Loglan Reference Guided by the Parsing Expression Grammar

M. Randall Holmes

August 8, 2021

1 Introduction

The intention of this document is to present a complete description of the phonetics, lexicography, and grammar of the latest proposed dialect of TLI Loglan in a format which follows the structure of the rules in the Parsing Expression Grammar (PEG) used to check Loglan utterances for correctness.

The PEG is not a perfect peg on which to hang a complete description of Loglan; Loglan is not *all* about disambiguation. But a lot of it is about disambiguation on various levels.

Contents

2 The Parsing Expression Grammar, with commentary

2.1 Phonetics

2.1.1 Comments from the PEG file on phonetics

This subsection is not part of the self-contained development of Loglan concepts intended in this document: it presupposes prior awareness of the language and

issues in its history. It is an edited version of a note found in the PEG file as a comment, and it may be useful to people who are already familiar with earlier versions of the language.

Mod bugs, I have implemented all of Loglan phonetics as described in my proposal. Borrowing djifoa are pretty tricky.

I have now parsed all the words in the dictionary, and all single words of appropriate classes parse successfully. I have added alien text and quotation constructions which do not conform to these rules; so actually all Loglan text should parse, mod some punctuation and capitalization issues. The conventions for alien text here are not the same as those in previous provisional parsers.

I believe the conventions for forcing comma pauses before vowel initial cmapua and after names except in special contexts have been enforced. In a full grammar, one probably would want to disable pauses before vowel initial letteralls (done). This grammar also does not support the lingering irregularities in acronyms (and won't).

This grammar (in Part I) is entirely about phonetics: all it does is parse text into names (with associated initial pauses or name markers), cmapua (qua unanalyzed streams of cmapua units), borrowings and complexes, along with interspersed comma pauses and marks of terminal punctuation. It does support conventions about where commas are required and a simple capitalization rule. Streams of cmapua break when markers initial in other forms are encountered (and may in some cases resume when the markers are a deception).

a likely locus for odd bugs is the group of predstartX rules which detect apparent cmapua which are actually preambles to predicates. These are tricky! (and I did indeed find some lingering problems when I parsed the dictionary). Another reason to watch this rule predstart is that it carries a lot of weight: !predstart is used as a lightweight test that what follows is a cmapua (a point discussed in more detail later).

In reviewing this, I think that very little is different from 1990's Loglan (the borrowing djifoa are post-1989 L1, but not my creation). Some things add precision without making anything in 1990's Loglan incorrect.

The requirement that syllabic consonants be doubled is new, and makes some 1990's Loglan names incorrect.

The requirement that names resolve into syllables is new, and makes some 1990's Loglan names incorrect, usually because they end in three consonants.

The rule restricting final consonant pairs from being noncontinuant/continuant is new, but does not affect any actual predicate ever proposed.

Enhancing the VccV rule to also forbid CVVV...ccV caused one predicate to be changed (`haiukre` became `haiukrre`, and `haiukre` was a novelty anyway, using a new name for X in X-ray) The exact definition of syllables and use of syllable breaks and stress marks is new (the close comma was replaced with the hyphen, so `Lo,is` becomes `Lo-is`); but this does not make anything in 1990's Loglan incorrect, it merely increases precision and makes phonetic transcript possible.

Forbidding doubled vowels in borrowings was new, was already approved, and caused us to change `alkooli` to `alkoholi`.

Formally allowing the CVccVV and CVcccV predicates without y-hyphens took a proposal in 2013 because Appendix H was careless in describing their abandonment of the slinkui test, but the dictionary makes it evident that this was their intent all along. The slinkui test had already been abandoned in the 1990s.

Formally abandoning **qwx** was already something that the dictionary workers in the 1990's were working on; we completed it.

Allowing glottal stop in vowel pairs and forbidding it as an allophone of pause is a new phonetic feature in the proposal but not reflected in the parser, of course. Alternative pronunciations of y and h and allowing h in final position are invisible or do not make any 1990's Loglan incorrect. Permitting false name markers in names was already afoot in the 1990's and the basic outlines of our approach were already in place. The rule requiring explicit pauses between a name marker not starting a name word and the beginning of the next name word is new, but reflects something which was already a fact about 1990's Loglan pronunciation: those pauses had to be made in speech (and in the 1990's they had no tools to do relevant computer tests)! The requirement that names resolve into syllables restricts which literal occurrences of name markers are actually false name markers (the tail they induce in the name must itself resolve into syllables).

Working out the full details of borrowing djifoa was interesting: I'm not sure that I've done anything *new* there; explicitly noting the stress shift in borrowing djifoa might be viewed as something new but it is a logical consequence of JCB's permission to pause after a borrowing djifoa, which contains explicit language about how it is to be stressed, and the final definition of a borrowing djifoa as simply a borrowing followed by -y. The shift strikes me as a really good idea anyway, because it marks djifoa with a pause after it as phonetically different in an additional way other than ending with the very indistinct vowel y. My rules as given here do not directly enforce the rule that a medial borrowing djifoa must be preceded by y but I think they indirectly enforce it in all or almost all cases: the parser tries to read a borrowing djifoa before reading any other kind of djifoa, so it is hard to see how to deploy a short djifoa in such a way that it would fall off the head of a borrowing without using y.

These phonetics do not support certain irregularities in acronyms. We note that it is now allowed to insert `, mue` into an acronym, which would be necessary for example between a Ceo letteral and a following VCV letteral.

