# The Morphosyntactic analysis of Georgian

*Paul Meurer*
*Uni Research Computing, Bergen, Norway*

This article presents the morphosyntactic analysis of Georgian, as it is carried out in the Georgian National Corpus (GNC) project.

By morphosyntactic analysis we mean the assignment of a lemma form and morphosyntactic features to each word or token in a text. This is an intuitive but rather vague formulation; below, I will give a more precise and technical definition of this process.

The morphosyntactic analysis of a language is a complex task that involves several steps, each having its theoretical underpinnings and requiring a dedicated tool. In the case of the Georgian National Corpus, additional complexity is introduced by the diachronic nature of the corpus: the analyzer should be able to handle not only Standard Modern Georgian, but also the historical varieties Old Georgian and Middle Georgian, with all their variants and the variation within each of them, as well as older, non-standard and dialectal variants of Modern Georgian.

Because of this, I will not be able to give a full description of the morphosyntactic analyzer, rather, I will try to draw a broad outline of the steps involved, and to explain the techniques used by way of typical examples. I will neither be able to give a full account of the software tools used, or to fully explain their theoretical background; here, I refer to the literature listed in the bibliography section.

At the end of the article, I offer a full overview of the morphosyntactic feature inventory used in the analyzer.

A morphosyntactic analyzer operates in a modular fashion. The input is a text in the form of a string of characters; the result of one processing step is handed over to the next module, and the output of the last step is the desired morphosyntactic analysis.

In which concrete format the input text and the output analysis is given is very much dependent on the application; in the context of the Georgian National Corpus, the input consists of XML documents adhering to TEI (Text Encoding Initiative) standards, and the output is again a TEI-conformant XML document with added tokenization markup and stand-off grammatical annotation. (That is, the grammatical annotation resides in an extra document, where readings of words are linked to the words in the main document by unique identifiers.)

The main analysis steps are the following:

1. Tokenization
2. Sentence splitting
3. Lexicon lookup
4. Disambiguation

Optional markup: Named entity recognition

## 1. Tokenization

A text is given as a string of characters (e.g., Unicode code points), where characters can be alphabetical, numerical, punctuation, symbol, or whitespace characters. Roughly, a token is any minimal meaningful textual element: a word, a symbolic or numerical expression, or a punctuation mark (full stop, comma, left/right quote, braces etc.). Normally, tokens are separated by whitespace, except when they are immediately preceded or followed by punctuation. Thus, tokenization is performed by splitting a string (representing a text) at spaces (whitespace characters like Space, Tab, Newline) and punctuation.

However, there are situations where this simple procedure is not adequate:

A dot can be used as an end-of-sentence full stop (in which case it should be split off as a separate token), but it also has several other usages: It marks abbreviations (e.g., ქ. თბილისი) and should then be part of the abbreviation. It is also used in ordinal numbers, in list numbering and in symbolic expressions, sometimes even inside an expression (e.g., მე-IV., 2., ა., VI., *www.gnc.org*). The tokenizer should be aware of all such cases and treat the dot as part of the token.

Similarly, parentheses, braces and quotes can be used and misused in several ways: a letter with a closing parenthesis (and missing opening parenthesis) might denote a list bullet point und thus should be treated as a single token.

A steady source of problems are wrong quotation marks: instead of proper opening and closing quotation marks („…“), simple straight quotes (") are often found in texts. Although they do not represent a problem in tokenization since quotes are always treated as separate tokens, straight quotes are unesthetical and simply wrong from a typographical perspective, and the information whether a straight quote represents an opening or a closing quote is lost when the token is viewed in isolation. This information is however crucial when a text is to be reconstructed from the list of its tokens, when whitespace information has been discarded and must be reconstructed from the tokens alone. (In a corpus application, only the tokens are stored, whereas whitespace is discarded, and whitespace reconstruction works perfectly well when all quotes are correct.) Similarly problematic are single straight quotes, which are often used instead of apostrophes, commas or correct single quotes. For this reason, quotes are normalized in a preprocessing step, such that all straight quotes are replaced by the proper left or right quotes, in accordance with Georgian orthography and typography standards („…“ or »…«; ‚…‘ or ›…‹) before a text is tokenized.

In some cases it is really difficult to distinguish between an abbreviation dot and a full stop. Here, we need heuristic rules to make a decision, which will not be correct in every case. Consider for example the abbreviation მან. for მანათი, as in

> ფასი 1 მან. სამუსლიმანო საქართველო ჰასან თაჰსინ ხიმშიაშვილის რედაქტორობით.

Since მან is also a common personal pronoun, it cannot be decided without considering the context that we have an abbreviation of მანათი in this example. Here, a heuristics could be to go for the abbreviation when it is preceded by a cardinal number. In this example, the interpretation of the dot even influences the division of the text into sentences, which leads us to the next step.

## 2. Sentence splitting

Sentence splitting is the process of dividing a tokenized text string into sentences. Normally, sentences do end in a full stop, a question mark or an exclamation mark. There are however a few exeptions to this which make sentence splitting more complex.

– A sentence may end in ellipsis (…), which can also be used sentence-internally.

– Headlines are normally missing punctuation; here we need additional markup (e.g., an XML element <*head*>…</*head*> enclosing the headline) that helps identifying the sentence end.

– As we have seen, dots can be used in abbreviations and number expressions.

In languages that have upper/lower case distinction (like most European languages), sentences written in standard orthography normally begin with a capitalized word (a word with uppercase first letter). This extra cue can often be used to unambiguously identify a preceding dot as an abbreviation marker. (If a dotted word is followed by a lowercase word the dot cannot normally be a full stop.)

Since the Georgian (მხედრული) script has no case distinction, it is slightly more difficult to do sentence splitting in Georgian. For the same reason, abbreviations cannot be recognized as easily. Additionally, proper names are more difficult to detect, as they do not, like in European languages, begin with an upper-case letter.

## 3. Lexicon lookup

Steps 1 (tokenization) and 2 (sentence boundary recognition) are part of the preprocessing stage; Lexicon lookup is the first step in the morphosyntactic annotation of the tokenized text.

In lexicon lookup, every word or token is looked at in isolation, and all possible morphosyntactic readings are attached to it, irrespective of context. A reading is a possible morphosyntactic analysis of a word, consisting of a lemma form and a set of morphosyntactic features (also called tags). Sometimes, lemmatization, the process of assigning one ore more lemma forms to a word, is considered a separate step; when using a transducer, lemmatization and the assignment of morphosyntactic features is done simultaneously in the lexicon lookup stage. In Georgian, as in virtually all languages, words are often ambiguous and have more than one possible reading.

*Examples*:

A word form can be ambiguous for case (case syncretism):

გოგოს →
  გოგო N Hum Gen Sg
  გოგო N Hum Dat Sg

A word form can be derived from two totally unrelated lemmas:

და →
  და Cj
  დ[ა] N Hum Nom Sg

ვაშლი →
ვაშლ-ი N Nom Sg
შლ[ა]/შლ V Act Pres LV …

Two homonyms may differ in their conjugation pattern, which is marked in the lemma form:

თვალი →
თვ[ა]ლ-ი N Nom Sg
თვალ-ი N Nom Sg

(The first one syncopates, the second does not.)

