b c> == d
      <d e f> == c
      <d e k> == j
```

```
        <h> == j
        <i> == "<h>"
        <j> == C
        <k l> == a
        <n> == C.
    C:
        <d e f> == b
        <j> == "<h>"
        <n> == "<h>".
```

According to Table 1(B), the value computed for the query `A:<a b c>` in the theory of (5) is the value computed for the query `B:<a b c>`; in accordance with Table 1(A), this latter value is `d`. In this instance, the evaluation of a query path at some query node amounts to evaluating the same query path at another query node.

According to Table 1(C), the value computed for the query `A:<d e f g>` in this theory is the value computed for the query `A:<a b c g>`, that is, for the query `A:<X g>`, where X is the result of concatenating the values of the terms `a`, `C`, and `B` relative to the query path `<d e f g>` and the LHS `<d e f>` at `A`; thus, in accordance with Table 1(A, B), the value computed for the query `A:<d e f g>` in this theory is `d`. In this instance, the evaluation of a query path at some query node amounts to the evaluation of another query path at that same query node.

According to Table 1(D), the value computed for the query `A:<g>` in this theory is the value computed for the query `B:<d e k>`, that is, for the query `B:<d e` X`>`, where X is the value of the term `<h>` relative to the query path `<g>` and the LHS `<g>` at `A`; thus, in accordance with Table 1(A-C), the value computed for the query `A:<g>` in this theory is `j`. In this instance, the evaluation of a query path at some query node amounts to the evaluation of another query path at another query node.

As these examples show, the evaluation of one query may entail the evaluation of a chain of subsequent queries. The node at which a chain of queries is initiated is the **INITIAL QUERY NODE**; the query path with which a chain of queries is initiated is the **INITIAL QUERY PATH**. These notions are essential for understanding the double-quote notation in Table 1(E-G).

Suppose that node `A` is the initial query node in the evaluation of a query `A:<i>` in this theory. According to Table 1(B), the value of this query is the value computed for the query `B:<i>`; according to Table 1(E), this latter value is in turn the result of initiating a new query `A:<h>`. Thus, the value computed for the query `A:<i>` is `k`. In this instance, the evaluation of a query path at some query node amounts to initiating a new query at the initial query node.

It is also possible to initiate a new query at a node distinct from the initial query node; in this case, the node at which the new query is initiated becomes the initial query node for purposes of dependent calculations. Suppose that a query `A:<j>` is initiated at node `A`. According to Table 1(F), the value computed for this query is the result of initiating a new query at node `B` with the initial query path `<j>`. For purposes of dependent calculations, `B` rather than `A` is the initial query node; thus, the ultimate value computed for `A:<j>` is `j` rather than `k`. In this instance, the evaluation of a query path at some query node amounts to initiating a new query at another node using the initial query path.

Suppose now that a query `A:<m>` is initiated at node `A`. According to Table 1(G), the value computed for this query is the result of initiating a new query `B:<n>`; accordingly, the ultimate value computed for `A:<m>` is again `j` rather than `k`. In this instance, the evaluation of a query path at some query node amounts to initiating a new query at another node using another query path.

General-term case: At *Node_q*, a query path *atom_path_q* is matched by the fact in (6), whose RHS is a sequence of zero or more terms.

(6) *atom_path_f* `==` *term_1... term_n*

If a query path *atom_path_q* is matched by the fact in (6) at *Node_q*, the value computed for the query *Node_q***:***atom_path_q* is the sequence of the values of *term_1 ... term_n* relative to the query path *atom_path_q* and the LHS *atom_path_f* at *Node_q*. The value of each of these terms relative to *atom_path_q* and *atom_path_f* at *Node_q* is determined exactly as in Table 1.

3 Some empirical challenges for Network Morphology

Recent work in Network Morphology (Brown 1996, 1998a,b,c; Brown & Hippisley 1994; Brown et al. 1996; Cahill & Gazdar 1997; Corbett & Fraser 1993; Fraser & Corbett 1995, 1997; Hippisley 1996, 1997, 1998) has demonstrated the exceptional utility of DATR for modelling inflectional systems. Nevertheless, certain kinds of morphological phenomena in natural language present important challenges for Network Morphology because they are not directly representable in existing forms of DATR. Here, we discuss three specific problems: the need to represent a word's morphosyntactic property set as unordered (3.1); the description of nonlocal sandhi phenomena (3.2); and the phenomenon of inflectional syncretism (3.3). One further problem, that of competing rules, is discussed later (5.1).

3.1 Unordered morphosyntactic property sets

Stump (2001) discusses a difficulty that DATR poses for the representation of natural-language rules of inflectional morphology. In network-morphologic applications of DATR, an inflectional rule realizing a morphosyntactic property-set σ is represented as a fact *f* whose LHS *atom_path_f* is an atom-path containing the members of σ; in (2), for instance, the inflectional rule realizing the morphosyntactic property set {present, participle} is represented as a fact whose LHS is the atom-path ***`<present participle>`***. This representation is, however, somewhat paradoxical, since the members of an atom-path are by definition ordered, while those of a property set are unordered. A consequence of this paradox is an inevitable redundancy in the formulation of inflectional rules.

The partial Swahili verb paradigm in Table 2 provides a clear illustration of this situation. As this table shows, a Swahili verb inflects for polarity, tense, and subject agreement: The prefix *ha-* appears as the exponent of negative polarity in slot iii; prefixal exponents of subject agreement appear in slot ii; prefixal exponents of tense, in slot i; and in the special case of first-person singular negative forms, the portmanteau prefix *si-* appears in slot iv, pre-empting slots iii and ii. Now, suppose the set σ of morphosyntactic properties realized by a Swahili verb form *W* is formulated as a path *P*. In that case, each of the inflectional exponents in *W* realizes a subset of σ; yet, no matter what ordering is assumed for the properties constituting *P*, the rules introducing the various inflectional exponents in *W* cannot be formulated without redundancy as facts having prefixes of *P* as their lefthand sides. Consider, for example, the DATR theory in (7).

		Positive			Negative			
					iv			
	Affix slot:	ii	i	(stem)	iii	ii	i	(stem)
Past Tense	1SG	*ni-*	*li-*	*taka*	*si-*		*ku-*	*taka*
	2SG	*u-*	*li-*	*taka*	*ha-*	*u-*	*ku-*	*taka* (phonetically *hukutaka*)
	3SG (CLASS 1)	*a-*	*li-*	*taka*	*ha-*	*a-*	*ku-*	*taka* (phonetically *hakutaka*)
	1PL	*tu-*	*li-*	*taka*	*ha-*	*tu-*	*ku-*	*taka*
	2PL	*m-*	*li-*	*taka*	*ha-*	*m-*	*ku-*	*taka*
	3PL (CLASS 2)	*wa-*	*li-*	*taka*	*ha-*	*wa-*	*ku-*	*taka*
Future Tense	1SG	*ni-*	*ta-*	*taka*	*si-*		*ta-*	*taka*
	2SG	*u-*	*ta-*	*taka*	*ha-*	*u-*	*ta-*	*taka* (phonetically *hutataka*)
	3SG (CLASS 1)	*a-*	*ta-*	*taka*	*ha-*	*a-*	*ta-*	*taka* (phonetically *hatataka*)
	1PL	*tu-*	*ta-*	*taka*	*ha-*	*tu-*	*ta-*	*taka*
	2PL	*m-*	*ta-*	*taka*	*ha-*	*m-*	*ta-*	*taka*
	3PL (CLASS 2)	*wa-*	*ta-*	*taka*	*ha-*	*wa-*	*ta-*	*taka*

Table 2 . Partial inflectional paradigm of Swahili *taka* `want'

(7) A DATR theory defining the Swahili paradigm in Table 2