2.1.2 Sounds

```
#all vowels

V1 <- [aeiouyAEIOUY]

#regular vowels
```

```
V2 <- [aeiouAEIOU]
```

The regular Loglan vowels are **aeiou**, with pronunciations one might describe as continental European.

a as in *father*

e as in *bed*

i as in *machine*

o as in *log*

u as in *rule*

The irregular vowel is **y**, for which the original official pronunciation was the unstressed schwa sound. We regard this as unfortunate (because English and Russian speakers tend to collapse unstressed final **a** to schwa), and propose instead that it be pronounced as *oo* in *look* or perhaps as the Cyrillic letter written bI, in either case a definite sound with a distinct value clearly different from the regular vowels.

```
#consonants

C1 <- [bcdfghjklmnprstvzBCDFGHJKLMNPRSTVZ]

#consonants in voiced/unvoiced pairs

Cvoiced <- [bdgjvzBDGJVZ]

Cunvoiced <- [ptkcfsPTKCFS]

# bad voice pair (or pair second term of which is h)
# forbidden as pairs of final consonants

Badvoice <- (Cvoiced (Cunvoiced/[Hh])/Cunvoiced (Cvoiced/[Hh]))
```

The Loglan consonants are **bcdfghjklmnprstvz**.
Most of these have values expected by an English speaker.

b as in *bed*

c as *sh* in *shoe* (**unexpected**)

d as in *dog*

f as in *fox*

g as in *gun*. (**never** soft as in *gem*).

h as in *horse* or (**unexpected**) as *ch* in Scottish *loch*. Only the second pronunciation is possible in final position in a syllable. 1989 Loglan did not have the second pronunciation, or syllable-final *h*.

j as *z* in *azure* (**unexpected**)

k as in *kill*

l as in *light*

m as in *mouse*

n as in *nose*

p as in *pop*

r as in *rose*

s as in *soap*

t as in *tease*

v as in *vanish*

z as in *zoo*

Some of these consonants appear in sets of a voiced and unvoiced consonant. There is a rule forbidding a pair of consonants of different voices from appearing in final position in a syllable, as well as pairs of consonants ending in **h** at the ends of syllables.

2.1.3 Capitalization and punctuation

```
letter <- (![qwxQWX] [a-zA-Z])


lowercase <- (![qwx] [a-z])

uppercase <- (![QWX] [A-Z])
```

The letters of the Loglan alphabet are the Latin letters other than **qwx** (which did appear in earlier versions of Loglan with misguided strange pronunciations). They have uppercase and lowercase forms as usual.

```
caprule <- [\"(]? &([z] V1
(!uppercase/&TAI0)/lowercase TAI0
(!uppercase/&TAI0)/!(lowercase uppercase).)
letter (&([z] V1 (!uppercase/&TAI0)/lowercase TAI0
(!uppercase/&TAI0)/!(lowercase uppercase).)
(letter/juncture))* !(letter/juncture)
```

A capitalization convention which allows what our current one allows and also allows all-caps. if case goes down from upper case to lower case, it can only go back up in certain cases. This does allow capitalization of initial segments of words. There is a forward reference to the grammar in that free capitalization of embedded literals is permitted, and capitalization of vowels guarded with z in acronyms as in DaiNaizA.

The exact details are that a correctly capitalized string of letters may start with a parenthesis or quote, in effect ignored. What follows may start with a uppercase letter or lowercase letter, and is a string of letters and junctures (see just below for junctures), in which a change to uppercase can occur only after a juncture or at the beginning of an embedded copy of a letteral (see below for letterals, which serve as pronouns and as letter names), or lowercase **z** may be followed by a capitalized vowel. Pragmatically, words can be uncapitalized, capitalized initially, or all-caps; parts of words separated by junctures may be capitalized independently; embedded letterals may be capitalized and vowels guarded by **z** in acronyms may be capitalized.

NOTE: Examples of capitalization should appear here or at a useful point or points in the text below.

```
juncture <- (([-] &letter)/[\'*]) !juncture

stress <- ['*] !juncture
```

The just promised junctures (syllable markers): the hyphen **-** is a marker of a syllable boundary and is always medial so must be followed by a letter. The stress marks **'*** signal a syllable break if there is an following syllable, and indicate that the preceding syllable (so there must be one) is stressed (***** indicates emphatic stress); the stress markers can be word final. A juncture is never followed by another juncture.

```
terminal <- ([.:?!;#])
```

A set of markers of terminal punctuation

```
character <- (letter/juncture)
```

Characters are the symbols which can occur in words (letters and junctures).

2.1.4 Forms with embedded alien text: strong quotation, foreign names, other similar things

An old comment embedded in the PEG on what follows:

to really get all Loglan text, we should add the alien text constructions and the markers of alien text, `<lie>,<lao>, <sao>, <sue>` and certain quotations which violate the phonetic rules.

we adopt the convention that all alien text may be but does not have to be enclosed in quotes. it needs to be understood that in quoted alien text, whitespace is understood as `<, y,>` ; in the unquoted version this is shown explicitly. This handling of alien text is taken from the final 1990's treatment of Linnaeans = foreign names, and extended by us to replace the impossible treatment of strong quotation in 1989 Loglan.

this is a little different from what is allowed in the previous provisional parser, but similar. A difference is that all the alien text markers are allowed to be followed by the same sorts of alien text.