Verb forms are often ambiguous:

დაუხატავს →
და-ხატვ[ა]/ხატ V Act Perf Pv S:3Sg DO:3
და-ხატვ[ა]/ხატ V Act Fut Pv OV S:3Sg DO:3 IO:3
და-უხატავ-ი/ხატ Part NegPart Dat Sg

## 4. Disambiguation

For corpus applications, it is desirable to disambiguate the morphological analysis as much as possible. Most word forms are ambiguous in isolation, but when we take the sentence context that a word in a text is found in into account, only one of all possible readings makes sense. In grammatical terms, it is the syntax of the sentence, and, to a lesser degree, its semantics, that can help disambiguating a word form. (Forms of homonymous lemmas can often only be disambiguated on semantic grounds, e.g., თვალი *eye* and თვ[ა]ლი *gemstone*.)

There are two principal approaches to disambiguation: statistical and rule-based taggers. A statistical tagger has no linguistic knowledge; it makes its decisions as to which reading to choose in a given context solely on the basis of a "gold standard" (a manually disambiguated set of sentences) it is trained on.

In contrast, in a rule-based tagger, the grammar rules are written by a linguist.

A rule-based approach has advantages over a statistical approach: it is better suited for rich tagsets; the ambiguity/precision ratio can be controlled, and residual ambiguity can be eliminated manually or with a statistical module. An obvious disadvantage is that it is quite time-consuming to write a rule set of adequate detail; it also requires a thorough knowledge of the language.

We have chosen a rule-based formalism: Constraint Grammar (CG) (Fred Karlsson version 1 (1990), Eckhard Bick version 3).

**Constraint Grammar**

A Constraint Grammar rule set consists of rules of several diffent types that operate on words and their lemma forms and readings in a sentence context. The most important rules are Select rules and Remove rules. A Select rule selects the readings it matches when applied to a word, and a Remove rule discards the readings it matches.

The Constraint Grammar rule set is applied to a single sentence at a time. The rules are applied successively, in the order they appear in the grammar, to all tokens in the sentence in token order. This process is repeated until no rule applies any more, and only then, the next sentence is considered.

A constraint grammar rule has a matching part and an optional context condition part. In general, a rule looks like this:

SELECT *match-condition* [ IF *context-condition* ] ;

or

REMOVE *match-condition* [ IF *context-condition* ] ;

The match-condition expresses a match in terms of a word form, a lemma form, and a boolean expression in the feature set. Each of these components is optional.

A simple example is a rule that unconditionally selects the verbal reading of არის:

SELECT ("<არის>" *V*) ;

Here, the match condition is specified in terms of a word form (არის; word forms are enclosed in quotes and angular brackets, whereas lemma forms are set in quotes only) plus a morphosyntactic feature (*V*).

In a Select rule, the readings that match the match condition are kept, whereas all others are discarded. Thus, the result of the application of this rule will be that from the original readings:

არის →
ყოფნ[ა]/არ V MedPass Pres <S> <S:Nom> S:3Sg
ყოფნ[ა]/არ V MedPass Pres <AuxIntr> <S:Nom> S:3Sg
არ[ა] N Gen Sg
არ[ე] N Gen Sg

only the first two, the verb (*V*) readings, will be kept. Since the rule is unconditional, missing the IF part, it will remove the noun (*N*) readings irrespective of context, which is a good approximation, but might be too strict.

The same effect could have been achieved by a Remove rule that discarded the matching noun readings:

REMOVE ("<არის>" *N*) ;

Context conditions constrain the context the matching word must occur in for the rule to apply. Context conditions can refer to numbered positions relative to the token

position; a negative number refers to positions before the token, and positive numbers to positions after the token.

The following rule selects the Optative/Conjunctive Perfect/Pluperfect reading of a verb (and discards possible Aorist or other readings) if the preceding token is a modal verb (that is, უნდა).

SELECT *OPT* IF (-1 (*V Modal*)) ;

Here, *OPT* is a shorthand for either *Optative, Conjunctive Perfect* or *Pluperfect*, defined by a *LIST* expression in the grammar file:

LIST *OPT* = *Opt ConjPres PluPerf* ;

Thus, the rule selects all *Opt, ConjPres* and *PluPerf* readings of a token if the preceding (-1) token has a *V Modal* reading. It is in fact sufficient that one of the readings is *V Modal*, even if other readings are present the rule will be triggered. Similarly, the rule

SELECT (*V Modal*) IF (1 *OPT*) ;

will select the modal reading of უნდა if it is followed by an Optative.

In the analysis of the sentence უნდა დაწერო, the first Select rule will select the *Opt* reading of დაწერო, while the second one selects the modal reading of უნდა, such that the readings marked with a star will be removed.

```
უნდა →
    უნდა V Modal
    ნდომ[ა]/ნდ V MedPass Inv Pres OV *
დაწერო →
    და-წერ[ა]/წერ V Act Opt …
    და-წერ[ა]/წერ V Act Fut … IndSpeech3 *
```

If it is necessary that all readings be unambiguously of a certain type, the symbol *C* (standing for *Cautious*) has to be added to the position number, as in the following rule that discards *V* readings if the token is preceded by an unambiguous negation item:

REMOVE (*V*) IF (1C *Neg*) ;

Context conditions can be negated, like in the following rule, which selects the conjunction reading of და except when it is preceded by a nominative possessive pronoun:

SELECT ("და" *Cj*) IF (NOT -1 (*Poss Nom*)) ;

Thus, in the phrase მე და შენი და, the Conjunction reading of the first და will be selected, whereas the readings of the second და will be unaffected by the rule:

მე →
  მე Pron Pers 1 Nom Sg
და →
  და Cj
  დ[ა] N Hum Nom Sg *
შენი →
  შენ-ი Pron Poss Poss2Sg Nom Sg
და →
  და Cj
  დ[ა] N Hum Nom Sg

The value a feature expression (e.g. *CASE* in the following example) takes in a match can be referred to later in the rule via the $$ shorthand:

SELECT *QUAL* $$*CASE* IF (1 *Prop* $$*CASE*) (1C *Prop*) ;

This rule selects a qualifier noun (as occurring in title expressions and the like: ბატონი, მინისტრი, მასწავლებელი, მწერალი,…) in reduced case if it is immediately followed by a proper noun (*Prop*) in the same case (here, the reference comes into play: $$*CASE*), and the token is unambiguously a proper noun.

*CASE* is defined as a *LIST* of all possible cases:

LIST *CASE = Nom Erg Dat Gen Advb Inst Voc*;

and *QUAL* is defined as a set of features:

SET *QUAL = N* (*Qual*) (*Red*);

In the phrase ბატონ გივისთან, this rule selects the correct Dative reading of ბატონ.