```
%variable declarations:
#vars $abc: a h i k l m n s t u w.
#vars $tense: future past.
#vars $polarity: negative positive.
SANDHI:
      <$abc> == $abc <>
      <> ==
      <a u> == u <>
      <a a> == a <>.
VERB:
      <slot_iv> == <slot_iii> <slot_ii>
      <slot_iv negative $tense 1 sg> == s i       %fact F1
      <slot_iii negative> == h a
      <slot_iii> ==
      <slot_ii $polarity $tense 1 sg> == n i      %fact F2
      <slot_ii $polarity $tense 2 sg> == u        %fact F3
      <slot_ii $polarity $tense 3 sg> == a        %fact F4
      <slot_ii $polarity $tense 1 pl> == t u      %fact F5
      <slot_ii $polarity $tense 2 pl> == m        %fact F6
      <slot_ii $polarity $tense 3 pl> == w a      %fact F7
      <slot_i positive past> == l i
      <slot_i negative past> == k u
      <slot_i $polarity future> == t a            %fact F8
      <> == SANDHI:<<slot_iv> <slot_i> "<root>">.
Want:
      <> == VERB
      <root> == t a k a.
#hide SANDHI VERB.
#show
```

```
<positive past 1 sg>
<positive past 2 sg>
<positive past 3 sg>
…
```

In this theory, the query **`Want:<positive future 3 pl>`** produces the value **`w a t a t a k a`**, the query **`Want:<negative past 1 sg>`** produces **`s i k u t a k a`**, and so on. The morphosyntactic properties in each query follow the sequence *polarity*, *tense*, *person*, *number*. This fact in turn requires that morphosyntactic properties be mentioned in this same sequence on the lefthand sides of the facts at the **`VERB`** node. More generally, if a fact's LHS mentions a property associated with position *n* in this sequence, then it must likewise mention a property associated with position *n* – 1. For instance, in fact **`F2`**, which specifies the prefix **`ni`** as the exponent of first-person singular subject agreement, the variables **`$polarity`** and **`$tense`** must be mentioned, even though **`ni`** is, strictly speaking, an exponent of neither polarity nor tense. Rather than say that the prefixation of **`ni`** realizes the path **`<slot_ii $polarity $tense 1 sg>,`** one would prefer to be able to say that the prefixation of **`ni`** simply realizes the property set **`{slot_ii`**, **`1`**, **`sg}`**, whatever the order might be in which these properties are specified in the query path. The problem cannot simply be solved by changing the sequence to *person, number, polarity, tense* in the query path, since that change would require that facts specifying the exponents of polarity and tense also include redundant variables over person and number values.

3.2 Nonlocal sandhi

In DATR, a rule of sandhi can be formulated as a fact whose LHS is a path that matches a sequence of adjacent phonological segments; some examples are the facts situated at the **`SANDHI`** node in (7). Though sandhi alternations usually involve a conditioning element adjacent to the conditioned alternant, it is not rare in natural language for a conditioning element to be nonlocal, in the sense that it is separated from the conditioned alternant by an indefinite amount of intervening material. Consider, for instance, the Sanskrit principle of *n* retroflexion. In Gonda's (1966:19) succinct formulation:

> An *n* which a vowel or *n m y v* follows is changed to *ṇ* if *ṛ ṝ r ṣ* immediately precede in the same word or no palatal, cerebral [= retroflex], or dental stands in between: *muṣ-nā-ti* > *muṣṇāti* "he steals"; *karman-ā* > *karmaṇā* "by the deed", but *rathena* "by the chariot"; *śuśrūṣaṇa-* "obedience", *sravaṇa-* "flowing", but *darśana-* "seeing", *grasana-* "swallowing".

In order to accommodate the expression of sandhi principles having this nonlocal character, DATR should make it possible to formulate a fact whose LHS matches a sequence of phonological segments in which an indefinite amount of material may intervene between two designated segments. Though DATR affords various indirect means of achieving this effect, it does not allow this effect to be achieved in the simplest, most direct way: by reference to a variable over strings.

3.3 Subtractive vs nonsubtractive rules

DATR facts have a subtractive quality. For instance, addressing the query **`A:<a b c d>`** to the sample theory in (8) yields the value **`f`** rather than **`e`**: Fact **`F1`** situated at the **`A`** node subtracts away the prefix `<a b c>`, causing the query `A:<a b c d>` to be subsequently evaluated as the query `B:<d d>` rather than as `B:<d a b c d>`.

(8)
```
A:
        <a b c> == B:<d>.    %fact F1
B:
        <d a b c d> == e
        <d d> == f.
```

In the definition of certain linguistic phenomena, however, this subtractive quality engenders redundancies. One phenomenon of which this is true is that of syncretism. In Sanskrit, for example, the default declensional pattern includes the following syncretisms, among others: (a) a nominal's ablative singular form is identical to its genitive singular form; (b) a nominal's locative dual form is identical to its genitive dual form; (c) a nominal's dative dual form is identical to its instrumental dual form; and (d) a neuter nominal's nominative forms are identical to the corresponding accusative forms. The paradigm of *manas-* `mind' in Table 3 illustrates. In inferential-realizational theories of morphology, such instances of syncretism are effected by rules of referral--rules that cause the realization of one morphosyntactic property set to mimic that of a distinct property set. In the DATR definition of the *manas-* paradigm in Table 3, these syncretisms are accounted for by four facts situated at the node from which all nominals inherit by default (specifically, by the facts `F1` through `F4` situated at the `NOMINAL` node of (9)).

	Singular	Dual	Plural
Nominative, Vocative, Accusative	*manas*	*manas-ī*	*manāṃs-i*
Instrumental	*manas-ā*	*mano-bhyām*	*mano-bhis*
Dative	*manas-e*		*mano-bhyas*
Ablative	*manas-as*		
Genitive		*manas-os*	*manas-ām*
Locative	*manas-i*		*manas-su*

Table 3. Declensional paradigm of the Sanskrit noun *manas-* `mind'

(9) A DATR theory defining the Sanskrit paradigm in Table 3 (without sandhi)
```
#atom M.
#vars $case: nom voc acc ins dat abl gen loc.
#vars $direct: nom voc acc.
#vars $number: sg du pl.
NOMINAL:
        <> == "<stem>" "<suff>"
        <stem> == "<vstem>"
        <vstem> == "<root>" "<stemvowel>"
        <longvstem> == "<vstem>" ;
        <stemobs> ==
        <suff nom sg> == s
        <suff acc sg neuter> ==
        <suff acc sg> == m
        <suff dat sg> == e
        <suff ins sg> == a a
        <suff gen sg> == a s
        <suff loc sg> == i
        <suff $direct du> == i i
        <suff gen du> == o s
        <suff ins du> == bh y a a m
        <suff acc pl neuter> == i
        <suff dat pl> == bh y a s
        <suff ins pl> == bh i s
        <suff gen pl> == a a m
        <suff loc pl> == s u
        <abl> == "<dat>"
        <voc> == "<nom>"
        <abl sg> == "<gen sg>"                            %fact F1
        <loc du> == "<gen du>"                            %fact F2
        <dat du> == "<ins du>"                            %fact F3
        <nom $number neuter> == "<acc $number neuter>".   %fact F4
CONSONANT_STEM:
        <> == NOMINAL
        <stem> == "<vstem>" "<stemobs>"
        <stem $direct pl neuter> == "<longvstem>" M "<stemobs>".
Manas:
        <> == CONSONANT_STEM
        <$case $number> == CONSONANT_STEM:<$case $number neuter>
        <root> == m a n
        <stemvowel> == a
        <stemobs> == s.
#hide NOMINAL CONSONANT_STEM.
#show
        <nom sg>
        <acc sg>
        <voc sg>
        …
```

