A GF-Grammar for Ancient Greek

Work in slow progress

Hans Leiß
Universität München
Centrum für Informations- und Sprachverarbeitung

3rd GF Summer School
Frauenchiemsee, August 18–30, 2013

Why this?

- Apply GF to an extremely well-studied language, in detail
- Get a feeling for the linguistic knowledge of the 19th century
- Learn more about Ancient Greek (and Aristotle's view of it)
- Learn how to use GF, know its pitfalls, improve teaching it
- Use GF grammar implementation as a grammar book checker.

Possible "application": connect it with efforts to reconstruct

- Aristotle's syllogism
- Euclid's reasoning (J.Avigard)

and check the Greek argumentation by a theorem prover.

Content

- Transliteration
- Phonological rules
  - Sound laws
- Accents and Aspirates
  - Accent rules
- Nominal Morphology
- Verbal Morphology
- NP-Syntax
  - Basic NP-rules
  - Numerals
- VP-Syntax
  - VP-constructions

Writing system

1. We use the transliteration of greek symbols

   ἀ, ἁ, ἂ, ἃ, ἄ, ἅ, ...

   by latin symbol combinations

   a), a(, a)‘, a(‘, a(’, a)’, ...

   using `gf/src/compiler/GF/Text/Transliterations.hs`.

   Vowels can have diacritics for: iota sub/ad-scriptum, 2 aspirates, 3 accents (and 2 indicators of vowel length).

   “Alphabet” size including vowel length indications: 224

2. The GF transliteration differs from ‘the standard’ one (where θ = `th`, υ=`u`) or the one in LaTeX (where θ = `j`).

   We do exploit the GF transliteration in string patterns.

   So far, we don’t use capitalized letters in the string patterns.

3. We use transliterated string input and output:

```
Lang> p -cat=N "a)'nvrwpos*" | l -table
s Sg Nom : a)'nvrwpos*
...
s Dl Voc : a)nvrw'pw
```

4. and apply -from/to_ancientgreek for greek symbols:

```
Lang> ps -from_ancientgreek "ἀνθρώπῳ" | p -cat=CN
UseN anthropos_N
```

```
Lang> p -cat=NP "o( a)'nvrwpos*" | l -table
-to_ancientgreek
s Nom :  ὁ ἄνθρωπος
s Acc :  τὸν ἄνθρωπον
...
```

Word structure

As in all languages, words are not arbitrary sound combinations.
As in some other languages, intonation at the word level is indicated in the script.

- "Sound laws" restrict the sound (resp.char) combinations.
- "Accentuation rules" restrict the intonation.

Problem: we have to deal with both when building the paradigms.

- Sound laws involve vowel changes, and vowel length influences accentuation;
- Conversely, accentuation is involved in sound laws as well.

Minor problem:

- Vowel length indicators are not part of the official script.
- Some combinations of length indicator and accent are not represented in Unicode (`a_'`, `a.'`).

We might use vowel length indicators to *produce* the paradigms and *then* drop the length indicators before rendering greek strings.

But: lexica show vowel lengths only rarely (when exactly?).

Sound laws

As a restriction on sound combinations, as sound law is just a constraint. But we use sound laws as functions

```
soundlaw : Type = Str*Str -> Str*Str
```

to ensure that these constraints don't get vioalated when composing word forms.

- the input type is `Str*Str`, since we apply a sound law at a specific point in a string, typically given as `<stem,ending>`,
- the output type is `Str*Str`, since sound laws are composed.

⟨*Sound laws as string operations*⟩≡

```
oper soundlaw = (Str*Str) -> (Str*Str) ;

-- c@(guttural or labial) + si > - + (c*s)i,
gutlabS : soundlaw =
 \se -> case se of { -- BR 41 6.
     <x + c@#guttural,  "si" + y> => <x, "xi" + y> ;
     <x + c@#labial,    "si" + y> => <x, "qi" + y> ;
      => se } ;

contractVowels : soundlaw = \se ->
case se of { <x + "a", "ai" + y>  => <x, "ai" + y> ;
             ... (22 cases) ...     -- BR 15 d)
           -                      => se  } ;
-- involved accent is put on the contraction, but
-- may be changed by applying an accent rule later
```