the forms with `<hoi>` and `<hue>` are required to have following quotes in written form to avoid unintended parses, which otherwise become likely in case of typos in non-alien text cases.

```
AlienText <- ([,]? [ ]+ [\"] (![\"].)+ [\"]/ [,]? [ ]+
(![, ]!terminal .)+ ([,]? [ ]+ [y] [,]? [ ]+
(![, ]!terminal .)+)*)

AlienWord <- &caprule ([Hh] [Oo] [Ii] juncture?
&([,]? [ ]+ [\"])/[Hh][Uu] juncture? [Ee] juncture?
&([,]? [ ]+ [\"]) / [Ll] [Ii] juncture? [Ee]juncture?
/[Ll] [Aa] [Oo]juncture? /[Ll] [Ii] juncture? [Oo] juncture?
/[Ss] [Aa] [Oo]juncture?/[Ss] [Uu] juncture?
 [Ee]juncture?) AlienText

# while reading streams of cmapua, the parser will watch for the markers of alien text.

alienmarker <- ([Hh] [Oo] [Ii] juncture? &([,]? [ ]+ [\"])/
[Hh][Uu] juncture? [Ee] juncture? &([,]? [ ]+ [\"]) /
[Ll] [Ii] juncture? [Ee] juncture? /[Ll] [Aa] [Oo] juncture? /
[Ll] [Ii] juncture? [Oo] juncture?  /
[Ss] [Aa] [Oo] juncture?/[Ss] [Uu] juncture? [Ee] juncture?) !V1
```

```
# 5/11/18 added <lio> as an alien text marker, to support numerals.
```

The classes above handle alien text, not conforming to Loglan rules, which is to be embedded in Loglan text or speech. Of course, Loglan rules of pronunciation do not indicate how alien text is to be pronounced.

Alien text may start with an optional comma pause (a comma followed by whitespace). The remaining part of the alien text may be either any string at all enclosed in double quotes or one or more strings of characters other than commas, spaces, or terminal punctuation, separated by the word **y** set off from the surrounding text by whitespace before and whitespace or a comma pause after.

Where quoted alien text contains spaces, its expression in speech will involve **y** even though it is not written: `lie "War and Peace"` and `lie War y and y Peace` are the same utterance, pronounced in the same way.

When pronounced, alien text is always preceded and followed by pauses, which may or may not be expressed as comma pauses in writing.

Alien text appearing in Loglan is always preceded by one of the alien text markers, which we list. NOTE: more examples should be embedded here.

hoi vocative (the alien text is a name of the person addressed and must be in quoted form)

hue inverse vocative (the alien text is a name of the speaker and must be in quoted form)

lii the alien text is the name of a foreign letter: `lii aleph` might denote the Hebrew letter aleph.

lie the alien text is quoted: `lie X` denotes the text `X`.

lao the alien text is a foreign name, as `lao Albert y Einstein`.

lio the alien text is marked as a numeral or other mathematical expression. `lio 123` is the number 123.

sao the alien text is a foreign predicate. `sao ice y cream`, ice cream.

sue the alien text is an onomatopoeic predicate `sue miau` for "to meow".

There is some history here which we briefly indicate. The mechanism of strong quotation (quotation of text not conforming to Loglan rules) in previous versions of Loglan is simply impossible to implement in BNF or a PEG in any reasonably economical way. We abandoned it. The **lao** form of foreign names was in older versions of Loglan reserved for Linnaean nomenclature: Steve Rice noted in his textbook that it could reasonably be used for foreign names in general and we have followed his lead. Dealing with breaks in Linnaeans by inserting **y** (originally not expressed in writing) was a suggestion of workers in the late 1990's. We applied this to all alien text forms and further require

that the word **y** be written if the alien text is not enclosed in quotes. So the rules for alien text have roots in earlier Loglan work in the final form of Linnaean nomenclature, this form being extended to various other applications of embedded alien text in Loglan.

NOTE: should **gao** be an alien text form?

2.1.5 The Loglan Syllable

I make a historical remark here briefly which I have made in other contexts at greater length. Although the notion of syllable does play a role in the phonetics of 1989 Loglan, neither 1989 Loglan nor any earlier version had a formal definition of syllables or syllable boundaries. It became clear to me in my work since 2013 that a precise definition of the syllable was needed to define the notion of borrowed predicate successfully. When I had defined it, it was natural to add the requirement that Loglan names must resolve into syllables as well. Earlier versions of Loglan got away with not having a formal definition of syllable because structure words (cmapua) and non-borrowed predicates break up into formally defined units (unit cmapua and combining forms (djifoa or "affixes") whose boundaries are always syllable boundaries, though some of them are multisyllabic. These units also appear in our grammar, of course.

```
# the continuant consonants and the syllabic pairs they can form

continuant <- [mnlrMNLR]

syllabic <- (([mM] [mM] !(juncture? [mM]))/
([nN] [nN] !(juncture? [nN]))/
([rR] [rR] !(juncture? [rR]))/
([lL] [lL] !(juncture? [lL])))
```

The continuant consonants of Loglan are **mnlr**. These consonants may be used syllabically ("vocalically") as the "vowel" in a syllable.

When they are used syllabically, they are doubled. A syllabic consonant will not be preceded or followed in a word (even with intervening juncture) by an occurrence of the same continuant.

