# Machine Parsing of Gilaki Verbs with Fieldworks Language Explorer

*Ron Lockwood*
*SIL International*

## Abstract

This paper will outline how machine parsing with Fieldworks Language Explorer (FLEx) can be done. To illustrate, this parsing will be applied to Gilaki verbs which have sufficient complexity in several areas to show the various methods of getting surface forms to parse correctly while ruling out incorrect forms. The paper will go through the steps to setup a system of parsing in FLEx. This includes setting up a lexicon for roots and affixes, defining templates for verb forms, applying features to lexical items, setting up environment rules and using ad-hoc rules. I will show how parsing can be done both in Arabic script and in Latin script using the same system. Lastly I will show some uses of machine parsing for creating interlinear texts and as a morphological spelling-checker.

## 1 Introduction to FLEx

FLEx is a tool that facilitates the recording and analysis of linguistic and anthropological data.[1] It provides a well-ordered set of fields to record data. It also provides a means of recording a text corpus with the built-in capability to interlinearize text. Running an optional automated morphological parser can make interlinearizing text an efficient task. Grammar tools are also available which allow the user to capture grammatical information. This information is used by the morphological parser, providing a way to check grammatical rules recorded against real language data. The grammar information can also be compiled in an automatically generated grammar sketch.

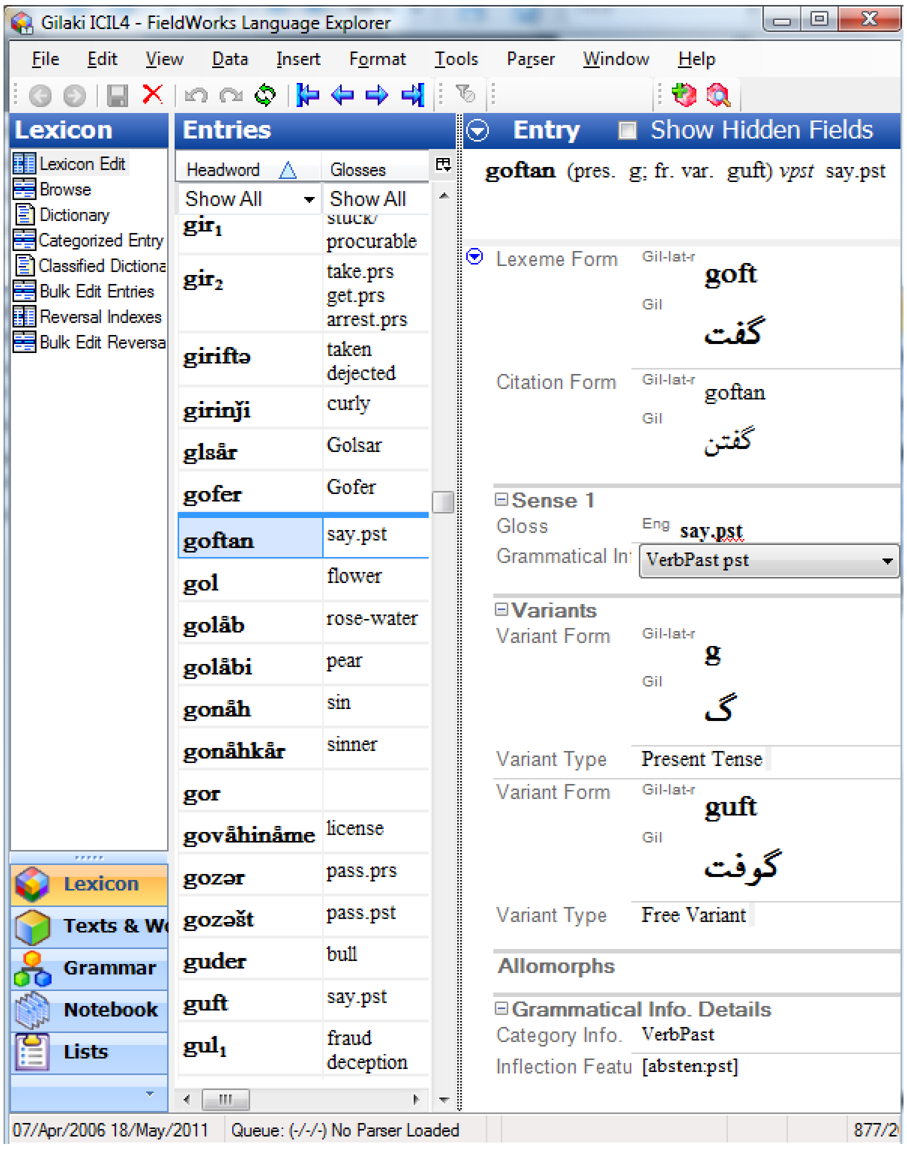
Here are some things you can use FLEx for:

- Enter lexical information for the development of a dictionary
- Rapidly enter words that were collected at a semantic domain word elicitation workshop
- Record vernacular texts and view a concordance of words within recorded texts
- Create an interlinear analysis of texts
- Analyze morphology
- Create a grammar sketch
- Record anthropological field notes

A couple of screen shots of the tools are shown below.

[1] In version 6.0 and earlier the Fieldworks software suite included a tool called Data Notebook for recording and analyzing anthropological data. In version 7.0 of FLEx this tool is integrated into FLEx.

(1) Lexicon edit view showing a sample lexicon entry



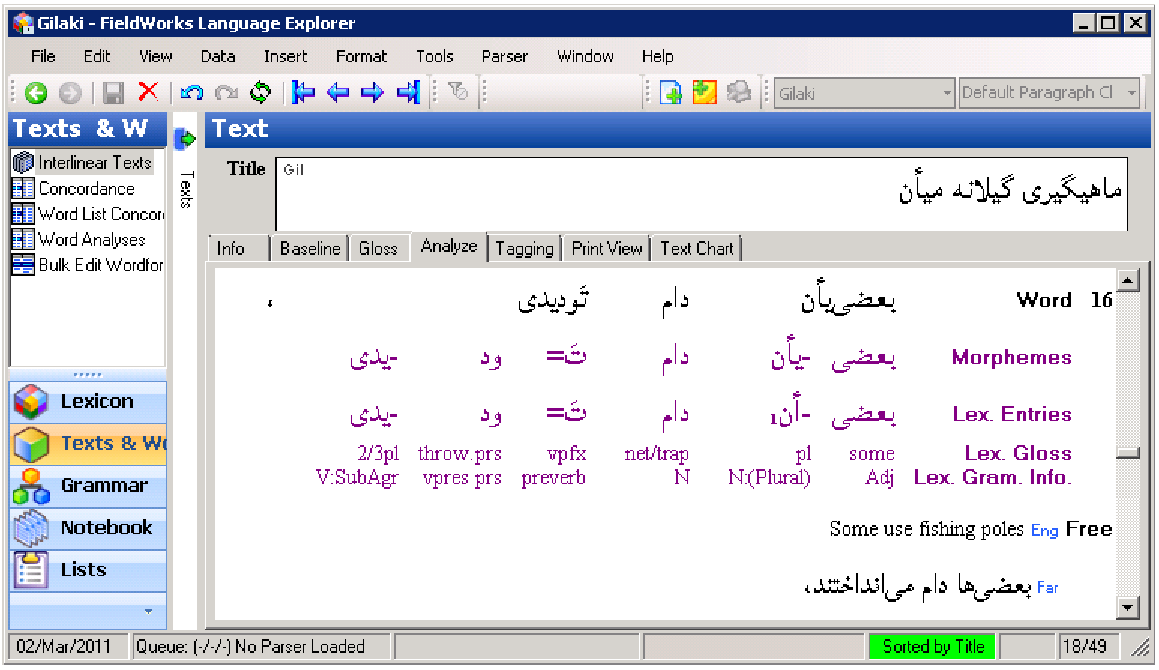
| Headword | Glosses |
|---|---|
| Show All | Show All |
| gir₁ | stuck/ procurable |
| gir₂ | take.prs get.prs arrest.prs |
| giriftə | taken dejected |
| girinǰi | curly |
| glsâr | Golsar |
| gofer | Gofer |
| goftan | say.pst |
| gol | flower |
| golâb | rose-water |
| golâbi | pear |
| gonâh | sin |
| gonâhkâr | sinner |
| gor | |
| govâhināme | license |
| gozər | pass.prs |
| gozəšt | pass.pst |
| guder | bull |
| guft | say.pst |
| gul₁ | fraud deception |

**goftan** (pres. g; fr. var. guft) *vpst* say.pst

| | | |
|---|---|---|
| Lexeme Form | Gil-lat-r | **goft** |
| | Gil | **گفت** |
| Citation Form | Gil-lat-r | goftan |
| | Gil | گفتن |
| **Sense 1** | | |
| Gloss | Eng | **say.pst** |
| Grammatical In | | VerbPast pst |
| **Variants** | | |
| Variant Form | Gil-lat-r | **g** |
| | Gil | **گ** |
| Variant Type | | Present Tense |
| Variant Form | Gil-lat-r | **guft** |
| | Gil | **گوفت** |
| Variant Type | | Free Variant |
| **Allomorphs** | | |
| **Grammatical Info. Details** | | |
| Category Info. | | VerbPast |
| Inflection Featu | | [absten:pst] |

07/Apr/2006 18/May/2011 Queue: (-/-/-) No Parser Loaded 877/2

(2) Interlinear text analysis view showing an interlinearized sentence



Title Gil ماهیگیری گیلانه میأن

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Word 16 | بعضی‌یأن | | دام | تودیدی | | | ، |
| Morphemes | بعضی | -یأن | دام | =تَ | ود | -یدی | |
| Lex. Entries | بعضی | -أن₁ | دام | =تَ | ود | -یدی | |
| Lex. Gloss | some | pl | net/trap | vpfx | throw.prs | 2/3pl | |
| Lex. Gram. Info. | Adj | N:(Plural) | N | preverb | vpres prs | V:SubAgr | |

Free Eng Some use fishing poles

Far بعضی‌ها دام می‌انداختند.

02/Mar/2011 Queue: (-/-/-) No Parser Loaded Sorted by Title 18/49

## 2 The FLEx Parser

The parser is a tool built into FLEx and it can be used to parse a single word, a single text or all words in all texts. It runs as a background process in FLEx so it does not mean you have to stop working in other parts of the tool. There is a Try a Word tool that works together with the parser for experimenting on word parsing. Built into the Try a Word tool is the ability to research why a word did or did not parse. This can be very helpful.

The default FLEx Parser uses an item and arrangement approach for parsing text, Black (2010). This means that the FLEx parser looks at the surface forms of words and creates possible parses from those forms. There is also an experimental parser that uses an item and process approach where underlying forms can be specified. This paper will deal with machine parsing using the first approach. For more information on using the second approach see appendix B of Black (2010).

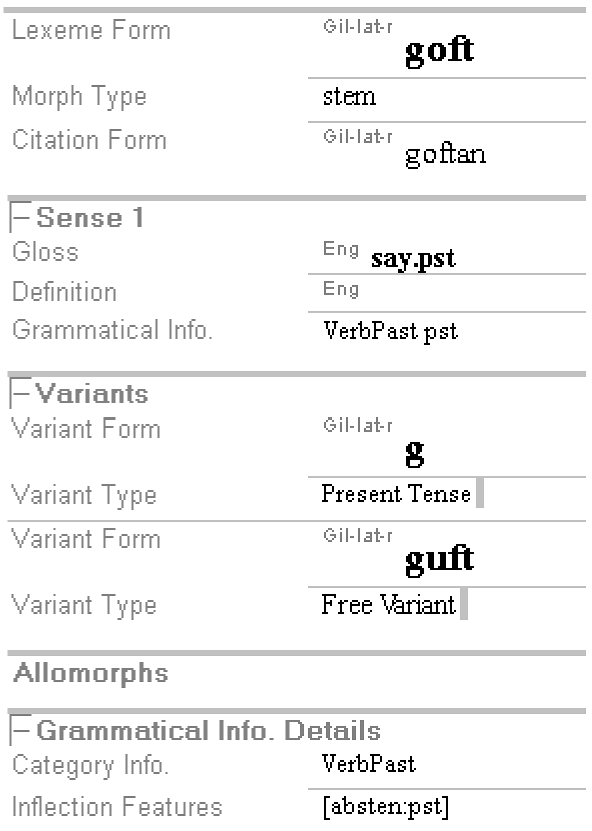
## 3 Setting up FLEx for Parsing

To be able to do a good job of machine parsing in FLEx several things must be in place.