Accentuation

- Every Greek word has an accent, acute (τόν), gravis (τὸν), or circumflex (τῶν) – except for
  - a few proclitics ὁ, ἡ, οἱ, αἱ, ἐν, ἐξ, εἰς, εἰ, ὡς, οὐ
  - a number of enclitics (Prons μου, τις, Advs που, Part γε, ...)
  - at the sentence end, a proclitic keeps its accent: πῶς γὰρ οὔ;
  - at sentence beginning, enclitics keep the accent: φημὶ τοίνυν ..
- The gravis replaces the acute on the last syllable of a word that is followed by another word: τὸν ἄνθρωπον
  - except for interrogatives: τίς ἄνθρωπος·

But: An (accentuated or proclitic) word may –according to specific rules– inherit an acute(!) on its last syllable from a following enclitic: ἄνθρωπός τις, εἴ τις

We assume a special lexer/unlexer replaces the gravis by an acute and moves the inherited accents to the enclitics which lost them.

Problem: Write such a lexer/unlexer!

General accent rules

1. The acute can be on a short or long vowel and diphtong, but only on one of the final three syllables. If the last syllable is long, it can only be on one of the final two syllables.
2. The circumflex can be on long vowels and diphtongs, and on one of the final two syllables. If the last syllable is long, it can only be on the final one.
3. If the last syllable is short and the second last is long and emphasized, the second last must carry a circumflex.

Since inflection may add/replace short or long endings, the accent moves in the paradigm. (Some diphtongs (αι, οι) count as short.)

Admissible accentuations in greek words, when the accent is on

3rd last vowel			2nd last vowel			last vowel		
A	N	N	N	A	N	N	N	A
L|S	L|S	S	L|S	S	L|S	L|S	L|S	L|S
			L|S	L	L			
			N	C	N	N	N	C
			L|S	L	S	L|S	L|S	L

Accent kinds: A=Acute, C=Circumflex, N=NoAccent
Vowel lengths: L=Long, S=Short

Example:

German:	Aristóteles	E N N + L S S
Greek:	Ἀριστοτέλης	N A N + S S L

Noun inflection

There are three major declension classes:

1. I (A-declension)
2. II (O-declension)
3. III (3rd declension)

Since vowels may change, accents are better treated independently.

Accent rule for noun declension:

1. the accent position (of SgNom) is only changed on demand.
2. a shift is demanded if a an ending with a long vowel is added and the accent was on the 3rd last vowel.
   ἄνθρωπος/ἀνθρώπων
3. when adding an ending with accent, drop the stem's accent.

We can produce the paradigm of a word

Alternative 1 from several forms that show the different accents and accent positions.

Alternative 2 from information about lengths of syllables/vowels in the stem and the endings.

We started with alternative 1 for noun declensions I and II (-A,-O), but moved to alternative 2 for declension III.

Alternative 1 seems hopeless for verb inflection (ca. 500 forms)

- too many different stems per word (with 7 aspect stems).
- too many changes in the stems (vowel lengths, consonant dropping)

Noun declension I, II

For nouns ending in α or η (without accent), infer vowel changes and accent shifts from `SgNom,SgGen,PlNom`:

⟨*A-declension, 1*⟩≡

```
noun3A : Str -> Str -> Str -> Noun =
  \valatta, valatths, valattai ->
  let valatt   = P.tk 1 valatta ;
      valatth  = P.tk 2 valatths ; -- omit "s*"
      valattPl = P.tk 3 valatths ; -- omit "hs*"|"as*"
  in
  mkNoun
    valatta valatths (valatth+"|") (valatta+"n") valatta
    valattai (dropAccent valatt +"w~n") (valattPl+"ais*")
             (valattPl+"as*")
    (valattPl+"a") (valattPl+"ain") Fem ; -- +"a"
```

`PlNom` is needed to see if short endings like αι cause an accent change on vowels ά, ί, ύ (i.e. if these are long).