Syllabic consonants occur only in Loglan names and in borrowed predicates.

```
# the obligatory monosyllables, and these syllables when broken
#  by a usually bad syllable juncture.
# The i-final forms are not obligatory mono when followed by another i.

MustMono <- (([aeoAEO] [iI] ![iI]) /([aA] [oO]))

BrokenMono <- (([aeoAEO] juncture [iI] ![iI])/([aA] juncture [oO]))
```

```
# the obligatory and optional monosyllables.  Sequences of three of the same letter
# are averted.  Avoid formation of doubled i or u after ui or ui.

Mono <- (MustMono/([iI] !([uU] [uU]) V2)/([uU] !([iI] [iI]) V2))
```

Certain adjacent vowels (without intervening juncture) form dipthongs as a rule. These are

ai as in *aisle*. Not read this way in the context **aii**.

ei as in *weigh*, Not read this way in the context **eii**.

oi as in *boil*. Not read this way in the context **oii**.

ao as in *Tao*, or as *ow* in *cow*. This is clearly an irregularity in the phonetics of the language, which we simply accept.

These are called the *mandatory monosyllables*.

The rule `BrokenMono` describes a mandatory monosyllable broken by insertion of a juncture. Such pairs do not occur in predicates but may occur in names, such as **Lo-is**.

Pairs of vowels in which the first is **i** (except for **ii** in the context **iiu** and **uu** in the context **uui**) *may* be read as monosyllables but do not have to be. If **iV** is read as a monosyllable, the initial *i* takes the consonantal value of English **y**. If **uV** is read as a monosyllable, the initial *u* takes the consonantal value of English **w**.

These are called the *optional monosyllables*.

```
# vowel pairs of the form found in cmapua and djifoa.
# (other than the special IY, UY covered in the cmapua rules)

# The mysterious prohibition controls a permitted phonetic exception in djifoa gluing.
# cmapua are never followed directly by vocalic continuants in any case.

VV <- !(!MustMono V2 juncture? V2 juncture?
   [Rr] [Rr]) (!BrokenMono V2 juncture? V2)
```

This class describes pairs of regular vowels, some monosyllabic, some disyllabic, some pronounceable either way, which can occur in cmapua (structure words) and in djifoa (the building blocks of native compound predicates). The basic form of such a pair is a pair of vowels, possibly with a juncture inserted between them, which is not a broken mandatory monosyllable. Such a pair can only be followed by **rr** if it is a mandatory monosyllable (NOTE: I should

consider a more general formulation of this rule): this implements an esoteric exception in djifoa gluing which we will describe in its proper place.

This seems as good a place as any to remark that a pair of vowels pronounced disyllabically, either because it is neither a mandatory nor an optional monosyllable, or because it is an optional monosyllable broken by a juncture (in which case it must be pronounced disyllabically) or optionally pronounced as a disyllable, will be pronounced as the value of the first vowel followed by the value of the second vowel with the option of an intervening glottal stop (not expressed in the orthography). Such a glottal stop is both written and required in Lojban; in previous dialects of RLI Loglan such a use of the glottal stop was forbidden. As we will see shortly, if the two vowels in such a pair are the same, one of them must be stressed.

```
# the next vocalic unit to be chosen from a stream of vowels
# in a predicate or name.  This is different than in our Sources
# and formally described in the proposal.

NextVowels <- (MustMono/(V2 &MustMono)/Mono/
!([Ii] juncture [Ii] V1) !([Uu] juncture [Uu] V1) V2)

# 5/11/18 forbidding consonantal vowels to follow the same vowel.
```

The rule `NextVowels` is used to resolve a string of regular vowels without junctures into syllables, in the context of borrowed predicates or Loglan names. A priority scheme is used: one chooses the first possible kind of item on the list which follows as the next syllable.

mandatory monosyllable: One chooses a mandatory monosyllable in preference to anything else. Recall that in **aii**, **eii**, **oii** there is no mandatory monosyllable.

single regular vowel followed by a mandatory monosyllable: If a single regular vowel is followed by a mandatory monosyllable, take that vowel as the first syllable (and the mandatory monosyllable as the second).

optional monosyllable: One chooses an optional monosyllable which is not mandatory and whose second vowel is not the first vowel of a mandatory monosyllable as the next syllable. Recall that **iuu** and **uii** do not contain initial optional monosyllables.

single regular vowel: One chooses a single vowel if none of the options above hold, which cannot be **i** or **u** followed by a juncture followed by a copy of the same vowel further followed by a vowel (even an irregular one). (this **i-iV** or **u-uV** situation signals a parsing failure).

There is history behind this: earlier versions of Loglan had a very implausible rule for this which was in fact applied only in names. This procedure is generally actually applied to sequences of three or at most four vowels.

```
# the doubled vowels that trigger the rule that one of them must be stressed

DoubleVowel <- (([aA] juncture? [aA])/([eE] juncture? [eE])/
([oO] juncture? [oO])/([iI] juncture [iI])/([uU] juncture [uU])/
[iI] [Ii] &[iI]/[Uu] [uU] &[uU])
```

It is a phonetic rule of Loglan that when two regular vowels with the same value occur consecutively in a word and not as a monosyllable, one of them must be stressed. The pairs **i-i** and **u-u** are double vowels that trigger stress, as are the initial pairs in **iii** and **uuu**. NOTE: Examine my ruling on the triple forms.

```
# the mandatory "vowel" component of a syllable

Vocalic <- (NextVowels/syllabic/[Yy])
```