### 3.1 Lexicon

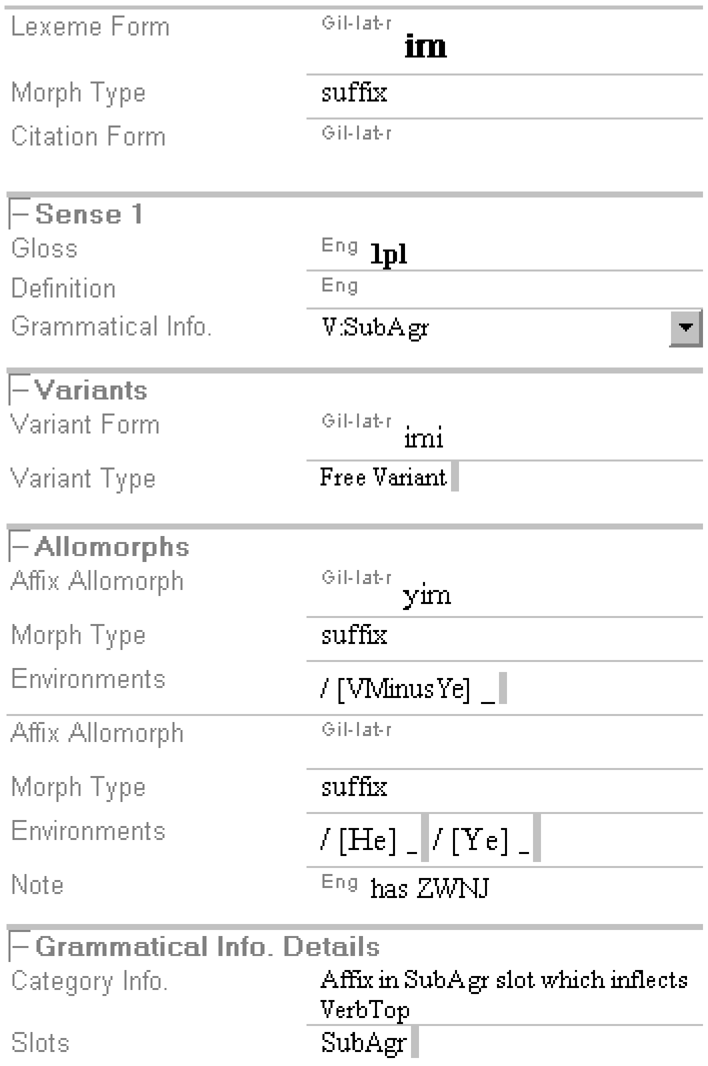
The first requirement is that you have a lexicon containing both the stems and affixes for your language. See (3) and (4).

(3) A stem entry for a Gilaki verb



| | |
|---|---|
| Lexeme Form | Gil-lat-r **goft** |
| Morph Type | stem |
| Citation Form | Gil-lat-r goftan |
| **Sense 1** | |
| Gloss | Eng **say.pst** |
| Definition | Eng |
| Grammatical Info. | VerbPast pst |
| **Variants** | |
| Variant Form | Gil-lat-r **g** |
| Variant Type | Present Tense |
| Variant Form | Gil-lat-r **guft** |
| Variant Type | Free Variant |
| **Allomorphs** | |
| **Grammatical Info. Details** | |
| Category Info. | VerbPast |
| Inflection Features | [absten:pst] |

(4) A suffix entry for a Gilaki verbal suffix



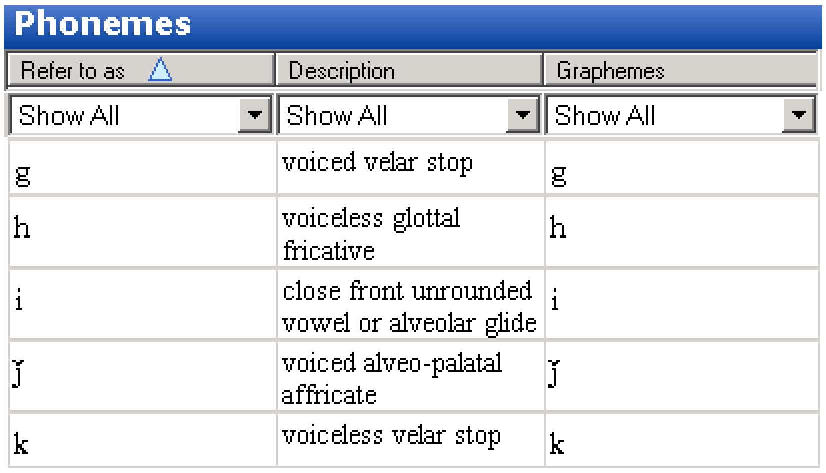
| | |
|---|---|
| Lexeme Form | Gil-lat-r **im** |
| Morph Type | suffix |
| Citation Form | Gil-lat-r |
| **Sense 1** | |
| Gloss | Eng **1pl** |
| Definition | Eng |
| Grammatical Info. | V:SubAgr |
| **Variants** | |
| Variant Form | Gil-lat-r imi |
| Variant Type | Free Variant |
| **Allomorphs** | |
| Affix Allomorph | Gil-lat-r yim |
| Morph Type | suffix |
| Environments | / [VMinusYe] _ |
| Affix Allomorph | Gil-lat-r |
| Morph Type | suffix |
| Environments | / [He] _ / [Ye] _ |
| Note | Eng has ZWNJ |
| **Grammatical Info. Details** | |
| Category Info. | Affix in SubAgr slot which inflects VerbTop |
| Slots | SubAgr |

The main difference between these two kinds of entries is that an affix has a different value for the Morph Type field. In (3) the morphtype is "stem" whereas in (4) the Morph Type is "suffix".
The parser in FLEx will examine a word and try to find matches of affixes and roots in the lexicon.

## 3.2 Phonemes

Another requirement is to set up phonemes in FLEx. These are required because phonemes are referred to in environment rules and can also be referenced in phoneme classes. Basically this is just defining what the phonemes are in the language and naming them so that they can be referred to by other elements of the program. Phonemes are accessed in FLEx by going to the Grammar area and clicking on the view: "Phonemes".

(5) A partial list of Gilaki phonemes



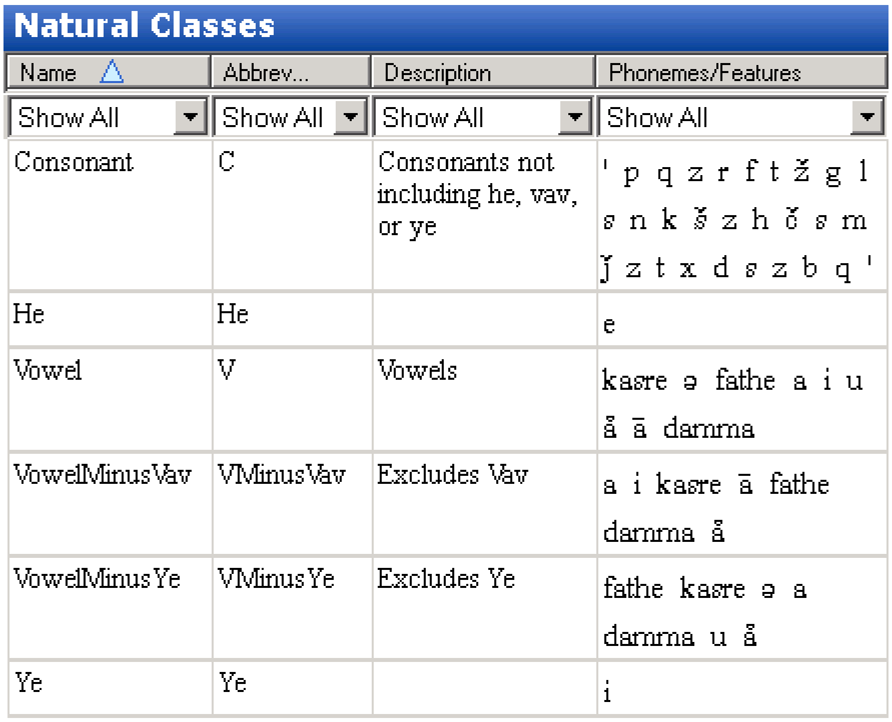
Phonemes

| Refer to as | Description | Graphemes |
|---|---|---|
| Show All | Show All | Show All |
| g | voiced velar stop | g |
| h | voiceless glottal fricative | h |
| i | close front unrounded vowel or alveolar glide | i |
| ǰ | voiced alveo-palatal affricate | ǰ |
| k | voiceless velar stop | k |

### 3.3 Natural Classes

In order to group multiple phonemes together it is necessary to set up natural classes in FLEx. The advantage of setting up phoneme classes is that an environment rule can be created that refers to a whole set of phonemes through the class name. Some common phoneme classes are consonants, voiceless stops and back vowels. Phonemes are accessed in FLEx by going to the Grammar area and clicking on the view: "Natural Classes".

(6) A list of phoneme classes in Gilaki



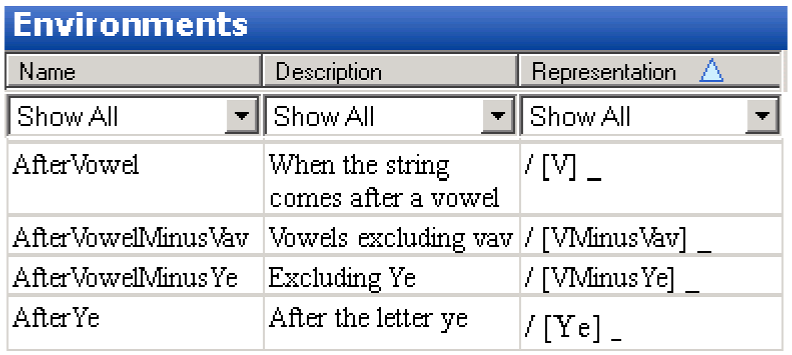
Natural Classes

| Name | Abbrev... | Description | Phonemes/Features |
|---|---|---|---|
| Show All | Show All | Show All | Show All |
| Consonant | C | Consonants not including he, vav, or ye | ' p q z r f t ž g l s n k š z h č s m ǰ z t x d s z b q ' |
| He | He | | e |
| Vowel | V | Vowels | kasre ə fathe a i u å ā damma |
| VowelMinusVav | VMinusVav | Excludes Vav | a i kasre ā fathe damma å |
| VowelMinusYe | VMinusYe | Excludes Ye | fathe kasre ə a damma u å |
| Ye | Ye | | i |

### 3.4 Environments

A fourth requirement is environments. Environments are things defined in FLEx in order to constrain where lexeme forms and allomorphs can occur according to the orthographic environment. They are key things to define for parsing in FLEx because they enable the parser to determine when one allomorph can or cannot be used.

(7) Some examples of environments defined for Gilaki



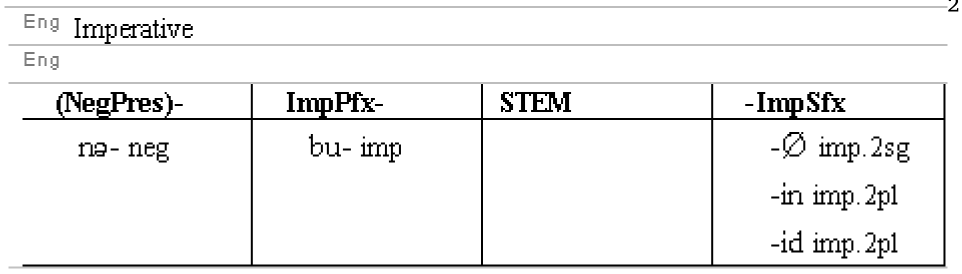
**Environments**

| Name | Description | Representation |
|---|---|---|
| Show All | Show All | Show All |
| AfterVowel | When the string comes after a vowel | / [V] _ |
| AfterVowelMinusVav | Vowels excluding vav | / [VMinusVav] _ |
| AfterVowelMinusYe | Excluding Ye | / [VMinusYe] _ |
| AfterYe | After the letter ye | / [Ye] _ |

### 3.5 Templates

The final requirement is templates. A template is a very powerful tool in FLEx to define affix slots for a grammatical category. In this paper we are focusing on templates for verbs, but templates can be defined for any grammatical category. It is possible to create multiple templates for a category and even have a hierarchy of templates where a template at a higher level applies to all grammatical subcategories.

(8) Example of a template[2]



Eng Imperative
Eng

| (NegPres)- | ImpPfx- | STEM | -ImpSfx |
|---|---|---|---|
| nə- neg | bu- imp | | -∅ imp.2sg |
| | | | -in imp.2pl |
| | | | -id imp.2pl |

## 4 Modeling Gilaki Verb Morphology with Templates

In this section I will discuss the basic morphology patterns of Gilaki verbs and how I have used FLEx to model this. See Rastorgueva, et al. (1971) for a good description of Gilaki grammar.