Clearly there is a redundancy in the formulation of the facts `F1` through `F4`: In each of these, both the LHS and the RHS must be specified for the same number property, and in `F4`, both are additionally specified for neuter gender. The source of this redundancy is the subtractive quality of DATR facts. In a redundancy-free formulation of these Sanskrit rules of referral, a fact's RHS would instead be interpreted nonsubtractively, that is, as sharing all of the properties of the fact's LHS by default (in the absence of any contrary stipulation).

4 An extension of DATR: KATR

In order to facilitate the formal representation of the morphological phenomena exemplified by the foregoing evidence from Swahili and Sanskrit, we propose an extension of the DATR language; we refer to this extension as KATR. In this section, we discuss three novel characteristics of KATR: that of allowing a fact's LHS to be formulated either as a path or as a set; that of allowing regular expressions to figure in the representation of a fact's LHS; and that of allowing facts to be defined in either a subtractive or a nonsubtractive way. As we show, these three characteristics afford a simple account of each of the problematic morphological phenomena identified in Section 3.

4.1 Sets vs paths in KATR

Syntactically, KATR differs very little from DATR: A query is always an atom-path, and the RHS of a fact is always a sequence of zero or more terms. The fundamental differences between DATR and KATR relate to the formulation of a fact's LHS and to query evaluation. The first fundamental difference between KATR and DATR is that while the LHS of a fact must be an atom-path in DATR, it may be either an atom-path or an atom-set in KATR (where an **ATOM-SET** is a sequence of atoms enclosed in curly brackets). In view of this difference, the matching of facts to queries proceeds somewhat differently in KATR and in DATR.

4.1.1 Matching facts to queries in KATR

When *atom_path_f* is the LHS of a fact *f* at *Node_q*, we say that *f* is a **POTENTIAL MATCH FOR** *atom_path_q* **AT** *Node_q* iff *atom_path_f* is a prefix of *atom_path_q*. Similarly, when *atom_set_f* is the LHS of a fact *f* at *Node_q*, we say that *f* is a **POTENTIAL MATCH FOR** *atom_path_q* **AT** *Node_q* iff each member of *atom_set_f* is a distinct member of *atom_path_q*. Now, the matching of facts to queries in KATR can be precisely characterized as follows: Where X is the LHS of a fact *f* at *Node_q*, *f* **MATCHES** *atom_path_q* **AT** *Node_q* iff (a) *f* is a potential match for *atom_path_q* at *Node_q* and (b) there is no fact *f'* at *Node_q* such that (i) *f'* is a potential match for *atom_path_q* at *Node_q* and (ii) the LHS of *f'* is greater in cardinality than X.

4.1.2 Evaluating queries in KATR

Suppose now that at *Node_q*, a query path *atom_path_q* is matched by the fact in (10a), whose RHS X is a sequence of zero or more terms; in that case, the value for the query *Node_q***:***atom_path_q* is computed exactly as in DATR. Suppose, on the other hand, that at *Node_q*, a query path *atom_path_q* is matched by the fact in (10b).

(10) a. *atom_path_f* `==` X
b. *atom_set_f* `==` *term_1 ... term_n*

In that case, the value computed for the query *Node_q***:***atom_path_q* is the sequence of the values of *term_1 ... term_n* relative to the query path *atom_path_q* and the LHS *atom_set_f* at *Node_q*. The value of each *term_i* ($1 \leq i \leq n$) is determined as in Table 4. Rows (A), (B), and (F) of Tables 1 and 4 are alike. The only significant difference between Tables 1 and 4 involves those instances in which the fact matching the query path has a RHS that includes a term-path (i.e. rows (C), (D), (E), and (G)). In those instances, the evaluation of the query depends on a new notion of path reduction. If *atom_set_f* is the LHS of a fact that matches *atom_path_q* at some node, the **PATH REDUCTION** of *atom_path_q* relative to *atom_set_f* is that atom-path P that is like *atom_path_q* except that the leftmost instances of *atom_set_f*s members appearing in *atom_path_q* are absent from P. For instance, if the atom-set ***`{a b c}`*** is the LHS of a fact matching the atom-path ***`<c a e b d a>`*** at some node, the path reduction of ***`<c a e b d a>`*** relative to ***`{a b c}`*** is the atom-path ***`<e d a>`***.[2]

Given a query *Node_q:atom_path_q* such that *atom_path_q* is matched by the fact in (10b) at *Node_q*:	
if *term_i* is of type	then the value of *term_i* relative to the query path *atom_path_q* and the LHS *atom_set_f* at *Node_q* is
(A) *atom*	*term_i*
(B) *Node_y*	the value computed for the query *Node_y:atom_path_q*
(C) <*term_1 ... term_n*>	the value computed for the query *Node_q*:<X Y>, where (i) X is the result of concatenating the values of *term_1 ... term_n* relative to the query path *atom_path_q* and the LHS *atom_set_f* at *Node_q*, and (ii) <Y> is the (possibly empty) path-reduction of *atom_path_q* relative to *atom_set_f.*
(D) *Node_y*:<*term_1 ... term_n*>	the value computed for the query *Node_y*:<X Y>, where X and Y are as above
(E) "<*term_1 ... term_n*>"	the value computed for the new query *Node_INIT*:<X Y>, where *Node_INIT* is the initial query node and X and Y are as above
(F) "*Node_y*"	the value computed for the new query *Node_y*:<Z>, where <Z> is the initial query path
(G) "*Node_y*:<*term_1 ... term_n*>"	the value computed for the new query *Node_y*:<X Y>, where X and Y are as above

Table 4. Terms and their values in KATR

Given this notion of path reduction, consider the KATR theory in (11).

(11)
```
A:
        {c a b} == <f g h>
        {d f h g a} == i
        {i j k} == B:<a b c>
        {l} == B
        {m n} == o
        {p} == "B:<q>".
B:
        {c a b} == m
        {c a l b} == n
        {l} == "<m n o>"
        {m} == C.
C:
        {} == "<a b c>"
        {s a b} == t
        {u a b} == v.
```

According to Table 4(C), the value computed for the query `A:<c a e b d a>` in theory (11) is the value computed for the query `A:<f g h e d a>`, namely `i` (in accordance with Table 4(A)). According to Table 4(D), the value computed for the query `A:<l k j i>` in this theory is the value computed for the query `B:<a b c l>`, namely `n`.

Suppose that node A is the initial query node in the evaluation of a query `A:<l>` in this theory. According to Table 4(B), the value of this query is the value computed for the query `B:<l>`; according to Table 4(E), this latter value is in turn the result of initiating a new query `A:<m n o>`. Thus, the value computed for the query `A:<l>` in this theory is `o`.

Suppose finally that a query `A:<m p>` is initiated at node `A` in this theory. According to Table 4(G), the value computed for this query is the result of initiating a new query `B:<q m>`; thus, the final value computed for `A:<m p>` is `m`.

Two facts situated at the same node can, in principle, match a query path equally well. For instance, what would be the value computed for the query `C:<a b s u>` in this theory? The problem is that there are two facts at node `C` that match the query path `<a b s u>` equally well. This ambiguity constitutes a flaw in this theory. Our KATR implementation therefore issues a warning when it computes this query, although it returns an answer, namely `v`: When two facts are overtly ambiguous, the second of these facts overrides the first. (In this respect, KATR follows DATR.)

4.1.3 An example of unordered property sets in KATR: Swahili verb inflection

Because KATR allows facts to mention morphosyntactic properties without having to adhere to a canonical sequence, the DATR theory in (7) can be expressed more simply as the KATR theory in (12).

(12) A KATR theory defining the Swahili paradigm in Table 2

[Identical to (7) except for the following substitutions at the `VERB` node.]