The mandatory vowel component of a syllable is either the regular vowel or pair of regular vowels extract by the previous rule, or a syllabic consonant, or **y**.

```
# the permissible initial pairs of consonants, and the same pairs possibly
# broken by syllable junctures.

Initial <- (([Bb] [Ll])/([Bb] [Rr])/([Cc] [Kk])/([Cc] [Ll])/([Cc] [Mm])/([Cc] [Nn])/([Cc] [

MaybeInitial <- (([Bb] juncture? [Ll])/([Bb]juncture?  [Rr])/([Cc]juncture?  [Kk])/([Cc] jun

# the permissible initial consonant groups in a syllable.  Adjacent consonants should be ini
# The group should not overlap a syllabic pair.  Such a group is of course followed by a voc

# this rule for initial consonant groups is stated in NB3.

# I forbid a three-consonant initial group to be followed by a syllabic pair.  This seems ob

InitialConsonants <- ((!syllabic C1 &Vocalic)/(!(C1 syllabic) Initial &Vocalic)/(&Initial C1

# the forbidden medial pairs and triples.  These are forbidden regardless of placement
```

```
# of syllable breaks.

# each of these is actually a single consonant followed by an initial, and the idea was to i
# would be hard to pronounce.  But the placement of the syllable break is not relevant to th
# Notice that the continuant syllabic pairs are excluded:  this prevents final consonants fr

NoMedial2 <- (([Bb] juncture? [Bb])/([Cc] juncture? [Cc])/([Dd] juncture? [Dd])/([Ff] junctu

NoMedial3 <- (([Cc] juncture? [Dd] juncture? [Zz])/([Cc] juncture? [Vv] juncture? [Ll])/([Nn

# The syllable.

# there are no formal rules about syllables as such in our Sources, which is odd since
# the definition of predicates depends on the placement of stresses on syllables.

# The first rule enforces the special point needed in complexes that
# a CVC syllable is preferred to a CV syllable where possible; we economically apply
# the same rule for default placement of syllable breaks everywhere, which is, with
# that exception, that the break comes as soon as possible.

# the SyllableB approach is taken if the following syllable would otherwise start with a syl

# the reason for this approach is that if one syllabizes a well formed complex in this way..
# the syllable breaks magically fall on the djifoa boundaries.  This does mean that the
# default break in <cabro> is <cab-ro>, which feels funny but is harmless.  Explicitly break
# it <ca-bro> will also parse correctly.

SyllableA <- (C1 V2 FinalConsonant (!Syllable FinalConsonant)?)

SyllableB <- (InitialConsonants? Vocalic (!Syllable FinalConsonant)? (!Syllable FinalConsona

Syllable <- ((SyllableA/SyllableB) juncture?)

# The final consonant in a syllable.  There may be one or two final consonants.  A pair of f
# consonants may not be a non-continuant followed by a continuant.  A final consonant may no
# start a forbidden medial pair or triple.

# The rule that a final consonant pair may not be a non-continuant followed by a continuant
# is natural and obvious but not in our Sources.  Such a pair of consonants would seem to
# naturally form another syllable.


# a pair of final consonants cannot be differently voiced

FinalConsonant <- !syllabic !(&Badvoice C1 !Syllable) (!(!continuant C1 !Syllable continuant
```

```
#!((!MaybeInitial)C1 juncture? !syllabic C1 juncture? !syllabic C1) !(&MaybeInitial C1 junct

# Here are various flavors of syllable we may need.

# this is a portmanteau definition of a bad syllable (the sort not allowed in a borrowing).

SyllableD <- &(InitialConsonants? ([Yy]/DoubleVowel/BrokenMono/&Mono V2 DoubleVowel/!MustMor

# this (below) is the kind of syllable which can exist in a borrowed predicate:
# it cannot start with a continuant pair, it cannot have a y as vocalic unit,
# and its vocalic unit (whether it has one or two regular vowels)
# cannot be involved in a double vowel or an explicitly broken
# mandatory monosyllable.

BorrowingSyllable <- !syllabic (!SyllableD) Syllable

# this is the final syllable of a predicate.  It cannot be followed
# without pause by a regular vowel.

VowelFinal <- InitialConsonants? Vocalic juncture? !V2

# syllables with syllabic consonant vocalic units
# this class is only used in borrowings, and we *could* reasonably
# require it to be followed by a vowel.  But I won't for now.
# for gluing this restriction would work, but we might literally borrow predicates
# with syllabic continuant pronunciations.

SyllableC <- (&(InitialConsonants? syllabic) Syllable)

# syllables with y

SyllableY <- (&(InitialConsonants? [Yy]) Syllable)

# an explicitly stressed syllable.

StressedSyllable <- ((SyllableA/SyllableB) [\'*])

# a final syllable in a word, ending in a consonant.

NameEndSyllable <- (InitialConsonants? (syllabic/Vocalic &FinalConsonant) FinalConsonant? Fi

# the pause classes actually hang on the letter before the pause.

# whitespace which might or might not be a pause.

maybepause <- (V1 [\'*]? [ ]+ C1)
```

```
# explicit pauses:  these are whitespace before a vowel or after a consonant, or comma marke

pause <- ((C1 [\'*]? [ ]+ &letter)/(letter [\'*]? [ ]+ &V1)/(letter [\'*]? [,] [ ]+ &letter)

# these are final syllables in words followed by whitespace which might not be a pause.
# the definition actually doesnt mention the maybepause class.

MaybePauseSyllable <- InitialConsonants? Vocalic ['*]? &([ ]+ &C1)