ბატონ →
  ბატონ-ი N Hum Qual Dat Red
  ბატონ-ი N Hum Qual Gen Red *
  ბატონ-ი N Hum Qual Inst Red *
  ბატონ-ი N Hum Qual Advb Red *
გივისთან →
  გივი N Prop Anthr FirstName Dat PP:თან

The context of a rule can also make reference to an indefinite range of positions: The position specifier 0* refers to all positions in the sentence, both to the left and to the right of the matching token. The range can be constrained by demanding that it should not cross a certain boundary or *BARRIER*.

The following rule tries to remove all Dative readings with a full, non-reduced case (- *Red*) that are not licensed by a verb having a 3rd Person Dative argument (*DatArg3*). Here, it makes sense to confine the context condition to a clause, which is the range that a verb licensing the Dative object has to be found in if there is one, so *BARRIER* is set to *CLB* (= Clause Boundary).

REMOVE *Dat - Red* IF (NOT 0* *DatArg3* BARRIER *CLB*) ;

A first approximation for *CLB* is the following list, which has to be refined further.

LIST *CLB* = "," ":" "რომ" "თუ" "რომელ-ი" "და" "როგორც" ;

*DatArg3* is defined as

SET *DatArg3* = (<*S:Dat*> *S:3Sg*) | (<*S:Dat*> *S:3Pl*)
| (<*DO:Dat*> *DO:3*) | (<*IO:Dat*> *IO:3*) ;

(The vertical bar (|) means "or".)

In the sentence კახამ გოგოს ძაღლი დაინახა, this rule correctly identifies გოგოს as Dative, since the verb დაინახა, which is in the same clause as გოგოს, has no Dative argument.

კახამ →
კახა N Prop Name FirstName Erg
გოგოს →
გოგო N Hum Gen Sg *
გოგო N Hum Dat Sg
ძაღლი →
ძაღლ-ი N Nom Sg
დაინახა →
და-ნახვ[ა]/ნახ V Act … <S-DO> <S:Erg> <DO:Nom> S:3Sg DO:3

Similarly, the following rule selects the *1st Singular Subject* reading of a verb if the verb unambiguously (0*C*) has a Nominative Subject (*S:Nom*) and there is a 1st person singular Nominative pronoun in the clause, where the *BARRIER* is either *CLB* or a second verb, but not a modal. (Modal უნდა is excluded from the barrier because it acts like an adverb in this respect and does not interfere with the argument and case syntax of the verb it modifies.)

SELECT (*V S*:1*Sg*) IF (0C *S:Nom*) (0* (*Pron Pers Nom* 1 *Sg*)
BARRIER *CLB* | (*V*) - (*Modal*)) ;

Thus, in the sentence მე უნდა გესაუბროთ, this rule selects the 1st Singular Subject reading of the verb:

მე →
მე Pron Pers 1 Erg Sg *
მე Pron Pers 1 Nom Sg
მე Pron Pers 1 Dat Sg *
უნდა →
უნდა V Modal
ნდომ[ა]/ნდ V MedPass Inv Pres … *
გესაუბროთ →
საუბარ-ი/საუბრ … Opt <S-IO> <S:Nom> <IO:Dat> S:3Sg IO:2Pl *

საუბარ-ი/საუბრ … Opt <S-IO> <S:Nom> <IO:Dat> S:1Sg IO:2Pl
საუბარ-ი/საუბრ … Opt <S-IO> <S:Nom> <IO:Dat> S:1Pl IO:2 *
საუბარ-ი/საუბრ … Opt <S-IO> <S:Nom> <IO:Dat> S:1Pl IO:2Sg *

Furthermore, the modal reading of უნდა will be selected by one of the rules we already encountered, and the Ergative reading of მე will be selected by the following rule:

SELECT (*Pron Pers Nom* 1 *Sg*)
IF (0* *S*:*Nom* *S*:1*Sg* BARRIER *CLB* | (*V*) - (*Modal*)) ;

There are still other constructions available in Constraint Grammar that are used in the rule set for Georgian; the interested reader is referred to the bibliography section.

As should be apparent from the preceding discussion, a CG rule set will not always totally disambiguate every sentence; in most cases, there will be tokens that still have more than one reading left. On the other hand, CG rules never discard all readings of a token; in this sense, CG is a robust formalism: it can handle any input, be it grammatical or not, and will give a more or less reasonable output. This sets CG and also statistical taggers off from deeper linguistic formalisms such as LFG (Lexical-Functional Grammar) or HPSG (Head-driven Phrase Structure Grammar), which can give very rich and linguistically meaningful analyses, but normally fail miserably on slightly ungrammatical (in the sense of not conforming to the implemented grammar) input. In addition, CG parsing is very fast and thus well-suited for the grammatical annotation of large corpora.

When writing a Constraint Grammar rule set, the grammar writer has to balance two goals against each other: the desire to make only correct disambiguations versus the aim to disambiguate as much as possible, that is, precision versus recall. Normally, grammar writing is an evolutionary process: first, only safe rules are written, then, more and more rules are added, in order to capture rarer and/or more complex phenomena. By testing the grammar on new material, or on a gold standard (a manually disambiguated subcorpus), rules will be revised, and new ones added. Here, the debugging and tracing facilities of the used Constraint Grammar implementation are important: when a rule set is run on a text, all rules that were triggered are marked on the respective (selected or discarded) readings, so it is easy to find out which rules applied in the wrong place.

If the ultimate goal is to end up with a fully and maximally correctly disambiguated corpus, measures have to be taken that go beyond Constraint Grammar rules.

Residual ambiguity can be eliminated by a statistical module, for example by always selecting the most probable reading, or in a more sophisticated manner. (It is still open how this will be done in the Georgian National Corpus project.)

As a last resort, errors can be corrected manually. To ease this work, manual correction will be available in a corpus search environment (e.g., Corpuscle, the software tool used in the GNC project), such that common errors can be found and corrected across a whole corpus.

**Named-Entity Recognition**

Named-entity recognition (NER) is the task of recognizing and classifying names (named entities) of diverse types, among them personal names, ethnonyms, toponyms, organization and brand names, book and film titles etc. Many named entities consist of more than one word. (As mentioned before, in European languages, named entities normally begin with an upper case letter, or are in all-uppercase. This, of course, does not apply to Georgian.)

There are several approaches to the Named-entity recognition task. The most straightforward one makes use of gazetteers, extended lists of precompiled names which can be consulted when tagging. More sophisticated methods exploit the syntactic and semantic surroundings a name typically occurs in. This can again be done with statistical and with linguistic, grammar-based techniques.

In the GNC project, we will mostly rely on the gazetteer approach, which will be backed up by Constraint Grammar rules that make context-based disambiguation decisions. In addition, many of the texts that are entering the Georgian National Corpus are already manually marked up for named entities, so little extra work has to be done here.

**The Morphosyntactic Analyzer**

The Morphosyntactic analyzer used in the GNC project is based on an analyzer for Modern Standard Georgian, which has been modularized to ease extension to other, both older and dialectal, varieties of Georgian. It is implemented as a finite state transducer in the software tool FST (Xerox Finite State Tool). Finite state transducers are the tool of choice for the implementation of morphological analyzers; there exist implementations for many languages.