```
VERB:
     {slot_iv negative 1 sg} == s i   %replaces fact F1
     {slot_ii 1 sg} == n i            %replaces fact F2
     {slot_ii 2 sg} == u              %replaces fact F3
     {slot_ii 3 sg} == a              %replaces fact F4
     {slot_ii 1 pl} == t u            %replaces fact F5
     {slot_ii 2 pl} == m              %replaces fact F6
     {slot_ii 3 pl} == w a            %replaces fact F7
     {slot_i future} == t a           %replaces fact F8
```

The analysis in (12) resolves the problem embodied in (7). In place of the highly redundant facts `F1-F8` situated at the `VERB` node in (7), the theory in (12) involves redundancy-free facts. The output of the theory in (12) is nevertheless identical to that of the theory in (7).

One reason why KATR retains the option of allowing a fact's LHS to be an atom-path (instead of simply requiring that every LHS be an atom-set) is upward compatibility: Any well-formed theory expressed in DATR remains a well-formed theory (though not necessarily the optimal theory) in KATR; thus, KATR is literally an extension of DATR. A second reason is that certain facts presuppose a linear ordering of the atoms constituting their LHS. A case in point is any fact describing sandhi phenomena; for instance, the fact `<a u> == u <>` at the `SANDHI` node in the theory of (7) makes essential reference to the linear ordering of two adjacent vowels.

4.2 Nonlocal sandhi and regular expressions

The second fundamental difference between KATR and DATR is motivated by the need to accommodate instances of nonlocal sandhi. Unlike DATR, KATR allows the LHS of a fact to be stated as an atom-path incorporating one or more regular expressions. A regular expression is a pattern that describes a set of strings; regular expressions are constructed analogously to arithmetic expressions by using various operators to build larger patterns out of smaller ones.

4.2.1 Regular-expression operators and their usage

Table 5 shows the (restricted) list of regular-expression operators implemented in KATR. Atoms or variables can join with regular-expression operators in the specification of a regular expression. Regular expressions are only allowed in an atom-path on a fact's LHS; that is, regular expressions are disallowed both in an atom-set on a fact's LHS and in queries. The KATR theory in (13a) and its output in (13b) illustrate the usage of regular expressions in KATR; in (13), ***$c*** is a variable whose only permitted value is the atom ***c***.

Regular Expression	Description
`*`	The preceding item is matched zero or more times.
`+`	The preceding item is matched one or more times.
`?`	The preceding item is optional and matched at most once.
`(n, )`	The preceding item is matched *n* or more times.
`( ,m)`	The preceding item is matched at most *m* times.
`(n, m)`	The preceding item is matched at least *n* times, but not more than *m* times.

Table 5. Regular-expression operators in KATR

(13) a.
```
#vars $c: c.
Node:
      <> == does not match
      <a b* $c(2, )> = matches.
%a followed by zero or more occurrences of b followed by two
%or more occurrences of c.
```
b.
```
Node:<a c c> ==> matches
Node:<a b c c> ==> matches
Node:<a b b c c c> ==> matches
Node:<a c> ==> does not match
```

4.2.2 Strings of atoms that match regular expressions

As KATR evaluates a given query, it captures strings of atoms that match a regular expression and binds them to an identifier sharing the name of that regular expression. The KATR theory in (14a) and its sample output in (14b) illustrate this kind of binding. In (14), fact `F1` matches the query path `<a b b b c>`; in particular, the regular expression `b*` in the atom-path in fact `F1`'s LHS matches the string `b b b` present in the query path. This string is bound to a newly-introduced identifier `b*`; thus, while the expression `b*` in the atom-path on fact `F1`'s LHS is a regular expression, its namesake `b*` in the atom-path on `F1`'s RHS is an identifier. Regular expressions are, again, only allowed in an atom-path in a fact's LHS, and any identifier named for a regular expression appearing in some fact has its scope limited to the RHS of that same fact.

(14) a.

```
Node:
        <a b* c> == <c b* d>          %fact F1
        <c b* d> == hello.            %fact F2
```

b.

```
Node:<a b b b c> == hello.
```

As it is formulated, fact `F2` causes the theory in (14a) to produce the same result `hello` for any of the following queries: `Node:<a c>`, `Node:<a b c>`, `Node:<a b b c>`. When these same queries are addressed to the sample theory in (15), only the first two produce output.

(15)

```
Node:
        <a b* c> == <c b* d>
        <c d> == no b
        <c b d> == one b.
```

Identifiers associated with regular expressions can be present in any order and any number of times, as illustrated by theory (16a) and its sample output (16b). In (16), for query `Q1`, fact `F1`'s RHS expands to `<b b e b b d d>`, since the identifier `b(2,2)` refers to the string `b b` that matches the regular expression `b(2,2)`, and the identifier `d(2,3)` refers to the string `d d d` that matches the regular expression `d(2,3)`.

(16) a.

```
Node:
<a b(2,2) c d(2,3)> == <b(2,2) e b(2,2) d(2,3)>  %fact F1
<b b e b b d d> == 2 d                            %fact F2
<b b e b b d d d> == 3 d.                         %fact F3
```

b.

```
Node:<a b b c d d> ==> 2 d                        %query Q1
Node:<a b b c d d d> ==> 3 d                      %query Q2
```

4.2.3 Handling identical regular expressions

Because the string of atoms that matches a given regular expression is bound to an identifier named after that regular expression, the presence of two identical regular expressions leads to a name-resolution conflict. Example (17) shows a KATR theory involving a name-resolution conflict owing to the presence of two instances of `b*` in the path on `F1`'s LHS. Example (18) is the corrected version of example (17).

(17)
```
Node:
        <a b* c b*> == b*.                    %fact F1
```

(18)
```
Node:
        <a b*#name1 c b*#name2> == b*#name1.  %fact F1
```

In (18), the two instances of `b*` are made unique by renaming. Strings matched by the first regular expression `b*` are bound to the newly-introduced identifier `b*#name1`, while strings matched by the second regular expression `b*` are bound to the newly-introduced identifier `b*#name2`. The names `name1`, `name2` can be arbitrary so long as they are distinct.

The theory in (19a) and its sample output (19b) demonstrate renaming. When `F1` matches `Q1` in example (19), the identifiers `b+#1` and `b+#2` are mapped to the contents `b` and `b b`, respectively.

(19) a.
```
Node:
        <a b+#1 c b+#2> == <c b+#2 a b+#1>          %fact F1
        <c b b b a>  == long                        %fact F2
        <c b b a> == short.                         %fact F3
```

b.
```
Node:<a b c b b> ==> short                          %query Q1
Node:<a b c b b b> ==> long                         %query Q2
```

4.2.4 Handling similar variables

Every instance of a given named variable in some path or set is interpreted as matching the same atom. Theory (20a) and its sample output (20b) illustrate. All three instances of `$abc#1` in the path of `F1` are required to match the same atom. Fact `F1` matches queries `Q1` and `Q2`, but only fact `F5` matches queries `Q3` and `Q4`. For example, when query `Q3` is given to the program in example (20), at `F1`, `$abc#1` matches `a` in `Q3`.

(20) a.
```
#vars $abc: a b c.
Node:
        <$abc#1 x $abc#1 y $abc#1> == <$abc#1 z>    %fact F1
        <a z> == a                                  %fact F2
        <b z> == b                                  %fact F3
        <c z> == c                                  %fact F4
        <>  == does not match.                      %fact F5
```

b.
```
Node:<a x a y a> ==> a                              %query Q1
Node:<b x b y b> ==> b                              %query Q2
Node:<a x a y b> ==> does not match                 %query Q3
Node:<a x b y c> ==> does not match                 %query Q4
```

The next three elements (atoms `x`, `a`, `y`) in the path on `F1`'s LHS also successfully match the next elements in query `Q3`. The match fails when the next element (`$abc#1`) in the path on `F1`'s LHS tries to match atom `b` in query `Q3`, since it is expecting the atom `a`.

By renaming the three instances of `$abc` to separate them, we can have them match different atoms. Theory (21a) modifies theory (20a) to produce the new results in (21b).

(21) a. [Identical to (20a) except for the following substitution at the `Node` node.]

```
Node:
<$abc#1 x $abc#2 y $abc#3> == <$abc#1 z>    %replaces fact F1
```

b.