# The full analysis of names.

# a name word (without initial marking) is resolvable into syllables and ends with a consona

PreName <- ((Syllable &Syllable)* NameEndSyllable)

# this is a busted name word with whitespace in it -- but not whitespace at which one has to

BadPreName <- (MaybePauseSyllable [ ]+/Syllable &Syllable)* NameEndSyllable

# This is a name marker followed by a consonant initial name word without pause.

# I deployed a minimal set of name marker words; I can add the others whenever.
# I have decided (see below) to retain the social lubrication words as vocative markers
# *without* making them name markers, so one must pause <Loi, Djan>.  By not allowing
# freemods right after vocative markers in the vocative rule, I make <Loi hoi Djan> work as
# without pause.

# MarkedName <- &caprule ((([Ll] !pause [Aa] juncture?)/ ([Hh] [Oo] !pause [Ii] juncture?) /

MarkedName <- &caprule ((([Ll] !pause [Aa] juncture?)/ ([Hh] [Oo] !pause [Ii] juncture?) /


# This is an unmarked name word with a false name marker in it.

FalseMarked <- (&PreName (!MarkedName character)* MarkedName)

# This is the full definition of name words.  These are either marked consonant initial name
# names without false name markers beginning with explicit pauses (either comma marked or vo
# and name markers followed, with or without pause, by name words.  In the latter case there
# whitespace before a vowel initial name.

# a series of names without false name markers and names marked with ci, separated by spaces

# there is a look ahead at the grammar: a NameWord can be followed without explicit pause (t
# a pause in speech!) by another
```

```
# kind of utterance only in a serial name when what follows is of the form <ci> predunit, to
# in the name.

NameWord <- (&caprule MarkedName/([,] [ ]+ !FalseMarked &caprule PreName)/(&V1 !FalseMarked

# this is the minimal set of name marker words we are using.  We may add more.

# I am contemplating adding the words of social lubrication as name markers, but in a more r
# way that in the last provisional parser, in which I made them full-fledged vocative marker
# I preserved their status as vocative markers without restoring their status as name marker

# adding <mue> as a name marker

namemarker <- ([Ll] [Aa] juncture?/[Hh][Oo][Ii] juncture?/([Hh] [Uu] juncture? [Ee] juncture

# this is the bad name marker phenomenon that needs to be excluded.  This captures the idea
# that what follows the name could be pronounced without pause as a name word according to t
# orthography, but the fact that whitespace is present shows that this is not the intention.

# it is worth noting that name markers at heads of name words pass this test
# (because I omitted the test that what follows is not a PreName in the interests
# of minimizing lookahead);
# but this test is only applied to strings that have already been determined not to
# be of class NameWord.

badnamemarker <- namemarker !V1 [, ]? [ ]* BadPreName

# we test for the bad name marker condition at the beginning of each stream of cmapua,
# and streams of cmapua stop before name markers (and may resume at a name marker
# if neither a NameWord nor the bad marker condition is found).

# We have at any rate completely solved the phonetic problem of names and their markers.

# predicate start tests:  the idea is the same as class "connective" above, to recognize
# the start of a predicate without recursive appeals to the whole nasty definition of predic
# The reason to do it is to recognize when CV^n followed by CC cannot be a cmapua unit.

# New implementation 4/28/2019.  This allows only (C)V(V)(V) before the pair of vowels, for
# potential lookahead.

Vthree <- (V2 juncture?) (V2 juncture?) (V2 juncture?)

Vfour <- (V2 juncture?) (V2 juncture?) (V2 juncture?) (V2 juncture?)

# predicate starting with two or three consonants:  rules out CC(C)V(V) forms.  Junctures in
# the initial consonant group ignored.
```

```
predstartA1 <- (&MaybeInitial C1 juncture? MaybeInitial/MaybeInitial) &V2 !(V2 stress !Mono

# an apparent cmapua unit followed by a consonant group which cannot start a predicate -- CV

predstartA2 <- C1 V2 juncture? (V2 juncture?)? !predstartA1 C1 juncture? C1

# a stressed CV^n before a consonant group (CV(V) case)

predstartA3 <- C1 !Vthree (!StressedSyllable V2 juncture?)? &StressedSyllable V2 V2? junctur

# other (C)V^n followed by nonpredicate

predstartA4 <- C1? V2 juncture? (V2 juncture?)?  (V2 juncture?)? !predstartA1 !(MaybeInitial

# other stressed (C)V^n followed by consonant group

predstartA5 <- C1? !Vfour (!StressedSyllable V2 juncture?)? (!StressedSyllable V2 juncture?)

# forms with y; implemented CVVhy alternative for CVV cmapua

predstartA6 <- C1 (V2 juncture?) (V2 juncture? [Hh]?/C1 juncture? (C1 juncture?)?) [Yy]

predstart <- predstartA1/predstartA2/predstartA3/predstartA4/predstartA5/predstartA6