### 4.1 Forms Built on the Present Stem

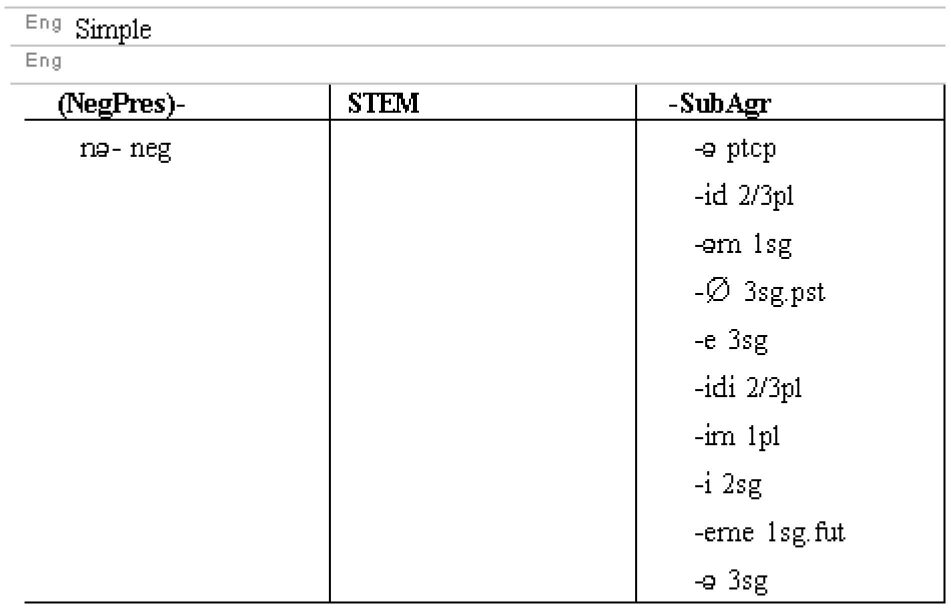
As in many Iranian languages, Gilaki verbs are built on two different stems. We can refer to these two kinds of stems as present and past. In my model for parsing verbs I have subcategorized verbs into two main categories, present and past.[3] The following three sections show forms built on the present stem.

[2]Note that some verbs take the -in form and some take the -id form.
[3]Another possibility for modeling two different stems would be to use "stem names" as described in Black (2010).

### 4.1.1 Simple Aspect

The simple aspect verb form is also simple in terms of its affixation. It has a set of subject agreement suffixes that are added to the present stem. The negative simple aspect form is formed with the prefix nə-.[4] This is modeled in my FLEx project by having a template called "Simple" which has an optional negative prefix slot and an obligatory subject agreement template slot. The template is shown in (9). Note that the SubAgr slot contains all the subject agreement suffixes including a past suffix and the participle suffix.[5]

(9) Simple aspect verb template

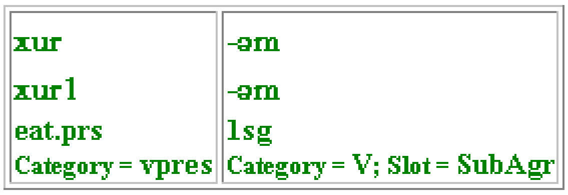


Eng Simple
Eng

| (NegPres)- | STEM | -SubAgr |
|---|---|---|
| nə- neg | | -ə ptcp |
| | | -id 2/3pl |
| | | -əm 1sg |
| | | -∅ 3sg.pst |
| | | -e 3sg |
| | | -idi 2/3pl |
| | | -im 1pl |
| | | -i 2sg |
| | | -eme 1sg.fut |
| | | -ə 3sg |

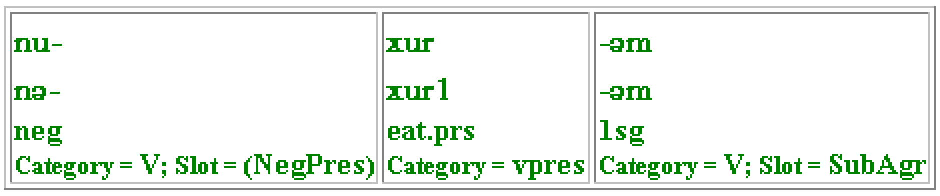
The parser gives the parses shown in (10) and (11) for the verb forms xurəm and nuxurəm which mean I eat and I do not eat respectively.

(10)



| | |
|---|---|
| xur | -əm |
| xur1 | -əm |
| eat.prs | 1sg |
| Category = vpres | Category = V; Slot = SubAgr |

(11)



| | | |
|---|---|---|
| nu- | xur | -əm |
| nə- | xur1 | -əm |
| neg | eat.prs | 1sg |
| Category = V; Slot = (NegPres) | Category = vpres | Category = V; Slot = SubAgr |

### 4.1.2 Subjunctive Mood

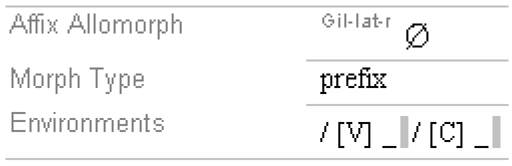
Another verb form that is built on the present stem is the subjunctive mood. Like the simple aspect and many other verb forms, it has a full set of subject agreement suffixes. It also has the prefix bu-[6] that marks the verb as being subjunctive. This form can also have a negative form with the prefix nə- and when the form is negative, the subjunctive prefix drops out. This is modeled in my FLEx project with a template that has an optional negative prefix slot, an obligatory subjunctive prefix

[4]Four forms are possible n-//nə-//nu-//ni- depending on the context.
[5]See section 5.1 for a discussion of why there are some past suffixes in a present template.
[6]Four forms are possible b-//bə-//bu-//bi- depending on the context.

slot and an obligatory subject agreement slot.

How is the subjunctive prefix that drops out handled? For this prefix I have defined a null allomorph which is conditioned by the environment of having some letter preceding it. The allomorph is defined as shown in (12) in my FLEx lexicon. When the subjunctive template is being used to parse a negative subjunctive form, the parser requires there to be a prefix in the subjunctive slot. There is no overt subjunctive prefix to be found, so the parser uses the null allomorph for the parse.

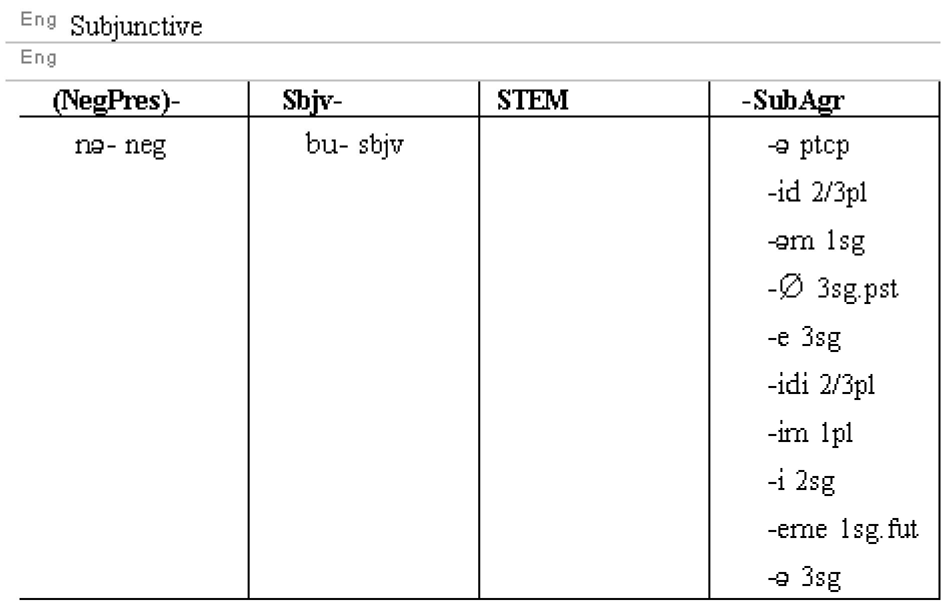
(12) Allomorph section of the subjunctive prefix entry showing a null allomorph



| | |
|---|---|
| Affix Allomorph | Gil-lat-r ∅ |
| Morph Type | prefix |
| Environments | / [V] _ / [C] _ |

See (13) which shows what the FLEx template for the subjunctive mood looks like.

(13) Subjunctive mood verb template

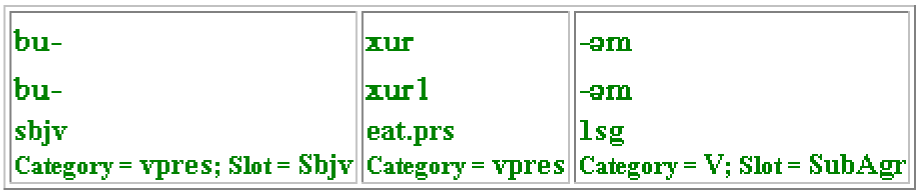


Eng Subjunctive
Eng

| (NegPres)- | Sbjv- | STEM | -SubAgr |
|---|---|---|---|
| nə- neg | bu- sbjv | | -ə ptcp |
| | | | -id 2/3pl |
| | | | -əm 1sg |
| | | | -∅ 3sg.pst |
| | | | -e 3sg |
| | | | -idi 2/3pl |
| | | | -im 1pl |
| | | | -i 2sg |
| | | | -eme 1sg.fut |
| | | | -ə 3sg |

The parser gives the parses shown in (14) and (15) for the verb forms buxurəm and nuxurəm which means I might eat and I might not eat respectively. Note that nuxurəm was used in (11) above illustrating the simple aspect form. This form can have two meanings and thus by having the templates (9) and (13) in place, the parser gives the two possible parses shown in (15).

(14)



| | | |
|---|---|---|
| bu- | xur | -əm |
| bu- | xur1 | -əm |
| sbjv | eat.prs | 1sg |
| Category = vpres; Slot = Sbjv | Category = vpres | Category = V; Slot = SubAgr |

(15)

| | | |
|---|---|---|
| nu- | xur | -əm |
| nə- | xur1 | -əm |
| neg | eat.prs | 1sg |
| Category = V; Slot = (NegPres) | Category = vpres | Category = V; Slot = SubAgr |

| | | | |
|---|---|---|---|
| nu- | ∅- | xur | -əm |
| nə- | bu- | xur1 | -əm |
| neg | sbjv | eat.prs | 1sg |
| Category = V; Slot = (NegPres) | Category = vpres; Slot = Sbjv | Category = vpres | Category = V; Slot = SubAgr |

### *4.1.3 Imperative Mood*

The imperative is somewhat similar to the subjunctive mood. It also can have an optional negative prefix nə- and it has a prefix bu-. This indicates imperative mood which like the subjunctive drops out when the negative prefix is present. The suffixes that can occur with the imperative mood are limited. This verb form is modeled in FLEx as shown in (16). The template has the same optional negative prefix slot that we have seen before. It has an obligatory imperative prefix slot and an obligatory imperative suffix slot.

The dropping out of the imperative prefix in the negative form is handled in the same way as the subjunctive case. In fact, the forms of the subjunctive and imperative are the same, so the same lexical entry in the lexicon is used and I have defined two senses of this prefix: one for subjunctive and the other for imperative. The null allomorph is one and the same. See (12).

(16) Imperative mood verb template

Eng Imperative
Eng

| (NegPres)- | ImpPfx- | STEM | -ImpSfx |
|---|---|---|---|
| nə- neg | bu- imp | | -∅ imp.2sg |
| | | | -in imp.2pl |
| | | | -id imp.2pl |

The parser gives the parses shown in (17) and (18) for the verb forms buxur and nuxurin which mean eat! and do not eat! respectively. There are three possible parses of nuxurin meaning you do not eat, do not eat!, or you might not eat. They are all legitimate meanings. These are shown in (19). These parses are possible because the form nuxurin matches the three templates shown in (9), (16) and (13).