For those nouns ending in ά/ή, SgGen, SgDat, PlDat take ᾶ/ῆ:

⟨*A-declension, 2*⟩≡

```
nounA' : Str -> Noun = \tima' ->  -- accent on endvowel
  let tim = Predef.tk 2 tima' ;
      a   = Predef.tk 1 (Predef.dp 2 tima')
  in
  mkNoun
    tima' (tim+a+"~s*") (tim+a+"|~") (tim+a+"'n") tima'
    (tim+"ai'") (tim+"w~n") (tim+"ai~s*") (tim+"a's*")
    (tim+"a'") (tim+"ai~n") Fem ;
```

Similar declension functions can be written this way and combined to a "smart paradigm" for declensions I/II. But:

- Regularities on accentuation are not explicitly expressed.
- Phonological regularities (sound laws) are likely to be violated.

Noun declension III

For nouns whose stem ends in a consonant or ι, υ, or diphthong

- the stem is found by stripping off ending -ος from SgGen
- use special endings with adaptions to the stem due to phonological rules (stem + ς + ending, vowel changes)
- for monosyllabic stems, shift accent to the ending in Gen/Dat

To build the paradigms, we transform given forms (of type `Str`) to structured data (of type `Word`) and compute with these in order to

- need less pattern matching to find parts of strings,
- reuse information extracted from the given strings.

Basically: isolate the three final vowels and non-vowel parts around.

⟨*Word patterns: c1+v1+c2+v2+c3+v3+c4*⟩≡

```
oper
  Position : PType = Predef.Ints 3 ;
param
  Accent = Acute Position
         | Circum Position | NoAccent ;
  Syllability = Mono | Bi | Multi ;
  Length = Zero | Short | Long ;
oper
  Word = { s : Syllability ;      -- # end syllables
           v : Str * Str * Str ; -- end vowels
           l : Length * Length * Length ; -- |vowels|
           c : Str * Str * Str * Str ; -- consonants
           a : Accent
         } ;
  toWord : Str -> Word = ...
```

⟨*Example: word Ἀριστοτέλης as Word*⟩≡

```
Lang> cc -unqual toWord "A)ristote'lhs*"
{s : Syllability = Multi;
 a : Accent = Acute 2;
 c : {p1 : Str; p2 : Str; p3 : Str; p4 : Str}
   = {p1 = "A)rist"; p2 = "t"; p3 = "l"; p4 = "s*"};
 l : {p1 : Length; p2 : Length; p3 : Length}
   = {p1 = Short; p2 = Short; p3 = Long};
 v : {p1 : Str; p2 : Str; p3 : Str}
   = {p1 = "o"; p2 = "e"; p3 = "h"}}

Lang> cc toStrT (toWord "A)ristote'lhs*")
"A)rist-o-t-e'-l-h-s*"
```

⟨*Auxiliary conversions from Words to strings*⟩≡

```
toStrT: Word -> Str = .. -- show segmentation
toStr0: Word -> Str = .. -- ignore accent rules
```

To enforce accent rules, we may have to change the accent type or position where we want to put it, depending on lengths of vowels.

⟨*Adding an accent to a Word*⟩≡

```
addAccentW : Accent -> Word -> Str = \accent, w ->
  let v1 = w.v.p1 ;  -- third last vowel
  ... l3 = w.l.p3 ;  -- length of last vowel
  in case accent of {
       Acute 3  => merge w.c <v1, v2, v3 + "'"> ;
       Circum 3 => case l3 of {
             Long  => merge w.c <v1, v2, v3 + "~"> ;
                   => merge w.c <v1, v2, v3 + "'"> } ;
       ...
        => Predef.error ("Illegal accentuation")
  } ;
```

⟨*Enforcing accent rules for the stored accent*⟩≡

```
  toStr : Word -> Str = \w -> addAccentW w.a w ;
```

It's more expensive to do this with `Str` instead of `Word`:

⟨*Adding an accent to a string*⟩≡

```
addAccent : Accent -> Str -> Str = ...
```

For example, this covers accent rule 3 above:

⟨*Adding an accent to a string*⟩+≡

```
Lang> cc -unqual addAccent (Acute 2) "a)'nvrwpwn"
"a)nvrw'pwn"
Lang> cc -unqual addAccent (Acute 2) "a)'nvrwpos*"
"a)nvrw~pos*"  -- (not greek)
```

The accent to be added may be computed, not explicitly given.