# it is worth noting that in the sequel we have systematically replaced tests &Cmapua
# with !predstart.  The former involves lots of lookahead and was causing recursion crashes
# in Python.  The phonetics and the grammar are both structured so that any string
# starting with a name marker is tested for NameWord-hood before it is tested for
# cmapua-hood; the only thing it is tested for later is predicate-hood, and predstart
# is a rough and ready test that something might be a predicate (and at any rate
# cannot be a cmapua).

# this class requires pauses before it, after all the phonetic word classes.
# what is being recognized is the beginning of a logical connective.

# To avoid horrible recursion problems, giving this a concrete phonetic definition
# without much lookahead.  This can go right up in the phonetics section if it works
# (and here it is!).

# single vowel cmapua syllables early for connectives

a <- ([Aa] !badstress juncture? !V1)

e <- ([Ee] !badstress juncture? !V1)
```

```
i <- ([Ii] !badstress juncture? !V1)

o <- ([Oo] !badstress juncture? !V1)

u <- ([Uu] !badstress juncture? !V1)


Hearly <- (!predstart [Hh])

Nearly <- (!predstart [Nn])


# these appear here for historical reasons and could be moved later


connective <- [ ]* !predstart ([Nn] [Oo] juncture?)? (a/e/o/u/Hearly a/Nearly UU) juncture?


# cmapua units starting with consonants.  This is the exact description from NB3.  The fancy
# three cases is enforcing the rule about pausing before a following predicate if stressed.

# consonant initial cmapua units may not be followed by vowels without pause.

# I am adding <iy> and <uy> (always monosyllable, yuh and wuh) as vowel pairs permitted in V
# it is worth noting that the "yuh" and "wuh" pronunciations of these diphthongs
# are surprising to the English-reading eye.
# The use for this envisaged is that the name <ziy> of Y becomes easy to introduce.  Adding
# is always nice, and these words seem pronounceable.  I also made <yfi> possible:  Y now ha
# regular names.

CmapuaUnit <- (C1 Mono juncture? V2 !(['*] [ ]* &C1 predstart) juncture? !V1/C1 (VV/[Ii][Yy]

# A stream of cmapua is read until the start of a predicate or a name marker word or an alie
# the stream might resume with a name marker word if it does not in fact start a name word a
# word due to inexplicit whitespace (doesn't satisfy the bad name marker condition).

# we force explicit comma pauses before logical connectives, but not before vowel initial cm
# other conditions force at least whitespace, which does stand for a pause, before such word

# detect starts of quotes or parentheses with <li> or <kie>

likie <- ([Ll] [Ii] juncture? !V1/[Ki] [Ii] juncture? [Ee] juncture? !V1)

# a special provision is made for NO UI forms as single words.  <yfi> is supported.

Cmapua <- &caprule !badnamemarker (!predstart (VV/[Ii][Yy]/[Uu][Yy]) !(['*] [ ]* &C1 predsta
```

```
# I have apparently now completely solved the problem of parsing cmapua as well as name word

# Now for predicates.

# the elementary djifoa (not borrowings)

# various special flavors of these djifoa will be needed.
# These are the general definitions.

# The NOY and Bad forms are for use for testing candidate borrowings for resolution
# with bad syllable break placements.  Borrowings do not contain Y...

# CVV djifoa with phonetic hyphens.

# added checks to all cmapua classes:  the vowel final ones, when not phonetically hyphenate
# be followed by a regular vowel.  This is crucial for getting the syllable analysis and the
# analysis to end at the same point.

# allowing h to be inserted before y in CVVy djifoa for a CVVhy form.

# allowing -r glue to be expressed as -rr

CVV <- C1 VV (juncture? [Hh]? [Yy] [-]? &(Complex) /juncture? [Rr] [Rr]? juncture? &C1/[Nn]

CVVNoHyphen <- C1 VV juncture? !V2

CVVHiddenStress <- C1 &DoubleVowel V1 [-]? V1 ([-]? [Hh]? [Yy] [-]? &Complex /[Rr] [-]? &C1/

CVVFinalStress <- C1 VV (['*] [Hh]? [Yy] [-]? &Complex /[Rr] ['*] &C1/['*] [Rr] [Rr] junctur

CVVNOY <- C1 VV (juncture? [Rr] [Rr]? juncture? &C1/[Nn] juncture? &[Rr]/juncture? !V2)

CVVNOYFinalStress <- C1 VV ([Rr] ['*] &C1/['*] [Rr] [Rr] juncture? &C1/[Nn] ['*] &[Rr]/['*]

CVVNOYMedialStress <- C1 !BrokenMono V2 ['*] V2 [-]? !V2

# CCV djifoa with phonetic hyphens.

CCV <- Initial V2 (juncture? [Yy] [-]? &letter/juncture? !V2)

CCVStressed <- Initial V2 (['*] [Yy] [-]? &letter/['*] !V2)

CCVNOY <- Initial V2 juncture? !V2

CCVBad <- MaybeInitial V2 juncture? !V2
```

```
CCVBadStressed <- MaybeInitial V2 ['*] !V2


# CVC djifoa with phonetic hyphens.  These cannot be final and are always followed by a cons
# -y form may be followed by a vowel...
# an eccentric syllable break is supported if the CVC is y-hyphenated:
# <me-ky-kiu> and <mek-y-kiu> are both legal.  The default is the latter.

CVC <- (C1 V2 !NoMedial2 !NoMedial3 C1 (juncture? [Yy] [-]? &letter/juncture? &C1)/C1 V2 jun

CVCStressed <- (C1 V2 !NoMedial2 !NoMedial3 C1 (['*] [Yy] [-]? &letter/['*] &letter)/C1 V2 |

CVCNOY <- C1 V2 !NoMedial2 !NoMedial3 C1 juncture? &C1

CVCBad <- C1 V2 !NoMedial2 !NoMedial3 juncture? C1 &C1

CVCNOYStressed <- C1 V2 !NoMedial2 !NoMedial3 C1 ['*] &C1

CVCBadStressed <- C1 V2 !NoMedial2 !NoMedial3 ['*] C1 &C1

# the five letter forms (always final in complexes)

CCVCV <- Initial V2 juncture? C1 V2 [-]? !V2

CCVCVStressed <- Initial V2 ['*] C1 V2 [-]? !V2