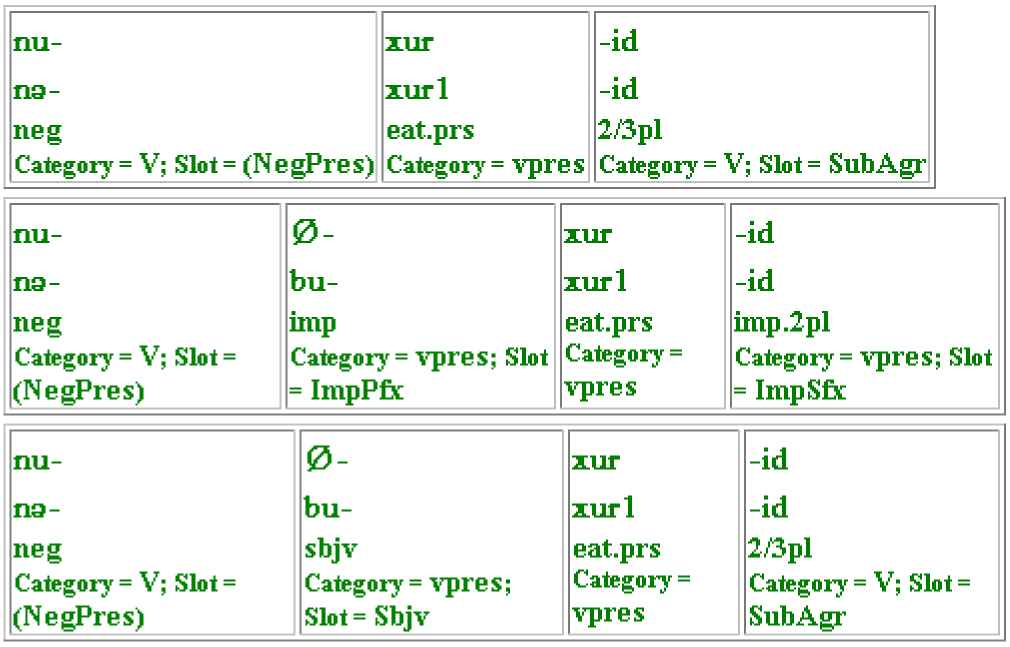
(17)

| | | |
|---|---|---|
| bu- | xur | -∅ |
| bu- | xur1 | -∅1 |
| imp | eat.prs | imp.2sg |
| Category = vpres; Slot = ImpPfx | Category = vpres | Category = vpres; Slot = ImpSfx |

(18)

| | | | |
|---|---|---|---|
| nu- | ∅- | xur | -in |
| nə- | bu- | xur1 | -in3 |
| neg | imp | eat.prs | imp.2pl |
| Category = V; Slot = (NegPres) | Category = vpres; Slot = ImpPfx | Category = vpres | Category = vpres; Slot = ImpSfx |

(19)



| | | |
|---|---|---|
| nu- | xur | -id |
| nə- | xur1 | -id |
| neg | eat.prs | 2/3pl |
| Category = V; Slot = (NegPres) | Category = vpres | Category = V; Slot = SubAgr |

| | | | |
|---|---|---|---|
| nu- | Ø- | xur | -id |
| nə- | bu- | xur1 | -id |
| neg | imp | eat.prs | imp.2pl |
| Category = V; Slot = (NegPres) | Category = vpres; Slot = ImpPfx | Category = vpres | Category = vpres; Slot = ImpSfx |

| | | | |
|---|---|---|---|
| nu- | Ø- | xur | -id |
| nə- | bu- | xur1 | -id |
| neg | sbjv | eat.prs | 2/3pl |
| Category = V; Slot = (NegPres) | Category = vpres; Slot = Sbjv | Category = vpres | Category = V; Slot = SubAgr |

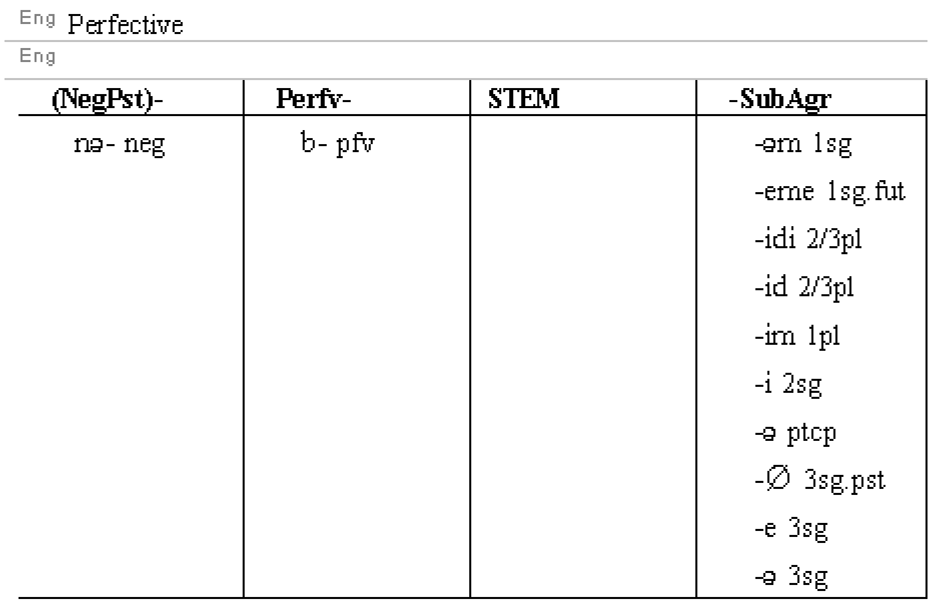
## 4.2 Forms Built on the Past Stem

Two main forms are built on the past stem form of the verb.

### *4.2.1 Perfective Aspect*

The perfective aspect of the verb is built on the past stem of the verb. It takes the full set of subject agreement suffixes and has the prefix bu-. It can take a negative prefix and like the subjunctive prefix, the perfective prefix drops out when the negative prefix is present. In FLEx this form is modeled with an optional negative prefix slot and obligatory perfective prefix and subject agreement suffix slots. The template is shown in (20).

(20) Perfective aspect verb template

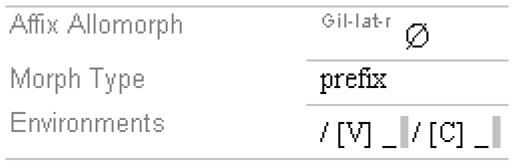


Eng Perfective
Eng

| (NegPst)- | Perfv- | STEM | -SubAgr |
|---|---|---|---|
| nə- neg | b- pfv | | -əm 1sg |
| | | | -eme 1sg.fut |
| | | | -idi 2/3pl |
| | | | -id 2/3pl |
| | | | -im 1pl |
| | | | -i 2sg |
| | | | -ə ptcp |
| | | | -Ø 3sg.pst |
| | | | -e 3sg |
| | | | -ə 3sg |

The perfective prefix that drops out is handled in the same way as in the subjunctive case. For this prefix I have defined a null allomorph which is conditioned by the environment of having some letter preceding it. The allomorph is defined as shown in (21) in my FLEx lexicon. When the perfective template is being used to parse a negative perfective form, the parser requires there to be a prefix in the Perfv slot. There is no overt perfective prefix to be found, so the parser uses the null allomorph for the parse.

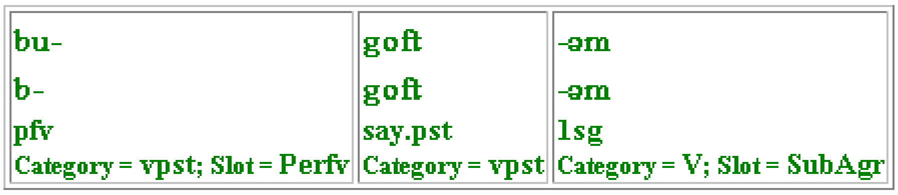
(21) Allomorph section of the perfective prefix entry showing a null allomorph



| | |
|---|---|
| Affix Allomorph | Gil-lat-r ∅ |
| Morph Type | prefix |
| Environments | / [V] _ / [C] _ |

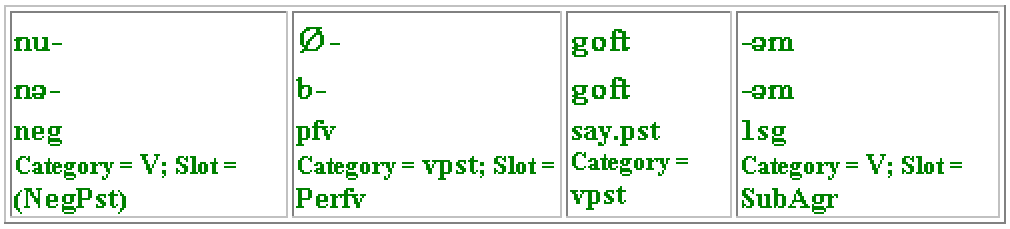
The parser gives the parses shown in (22) and (23) for the verb forms goftəm and nugoftəm which mean I said and I did not say respectively.

(22)



| | | |
|---|---|---|
| bu- | goft | -əm |
| b- | goft | -əm |
| pfv | say.pst | 1sg |
| Category = vpst; Slot = Perfv | Category = vpst | Category = V; Slot = SubAgr |

(23)

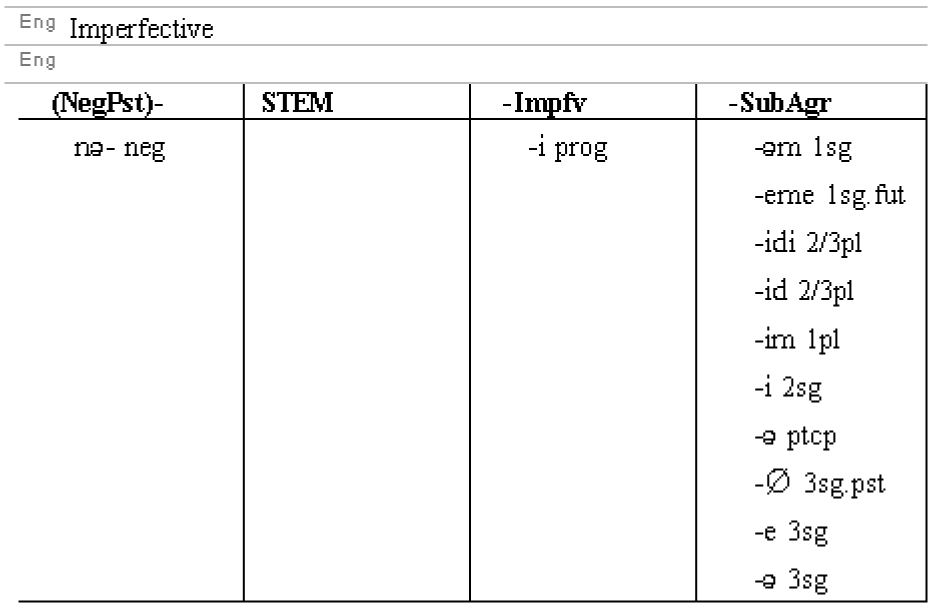


| | | | |
|---|---|---|---|
| nu- | ∅- | goft | -əm |
| nə- | b- | goft | -əm |
| neg | pfv | say.pst | 1sg |
| Category = V; Slot = (NegPst) | Category = vpst; Slot = Perfv | Category = vpst | Category = V; Slot = SubAgr |

### *4.2.2 Imperfective Aspect*

The imperfective aspect is also built on the past stem. This form is built by having an imperfective suffix -i attach to the stem followed by a subject agreement suffix. It also takes an optional negative prefix. This form is modeled in FLEx with an optional negative prefix slot and two obligatory suffix slots, first an imperfective suffix slot and then a subject agreement slot. The two suffix slots, merely by their position relative to the stem, require the affixes contained in those slots to come in that order. The template is shown in (24).