To build paradigms based on structured data `Word*Ending`, we use

⟨*Type of noun endings*⟩≡

```
NEnding = { a : Accent ;
            v : Str ; l : Length ;  -- vowel with length
            c : Str * Str } ;  -- surrounding consonants
toNEnding : Str -> NEnding = \str -> ...
```

⟨*Building a word form from w:Word and e:Str*⟩≡

```
toStrN : Word -> Str -> Str =
    \w,e -> toStr (concat <w, toNEnding e>) ;

concat : (Word * NEnding) -> Word = ... ;
-- Append an ending of constants to the stem's end c's.
-- If the ending has an accent, drop the one in the stem;
-- if it has an unaccentuated vowel, use the stem's
-- accent. Combine the s,v,l,c components modulo the
-- length of the ending's vowel.
```

Lift the string-level `soundlaw` to operations on `Words`:

⟨*Sound laws on structured words*⟩≡

```
Soundlaw = (Word * NEnding) -> (Word * NEnding) ;

toSL : soundlaw -> Soundlaw = \sl -> \we ->
   -- toStr0 to not apply accent rules:
   -- sw'mat+si > sw'ma+si, not sw~mat+si > ...
   let se = sl <toStr0 we.p1, toStr we.p2>
    in adjustAccent <toWord se.p1, toNEnding se.p2> ;

adjustAccent : Soundlaw = \<W,E> -> ..
-- move and change accent in W, if needed when adding E
-- according to accent rules for W+E (does not change E).
```

⟨*Example* ⟩+≡

```
Lang> cc toStrT (adjustAccent <toWord "ge'ne",
                                toNEnding "wn">).p1
"--g-e-n-e'-"
```

⟨*Example* ⟩+≡

```
gutlabS : soundlaw = .. ; -- guttural+s > 0+x,
                          --   labial+s > 0+q
glS : Soundlaw = toSL gutlabS ;
```

A sound law ought to adjust accent and syllability in the stem, if a vowel is added/dropped/changed in length in an ending, so that soundlaws can be combined. We want to have:

- `(soundlaws o adjustAccent) : Soundlaw`,
- `(adjustAccent o soundlaws) : Soundlaw`.

⟨*Example: compute accent position, then drop s and contract vowels*⟩≡

```
toStrNs : Word -> Str -> Str =
    \w,e -> let we = adjustAccent <w, toNEnding e> ;
                we' = cVdS we ;
             in toStr (concat we') ;
cVdS ue = case (toStr ue.p2) of {
            #vowel +  => cV (dS ue) ;  => ue } ;
```

A Paradigm is constructed in three steps:

1. turn the given forms (and stem) into Words,

⟨*Step 1*⟩≡

```
noun3LGL : Str -> Str -> Gender -> Noun =
\rhtwr, rhtoros, g ->
  let -- stem ends in l|r, k|g|c, p|b|f
      stem : Str = case rhtoros of {
        stm + ("os*"|"o's*") => stm ;  => rhtwr } ;
      rhtwr : Word = toWord rhtwr ;
      -- Ablaut: undo vowel lengthening in SgNom
      rhtor : Word = let stem' = toWord stem in ...
  in noun3LGLw rhtwr rhtor g ;
```

2. construct the combined Word using the isolated informations, applying soundlaws if necessary,
3. collapse the combined Word to a string (using accent rules).

⟨*Steps 2 and 3*⟩≡

```
noun3LGLw : Word -> Word -> Gender -> Noun =
\rhtwr,rhtor,g ->
  let syl = rhtwr.s ;
      rhtwr   = toStrN rhtwr "" ;
      rhtoros = toStrN rhtor (endingsN3!Sg!Gen!g!syl) ;
      rhtorsi = toStrNsl glS -- BR 43, BR 41 6.
                             rhtor (endingsN3!Pl!Dat!g!syl) ;
      ...
  in mkNoun rhtwr rhtoros ... rhtoroin g ;
```

Sound law `glS` is applied before combining the parts, using

```
toStrNsl : Soundlaw -> Word -> Str -> Str =
   \sl,w,e -> toStr (concat (sl <w, toNEnding e>)) ;
```

Verb inflection

There are two main conjugation classes:

- ω-conjugation: παιδεύω
- μι-conjugation: δείκνυμι

There are three diatheses, active, medium, passive. The verbal system is organized by aspect (not by tense), with seven stems:

Stem	w-	mi-	conjugation form:
act/med/pass Pres	paideyw	didwmi	VAct (VPres VInd) Sg P1
act/med Fut	paideysw	dwsw	VAct VFutInd Sg P1
act/med Aor	epaideysa	edwka	VAct (VAor VInd) Sg P1
act Perf	pepaideyka	dedwka	VAct (VPerf VInd) Sg P1
med/pass Perf	pepaideymai	dedwmai	VMed (VPerf VInd) Sg P1
pass Aor	epaideythn	edothn	VPass (VAor VInd) Sg P1
VAdj	paideytos	dotos	VAdj Masc Sg Nom

Parametrization of the verb forms:

- Full verbs have three voices (medium: $\simeq$ reflexive use).
- Greek has two kinds of deponent verbs lacking active forms.
- There are four "main" tenses, which except `GFut` correspond to the three aspects: imperfective, perfective and stative.

⟨*Verb form parametrization*⟩≡

```
param
  Voice = Act | Med | Pass ;  -- Active, Medium, Passiv
  VType = VFull | DepMed | DepPass ;    -- used in predV
  VTmp = GPres | GFut | GAor | GPerf ;  -- main tenses

  --   VAspect = Imperfective (Present-stem)
  --           | Perfective   (Aorist-stem)
  --           | Stative      (Perfect-stem)
  --           ;
```

⟨*Finite verb forms*⟩≡

```
-- 'main' tenses Pres,Fut,Aor,Perf have moods:
VTense = VPres Mood   -- (in the order of verbstems)
       | VImpf        -- imperfect: just Ind mood
       | VFut MoodF   -- future: just Ind and Opt mood
       | VAor  Mood
       | VPerf Mood
       | VPlqm ;      -- plusquamperfect: just Ind
Mood  = VInd | VConj | VOpt; -- | VImp
MoodF = FInd | FOpt ; -- Conj, Imp don't exist in Fut
```

Imperatives exist in all voices but only three of the main tenses and of course not all (Pers,Num)-combinations (Dual?)

⟨*Tense and person for imperatives:*⟩≡

```
ITmp   = IPres | IAor | IPerf ;
NumPers = SgP2 | SgP3 | PlP2 | PlP3 ;
```

There are no imperative forms in Active IPerf: deliver `NonExists`.

The main tenses have infinite Forms: infinitives and participles.
And there are two verbal adjectives (modalized passive participles).

⟨*Finite and infinite verb forms*⟩≡

```
param                   -- Voice: omitted here, cf. Verb
  Vform = Fin VTense Number Person
        | Imp ITmp NumPers
        | Inf VTmp
        | Part VTmp AForm ;
oper                    -- type of morphological verb
  Verb : Type = {
    act : Vform => Str ;   -- Voices:
    med : Vform => Str ;   -- define med, derive pass
    pass : Vform => Str ;
    vadj1 : Adj ;  -- paideyto's  = who can be educated
    vadj2 : Adj ;  -- paideyte'os = who must be educated
    vtype : VType
    -- stems : Str * ... * Str  might be useful to have
    } ;
```

One might want to omit participles and verbal adjectives from the morphological verb and make them morphological adjectives, using

⟨*Constructors for participles*⟩≡

```
mkAdj : Verb -> Voice -> VTmp -> Adj ;
```

that have to find the proper verb stem from the `VTmp`, and

⟨*Constructors for verbal adjectives*⟩≡

```
vadj1, vadj2 : Verb -> Adj ;
```

These could perhaps take the `verbstem:Str` instead of `v:Verb`.

⟨*Number of verb forms*⟩≡

```
   3 * |Vform| + 2 * |AForm|
 = 3 * (  |VTense|*|Number|*|Person|
        + |ITmp|*|NumPers| + |VTmp| )
   + (3 * |VTMP| + 2 ) * |AForm|  -- particples + vadjs
 = 3 * ( 13*3*3 + 3*4 + 4 )  + (3*4 + 2)*(3*3*5)
 = 399 + 630 =~ 1030
```

Creating all forms for `give_V3` = δίδωμι takes 3 sec.