A finite state transducer is a formalism that is an extension of regular expressions and finite state automata.

A regular expression is a string of characters that is composed of ordinary, literal characters and characters that have a special operator meaning, expressing for example optionality (?), arbitrary repetition (*), and alternative choices (|) of the ordinary characters or sub-expressions around them. Parentheses are used for grouping into sub-expressions. A technical definition of what it means for a string to match a regular expression would lead us too far. Instead, we consider as a basic, intuitive example the following expression: "(*da*(*m*|*gv*)*exmar*|(*m*|*gv*)*išvel*)*ee***t*?!*", which contains all the mentioned operators. It is easy to see that the strings *dagvexmare* and *mišveleeeeet!!!* both match this regular expression.

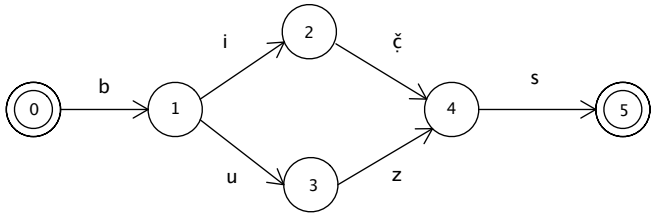
Whereas a regular expression is associated with a set of strings that match the expression, a transducer transforms a matching string into a set of other strings. Transducers are normally given as a set of transducer rules.

Both transducer rules and regular expressions can be transformed into and are equivalent to finite state automata. A finite state automaton is a set of state nodes and directed labeled edges between the nodes. (Not every pair of nodes needs to be connected by an edge.) There is one distinguished start node and a non-empty set of end nodes.

In the case of finite state automata corresponding to regular expressions, the edges are labeled by characters. A string matches a regular expression if there is a path

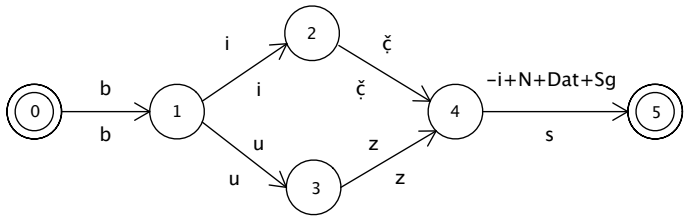
through the corresponding finite state automaton from the start state to an end state such that the characters encountered at the traversed edges amount to that string. If there is no such path, the string does not match the regular expression.

The following picture shows a finite state automaton that recognizes the two strings *biǩs* and *buzs* and corresponds to the regular expression "*b(iǩ|uz)s*" (or to any other equivalent expression). The node labeled 0 is the start node and 5 is the end node.




In contrast, the edge labels of transducer automata are pairs of characters, called upper and lower characters. It is easy to see that the lower side and the upper side of a transducer (obtained by considering only the upper resp. the lower characters) correspond to regular expressions. An input string that matches the lower side of a given transducer can be transformed into an output string that matches the upper side of the transducer: As we have seen, a matching (lower side) string corresponds to a path through the transducer such that the encountered lower side characters amount to the string. If we instead collect all upper side characters, we get an upper side string that obviously matches the upper side transducer. This string is the output string.

The transducer in the following picture accepts on its lower side the same two strings as the finite state automaton above, but it transduces them into the strings *biǩ-i+N+Dat+Sg* and *buz-i+N+Dat+Sg* on the upper side. If we, on the other hand, feed in e.g. *buz-i N Dat Sg* on the upper side, we get *buzs* as output on the lower side.




However, there might be more than one matching path through the lower side transducer. Since the corresponding upper side strings do not need to be equal, we see that an input string can have a set of strings as output. In this way, a transducer that implements the morphology of a language will be able to transform a word form on the lower side to a set of readings on the upper side, accounting for homonymy in the language.

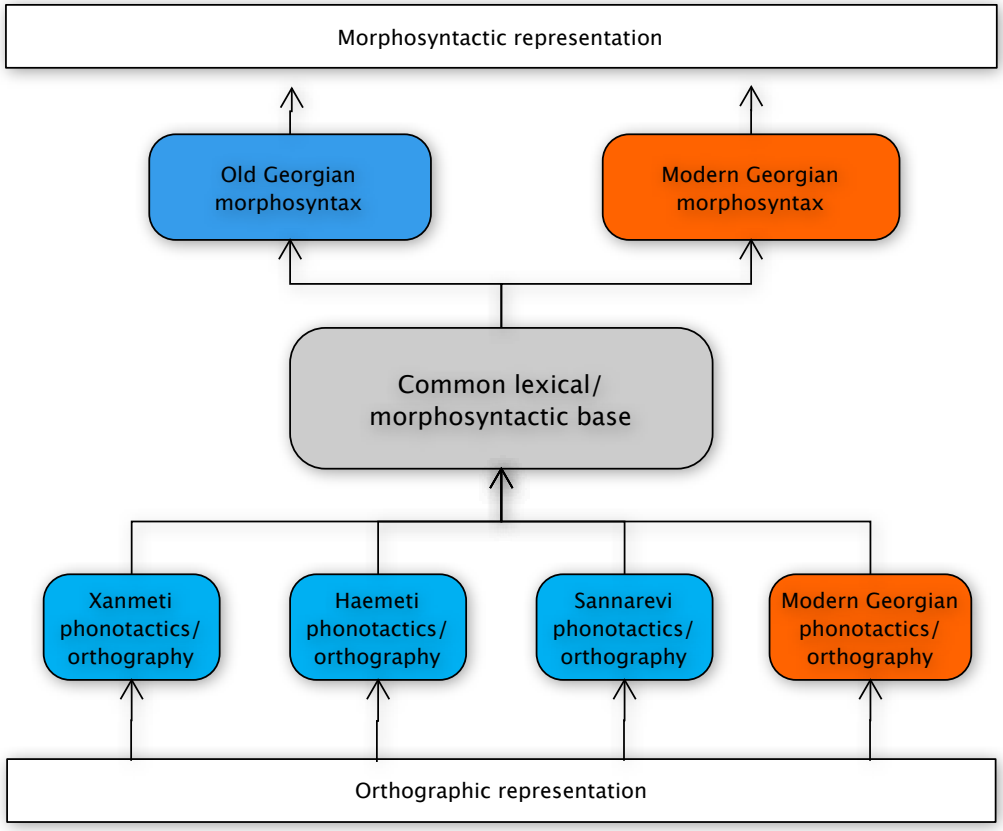
There is no formal difference between the upper and the lower side of a transducer, so it can be run in either direction. We can thus input a string on the upper side and receive an output on the lower side. In this way, a morphological analyzer can be run in generation mode, with all word forms that correspond to a given reading as output.

Transducers can be composed: the upper side output of one transducer is fed into an other transducer as input on the lower side. In this way, a morphological analyzer can be built up of modules, and modules can be reused.