(24) Imperfective aspect verb template

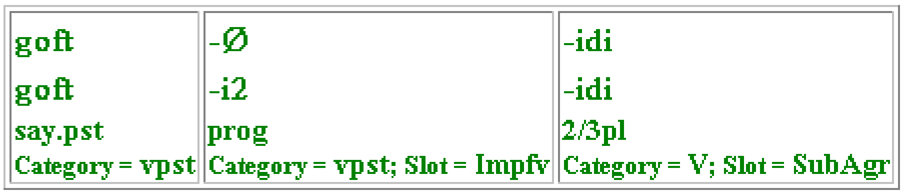


Eng Imperfective
Eng

| (NegPst)- | STEM | -Impfv | -SubAgr |
|---|---|---|---|
| nə- neg | | -i prog | -əm 1sg |
| | | | -eme 1sg.fut |
| | | | -idi 2/3pl |
| | | | -id 2/3pl |
| | | | -im 1pl |
| | | | -i 2sg |
| | | | -ə ptcp |
| | | | -∅ 3sg.pst |
| | | | -e 3sg |
| | | | -ə 3sg |

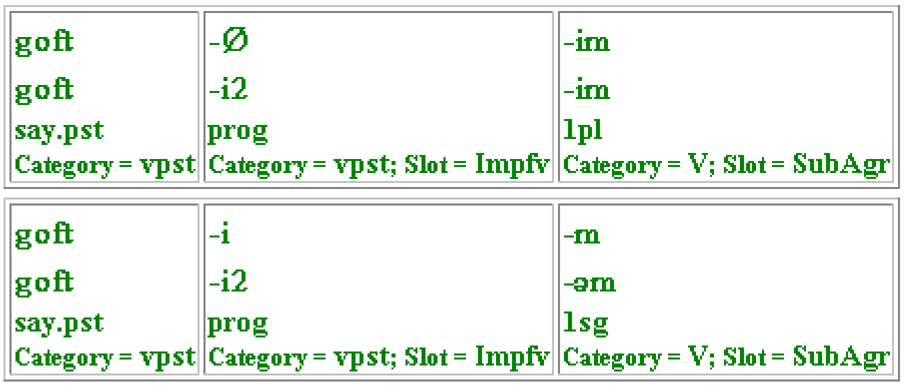
In Gilaki the progressive suffix is only seen overtly in the first person singular and in the third person singular. In the other person/number cases the progressive suffix is merged into the subject agreement suffix. For my analysis I have assumed that the progressive suffix is null in these cases. For example, the word goftidi means you(pl)/they were saying and the parser gives the parse shown in (25). Note that the progressive suffix is null in this case. For the word goftim there are two possible meanings: We were saying or I was saying. Accordingly there are two parses for this word both in the imperfective aspect. One has the overt progressive suffix and the other has the null progressive suffix. The parses are shown in (26). To illustrate the negative form takes the form nugoftidi. Besides the perfective meaning there is also an imperfective meaning. (27) shows two parses meaning you(pl)/they were not saying and you(pl)/they did not say.

(25)



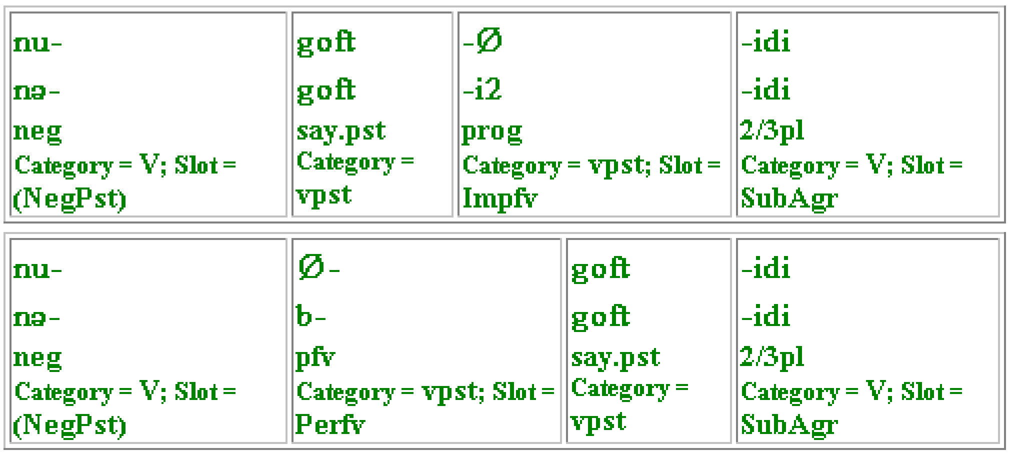
| | | |
|---|---|---|
| goft<br>goft<br>say.pst<br>Category = vpst | -Ø<br>-i2<br>prog<br>Category = vpst; Slot = Impfv | -idi<br>-idi<br>2/3pl<br>Category = V; Slot = SubAgr |

(26)



| | | |
|---|---|---|
| goft<br>goft<br>say.pst<br>Category = vpst | -Ø<br>-i2<br>prog<br>Category = vpst; Slot = Impfv | -im<br>-im<br>1pl<br>Category = V; Slot = SubAgr |
| goft<br>goft<br>say.pst<br>Category = vpst | -i<br>-i2<br>prog<br>Category = vpst; Slot = Impfv | -m<br>-əm<br>1sg<br>Category = V; Slot = SubAgr |

(27)



| | | | |
|---|---|---|---|
| nu-<br>nə-<br>neg<br>Category = V; Slot = (NegPst) | goft<br>goft<br>say.pst<br>Category = vpst | -Ø<br>-i2<br>prog<br>Category = vpst; Slot = Impfv | -idi<br>-idi<br>2/3pl<br>Category = V; Slot = SubAgr |
| nu-<br>nə-<br>neg<br>Category = V; Slot = (NegPst) | Ø-<br>b-<br>pfv<br>Category = vpst; Slot = Perfv | goft<br>goft<br>say.pst<br>Category = vpst | -idi<br>-idi<br>2/3pl<br>Category = V; Slot = SubAgr |

## 5 Tools for Parsing Gilaki Verbs

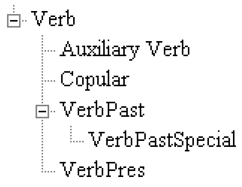
### 5.1 Using Templates

In section 4 above I showed different templates that I employ to model Gilaki verb morphology. The power of using templates in FLEx is that it quickly constrains the parser as to which affixes can be used and where. The FLEx parser can be used without employing templates, but then any number of affixes can occur on a stem and there is no constraint on what order the affixes can occur in. If the parser is not constrained with templates and other tools discussed below, it can find all kinds of possible parses with your inventory of stems and affixes.

A template can have one or more slots. In each slot you define which affixes are valid for that slot. The affixes come from the lexicon. A slot can be optional or obligatory. By default a slot in FLEx is obligatory unless it is marked as optional. If a slot is obligatory the parser will only successfully parse a word if an affix in the slot is present.

In FLEx grammatical categories can be arranged in a hierarchy. I have defined several subcategories of the category verb. The hierarchy I have used is shown in (28).

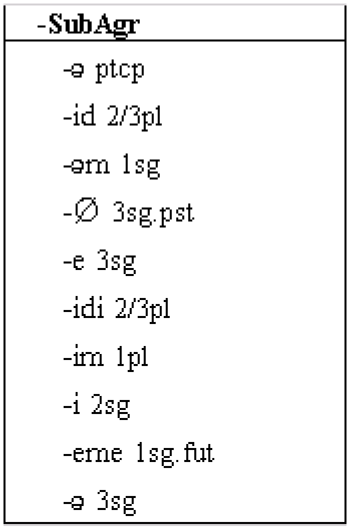
(28) Verb heirarchy



```
Verb
    Auxiliary Verb
    Copular
    VerbPast
        VerbPastSpecial
    VerbPres
```

At every level in the verb hierarchy, one or more templates can be defined. In section 4 I showed several templates that I have defined for the categories VerbPast and VerbPres. Since the categories are set up in a hierarchy, template slots that I have defined at the top level (Verb) can be used at any level below. In my model for Gilaki verbs, I defined a slot called SubAgr at the top level (Verb) which contains all the subject agreement affixes and I use this slot throughout many templates at lower levels of the hierarchy. The slot is shown in (29). Besides the advantage of using the same slot in multiple templates, having one slot means that you have to only maintain one slot. So when you discover you need to add another affix to the subject agreement slot for example, you only have to add it in one place and it applies to all templates that are using that slot.

(29) Subject agreement template slot



| -SubAgr |
|---|
| -ə ptcp |
| -id 2/3pl |
| -əm 1sg |
| -∅ 3sg.pst |
| -e 3sg |
| -idi 2/3pl |
| -im 1pl |
| -i 2sg |
| -eme 1sg.fut |
| -ə 3sg |

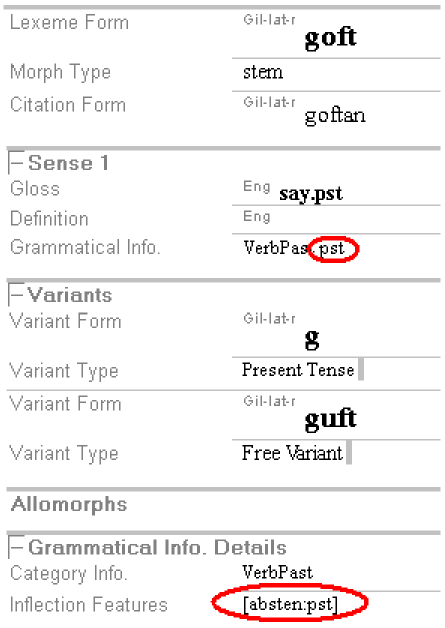
In regards to this subject agreement slot, you may ask why there are suffixes that go on past stems mixed in with suffixes that go on both past and present stems. The suffixes glossed ptcp and 3sg.pst are two that only suffix to past stems. I experimented in FLEx with several models for handling this. One option is to have one slot for past stem suffixes and another for present stem suffixes, but this unnecessarily complicates the analysis. I found that it was simpler to put all the suffixes in one slot and then disallow their joining to present stems using features. See section 5.2 for more information on features.

## 5.2 Using Features to Prevent Affixes from Attaching to Incorrect Verb Stems

FLEx allows you to define a morpho-syntactic feature system for the language you are working on. Things like gender, case, noun classes and much more can be modeled with features. For the

Gilaki project I have used inflection features to mark stems and affixes as being past tense or present tense. Every verb stem in the lexicon has been marked as either past or present. By examining the Grammatical Info. Details section of the stem entry shown in (30), we see that the entry is tagged with the inflection feature [absten:pst].[7] In the sense section of the entry the feature "pst" is also shown.

(30) Verb stem entry marked with the "pst" inflection feature

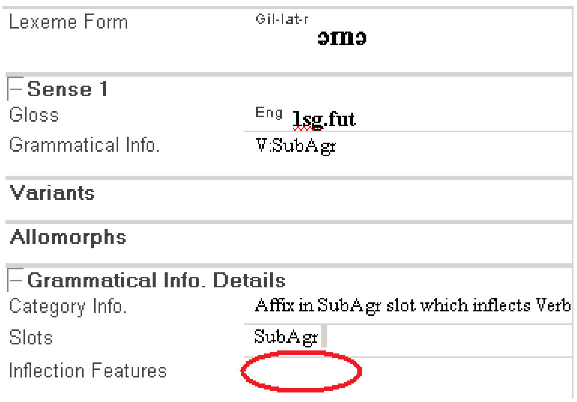


| | |
|---|---|
| Lexeme Form | Gil-lat-r **goft** |
| Morph Type | stem |
| Citation Form | Gil-lat-r goftan |
| **Sense 1** | |
| Gloss | Eng **say.pst** |
| Definition | Eng |
| Grammatical Info. | VerbPast pst |
| **Variants** | |
| Variant Form | Gil-lat-r **g** |
| Variant Type | Present Tense |
| Variant Form | Gil-lat-r **guft** |
| Variant Type | Free Variant |
| **Allomorphs** | |
| **Grammatical Info. Details** | |
| Category Info. | VerbPast |
| Inflection Features | [absten:pst] |

Stems that are present stems are similarly marked with "prs".

How can this marking with features keep incorrect affixes from co-occurring with certain stems? When the parser in FLEx checks to see if an affix can attach to a stem, it checks to see if there is a mismatch for a feature value. If there is a mismatch the possible parse is disallowed. As an example let's take the 1sg.fut suffix -əmə. The entry is shown in (31).

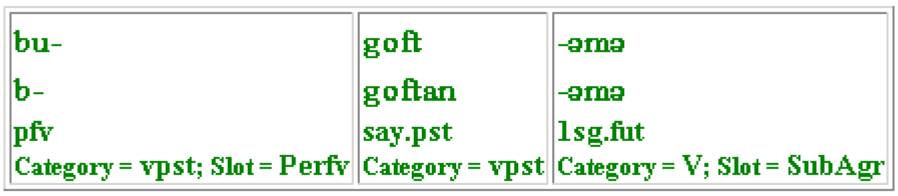
(31) Verb suffix entry unmarked for tense



| | |
|---|---|
| Lexeme Form | Gil-lat-r **əmə** |
| **Sense 1** | |
| Gloss | Eng **1sg.fut** |
| Grammatical Info. | V:SubAgr |
| **Variants** | |
| **Allomorphs** | |
| **Grammatical Info. Details** | |
| Category Info. | Affix in SubAgr slot which inflects Verb |
| Slots | SubAgr |
| Inflection Features | |

If we do not have -əmə marked for tense and we parse the word *bugoftəmə we will get the following incorrect parse:

(32)

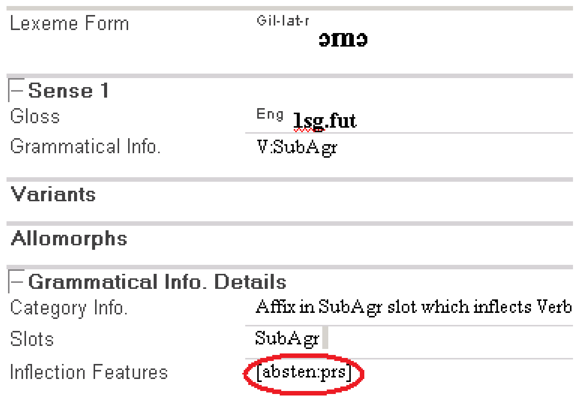


| | | |
|---|---|---|
| bu- | goft | -əmə |
| b- | goftan | -əmə |
| pfv | say.pst | 1sg.fut |
| Category = vpst; Slot = Perfv | Category = vpst | Category = V; Slot = SubAgr |

[7]absten is an abbreviation for absolute tense.