Noun phrases

Positions of the adjectival attribute:

μεγάλη πόλις	A N	a big city
πόλις μεγάλη	N A	
ἡ μεγάλη πόλις	DefArt A N	the big city
ἡ πόλις ἡ μεγάλη	DefArt N DefArt A	

But: predicatively used adjective: (Sentence)

μεγάλη ἡ πόλις	A DefArt N (Cop)	the city is big
ἡ πόλις μεγάλη	DefArt N A (Cop)	

Positions of the genitive attribute:

ἡ τῶν Πέρσων ἀρχή	DefArt NP^{gen} N2	the reign of the Persians
ἡ ἀρχὴ τῶν Πέρσων	DefArt N2 NP^{gen}	
τῶν Πέρσων ἡ ἀρχή	NP^{gen} DefArt N2	

Positions of demonstrative pronoun:

ἐκεῖνη ἡ γυνή	DemPron DefArt N	that woman
ἡ γυνὴ ἐκεῖνη	DefArt N DemPron	

Some word order variations are handled by parameterizing the NP:

⟨*Parameter for attribute position*⟩≡

```
AOrder = Pre | Post ;  -- before or after the noun
```

One would expect the paradigm of NPs to have type

```
s : AOrder => Case => Str
```

giving rise to 2*5 = 10 strings.

However, reflexive possessive pronouns complicate things:

ὁ ἐμός φίλος	DefArt PossPron^{stressed} N	my friend
ὁ φίλος μου	DefArt N PossPron^{unstressed}	my friend
τὸν ἐμαυτοῦ φίλον	DefArt Pron^{Gen} N	my own friend

The reflexive possessive ("*my own*") in Greek is expressed by the Pron^{Gen} *agreeing with the subject in gender, number and person*.

Hence the paradigm of NPs depends on agreement parameters:

⟨*lintype of noun phrases*⟩≡

```
NP = { s : Agr => AOrder => Case => Str ;
           -- reflexive and reflexive possessive use
       isPron : Bool ;
       e : Case => Str ; -- emphasized pron, or ignored
       a : Agr } ;
```

This blows up the NP paradigm to |Agr|*2*5 = 3*3*3*2*5 = 270 strings, or 180, if we ignore the dual number.

Problem: In principle, one has to expect

`NP.s : Agr => Case => Str` rather than `Case => Str`

for other languages as well. Can we afford to embed NPs in VPs?

We need to make the CN-attribute depend on Agr as well, because

- CNs can be modified by reflexive possessives (dep. on Agr),
- CNs built from N2s can have NP objects (dep. on Agr).

⟨*lintype of common nouns*⟩≡

```
CN = { s : Number => Case => Str ;   -- noun only
       s2 : Agr => Number => Case => Str ; -- attribute
       isMod : Bool ;          -- attribute nonempty?
       rel : Number => Str ;  -- relative clause (Agr?)
         g : Gender } ;
```

Participle phrases also may have reflexive possessive parts,

Alexander, having killed his own friend, . . .

Verb Phrases and Clauses

We construct verb phrases from verbs by fixing a voice and storing objects and modifiers as separate fields of a record:

⟨*lintype of verb phrases*⟩≡

```
oper
  VP : Type = {
    s : VPForm => Str ;
    neg : Bool ;          -- TODO: need Pos, Ouk, Mh
    obj : Agr => Str ; -- nominal complement
    adj : Agr => Str ; -- predicative adj
    adv : Str ;           -- adverb
    ext : Str             -- sentential complement
  } ;
```

VPs have finite forms in all tenses, and infinitives and participles in the main tenses:

⟨*Paramers of verb phrases*⟩≡

```
param
  VPForm = VPFin VTense Number Person
         | VPImp VPImpForm
         | VPInf VTmp
         | VPPart VTmp AForm
         | VPAdj1 AForm
         | VPAdj2 AForm
         ;
  VPImpForm = ImpF ITmp NumPers ;
```

As participle forms are more common than subordinate clauses, they belong to the VP and should not be separate APs.