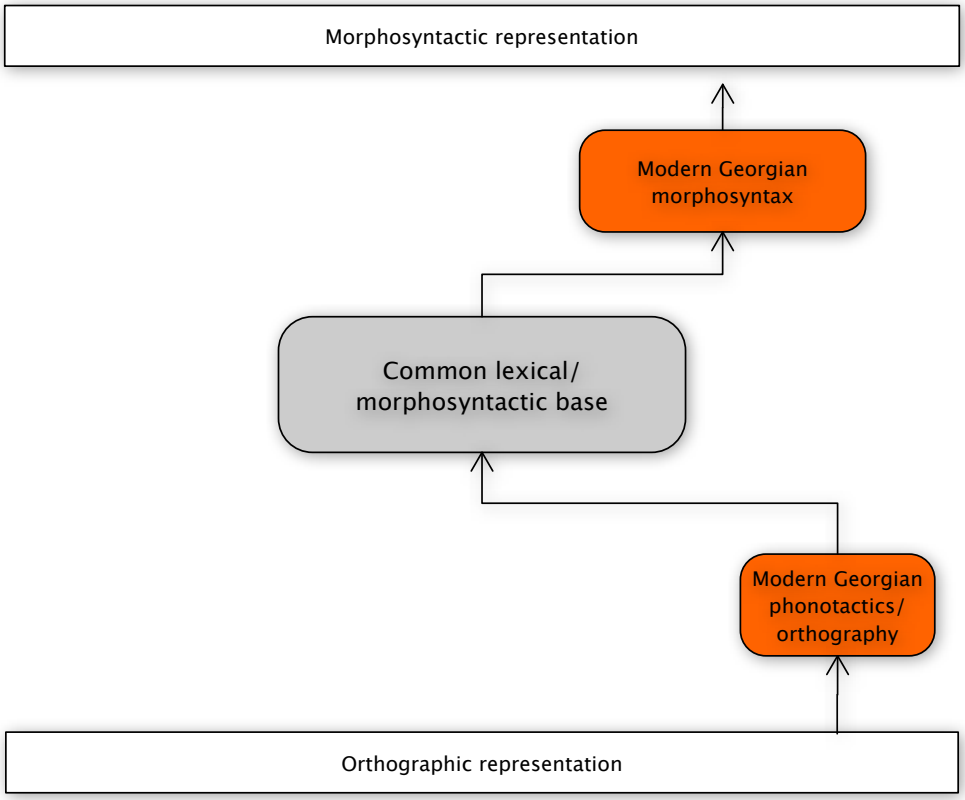
In the implementation of the Georgian morphological analyzer, there is a core module that contains the common lexicon for all historical varieties (Old Georgian with its varieties Xanmeti, Haemeti and Sannarevi, „Standard Old Georgian“, Middle Georgian, Modern Georgian) and the concatenative part of the morphology (morphophonemic processes are treated in other modules). The core module is common for all varieties, but certain affixes and forms are marked for language variety. The lexical basis of the core module is derived from major dictionaries of Old and Modern Georgian, as well as various word lists.

The input on the lower side of the core module is an intermediate morphophonemic representation. The output of the upper side are lemma forms plus morphosyntactic features. The core transducer is composed both on the lower and on the upper side with transducers that are specific for each language variety. The lower side transducers implement phonotactic and orthographic pecularities of the language varieties, whereas the upper side transducers treat morphosyntactic phenomena.

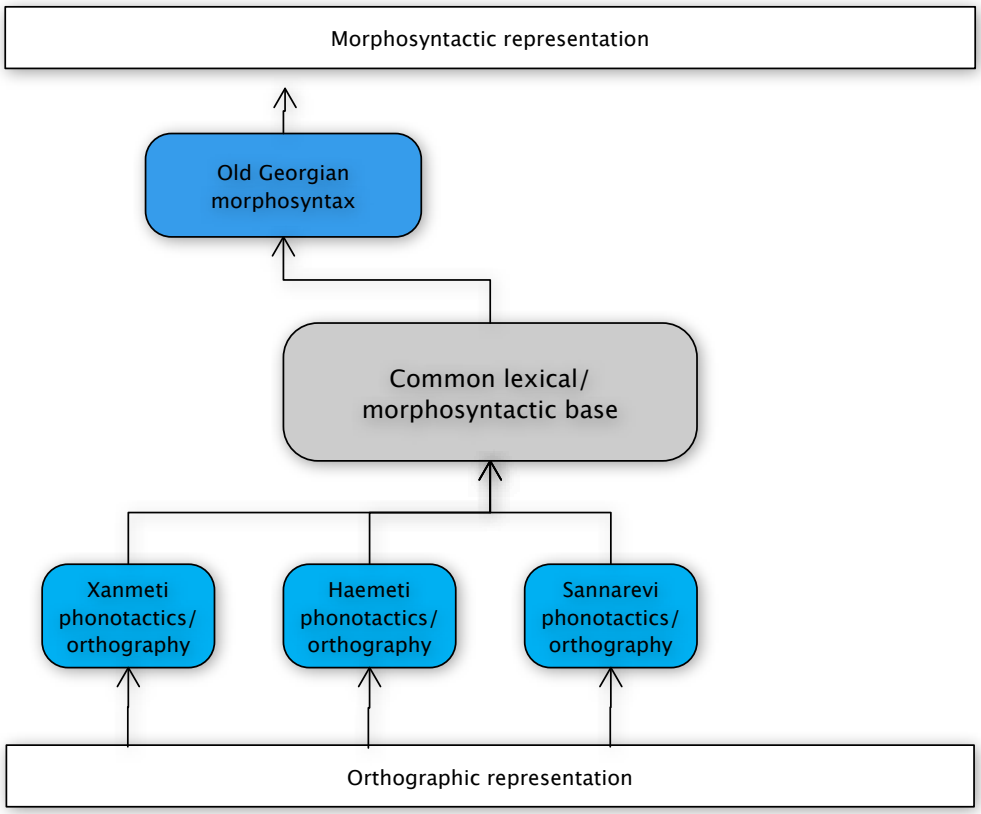
The following diagram gives an outline of the morphological analyzer in its entirety. (In order not to overload the picture, only Old Georgian and Standard Mordern Georgian have been included.) Each box represents a transducer module, arrows denote the composition of modules; that is, an arrow goes from the upper side of one transducer to the lower side of the next transducer. Altogether, a path through the analyzer goes from the orthographic representation of a word to its morphosytactic representation, which is the set of readings that can be assigned to the word.




When analyzing a text in a given language variety, only the modules belonging to that variety are active. Thus, for Modern Georgian, we get the following picture:




In the analysis of Old Georgian, the transducers depicted below are used.



Here, we have the choice between three sub-varieties: Xanmeti, Haemeti and Sannarevi. Depending on the text, one of the three phonotactics/orthography modules on the bottom has to be chosen. There are varieties of Old Georgian (e.g. those used in the manuscripts from the Sinai, Palestine and Athos) that exhibit orthographic pecularities; those will be integrated into the existing transducers as extensions.