It is wrong because the 1sg.fut should not be allowed to attach to a past stem. If we add the inflectional feature of present tense ("prs") to this affix as in (33) and try to parse this word again we get the message: This word failed to parse successfully.

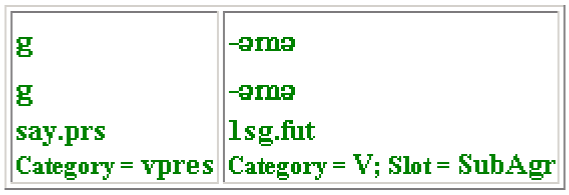
(33) Verb suffix entry marked for "prs" tense



| | |
|---|---|
| Lexeme Form | Gil-lat-r əmə |
| **Sense 1** | |
| Gloss | Eng 1sg.fut |
| Grammatical Info. | V:SubAgr |
| **Variants** | |
| **Allomorphs** | |
| **Grammatical Info. Details** | |
| Category Info. | Affix in SubAgr slot which inflects Verb |
| Slots | SubAgr |
| Inflection Features | [absten:prs] |

If we research further with FLEx's "Try a Word" tool we see that the reason for the failed parse is the incompatibility of the feature "pst" on the stem with the feature "prs" on the suffix.[8] If we try parsing the word gəme - I will say which is using the present stem we get the following successful parse:

(34)



| | |
|---|---|
| g | -əmə |
| g | -əmə |
| say.prs | 1sg.fut |
| Category = vpres | Category = V; Slot = SubAgr |

This parse succeeds because the feature "prs" on the stem matches the feature "prs" of the suffix.

Similarly if we tried to parse a present stem with a past suffix, it will fail because of the feature mismatch.

## 5.3 Using Environments to Force the Parser to Choose the Correct Allomorph

In Gilaki as in many languages, there are different allomorphs in the language's lexicon and these are conditioned by certain environments. When multiple allomorphs exist we would like the parser to allow allomorphs when the environment condition is satisfied and disallow the allomorphs that do not satisfy the environment condition.

One example of allomorphs and environments in Gilaki verbs is the bu- prefix which has allomorphs bi-, bə-, b- and ∅-. The allomorph that attaches to a verb stem depends on the environment. See the following table of example words:

Table 1 Allomorphs of bu-

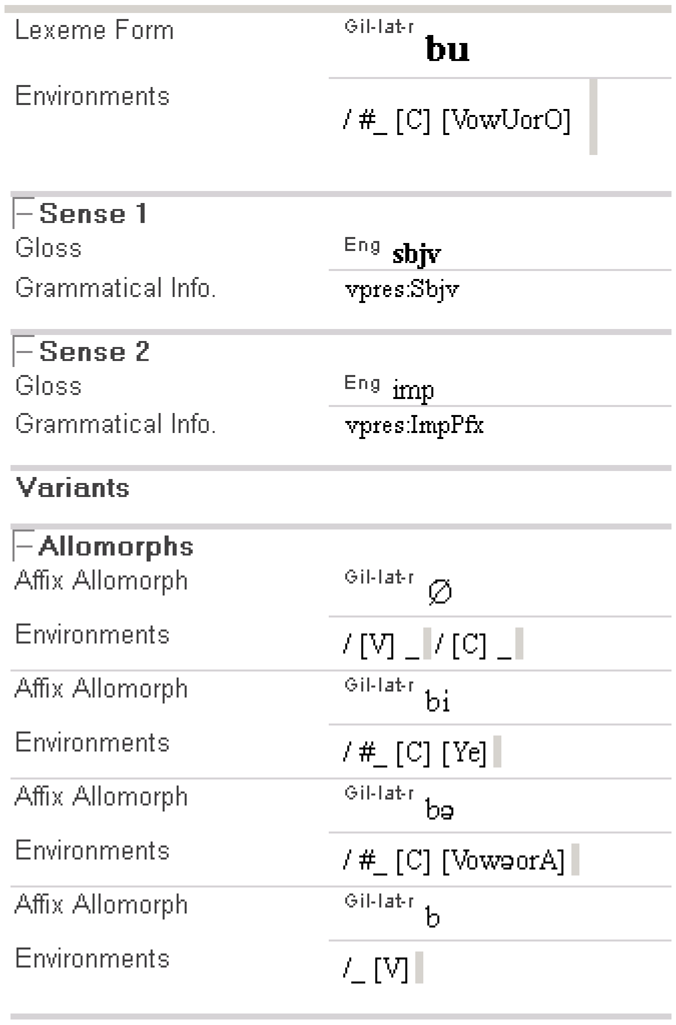
| Prefix | Word | Gloss |
|---|---|---|
| bu- | bukunəm | I might do |
| bu- | buxosəm | I might sleep |
| bi- | bigirəm | I might take |

[8]The actual message is the following: The inflectional template named 'Perfective' for category 'VerbPast' failed because at least one inflection feature of the stem is incompatible with the inflection features of the inflectional suffix (-əmə (1sg.fut): -əmə). The incompatibility is for feature absten. This feature for the stem has a value of pst but the corresponding feature for the inflectional suffix (-əmə (1sg.fut): -əmə) has a value of prs.

| Prefix | Word | Gloss |
|---|---|---|
| bə- | bəkəšəm | I might pull |
| b- | bayəm | I might come |
| ∅- | nigirəm | I might not take, I do not take |

So how do we model this in FLEx so that the parser correctly parses all these different forms, but rejects malformed combinations such as *bikəšəm, *bugirəm, *bəxosəm and *bkunəm? What we do is specify an environment for each allomorph and the parser will check if the environment applies. Example (35) shows what the entry for prefix bu- looks like. The order of the allomorph is important. The parser works from the top allomorph to the bottom. For each allomorph the parser assumes the previous environment is negated. Note that the lexeme form is one of the allomorphs for the entry and serves as the elsewhere allomorph, i.e. the last allomorph after all other allomorph environments have not been satisfied. Also, to use classes in environments you need to have the class defined in FLEx, and a class relies on phonemes having been defined in FLEx. (5), (6) and (7) above showed some definitions of phonemes, natural classes and environments.

(35) Prefix entry with environment conditioned allomorphs



| | |
|---|---|
| Lexeme Form | Gil-lat-r **bu** |
| Environments | / #_ [C] [VowUorO] |
| **Sense 1** | |
| Gloss | Eng **sbjv** |
| Grammatical Info. | vpres:Sbjv |
| **Sense 2** | |
| Gloss | Eng imp |
| Grammatical Info. | vpres:ImpPfx |
| **Variants** | |
| **Allomorphs** | |
| Affix Allomorph | Gil-lat-r ∅ |
| Environments | / [V] _ / [C] _ |
| Affix Allomorph | Gil-lat-r bi |
| Environments | / #_ [C] [Ye] |
| Affix Allomorph | Gil-lat-r bə |
| Environments | / #_ [C] [VowəorA] |
| Affix Allomorph | Gil-lat-r b |
| Environments | /_ [V] |

The table below shows which environment applies for which word:

Table 2 Environments for allomorphs of bu-

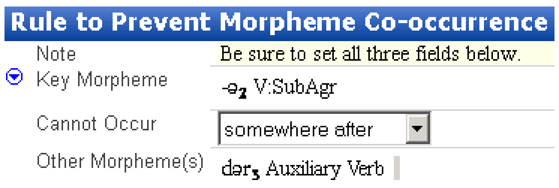
| Word | Environment(s) | Comment |
|---|---|---|
| bukunəm | / #_ [C] [VowUorO] | Use bu- when followed by a consonant and a "u" or "o". VowUorO is a class containing "u" and "o". |
| buxosəm | / #_ [C] [VowUorO] | Use bu- when followed by a consonant and a "u" or "o". VowUorO is a class containing "u" and "o". |
| bigirəm | / #_ [C] [Ye] | Use bi- when followed by a consonant and a "i". Ye is a class containing "i". |
| bəkəšəm | / #_ [C] [VowəorA] | Use bə- when followed by a consonant and a "ə" or "a". VowəorA is a class containing "ə" or "a". |
| bayəm | /_ [V] | Use b- when followed by a vowel. |
| nigirəm | / [V] _; / [C] _ | Use ∅- when preceded by any letter (vowel or consonant). |

Environments can apply to any kind of allomorph. In Gilaki many verbs have more than one allomorph for a stem. For example, the verb meaning to have has forms dašt and ašt. The form ašt occurs in the negative form. So I have two allomorphs listed for this entry and an environment for the form ašt which says it can only occur following an "n".

## 5.4 Using Ad Hoc Rules to Prevent the Co-occurrence of Morphemes

Using templates, features and environments can really make the parsing model work well. But there are always situations where the parser is incorrectly allowing morphemes or allomorphs to occur together. In this situation you can have some rules in FLEx that will disallow co-occurring morphemes. I have primarily used these ad hoc rules to prevent clitics from co-occurring, but an example in Gilaki verbs is the following: In Gilaki there is a present progressive auxiliary dər. Being an auxiliary, this form would never have the participle suffix -ə. I have created an ad hoc rule that prevents the participle from occurring after this stem. (36) shows the rule as it is defined in FLEx. If I did not have this rule, I would get the two parses shown in (37). The 2nd is invalid and does not get suggested by the parser when I have the rule in (36) defined.

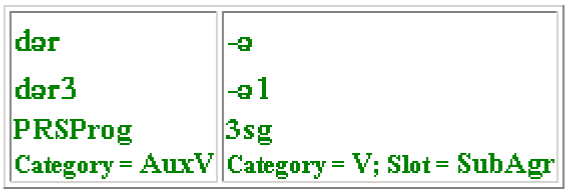
(36) Ad Hoc Rule



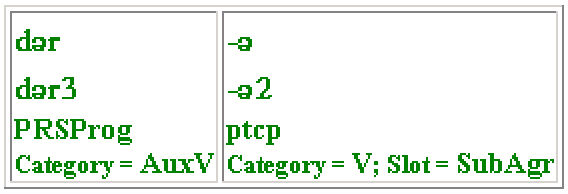
**Rule to Prevent Morpheme Co-occurrence**

| | |
|---|---|
| Note | Be sure to set all three fields below. |
| Key Morpheme | -ə$_2$ V:SubAgr |
| Cannot Occur | somewhere after |
| Other Morpheme(s) | dər$_3$ Auxiliary Verb |

(37)



| | |
|---|---|
| dər | -ə |
| dər3 | -ə1 |
| PRSProg | 3sg |
| Category = AuxV | Category = V; Slot = SubAgr |



| | |
|---|---|
| dər | -ə |
| dər3 | -ə2 |
| PRSProg | ptcp |
| Category = AuxV | Category = V; Slot = SubAgr |

## 5.5 Other Tools

In addition to the tools described above, FLEx has other tools that can help constrain the parser. These tools include Inflection Classes, Exception Features, Compound Rules and Stem Names. See Black (2010) for a thorough treatment of how to use FLEx for various linguistic phenomena. This paper is available under the Help -> Resources -> Introduction to Parsing menu in FLEx. I would recommend a person read this paper before starting to develop a model for parsing in FLEx.

## 6 Using Arabic Script

FLEx is a program that fully supports the Unicode standard of representing character encodings. Because of this nearly any complex script can be used in FLEx. Even scripts that are not supported by current operating systems can be supported by using an SIL rendering engine called Graphite.

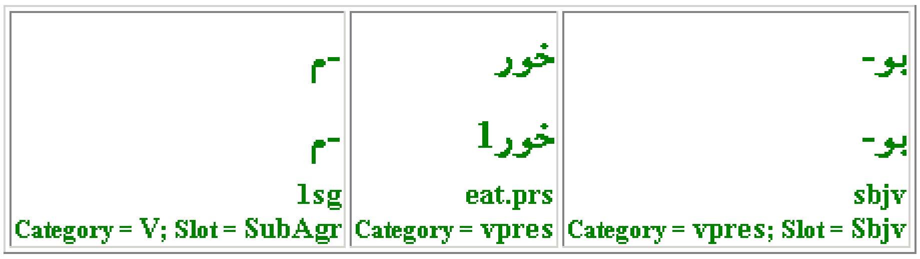
Arabic script is supported in FLEx, and even the user interface can be changed to use a language written in Arabic script. The interface has been translated to Persian for most common interface items. Whereas right to left support in FLEx could be better, I think it is usable and hopefully we can encourage the FLEx development team to make it better.