```
Node:<a x a y a> ==> a
Node:<b x b y b> ==> b
Node:<a x a y b> ==> a
Node:<a x b y c> ==> a
```

To allow for backward compatibility, the program listed in example (19) works in KATR even without naming the instances of `$abc`, but a warning message is displayed during compilation, alerting the user to possible ambiguity in variable usage.

There are some limitations on the use of regular expressions in KATR: (a) The regular-expression operator `*` doesn't fully implement Kleene's closure (thus, the query `<a b b b c>` would not match a path `<a b* b c>`, because the regular expression `b*` absorbs all instances of `b` present in the query; the query would match if the principles of Kleene's closure were strictly adhered to); (b) the regular-expression operator `{n, }` has its upper bound set at 10,000 (representing infinity); (c) nested regular expressions (such as `(b* c)+`) are disallowed in KATR; and (d) since regular expressions presume a linear ordering that sets do not possess, regular expressions are not allowed for sets appearing on a fact's LHS.

4.2.5 An example of regular expressions in KATR: Sanskrit n-retroflexion

The following KATR theory expresses the Sanskrit principle of *n*-retroflexion given in Section 3.2 above. We code Sanskrit phonological segments using the ITRANS coding scheme; thus, `N`, `sh`, and `Sh` represent ṇ, ś, and ṣ, respectively. The regular expression `$non_cor_cons*` designates a string of zero or more instances of the variable `$non_cor_cons`.

(22) a. A KATR theory of the Sanskrit principle of *n*-retroflexion

```
%Declaration of atoms beginning with capital letters:
#atom Ri RI Li LI M H ~N Ch ~n T Th D Dh N Sh. %etc.
#vars $abc: Ri RI Li LI M H ~N Ch ~n T Th D Dh N Sh a ch d dh
     e g i j jh k m n o r s sh t th u v y.  %etc.
#vars $cor_cons: ch Ch j jh ~n T Th D Dh N t th d dh n sh Sh
     s.  %coronal consonants
#vars $non_cor_cons: $abc - $cor_cons.
#vars $retr_cont: r Ri RI Sh.  %retroflex continuants
#vars $vowel_nasal_glide: a e i o u Ri n m y v.
SANDHI:
  <$abc> == $abc <>
  <> ==
  <$retr_cont $non_cor_cons* n $vowel_nasal_glide>
    == <$retr_cont $non_cor_cons* N $vowel_nasal_glide>.
```

b. Output for sample queries

```
SANDHI:< m u Sh n a a t i > ==> m u Sh N a a t i
SANDHI:< k a r m a n a a > ==> k a r m a N a a
SANDHI:< r a th e n a > ==> r a th e n a
SANDHI:< sh u sh r u u Sh a n a > ==> sh u sh r u u Sh a N a
SANDHI:< s r a v a n a > ==> s r a v a N a
SANDHI:< d a r sh a n a > ==> d a r sh a n a
SANDHI:< g r a s a n a > ==> g r a s a n a
```

4.3 Nonsubtractive rules in KATR

The third fundamental difference between KATR and DATR concerns the form and interpretation of facts having term-paths on their RHS. Consider again Table 4(C). According to this property, the value computed for the query `Node:<a b c e>` in the KATR theory (23) is `this` rather than `that`.

(23)
```
Node:
        {a b c} == <d>          %fact F1
        {d} == this
        {a b c d} == that.
```

In KATR as in DATR, the `==` operator has a subtractive quality, in the sense that it causes the atoms `a`, `b`, and `c` constituting the LHS of fact `F1` in (23) to be eliminated from the query path prescribed for subsequent evaluation. This subtractive quality is appropriate for the expression of many morphological phenomena, but not for all, as we showed in Section 3.3. In addition to the double equal sign `==`, whose interpretation involves the subtractive quality exemplified in (23), KATR introduces the `=+=` sign, whose interpretation does not involve this quality. Accordingly, the value computed for the query `Node:<a b c e>` in the KATR theory in (24) is `that` rather than `this`.

(24)
```
Node:
        {a b c} =+= <d>          %fact F1
        {d} == this
        {a b c d} == that.
```

The nonsubtractive fact `F1` in (24) causes the entire query path to be included in the path used for subsequent evaluation. KATR also makes it possible to formulate nonsubtractive facts that cause a modified form of the query path to be included in the path used for subsequent evaluation. Substitution is accomplished by means of the slash operator exemplified in the KATR theory in (25); this operator is defined in such a way that the value computed for the query `Node:<a b d>` in the KATR theory (25) is `the other` rather than `this` or `that`.

(25)
```
Node:
        {a b/c d} =+= <e>
        {e} == this
        {a b d e} == that
        {a c d e} == the other.
```

Thus, in general, the slash operator makes it possible to evaluate a path *P* as some distinct path *P'* that is derived from *P* by means of particular substitutions. In order to define the evaluation of a fact whose LHS includes the slash operator, we must define a number of ancillary notions.

4.3.1 Defining the **=+=** operator

The expression *atom_1/atom_2* is a **SLASH-ATOM** if *atom_1* and *atom_2* are atoms. For any *atom_1*, *atom_2*, the slash-atom *atom_1/atom_2* is an **L-INSTANCE** of *atom_1*. A **SLASH-PATH** is a sequence (enclosed in angle brackets) that contains an instance of at least one slash-atom and whose other members (if any) are instances of atoms or slash-atoms. A **SLASH-SET** is a sequence (enclosed in curly brackets) that contains an instance of at least one slash-atom and whose other members (if any) are instances of atoms or slash-atoms. The members of a slash-path, like those of an ordinary atom-path, are linearly ordered. The members of a slash-set S are partially ordered; in particular, they are unordered except to the extent that all L-instances of a given atom in S are linearly ordered with respect to each other (though not with respect to L-instances of other atoms).

We define a relation of **L-MATCHING** as follows: (a) for any *atom_1*, *atom_2*, both *atom_1* and *atom_1/atom_2* L-match *atom_1*; (b) an *n*-member slash-path SP L-matches *atom_path_q* iff the *i*th member of SP L-matches the *i*th member of *atom_path_q* ($1 \leq i \leq n$); and (c) a slash-set SS L-matches *atom_path_q* iff every member of SS L-matches a distinct member of *atom_path_q*.

If the LHS of fact *f* at *Node_q* is a slash-path or slash-set S, then *f* is a **POTENTIAL MATCH FOR** *atom_path_q* at *Node_q* iff S L-matches *atom_path_q*. As before, where X is the LHS of a fact *f* at *Node_q*, *f* **MATCHES** *atom_path_q* **AT** *Node_q* iff (a) *f* is a potential match for *atom_path_q* at *Node_q* and (b) there is no fact *f'* at *Node_q* such that (i) *f'* is a potential match for *atom_path_q* at *Node_q* and (ii) the LHS of *f'* is greater in cardinality than X.

Given a fact *f* which matches *atom_path_q*, the **SLASH-ALTERNANT** SA of *atom_path_q* relative to *f* is that path that is like *atom_path_q* except that if *atom_1/atom_2* is the *n*th L-instance of *atom_1* in *f* 's LHS and the *n*th instance of *atom_1* in *atom_path_q* is the *x*th member of *atom_path_q*, then *atom_2* is the *x*th member of SA. If the LHS of fact *f* is an atom-path or an atom-set (and not a slash-path or slash-set), the slash-alternant of *atom_path_q* relative to *f* is *atom_path_q* itself.

Thus, suppose that at *Node_q*, a query path *atom_path_q* is matched by fact (26), whose LHS X is an atom-path, an atom-set, a slash-path, or a slash-set and whose RHS is *term_a_1 ... term_a_m*. In that case, the value computed for the query *Node_q:atom_path_q* is the sequence of the values of *term_a_1 ... term_a_m* relative to the query path *atom_path_q* and the LHS X at *Node_q*. The value of each *term_a_i* ($1 \leq i \leq m$) is determined as in Table 6. The evaluation principles in Table 6 are exemplified by the KATR theory in (27).

(26) X **=+=** *term_a_1 ... term_a_m*

Given a query *Node_q:atom_path_q* such that *atom_path_q* is matched by fact (26) at *Node_q*:	
if *term_a_i* is	then the value of *term_a_i* relative to the query path *atom_path_q* and the LHS X at *Node_q* is
(A) *atom_1*	*term_a_i*
(B) *Node_y*	the value computed for the query *Node_y:atom_path_q*
(C) *<term_b_1 ... term_b_n>*	the value computed for the query *Node_q:*<Y Z>, where (i) Y is the result of concatenating the values of *term_b_1 ... term_b_n* relative to the query path *atom_path_q* and the LHS X at *Node_q*, and (ii) <Z> is the slash-alternant of *atom_path_q* relative to (26)
(D) *Node_y:<term_b_1 ... term_b_n>*	the value computed for the query *Node_y:*<Y Z>, where Y and Z are as above
(E) *"<term_b_1 ... term_b_n>"*	the result of initiating a new query *Node_INIT:*<Y Z>, where *Node_INIT* is the initial query node and Y and Z are as above
(F) *"Node_y"*	the result of initiating a new query *Node_y:*<W>, where <W> is the initial query path
(G) *"Node_y:<term_b_1 ... term_b_n>"*	the result of initiating a new query *Node_y:*<Y Z>, where Y and Z are as above

Table 6. Terms and their values in KATR

(27)