**Phonotactics**

To give you a clearer picture of how these transducers are built up and work, I will show you the lower side output of the four phonotactics transducers (those on the bottom of the first diagram) for the following upper side input (going from the upper to the lower side):

*da#H<c̣er>, mi#vH<c>e, H<c̣er>, Hi<tku>mis*

In this intermediate representation, additional symbols are inserted that mark different morpheme boundaries: the hash symbol (#) marks the verbal prefix, and the root is inside angular brackets (<…>).

Here are the essential parts of those phonotactics transducers that derive the surface representation of the 2nd person Subject/3rd person Object/*i*-passive marker *H* (*x*/*h*/*s*). (Please keep in mind that these transducers are a much simplified version of the transducers in the actual implementation; they do not cover exceptions like *H* → *š*, or forms such as *xval*, *xar* etc., which are treated in different, specialized transducers.) The uppercase characters (e.g., *H*) are abstract symbols, only the lowercase characters correspond to Georgian letters.

```
define phtXanmeti
      [ v H → H    || _ u "<" ]
  .o. [ v H → H u || _ (?) "<"]
  .o. [ H → x ] ;

define phtHaemeti
      [ v H → H    || _ u "<" ]
  .o. [ v H → H u || _ (?) "<"]
  .o. [ H    → h ] ;

define phtSannarevi
      [ H → s  || _ "<" sRoot ]
  .o. [ H → 0  || _ ("<") Vowel ]
  .o. [ H → h ] ;

define phtMG
      [ H  → 0  || _ "<" sRoot ]
  .o. [ H  → 0  || _ ("<") Vowel ]
  .o. [ H  → h ] ;
```

All four transducers consist of three conditional replacement rules (in brackets: […])

that are composed (operator: .o.) to give the transducer. (There are many other rule types in the FST language; here we will focus on replacement rules only since they are both powerful and intuitive.)

Each conditional replacement rule is of the form

[ X → Y || L _ R ]

and operates by replacing *X* in the input (upper side) string by *Y* in the output (lower side) string, but only if *X* is preceded by *L* and followed by *R*. *L* and *R* are optional. If the context conditions are not matched, *X* appears unchanged on the output side.

Taking the example *mi#vH<c>e* in the Xanmeti case, the following is happening:

The first rule says: replace *vH* by *H* if it is followed by *u* and the root marker <. This is not the case in the input *mi#vH<c>e*, so the input goes unchanged to the next rule. The next rule says: replace *vH* by *Hu* if it precedes the root marker <, possibly with one character in between. (A pair of parentheses denotes optionality, whereas a question mark stands for an arbitrary character, so (?) denotes an optional arbitrary character.) Thus, the output will be *mi#Hu<c>e*.

In the next rule, *H* is unconditionally replaced by *x*, and we get *mi#xu<c>e*.

Finally, additional transducers remove the extra symbols and transform the output into the Georgian script, so we get *mixuce* and მიხუცე.

The table below shows the output of the four examples for the four different phonotactics transducers.

| Function | Intermediate rep. | Xanmeti | Haemeti | Sannarevi | Modern G. |
|---|---|---|---|---|---|
| DO3 | *daH<ċer>* | *daxċer* | *dahċer* | *dasċer* | *daċer* |
| IO3 | *mi#vH<c>e* | *mixuce* | *mihuce* | *mivsce* | *mivsce* |
| S2 | *H<ċer>* | *xċer* | *hċer* | *sċer* | *ċer* |
| i-Passive | *Hi<tku>mis* | *xitkumis* | *hitkumis* | *itkumis* | *itkmis* |

**Lemma form and feature set**

As discussed above, the morphosyntactic analysis of a word consists of one or more readings, and each reading is composed of a lemma (or base) form and a set of morphosyntactic features.

For nominals, the lemma form consists of the nominative form, where syncopation and apocope are marked by setting the vowel(s) in brackets. The nominative ending is set off with a hyphen.

Modern Georgian: გოგო, ძმ[ა], მეგობ[ა]რ-ი, ფანჯ[ა]რ[ა]
Old Georgian: მეფ[ე]-ჲ, მამ[ა]-ჲ

For verbs, the lemma form consists of a combination of the masdar (as a nominal in the nominative) and the verbal stem, if it exists.

წერ[ა]/წერს, ყოფნ[ა]/არ, ჯდომ[ა]/ზი

The lemma form of a participle, as a mixed nominal and verbal category, is a

combination of the participle and its verbal stem.

    და-წერილ-ი/წერ, ნაჭ[ე]რ-ი/ჭ[ე]რ

In the feature set of a reading, the order of the features is insignificant; however, the features are consistently ordered in all analyses. The first feature is always the part of speech (word class), the second, if present, is the subcategory, followed by morphological and syntactic features. For names, there may be name features after the subcategory.

Here is a list of almost all morphosyntactic and named-entity features, loosely ordered by feature type. For most features, an example reading is given in parentheses. (If a corresponding word form (followed by →) is given, the example reading is typically only one of several possible readings. The examples are from Modern Georgian if not specified otherwise.) The list of features is not fully set, there may occur slight changes in the future.

Word class features:

*N* – Noun (ღვინო N Nom Sg), with subclasses:
*Prop* – Proper noun (თბილის-ი N Prop Top Place Nom)
*Hum* – Human (კაც-ი N Hum Nom Sg)

Some nouns are additionally classified as:
*Qual* – Qualifying noun (პროფესორ-ი N Hum Qual Nom Red)
*Meas* – Measurement noun (ლიტრ-ი N Meas Nom Red)
*Temp* – Temporal noun (დრო N Temp Nom Sg).

Proper nouns can have these name categories:
*Anthr* – Anthroponym; *FirstName* (გია N Prop Anthr FirstName Nom)
*LastName* (ჩხატარაშვილ-ი N Prop Anthr LastName Nom)
*Top* – Toponym; *Place, Area, River, Sea* (ნორვეგი[ა] N Prop Top Area Nom)
*Org* – Organization (გაერო N Prop Org Nom Abbrev)

Abbreviations receive an *Abbrev* feature, in addition to the regular features of the word in question. (If an abbreviation is not marked for case etc., the respective feature is not set.) (ქ. → ქალაქ-ი N Abbrev)

*A* – Adjective (დიდ-ი A Nom Red), with subclass:
*Quant* – Quantifier (ყველა A Quant Nom Sg)

*V* – Verb (ცეკვავს → ცეკვ[ა]/ცეკვ V MedAct Pres <S> <S:Nom> S:3Sg)

*Part* – Participle, with subclasses:
*PresPart* (მოცეკვავ[ე]/ცეკვ Part PresPart Nom Red)
*PastPart* (გა-კეთებულ-ი/კეთ Part PastPart Nom Red)
*FutPart* (და-მწერ-ი/წერ Part PresPart Nom Red)
*NegPart* (და-უწერ[ე]ლ-ი/წერ Part NegPart Nom Red)

*Masdar* (წერ[ა]/წერ Masdar Nom Sg)