Since Gilaki is most often written in Arabic script, I have done quite a bit of work getting FLEx to parse Gilaki texts written in Arabic script. I have even been fairly successful at having a system that will work with both scripts. I want to list here some of the issues that come up with using Arabic script in general and show specific examples of parsing Gilaki verbs.

### 6.1 Sample Parse

The Arabic script equivalent of the parse in (14) is shown below in (38).

(38)

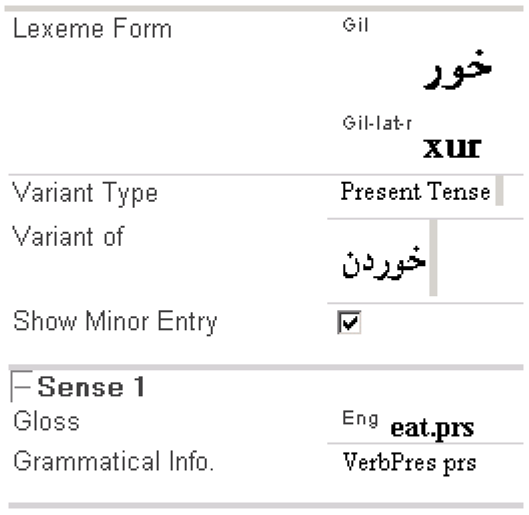


| | | |
|---|---|---|
| -م | خور | بو- |
| -م | خور1 | بو- |
| 1sg | eat.prs | sbjv |
| Category = V; Slot = SubAgr | Category = vpres | Category = vpres; Slot = Sbjv |

The parse looks as you would expect with the first affix starting in the right column. What differences are there in the lexicon to support multiple scripts? (39) shows the stem entry used above and (40) shows the prefix entry. Note how the lexeme forms are written in both Arabic script and Latin script.[9] Note that there are two scripts written for the allomorph shown in (40) as well.

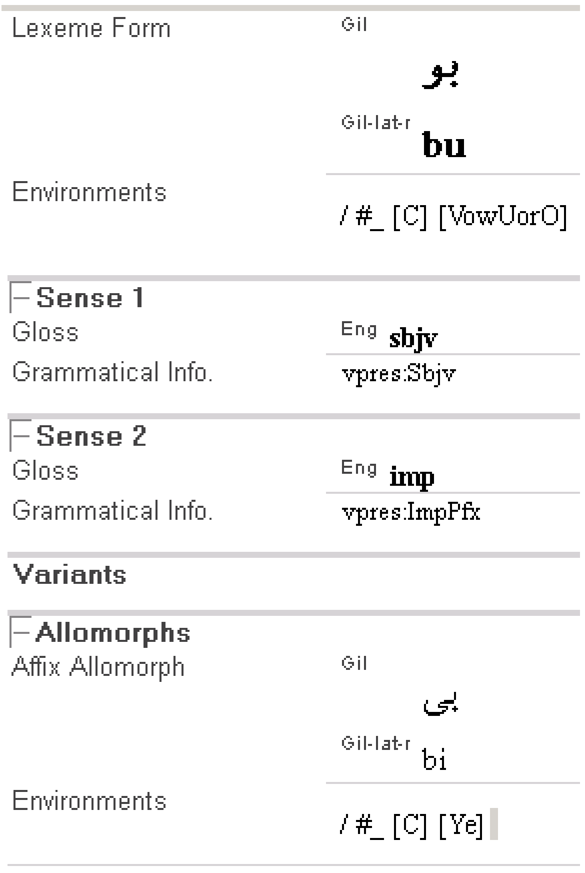
[9]In (3) and in other examples, the Arabic script line was hidden by choosing to hide the Gilaki Arabic writing system which is indicated by Gil in example (39).

(39) Stem entry with two scripts



| | |
|---|---|
| Lexeme Form | Gil خور |
| | Gil-lat-r **xur** |
| Variant Type | Present Tense |
| Variant of | خوردن |
| Show Minor Entry | ☑ |
| **Sense 1** | |
| Gloss | Eng **eat.prs** |
| Grammatical Info. | VerbPres prs |

(40) Prefix entry with two scripts



| | |
|---|---|
| Lexeme Form | Gil بو |
| | Gil-lat-r **bu** |
| Environments | / #_ [C] [VowUorO] |
| **Sense 1** | |
| Gloss | Eng **sbjv** |
| Grammatical Info. | vpres:Sbjv |
| **Sense 2** | |
| Gloss | Eng **imp** |
| Grammatical Info. | vpres:ImpPfx |
| **Variants** | |
| **Allomorphs** | |
| Affix Allomorph | Gil بی |
| | Gil-lat-r bi |
| Environments | / #_ [C] [Ye] |

## 6.2 Dealing with Diacritics

In Gilaki some clitics are written only with a diacritic. While it is sometimes tricky to work with diacritics and get your cursor where you think it should be,[10] it does actually work to give a diacritic its own entry.

## 6.3 Phonemes and Natural Classes

When defining phonemes in FLEx in a system that uses both Arabic and Latin script it can be tricky deciding how to set it up because of the lack of a one-to-one relationship between Arabic

[10]In FLEx to move your cursor left or right including going before and after diacritics, use the F7 and F8 keys.

script phonemes and Latin script phonemes. For example a "ye" in Gilaki can correspond to "i" or "y" in Latin script. A Latin "s" can correspond to "sin", "sād" or "se" in Gilaki. The approach I've taken thus far is to define each Arabic script letter as a phoneme and give one corresponding Latin letter even though this creates duplicate Latin phonemes.

As far as Natural Classes are concerned, I've found it helpful to define a class that will cover both the Arabic script phonemes and the Latin. Even if you have to refer to a single phoneme, it is better to create a class that will correspond to the Arabic and Latin versions. If instead you write a literal phoneme into an environment rule, that rule will only be valid for the script in which that literal phoneme was written. The class [Ye] is an example you have noticed above.

## 6.4 Issues Arising from the ZWNJ character

In Gilaki Arabic script writing, the Zero Width Non-Joiner (ZWNJ[11]) character is used to keep two letters from joining while keeping them next to each other. For some enclitics in Gilaki there is an allomorph with a ZWNJ and one without it. Which one is used depends on the preceding letter.[12] In Latin script this is not a problem; there would be one allomorph instead of two. So how do we have an entry that contains both the Arabic script versions of the allomorphs and the Latin script version? My solution has been to keep the Latin form with the Arabic form that has the ZWNJ. The Arabic allomorph form that does not have the ZWNJ would stand alone without a Latin version.

## 6.5 Entering Texts and Interlinearizing Them

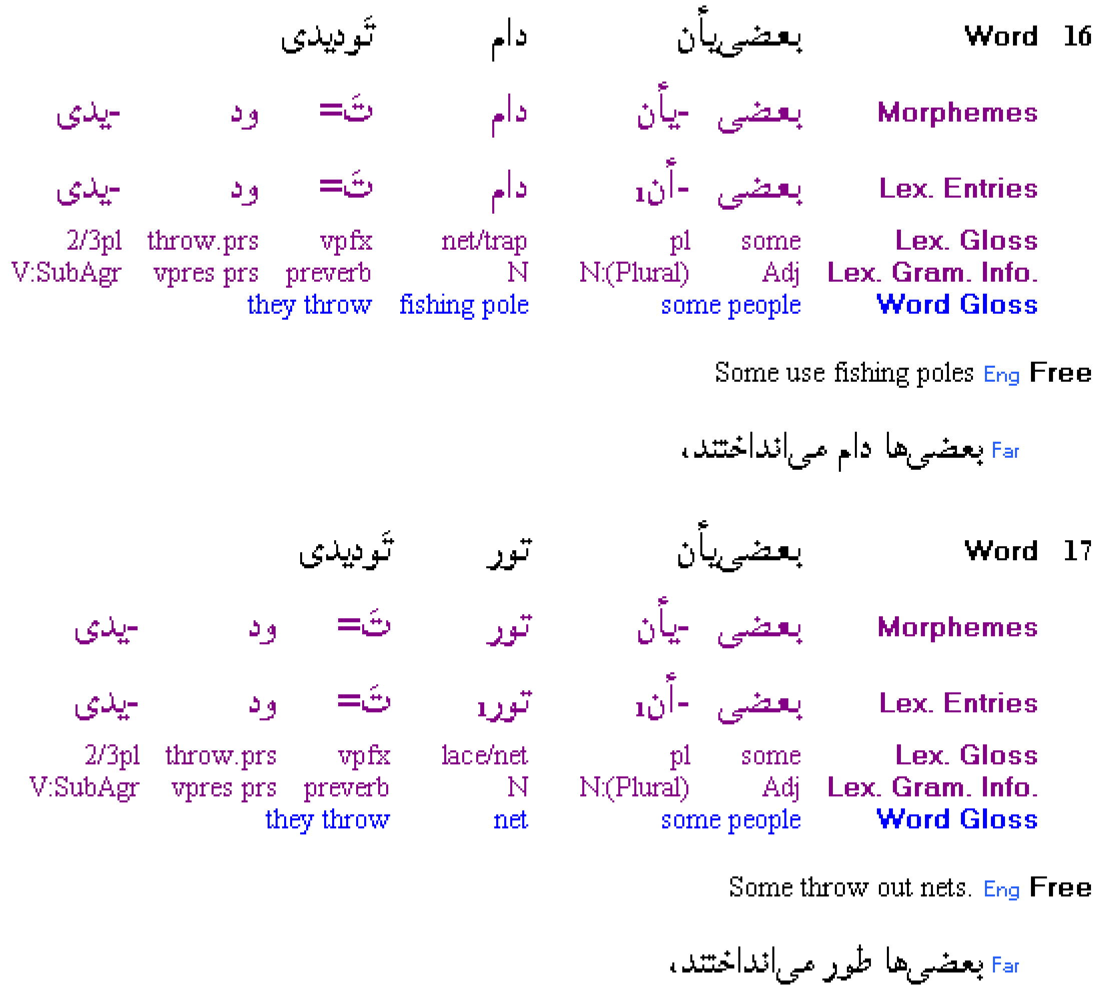
When using multiple writing systems in FLEx (i.e. Arabic script and Latin script), you have to keep straight which script you are currently working with. One place this comes up is when you are inserting a new text into FLEx. When you make the insert you are asked which writing system the text is written in. Once this is chosen the text must be written in that writing system and you cannot change to another writing system.

Working on interlinear texts in FLEx that are in Arabic script is pretty much the same as with Latin script. The main thing you have to get used to is working with the right-to-left orientation of everything. Here is a sample of interlinear text with Arabic script.

[11]Unicode value: 200C.

[12]I had a choice to make in my analysis. Either I had to create two allomorphs of the stem, one with the ZWNJ and one without or create two allomorphs of the enclitic. The simpler solution seemed to be to have the duplication in one enclitic entry instead of in many root entries.