To build a VP from a V, we choose active (for full verbs) or medium resp. passive voice (for deponents):

⟨*Basic VP-construction, depending on vtype*⟩≡

```
lin  UseV = predV ;      -- use active voice
oper predV : Verb -> VP = \v ->
   { s = table {
           VPFin t n p => case v.vtype of {
               -- DepPass has "active" forms in v.med
               VFull => v.act ! (Fin t n p) ;
                   => v.med ! (Fin t n p) } ;
           ...
           VPAdj2 a => v.vadj1.s ! a
         } ;
     neg = False ; obj, adj = \\ => [] ;  adv, ext = []
  } ;
```

Using reflexive arguments in `ReflVP: VPSlash -> VP` in English generalizes to using the medium voice `MedV2: V2 -> VP` in Greek:

⟨*VP construction using medium voice*⟩≡

```
lin MedV2 = predVmed ; -- use medium voice

oper predVmed : Verb -> ResGrc.VP = \v ->
  { s = table { VPFin t n p => v.med ! Fin t n p ;
                VPInf tmp   => v.med ! Inf tmp ;
                ...
                VPAdj2 a    => v.vadj1.s ! a
              } ;
    neg = False ;
    obj, adj = \\ => [] ;  adv, ext = []
  } ;
```

The Greek school-tablet

The British Museum contains a nice greek school tablet of 450 BC (?) showing the following sentence:

Πυθαγορας φιλοσοφος	Pythagoras the philosopher,
αποβας και γραμματα διδασκων	when going to read letters,
συνεβουλευεν τοις εαυτοις μαθηταις	advised his (own) students
εναιμονων απεξεσται	to abstain from meat

The tablet demonstates

- how the participles agree with the subject as it varies in number (Sg,Pl,Dl), and
- how Πυθαγορας φιλοσοφος appears in all cases when the sentence is embedded under a suitable main verb

F.G.Kenyon: Two Greek school-tablets.
Journal of the Hellenistic Society

```
AllGrcAbs> l -table
(PredVP (DetCN (DetQuant DefArt NumPl) (UseN filosofos_N))
  (ComplSlash (SlashV2V advise_V2V
     (ComplSlash (SlashV2a abstain_V2)
        (UseCNSg (PossNP (UseN meat_N) (DetNP every_Det))))
          (DetCN (DetQuant DefArt NumPl)
                 (ReflCN (UseN student_N))))))
s (VPres VInd) Pos SVO :
  oi( filo'sofoi symboyley'oysi a)pe'cein
        panto's* e)naimo'noy toi~s* e(aytw~n mavhtai~s*
...
s VPlqm Neg VSO :
  oy) synebeboyley'kesan oi( filo'sofoi a)pe'cein
      panto's* e)naimo'noy toi~s* e(aytw~n mavhtai~s*
```

Problems with GF

1. The Greek possessive pronoun is an adjective, not a quantifier.
   ἡ ση βίβλος DefArt PossPron N your book

   GF's `PossPron : Pron -> Quant` gives `PossPron pers : Quant`, whence we need some structural transfer

   ⟨*turning a possessive quantifier to possessive NP-attribute*⟩≡

```
fun possAdj : NP -> NP ;
def possAdj (DetCN (DetQuant (PossPron pers) num) cn)
            = (DetCN (DetQuant DefArt num)
                     (PossNP cn (UsePron pers))) ;
```

   But in GF *we cannot define a function by recursion on the structure of NPs*: NP-constructors aren't `data` constructors.

   Moreover, there is no `posses_Prep : Prep` in Greek, like in `book of mine`. There should not be a `possess_Prep` in the abstract syntax, nor a `part_Prep`.

2. Using the `DefArt` is special. Sometimes we'd like to do a *case analysis of the NP* given. For example:
   - In predicative NPs, drop the article: πόλεμος πατὴρ πολλῶν
   - With postponed adjective, repeat the article: ἡ πόλις ἡ μεγάλη
   - At sentence beginning, insert *but* between article and noun: ὁ δὲ Σωκράτης ... vs. ἐγὼ δὲ ...

   We can add to `lincat NP` a separate `Str`-field for the definite article, but it would *only* be used in such cases.

   Doing case analysis on abstract syntax trees would be more general and natural than storing properties of the absyn trees in fields of the concrete syntax (like `isPron`, `isMod`).

Summary

1. Accentuation and sound laws interact
2. Full linearization type of NPs has paradigm depending on agreement features
3. Deponent verbs use medium resp. passive forms as active
4. Medium voice of `V2`-verbs generalizes reflexive use

- Extended the Ancient Greek grammar by numerals, reflexives, possessives, deponent verbs, participle constructions,
- Other constructions in `ExtraGrc` like `ReciVP`, `PartAorVP`
- Sound laws not yet applied in some noun and verb inflections
- Still a lot of constructions wrong or missing.