```
A:
      <a> =+= b
      <a b/c d> =+= e
      {b} =+= c
      <e> =+= <f g h>
      <e f/i f> =+= <f g h>
      {e g/i g} =+= <f g h>
      {e h/i f h g h/k} =+= B:<n o p>
      <f g h e> =+= k
      <f g h e i f> =+= m
      <f g h e i g> =+= n
      {f f/g} =+= B
      {h h h} =+= "B"
      {h i f g} =+= l
      {i} =+= <f g h>
      {i h/a g} =+= f
      <m> =+= B:<n o p>
      <q> =+= B
      <s> =+= a
      {s f h} =+= q
      <t> =+= "B"
      <u> =+= "B:<v>"
      <v> =+= "B:<w>"
      {w} =+= d.
```

```
B:
        <a> =+= A
        <a b/c d> =+= A
        {b} =+= A
        <f f/h> =+= "<s>"
        {h h h} == C
        {h q s t} == i
        {i h/a g} =+= A
        {m o p} == w
        <n o p e f g i k h> == o
        <q> == "<s>"
        <s> == q
        <t> == C
        <u> == x
        <v> == C
        {w v} == C.
C:
        {h h/q h/t} =+= "<s>"
        <t> == "<s>"
        <v> == <t>
        <w v> == "<u>" "A"
        <v w> == "A" "<u>".
```

In accordance with Table 6(A), the values computed for the queries `A:<a>`, `A:<b>`, `A:<a b d>`, and `A:<g h i>` in the theory of (27) are `b`, `c`, `e`, and `f`, respectively. In accordance with Table 6(A, B), the values computed for the queries `B:<a>`, `B:<b>`, `B:<a b d>`, and `B:<g h i>` are `b`, `c`, `e`, and `f`, respectively. In accordance with Table 6(A, C), the values computed for the queries `A:<e>`, `A:<i>` , `A:<e f f>`, and `A:<e g g>` are `k`, `l`, `m`, and `n`, respectively. In accordance with Table 6(A, D), the values computed for the queries `A:<m>` and `A:<e f g h h h>` are `w` and `o`, respectively. In accordance with Table 6(A, B, E), the values computed for the queries `A:<q>` and `A:<f f>` are `a` and `q`, respectively. In accordance with Table 6(A, B, E, F), the values computed for the queries `A:<t>` and `A:<h h h>` are `q` and `i`, respectively. In accordance with Table 6(A, B, C, E, G), the values computed for the queries `A:<u>` and `A:<v>` are `q` and `x d`, respectively.

4.3.2 An example of nonsubtractive rules in KATR: Sanskrit declensional syncretism

The option of using nonsubtractive rules in KATR makes it possible to streamline the analysis of Sanskrit declensional syncretism given in (9): The subtractive rules `F1-F4` at the `NOMINAL` node in (9) can now be replaced with the nonsubtractive rules in (28), eliminating the redundancy inherent in the former rules.

(28) A KATR theory defining the Sanskrit paradigm in Table 3

[Identical to (9) except for the following substitutions at the `NOMINAL` node.]

```
NOMINAL:
        <abl/gen sg> =+= "<>"                   %replaces fact F1
        <loc/gen du> =+= "<>"                   %replaces fact F2
        <dat/ins du> =+= "<>"                %replaces fact F3
        {nom/acc neuter} =+= "<>"            %replaces fact F4
```

5 An empirical challenge: Rules applying in “expanded mode”

As noted in Section 4.2, two facts situated at the same node in a KATR theory may match a query path equally well; a theory in which this happens is therefore flawed, and our implementation of KATR issues a warning to that effect. Nevertheless, the inflectional systems of natural languages sometimes seem to involve competition between rules that are equally narrow in their specification. Georgian presents a case in point: A transitive verb may exhibit agreement with both its subject and its direct object; the particular system of subject- and object- agreement markings that a transitive verb exhibits varies according to the conjugation class to which the verb belongs and to the specific temporal, modal, and aspectual properties for which it is inflected--see Stump (2001, Chapter 3) for additional details and references. At issue here is the default pattern of affixal agreement in Table 7, which is exhibited by verbs in any but the fourth conjugation in most temporal/modal/aspectual contexts. For instance, the first-conjugation verb *mo-ḳlav* 'kill' has the future-tense paradigm in Table 8; thus, the form *mo-gv-ḳlav-en* consists of the "preverb" *mo-*, the first-person plural object-agreement prefix *gv-*, the verb root *ḳlav*, and the third-person plural subject-agreement suffix *-en*, and hence means `they will kill us'. The forms in Table 8 can seemingly be generated by the KATR program in (29); thus, the query `Kill:<3PerSubj sgSubj 3PerObj plObj>` produces the value `m o K l a v s`, and so on.

	Subject-agreement affixes		Object-agreement affixes	
	Singular	Plural	Singular	Plural
1st Person	*v-*	*v-...-*t_1	*m-*	*gv-*
2nd Person	none	*-*t_1	*g-*	*g-...(-*t_2*)*[*]
3rd Person	*-s*	*-en*	none	none
*-t_2 appears only in the presence of singular subject agreement.				

Table 7. Default subject- and object-agreement affixes in Georgian

SUBJECT:		1SG	1PL	2SG	2PL	3SG	3PL
OBJECT	1SG			*mo-m-ḳlav*	*mo-m-ḳlav-*t_1	*mo-m-ḳlav-s*	*mo-m-ḳlav-en*
	2SG	*mo-g-ḳlav*	*mo-g-ḳlav-*t_1			*mo-g-ḳlav-s*	*mo-g-ḳlav-en*
	3SG	*mo-v-ḳlav*	*mo-v-ḳlav-*t_1	*mo-ḳlav*	*mo-ḳlav-*t_1	*mo-ḳlav-s*	*mo-ḳlav-en*
	1PL			*mo-gv-ḳlav*	*mo-gv-ḳlav-*t_1	*mo-gv-ḳlav-s*	*mo-gv-ḳlav-en*
	2PL	*mo-g-ḳlav-*t_2	*mo-g-ḳlav-*t_1			*mo-g-ḳlav-*t_2	*mo-g-ḳlav-en*
	3PL	*mo-v-ḳlav*	*mo-v-ḳlav-*t_1	*mo-ḳlav*	*mo-ḳlav-*t_1	*mo-ḳlav-s*	*mo-ḳlav-en*

Table 8. Future-tense paradigm of Georgian *mo-ḳlav* 'kill'

(29) A KATR theory generating the Georgian paradigm in Table 8

```
#atom K.
VERB:
      {} == "<preverb>" <prefix> "<root>" <suffix>
      {prefix 1PerSubj} == v                           %fact F1
      {prefix 1PerObj} == m                            %fact F3
      {prefix 1PerObj plObj} == g v
      {prefix 2PerObj} == g                            %fact F2
```