*Adv* – Adverb, with subclasses:
*Loc* – Local adverb (აქ Adv Loc)
*Temp* – Temporal adverb (დღეს Adv Temp)
*Mann* – Manner adverb (კარგად Adv Mann)
*Freq* – Frequency adverb (ერთხელ Adv Freq)
*Cause* – Cause adverb (ამისთვის Adv Cause)
*Int* – Interrogative adverb (რატომ Adv Int)
*Rel* – Relative adverb (სადაც → სად Adv Rel Encl:ც)
*Neg* – Negation adverb (არ Adv Neg), with subtypes:
*Pot* – Potential (ვერასოდეს Adv Neg Pot Temp) and
*Imp* – Imperative (ნუ Adv Neg Imp)

*Pron* – Pronoun, with subclasses:
*Pers* – Personal pronoun (მე Pron Pers 1 Erg Sg)
*Poss* – Possessive pronoun (შენ-ი Pron Poss Poss2Sg Nom Sg)
*Int* – Interrogative pronoun, with subtypes:
*Hum* (ვინ Pron Int Hum Nom) and
*Nonhum* (რა Pron Int Nonhum Nom)
*Indet* – Indeterminative pronoun with subtypes *Hum* and *Nonhum*
(ვინმე Pron Indet Hum Nom)
*Rel* – Relative pronoun, with subtypes *Hum* and *Nonhum*
(ვინც → ვინ Pron Rel Hum Nom Encl:ც)
*Refl* – Reflexive pronoun (თავ-ი Pron Refl Dat)
*Recip* – Reciprocal pronoun (ერთმანეთ-ი Pron Recip 3 Dat Sg)

*Dem* – Demonstrative (ეს Dem Nom)

*Num* – Numeral, with subclasses:
*Card* – Cardinal number, with subtypes:
*Digits* (2014 Num Card Digits)
*Alpha* (სამ-ი Num Card Alpha Nom Red)
*Range* (3-4 Num Card Digits Range)
*Approx* (ოთხიოდე → ოთხ-ი Num Card Alpha Nom Red Approx)
*Repet* (ორჯერ → ორ-ი Num Card Alpha Adv Repet)
*Roman* (XXI → 21 Num Card Roman)
*Letter* - Old Georgian letter numerals (კგ → 23 Num Card Letter)
*Ord* – Ordinal number, with the same subtypes
(მეშვიდ[ე] Num Ord Alpha Nom Red).

*Cj* – Conjunction (და Cj), with subclass:
*Sub* – Subordinating conjunction (რომ Cj Sub)

*Pp* – Postposition (and Preposition) (შესახებ Pp)

*Interj* – Interjection (ვაიმე Interj)

*Symbol* – Symbolic expression (& Symbol)

Nominals inflect for case and number, and can have double and triple case inflection. The features used are:

Case: *Nom, Erg, Dat, Gen, Advb, Inst, Voc* (კაცო → კაც-ი N Hum Voc Sg);
Old Georgian: *Abs*, *Dir*, *Ben*.

Number: *Sg*, *Pl*; for Plural, the two plural formants -ებ and -ნ/-თა are distinguished as *NewPl* (ბავშვ-ი N Hum Nom Pl NewPl) and *OldPl* (ხელთ → ხელ-ი N Dat Pl OldPl).

Double case: *DNom, DErg, DDat, DGen, DAdvb, DInst, DVoc*
(OG: კაცისაჲ → კაც-ი N Hum Gen Sg DNom DSg).

Double number: *DPl, DSg, DNewPl, DOldPl*
(კაცისანი → კაც-ი N Hum Gen Sg DNom DPl DOldPl,
ბავშვთაგანი → ბავშვ-ი N Hum Gen Pl OldPl PP:გან DNom DSg).

Triple case: *DDNom*, *DDVoc*, *DDAdvb*, *DDDat*, *DDErg*, *DDGen*, *DDInst*
(OG: კაცისაჲსასა → კაც-ი N Hum Gen Sg DGen DSg DDDat DDSg DDL).

Triple number: *DDSg, DDNewPl*, *DDOldPl*, *DDPl* (OG: კაცისაჲსანი → კაც-ი N Hum Gen Sg DGen DSg DDNom DDPl DDOldPl).

Long forms receive the feature *L* resp. *DL*, *DDL* (კაცისა → კაც-ი N Hum Gen Sg L, კაცისასა → კაც-ი N Hum Gen Sg L DDat DSg DL)

For reduced adjective or noun forms, the feature *Red* is used (კარგ → კარგ-ი A Advb Red). A truncated case ending (e.g., Advb. -ა instead of -ად) is marked as *Trunc* resp. *DTrunc* and *DDTrunc* (OG: ძედ იოსებისა → იოსებ N Prop Name FirstName Gen DAdvb DTrunc DSg).

When an opposition between bound and free pronouns exists, they are marked as *Bound* resp. *Free* (ვიღაც → ვიღაც Dem Indet Hum Bound; ვიღაცამ -> ვიღაც Dem Indet Hum Erg Sg Free).

For the person of personal pronouns, the features *1*, *2*, and *3* are used, whereas number is again marked by *Sg* and *Pl* (თქვენ Pron Pers 2 Erg Pl). For possessive pronouns, which mark the person and number of the possessor, the features *Poss1Sg*,… *Poss3Pl* are used (შენ-ი Pron Poss Poss2Sg Nom Sg).

Verbs are classified as:

*Act* – Active (აკეთებს → კეთებ[ა]/კეთ V Act Pres …)
*MedAct* – Medioactive (იმღერებს → მღერ[ა]/მღერ V MedAct Fut …)
*MedPass* – Mediopassive (ვიჯექ → ჯდომ[ა]/ჯექ V MedPass Aor …)
*Pass* – Passive (დავმალულვარ → და-მალვ[ა]/მალ V Pass Perf …)
*PassState* – Passive of State (ყრია → ყრ[ა]/ყრ V PassState Pres …)
*Caus* – Causative (დაარბევინებს → და-რბევინებ[ა]/რბ V Caus Fut …)

Mediopassives that have inverted case syntax are marked as *Inv* (მიყვარხარ → სიყვარულ-ი/ყვარ V MedPass Inv Pres OV …).

The following tense features are used:

Modern Georgian:
Present tense group: *Pres, Impf, ConjPres*
Future tense group: *Fut, Cond, ConjFut*
Aorist group: *Aor, Opt, Impv*
Perfect group: *Perf, PluPerf, ConjPerf*

Imperfective aorists and optatives get the *Imperfective* feature (ვრეკე → რეკვ[ა]/რეკ V Act Aor Imperfective …).

Old and Middle Georgian:
Present tense group: *Pres, IterPres, Impf, Impv-I, Conj-I, IterImpf*; *Fut* (Middle G.)
Aorist group: *Aor, Impv-II, Conj-II, Iter-II*.
Perfect group: *Perf, PluPerf, Conj-III, Iter-III*.

Verbal version is marked by the following features:
*SV* – Subjective version (ვიწერ → წერ[ა]/წერ V Act Pres SV …)
*OV* – Objective version (ვუწერ → წერ[ა]/წერ V Act Pres OV …)
*LV* – Locative version (ვაწერ → წერ[ა]/წერ V Act Pres LV …)

If a version marker is referentless, the feature is set in brackets: [*OV*] etc. (დავუკრავ → და-კვრ[ა]/კრ V Act Fut Pv [OV] <S-DO> …).