(41) Interlinearized Arabic script text



| | | | | | | |
|---|---|---|---|---|---|---|
| Word 16 | بعضی‌یأن | | دام | تَودیدی | | |
| Morphemes | بعضی | -یأن | دام | تَ= | ود | -یدی |
| Lex. Entries | بعضی | -أن1 | دام | تَ= | ود | -یدی |
| Lex. Gloss | some | pl | net/trap | vpfx | throw.prs | 2/3pl |
| Lex. Gram. Info. | Adj | N:(Plural) | N | preverb | vpres prs | V:SubAgr |
| Word Gloss | some people | | fishing pole | they throw | | |
| Free | Eng Some use fishing poles | | | | | |
| | Far بعضی‌ها دام می‌انداختند، | | | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| Word 17 | بعضی‌یأن | | تور | تَودیدی | | |
| Morphemes | بعضی | -یأن | تور | تَ= | ود | -یدی |
| Lex. Entries | بعضی | -أن1 | تور1 | تَ= | ود | -یدی |
| Lex. Gloss | some | pl | lace/net | vpfx | throw.prs | 2/3pl |
| Lex. Gram. Info. | Adj | N:(Plural) | N | preverb | vpres prs | V:SubAgr |
| Word Gloss | some people | | net | they throw | | |
| Free | Eng Some throw out nets. | | | | | |
| | Far بعضی‌ها طور می‌انداختند، | | | | | |

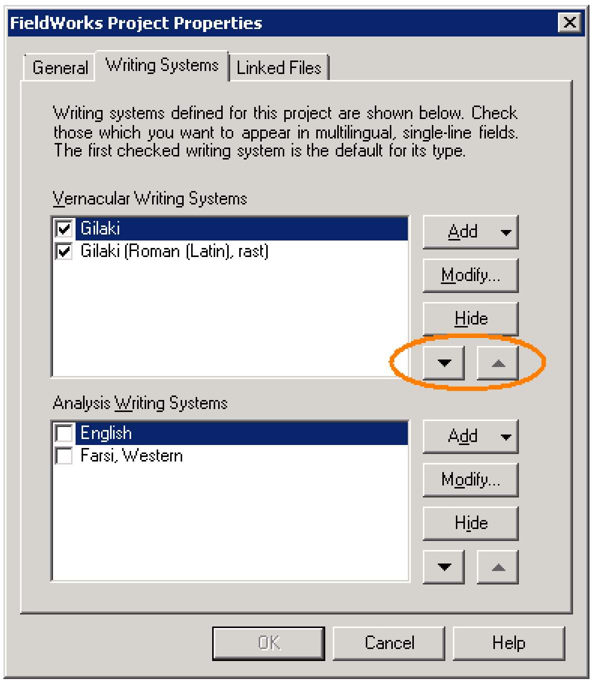
## 6.6 Default Writing System

When using a FLEx project with multiple writing systems, one writing system is always assigned as the default vernacular writing system. It is important to know which is your default because that will be the writing system that searches usually will be done in[13] and some columns will be initially set to display the default writing system. In relation to parsing the default vernacular writing system is what will be used in the "Try a Word" parsing tool. Also phonemes and environments will be keyed off of this writing system.

It is easy to switch which writing system is your default vernacular writing system using the arrow buttons shown in the window below.

[13]In some searches like Find Entry in the Lexicon Edit view the default writing system is selected, but you can choose to search in another writing system.

(42) Buttons for changing default writing system



FieldWorks Project Properties

General | Writing Systems | Linked Files

Writing systems defined for this project are shown below. Check those which you want to appear in multilingual, single-line fields. The first checked writing system is the default for its type.

Vernacular Writing Systems

- [✓] Gilaki
- [✓] Gilaki (Roman (Latin), rast)

Analysis Writing Systems

- [ ] English
- [ ] Farsi, Western

## 7 Uses for Machine Parsing in FLEx

One could argue that it is very helpful to set up a parsing system for a language just in order to develop a good model of how the morphology of the language works. It also could be argued that it is a good tool for discovering what is going on in a language. Besides these things I want to discuss two practical benefits of setting up parsing for a language in FLEx.

### 7.1 Automated Interlinear Texts

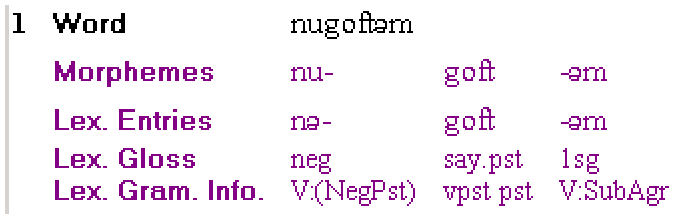
Interlinear texts are a critical component of language documentation and analysis. FLEx does a very good job of helping you create interlinear texts and has a tight link between the lexicon and texts. Breaking words into morphemes and glossing them can be done manually in the Interlinear Texts view of FLEx or it can be done more automatically with the parser.

Using the manual method is not bad and FLEx assists greatly by looking up morphemes and giving suggestions as to which lexical item it may be. It will also remember a word that was "approved" as an analysis and suggest that analysis the next time the word appears in the text.

Using the parser to assist in breaking words into morphemes is an even better method assuming you have at least some of the preparation work above. Of course the more preparation work you do the more information you make available to the parser allowing for better overall results.

The biggest advantage of using the parser for automatically creating analyses for words is that it prevents human error. It is quite easy to choose morphemes the manual way, but they may actually be the wrong forms and not what you want. These errors magnify when there are many homophonous forms that are distinct lexical entries. Let's look at an example of a manual parse versus a machine parse. Let's take the word nugoftəm as an example.

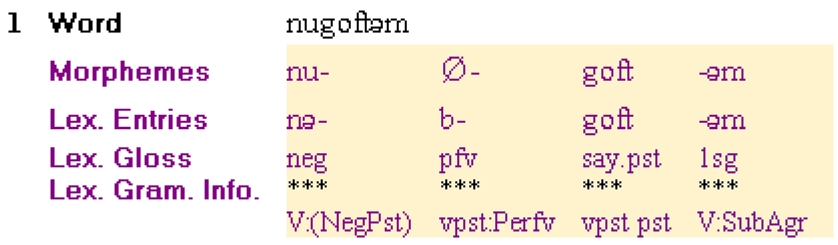
(43) Manual parse



| | | | | |
|---|---|---|---|---|
| 1 | **Word** | nugoftəm | | |
| | **Morphemes** | nu- | goft | -əm |
| | **Lex. Entries** | nə- | goft | -əm |
| | **Lex. Gloss** | neg | say.pst | 1sg |
| | **Lex. Gram. Info.** | V:(NegPst) | vpst pst | V:SubAgr |

In (43), I have broken the word into morphemes manually. One prefix and one suffix. There's a problem here though. In (23) above, we were requiring there to always be a perfective prefix which is sometimes realized as a null prefix. So in (43) I have forgotten to put the null prefix there. If I do a machine parse of this word, a parse such as (43) will not even be suggested because it is not valid according to my perfective template in (20). The parse I get is shown below in (44). The background is a peach color which indicates that the suggested parse or analysis is from the parser. Note that this parse includes the null perfective prefix.

(44) Machine parse

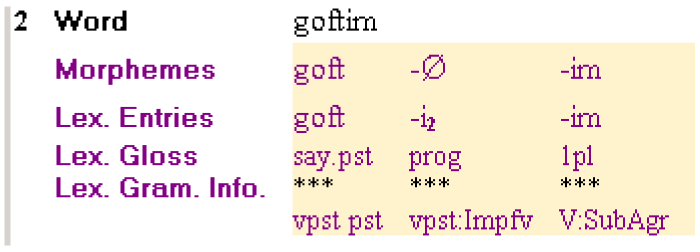


| | | | | | |
|---|---|---|---|---|---|
| 1 | **Word** | nugoftəm | | | |
| | **Morphemes** | nu- | ∅- | goft | -əm |
| | **Lex. Entries** | nə- | b- | goft | -əm |
| | **Lex. Gloss** | neg | pfv | say.pst | 1sg |
| | **Lex. Gram. Info.** | *** | *** | *** | *** |
| | | V:(NegPst) | vpst:Perfv | vpst pst | V:SubAgr |

Throughout your text the FLEx parser will have these parser suggested analyses, so a lot of the manual work of breaking the words into morphemes and selecting the correct lexical entry is saved. Also it prevents the human error of incorrectly parsing the word.

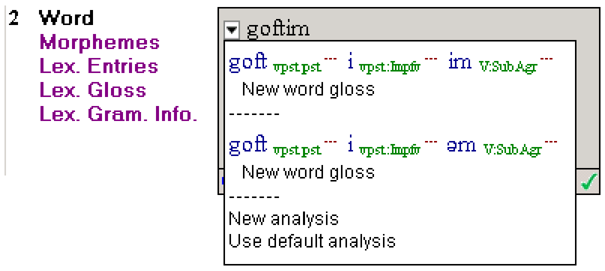
There are of course going to be times when the parser suggests more than one analysis in the interlinear text. How does this function? Let's take an example word we used above, nugoftidi. We expect the parser to give us two parses as shown in (27). By default, the word will get set to the first analysis that the parser comes up with. Again the background will be in a peach color. It will look like (45) below. When we visit this word in the text and click on the arrow to the left of the word we see two suggested parses in the list of analyses to choose from as shown in (46) below. If I select the second one the end result is shown in (47). This is the process you have to go through when there are multiple parses suggested by the parser.

(45) First machine parse



| | | | | |
|---|---|---|---|---|
| 2 | **Word** | goftim | | |
| | **Morphemes** | goft | -∅ | -im |
| | **Lex. Entries** | goft | $-i_2$ | -im |
| | **Lex. Gloss** | say.pst | prog | 1pl |
| | **Lex. Gram. Info.** | *** | *** | *** |
| | | vpst pst | vpst:Impfv | V:SubAgr |

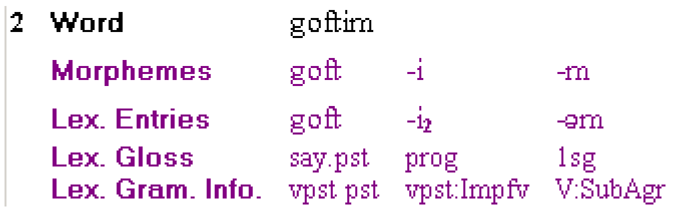
(46) List of two machine parses



| | |
|---|---|
| 2 Word | goftim |
| Morphemes | |
| Lex. Entries | |
| Lex. Gloss | |
| Lex. Gram. Info. | |

```
goft vpst:pst  i vpst:Impfv  im V:SubAgr
  New word gloss
-------
goft vpst:pst  i vpst:Impfv  əm V:SubAgr
  New word gloss
-------
New analysis
Use default analysis
```

(47) Second machine parse after being selected



| | | | |
|---|---|---|---|
| 2 Word | goftim | | |
| Morphemes | goft | -i | -m |
| Lex. Entries | goft | -i₂ | -əm |
| Lex. Gloss | say.pst | prog | 1sg |
| Lex. Gram. Info. | vpst pst | vpst:Impfv | V:SubAgr |

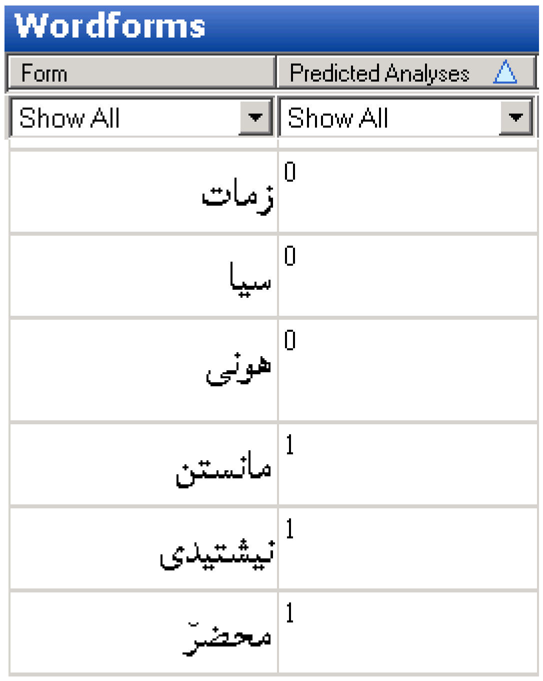
I think you can see how the parser does a lot of the work for you. This can save hours and hours of work when dealing with a lot of text data.

## 7.2 Morphological Spelling-Checking

Another practical help that I want to point out is that of doing morphological spelling-checking of a text. When you run the parser on a text FLEx will compile a list of all of the analyses for all the words in that text. By looking at the results, you can see very easily which words have been parsed successfully and which have not. Here are some examples to illustrate.

Let's say you have a new text where you want to check the spelling. The first step is to create a new text and paste in the text contents. Assign a Genre temporarily to this text that is unique from all other texts. Then tell FLEx to parse all the words in this text. Next, display the Word Analyses view and filter for the Genre you gave to the text. Sort the list of words by Predicted Analyses to quickly see which words have 0 or more analyses. (48) below shows a sample list.

(48) Portion of the wordforms view



**Wordforms**

| Form | Predicted Analyses |
|---|---|
| Show All | Show All |
| زمات | 0 |
| سیا | 0 |
| هونی | 0 |
| مانستن | 1 |
| نیشتیدی | 1 |
| محضر | 1 |

When the predicted analyses column shows 0, this means the parser could not parse the word. The word could be misspelled, it could be that the word stem is not in the lexicon or it could be that your analysis model is not allowing a well-formed word.

This kind of morphological spelling-checking is very useful for languages with lots of morphemes. It can get very cumbersome to maintain a list of all possible inflections of a word in a spelling list. By doing a morphological spelling-check with FLEx you quickly see which words are ill-formed.

## References

Black, Andrew H. 2010. A Conceptual Introduction to Morphological Parsing for Stage 1 of the Fieldworks Language Explorer. SIL International. Manuscript.

Rastorgueva, V.S. et al. 1971. *Giljanskij jazyk.* Moscow: Nauka.