```
        {suffix 3PerSubj sgSubj} == s
        {suffix 3PerSubj plSubj} == e n
        {suffix plSubj} == t_1
        {suffix sgSubj 2PerObj plObj} == t_2
        {prefix} ==
        {suffix} ==.
Kill:
        {} == VERB
        {root} == K l a v
        {preverb} == m o.
```

There is, however, a problem with this theory. When values are computed for the queries in Table 9, competition arises between facts `F1` and `F2` at the `VERB` node; that is, when theory (29) is queried for a verb form realizing first-person subject agreement and second-person object agreement, the appearance of the first-person subject prefix `v` (dictated by `F1`) is incompatible with that of the second-person object prefix `g` (dictated by `F2`). Both facts match the queries in Table 9, and their lefthand sides have the same cardinality. Consequently, our KATR implementation issues a warning (`More than one maximal match for node VERB with local query < prefix 1PerSubj sgSubj 2PerObj sgObj >`), and the values returned for the queries in Table 9 are determined by whichever of the two competing facts happens to be ordered last in theory (29):[3]

Query	Corresponding value in theory (29)	
	if `F1` precedes `F2`	if `F2` precedes `F1`
`Kill:<1PerSubj sgSubj 2PerObj sgObj>`	`m o g K l a v`	`m o v K l a v`
`Kill:<1PerSubj sgSubj 2PerObj plObj>`	`m o g K l a v t_2`	`m o v K l a v t_2`
`Kill:<1PerSubj plSubj 2PerObj sgObj>`	`m o g K l a v t_1`	`m o v K l a v t_1`
`Kill:<1PerSubj plSubj 2PerObj plObj>`	`m o g K l a v t_1`	`m o v K l a v t_1`

Table 9. Queries producing alternative values according to the relative ordering of `F1` and `F2` in (29)

From a theoretical standpoint, this sensitivity to the linear ordering of a node's facts is problematic. Most instances of competition among inflectional rules are resolved by Pāṇini's principle, without reference to language-specific relations of linear rule ordering: When two rules are in competition, the more narrowly specified rule wins. Considerations of theoretical parsimony would therefore favor the assumption that parochial relations of linear ordering are in principle irrelevant to the resolution of rule competition. How can Pāṇini's principle be relied upon to resolve the competition in the case at hand?

5.1 The +n and ++ notations

Inspection of the forms in Table 8 reveals that fact `F2` should determine the evaluation of the four queries in Table 9; that is, contrary to its formulation in (29), `F2` acts as if it were a narrower stipulation than fact `F1`. Stump (2001) argues that the Georgian rule introducing the inflectional prefix `g`- applies in "expanded mode": The application of this rule doesn't simply realize the property of second-person object agreement; instead, it realizes every well-formed extension of that property set. This argument implies that rule `F2` is more narrowly specified than every rule with which it enters into competition; Pāṇini's principle therefore correctly predicts that it should override any such rule.

In order to implement this notion of rules applying in expanded mode, we introduce the following new notation in KATR: where {X} and <X> each have cardinality *m*, {X +n} and <X +n> each have cardinality *m*+*n*, and {X ++} and <X ++> each have indefinitely great cardinality. Thus, in the context of the KATR theory in (30), the queries `A:<a b c d>`, `A:<a b c>`, `A:<a b>`, and `A:<a>` all yield `e` as their value, and the queries `A:<b a c d>` and `A:<b>` have the respective values `e` and `f`.

(30)
```
A:
    {a b c} == d
    {a ++} == e
    <b +3> == f.
```

5.2 *An example of a rule applying in "expanded mode": Georgian verb agreement*

Using this new notation, we replace fact `F2` situated at the `VERB` node in theory (29) with (31):

(31)
```
{prefix 2PerObj ++} == g   %replaces fact F2
```

Although the set `{prefix 2PerObj}` has cardinality 2, `{prefix 2PerObj ++}` is a set with indefinitely great cardinality whose only stipulated members are `prefix` and `2PerObj`. By virtue of the indefinitely great cardinality of its LHS, the revised formulation of fact `F2` in (31) overrides `F1` in matching the queries in Table 9, regardless of the order in which the two facts are given. Accordingly, once (31) is substituted into the theory in (29), our implementation of KATR no longer issues any warning, since for any query for which facts `F1` and `F2` are potential matches, `F2` is necessarily the better match.

6 Generative capacity

Notwithstanding the extensions that it incorporates, KATR is no more powerful than DATR. We derive this result from the fact that DATR itself is capable of emulating a Turing machine, so it can compute any partial recursive function, so it is quite powerful. The KATR enhancements cannot increase its power; they only provide a convenient way to express morphological rules that are otherwise clumsy to specify. (This result also suggests that DATR should perhaps be weakened.)

A Turing machine is composed of an infinite tape of 0's and 1's, a movable read/write head, and a finite control. To prove that DATR can emulate a Turing machine, we encode the initial tape as a query, where the alphabet is restricted to the symbols `0`, `1`, and `h` (for the read-write head). Without loss of generality, the tape starts with the symbols "`0 h`" (to avoid writing DATR code to cover the case when the head is at the very beginning). We encode each state in the finite control of the Turing machine as a DATR node. In addition, we provide the following utility DATR nodes:

(32)

```
First: % returns the part to the left of the h (minus the last symbol)
    <0 0> == 0 First:<0>
    <1 0> == 1 First:<0>
    <0 1> == 0 First:<1>
    <1 1> == 1 First:<1>
    <0 h> ==
    <1 h> ==.
Done: % the final node, which just leaves the head where it is.
    <0> == 0 <>
    <1> == 1 <>
    <h> == h <>
    <> =.
```

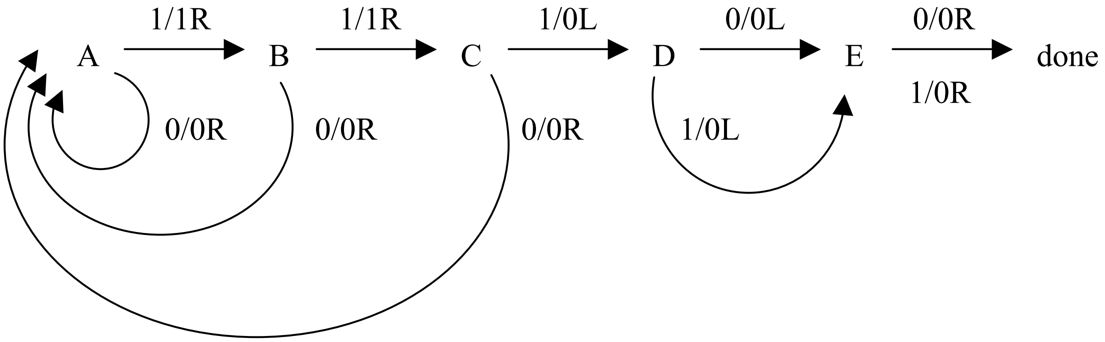
Every encoded state has at least the following three rules:

(33)

```
<0> == <>        % scan to the right looking for the h
<1> == <>        % scan to the right looking for the h
<> == "Done"     % got to the end of the tape; quit
```

In a Turing machine, transitions in the finite-state control at a state *N* are of the form "if the symbol at the head is x, replace it with y and move the head right/left one position, going to state *M*". We notate this transition by labelling an arrow between states *N* and *M* with the label x/yL or x/yR, as shown in Figure 1.

Figure 1. Finite control of a Turing machine



We encode the transition from state *N* to state *M* with two rules in *N*'s DATR node:

(34)

```
<0 h x> == "M:<"First" 0 y h>" % if the direction is right
<1 h x> == "M:<"First" 1 y h>" % if the direction is right
<0 h x> == "M:<"First" h 0 y>" % if the direction is left
<1 h x> == "M:<"First" h 1 y>" % if the direction is left
```

Thus, Figure 1 represents the finite control of a Turing machine that converts the first "1 1 1" to the right of the head to "0 0 0", leaving the head on the second 0 in the replacement. The DATR translation of this sample Turing maching is (35).[4]

(35) Translation of sample Turing machine into DATR