Verbs having a preverb are marked with *Pv*. In Old Georgian, there is a marker *PrePv* for preposed preverbs (წინა-დაუდვა → წინა-და-დებ[ა]-ე/დვ V Act Aor PrePv Pv OV <S-DO-IO> …). Composite verbs are marked with *VComp* (ღაღად-ყო → ღაღად-ყოფ[ა]-ე/ყ[ავ] V VComp Act Aor Pv …).

Verbs are marked for their arguments or actants (their subcategorization frames) by the following features:

<*Null*> – no arguments (თოვს → თოვ[ა]/თოვ V MedAct Pres <Null>)
<*S*> – Subject only
(დაბრუნდა → და-ბრუნებ[ა]/ბრუნ V Pass Aor Pv <S> <S:Nom> S:3Sg)
<*S-DO*> – Subject and Direct object
(გაუკეთებია → გა-კეთებ[ა]/კეთ V Act Perf Pv <S-DO> …)
<*S-IO*> – Subject and Indirect object
(უზის → ჯდომ[ა]/ზი V MedPass Pres OV <S-IO> …)
<*S-DO-IO*> – Subject, Direct object and Indirect object
(მომეცი → მი-ცემ[ა]/ც V Act Imper Pv <S-DO-IO> …)

Auxiliary verbs are differentiated in the following way:

<*AuxIntr*> – Intransitive auxiliary verb
(იყო → ყოფნ[ა]/ყ[ავ] V Pass Opt <AuxIntr> <S:Nom> S:2Sg)
<*AuxTrans*> – Transitive auxiliary (with non-Human object)

(მქონდა → ქონ[ა]/ქონ V MedPass Inv Impf <AuxTrans> …)
<*AuxTransHum*> – Transitive auxiliary with Human object (ეყოლებათ → ყოლ[ა]/ყოლ V MedPass Inv Fut <AuxTransHum> …)

The following features concern the verbal person marking and argument case syntax.

Subject person and number:
*S:1, S:1Pl, S:1Sg, S:2, S:2Pl, S:2Sg, S:3, S:3Pl, S:3Sg*.
Direct object person and number:
*DO:1, DO:1Sg, DO:1Pl, DO:2, DO:2Sg, DO:2Pl, DO:3, DO:3Sg, DO:3Pl*.
Indirect object person and number:
*IO:1* (OG), *IO:1Sg, IO:1Pl, IO:2, IO:2Sg, IO:2Pl, IO:3*.

(See above and below for examples.)

Exclusive/inclusive object Plural in Old Georgian:
[*Excl*] (მხატავს → ხატვ[ა]-ღ/ხატ V Act Pres <S-DO> … DO:1 [Excl])
[*Incl*] (გუხატავს → ხატვ[ა]-ღ/ხატ V Act Pres <S-DO> … DO:1 [Incl])

Argument case:

<*S:Nom*>, <*S:Dat*>, <*S:Erg*> – Subject in *Nom*, *Dat*, or *Erg*
(დამირეკავს → და-რეკვ[ა]/რეკ V Act Perf <S-DO> <S:Dat> … S:1Sg …)

<*DO:Dat*>, <*DO:Nom*> – Direct object in *Dat* or *Nom*
(ვხედავთ → ხედვ[ა]/ხედ V … <S-DO> <S:Nom> <DO:Dat> S:1Pl DO:3)

<*IO:Dat*>, <*IO:Gen*> – Indirect object in *Dat* or *Gen*
(მეშინია → შინებ[ა]/შინ V … Pres <S-IO> <S:Dat> <IO:Gen> S:1Sg IO:3)

Enclitics are marked with the feature *Encl*, followed by a semicolon and the concrete enclitical element:
*Encl*:ვე, *Encl*:მდა, *Encl*:რა, *Encl*:ღა, *Encl*:ც
(შენც → შენ Pron Pers 2 Erg Sg Encl:ც)

Postpositions are coded in a similar way by the feature *PP* followed by the postposisition:
*PP*:გან, *PP*:და, *PP*:დამ, *PP*:დან, *PP*:დმი, *PP*:ებრ, *PP*:ებრივ, *PP*:ვით, *PP*:ზე, *PP*:ზედ, *PP*:თან, *PP*:თვის, *PP*:კენ, *PP*:მდე, *PP*:მდინ, *PP*:მდის, *PP*:მებრ, *PP*:ურთ, *PP*:ქვეშ, *PP*:ში, *PP*:წინ and others
(ქალაქში → ქალაქ-ი N Dat Sg PP:ში).

The Indirect speech markers are:
*Encl:IndSp1, Encl:IndSp2, Encl:IndSp3*
(მოდი-მეთქი → მო-სვლ[ა]/დი V MedPass Imper <S> … Encl:IndSp1).

Punctuation marks have, in addition to the common feature *Punct*, one distinguished feature each:

*Period, ExclPoint, IntMark, Colon, Semicolon, Comma, Ellipsis, Quote, Paren, Bracket, Dash, Hyphen, Star* (! Punct ExclPoint, … Punct Ellipsis etc.).

Finally an example of a fully analyzed and disambiguated entire sentence:

| | |
|---|---|
| და | და Cj |
| აწ | აწ Adv Temp |
| დამტკიცებულად | და-მტკიცებულად/მტკიც Adv Mann |
| გითხრა | თხრობ[ა]-ჲ/თხ[ა]რ V Act Conj-II OV <S-DO-IO> <S:Erg> <DO:Nom> <IO:Dat> S:1Sg DO:3Sg IO:2 |
| თქუენ | თქუენ Pron Pers 2 Dat Pl |
| აღსასრული | აღსასრულ-ი N Nom Sg |
| წმიდისა | წმიდ[ა]-ჲ A Gen Sg L |
| და | და Cj |
| სანატრელისა | სანატრელ-ი/ნატრ Part FutPart Gen Sg L |
| შუშანიკისი | შუშანიკ N Prop Anthr FirstName Gen DNom DSg |
| . | . Punct Period |

**Acknowledgements**

I would like to thank Lela Tsikhelashvili, Nina Sharashenidze and Jost Gippert for many useful suggestions, comments and discussions.

**Bibliography**

Beesley, Kenneth R., and Lauri Karttunen, Finite State Morphology, CSLI Publications, 2003.

Bick, Eckhard: Basic Constraint Grammar Tutorial for CG-3 – http://beta.visl.sdu.dk/cg3_howto.pdf

Bresnan, Joan: Lexical Functional Syntax, Blackwell, 2001.

Meurer, Paul: Corpuscle – a new corpus management platform for annotated corpora. In: Gisle Andersen (ed.): «Exploring Newspaper Language: Using the Web to Create and Investigate a large corpus of modern Norwegian», Studies in Corpus Linguistics 49, John Benjamins, 2012.

Pollard, Carl, Ivan A. Sag: Head-Driven Phrase Structure Grammar. Chicago: University of Chicago Press, 1994.

TEI: Text Encoding Initiative – http://www.tei-c.org