```
First: % returns the part to the left of the h (minus the last symbol)
        <0 0> == 0 First:<0>
        <1 0> == 1 First:<0>
        <0 1> == 0 First:<1>
        <1 1> == 1 First:<1>
        <0 h> ==
        <1 h> == .
Done: % the final node, which just leaves the head where it is.
        <0> == 0 <>
        <1> == 1 <>
        <h> == h <>
        <> =.
A:
        <0> == <>
        <1> == <>
        <> == "Done"
        <0 h 0> == "A:<"First" 0 0 h>" % 0/0RA
        <1 h 0> == "A:<"First" 1 0 h>" % 0/0RA
        <0 h 1> == "B:<"First" 0 1 h>" % 1/1RB
        <1 h 1> == "B:<"First" 1 1 h>". % 1/1RB
B:
        <0> == <>
        <1> == <>
        <> == "Done"
        <0 h 0> == "A:<"First" 0 0 h>" % 0/0RA
        <1 h 0> == "A:<"First" 1 0 h>" % 0/0RA
        <0 h 1> == "C:<"First" 0 1 h>" % 1/1RC
        <1 h 1> == "C:<"First" 1 1 h>". % 1/1RC
C:
        <0> == <>
        <1> == <>
        <> == "Done"
        <0 h 0> == "A:<"First" 0 0 h>" % 0/0RA
        <1 h 0> == "A:<"First" 1 0 h>" % 0/0RA
        <0 h 1> == "D:<"First" h 0 0>" % 1/0LD
        <1 h 1> == "D:<"First" h 1 0>". % 1/0LD
D:
        <0> == <>
        <1> == <>
        <> == "Done"
        <0 h 0> == "E:<"First" h 0 0>" % 0/0LE
        <1 h 0> == "E:<"First" h 1 0>" % 0/0LE
        <0 h 1> == "E:<"First" h 0 0>" % 1/0LE
        <1 h 1> == "E:<"First" h 1 0>". % 1/0LE
E:
        <0> == <>
        <1> == <>
        <> == "Done"
        <0 h 0> == "Done:<"First" 0 0 h>" % 0/0Rdone
        <1 h 0> == "Done:<"First" 1 0 h>" % 0/0Rdone
        <0 h 1> == "Done:<"First" 0 0 h>" % 1/0Rdone
        <1 h 1> == "Done:<"First" 1 0 h>". % 1/0Rdone
#show
        <0 h 0 1 1 1 0 1> % expect 0 0 0 h 0 0 0 1
        <0 h 0 1 1 1 1 1> % expect 0 0 0 h 0 0 1 1
        <0 h 0 0 1 1 0 1>. % expect 0 0 0 1 1 0 1 h
#hide B C D E Done First.
```

7 Summary

KATR incorporates a number of formal features motivated by empirically observable characteristics of natural-language morphology. First, KATR allows the facts defining a language's exponence relations to be formulated without presuming any sort of ordering among a word's morphosyntactic properties (Section 4.1); second, it allows the facts defining a language's morphophonology to make reference to variables over strings of segments (Section 4.2); third, it affords a redundancy-free formulation of "rules of referral": facts referring the evaluation of one set (or sequence) of properties to that of some distinct set (or sequence) of properties (Section 4.3); and finally, it allows competition among a node's facts to be resolved in a uniform way, always by reference to the relative cardinality of the facts' lefthand sides (Section 5). These extensions make KATR especially well-suited for modelling systems of inflectional morphology. In particular, KATR facilitates a compact definition of rules specifying the exponence of a language's morphosyntactic properties, of rules regulating the incidence of a language's sandhi phenomena, and of rules determining a language's systematic patterns of syncretism. In addition, it is compatible with a highly restrictive conception of rule competition, according to which such competition is in all instances resolved by Pāṇini's principle.

The KATR software is freely available for download from the KATR website, http://www.cs.uky.edu/~gstump/katrsite/home.html.

Notes

* 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.

1. By convention, a node's name begins with a capital letter, while an atom's name begins with a lower-case letter. We further distinguish between an inheritance hierarchy’s leaf nodes (with initial capitalization only) and its internal nodes (in all capitals).
2. The "sets" in KATR are, to be precise, multisets (or bags), in that they may contain multiple tokens of a single type. Path reduction can therefore remove the same atom more than once from a query path.
3. A similar conflict might appear to be engendered by facts `F1` and `F3` for the queries `<1PerSubj sgSubj 1PerObj sgObj>` and `<1PerSubj plSubj 1PerObj sgObj>`; but because a rule of Georgian morphosyntax stipulates that a first-person object cannot occur with a first-person subject (Aronson 1990:169), these queries would be ill-formed in a comprehensive account of Georgian morphology.
4. We thank Alexander Dekhtyar for insights leading to the proof that DATR can emulate a Turing Machine.

References

Aronson, Howard I. 1990. *Georgian: A Reading Grammar*. Columbus, OH: Slavica.

Brown, Dunstan. 1996. Facts that influence the shape of inheritance hierarchies, Manuscript. [Surrey Morphology Group Document RP-34.]

Brown, Dunstan. 1998a. Defining `subgender': virile and devirilized nouns in Polish, *Lingua* 104: 187-233.

Brown, Dunstan. 1998b. From the General to the Exceptional, Unpublished PhD thesis, University of Surrey.

Brown, Dunstan. 1998c. Stem indexing and morphonological selection in the Russian verb. In R. Fabri, A. Ortmann and T. Parodi (eds), *Models of Inflection*, 196-221. Niemeyer: Tübingen.

Brown, Dunstan & Andrew Hippisley. 1994. Conflict in Russian genitive plural assignment: a solution represented in DATR, *Journal of Slavic Linguistics* 2: 30-48.

Brown, Dunstan, Greville Corbett, Norman Fraser, Andrew Hippisley & Alan Timberlake. 1996. Russian noun stress and Network Morphology, *Linguistics* 34: 53-107.

Cahill, Lynne J. & Gerald Gazdar. 1997. The inflectional phonology of German adjectives, determiners and pronouns, *Linguistics* 35: 211-245.

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

Evans, Roger & Gerald Gazdar. 1989a. Inference in DATR, in *Proceedings of the Fourth Conference of the European Chapter of the Association for Computational Linguistics*, 66-71, Manchester: Association for Computational Linguistics.

Evans, Roger & Gerald Gazdar. 1989b. The semantics of DATR, in A. G. Cohn (ed.), *Proceedings of the Seventh Conference of the Society for the Study of Artifical Intelligence and Simulation of Behaviour*, 79-87, London: Pitman/Morgan Kaufmann.

Evans, Roger & Gerald Gazdar. 1996. DATR: A language for lexical knowledge representation. *Computational Linguistics* 22, 167-216.

Fraser, Norman M. & Greville G. Corbett. 1995. Gender, animacy, and declensional class assignment: a unified account for Russian, in G. Booij and J. van Marle (eds.), *Yearbook of Morphology 1994*, 123-150, Dordrecht: Kluwer.

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

Gonda, Jan. 1966. *A Concise Elementary Grammar of the Sanskrit Language*, trans. by Gordon B. Ford, Jr. University of Alabama Press.

Hippisley, Andrew. 1996. Russian expressive derivation: a Network Morphology account, *The Slavonic and East European Review* 74 (2): 201-222.

Hippisley, Andrew. 1997. Declarative Derivation: A Network Morphology Account of Russian Word Formation with Reference to Nouns Denoting `Person', Unpublished PhD thesis, University of Surrey.

Hippisley, Andrew. 1998. Indexed stems and Russian word formation: a Network Morphology account of Russian personal nouns, *Linguistics* 36: 1039-1124.

Kiparsky, Paul. 1973. `Elsewhere' in phonology. In S. R. Anderson & P. Kiparsky, eds., *A Festschrift for Morris Halle*. New York: Holt, Rinehart & Winston.

Shen, Lei 1999, KATR: Software for morphological studies in computational linguistics, M.S. project, University of Kentucky Department of Computer Science. (Available at http://www.cs.engr.uky.edu/~raphael/studentWork/#KATR.)

Stump, Gregory T. 2001. *Inflectional Morphology*. Cambridge University Press.