CCVCVBad <- MaybeInitial V2 juncture? C1 V2 [-]? !V2

CCVCVBadStressed <- MaybeInitial V2 ['*] C1 V2 [-]? !V2

CVCCV <- (C1 V2 juncture? Initial V2 [-]? !V2/C1 V2 !NoMedial2 C1 juncture? C1 V2 [-]? !V2)

CVCCVStressed <- (C1 V2 ['*] Initial V2 [-]? !V2/C1 V2 !NoMedial2 C1 ['*] C1 V2 [-]? !V2)

# the medial five letter djifoa

CCVCY <- Initial V2 juncture? C1 [Yy] [-]?

CVCCY <- (C1 V2 juncture? Initial [Yy] [-]?/C1 V2 !NoMedial2 C1 juncture? C1 [Yy] [-]?)

CCVCYStressed <- Initial V2 ['*] C1 [Yy] [-]?

CVCCYStressed <- (C1 V2 ['*] Initial [Yy] [-]?/C1 V2 !NoMedial2 C1 ['*] C1 [Yy] [-]?)

# to reason about resolution of borrowings into both syllables and djifoa (we want to exclud
```

```
# but we need to define it adequately) we need to recognize where to stop.  A predicate word
# at a non-character (not a letter or syllable mark: whitespace, comma or terminal punctuati
# has an explicit or deducible penultimate stress.  Borrowings do not contain doubled vowels
# have to have explicit stress in the latter case.

# analysis:  the stressed tail consists of a stressed syllable followed by an unstressed syl
# identifying an unstressed final syllable is complicated by recognizing which CVV combinati
# be one syllable.  This will either be an explicitly stressed syllable followed by a single
# or a syllable suitable to be stressed followed by an explicitly final syllable.  CVV djifo
# contain both syllables in a tail and of course the five letter djifoa have to be tails.  A
# SyllableC (with a continuant) may intervene.

# tail of a borrowing with an explicit stress

BorrowingTail1 <- !SyllableC &StressedSyllable BorrowingSyllable (!StressedSyllable &Syllabl

# tail of a borrowing or borrowing djifoa with no explicit stress

BorrowingTail2 <- !SyllableC BorrowingSyllable (!StressedSyllable &SyllableC BorrowingSyllab

# tail of a stressed borrowing djifoa, different because stress is shifted to the end

BorrowingTail3 <- !SyllableC !StressedSyllable BorrowingSyllable (!StressedSyllable &Syllabl

BorrowingTail <- BorrowingTail1 / BorrowingTail2

# short forms that are ruled out:  CCVV and CCCVV forms.

CCVV <- (InitialConsonants V2 juncture? V2 juncture? !character / InitialConsonants V2 ['*]

# VCCV and some related forms are ruled out (rule predstartF above is about this)

# a continuant syllable cannot be initial in a borrowing and there cannot be successive cont
# syllables.  There really ought to be no more than one!

# borrowing, before checking that it doesnt resolve into djifoa

PreBorrowing <- &predstart!CCVV!Cmapua!SyllableC(!BorrowingTail!(StressedSyllable)!(Syllable

# ditto for an explicitly stressed borrowing

StressedPreBorrowing <- &predstart!CCVV!Cmapua!SyllableC(!BorrowingTail!(StressedSyllable)!(

# borrowing djifoa without explicit stress (before resolution check)

PreBorrowing2 <- &predstart!CCVV!Cmapua!SyllableC(!BorrowingTail!(StressedSyllable)!(Syllabl
```

```
# stressed borrowing djifoa (before resolution check).

PreBorrowing3 <- &predstart!CCVV!Cmapua!SyllableC(!BorrowingTail3!(StressedSyllable)!(Syllab

# Now comes the problem of trying to say that a preborrowing cannot resolve into cmapua.  Th
# recognizing the tail, so making sure that the two resolutions stop in the same place.

# we know because it is a borrowing that there is at most one explicit stress, and it has to
# in one of the cmapua!  This should make it doable.

# borrowing djifoa are terminated with y, so the final djifoa needs to take this into accoun

# the idea behind both djifoa analyses is the same.  If we end with a final djifoa followed
# a non-character, we improve our chances of ending the syllable analysis at the same point.
# this by identifying djifoa with stresses in them:  a medially stressed djifoa must be the
# (and the syllable analysis will find its stressed syllable and end at its final syllable,
# that djifoa cannot be followed by vowels ensuring that the syllable analysis cannot overru
# When the djifoa is finally stressed, the complex analysis ends with a further djifoa guara
# just one syllable, and the syllable analysis again will stop in the same place.  The media
# and borrowing djifoa of course are finally stressed mod an additional unstressed syllable
# by the syllable analysis, because it allows one to ignore an actually penultimate syllable
# a syllabic consonant.  In the case where we never find a stress and end up at a final djif
# analysis will carry right through to the same final point.

# in the attempted resolution of borrowings, our life is easier because we do not have
# borrowing djifoa or medial five letter forms to consider, or any forms with y-hyphens.

RFinalDjifoa <- (CCVCVBad/CVCCV/CVVNoHyphen/CCVBad/CVCBad) (&[Yy]/!character)

RMediallyStressed <- (CCVCVBadStressed/CVCCVStressed/CVVNOYMedialStress)

RFinallyStressed <- (CVVNOYFinalStress/CCVBadStressed/CVCBadStressed/CVCNOYStressed)

BorrowingComplexTail <- (RMediallyStressed/RFinallyStressed (&(C1 Mono) CVVNoHyphen/CCVBad)/

ResolvedBorrowing <- (!BorrowingComplexTail(CVVNOY/CCVBad/CVCBad))* BorrowingComplexTail

# borrowed predicates

Borrowing <- !ResolvedBorrowing &caprule PreBorrowing !([ ]* (connective))

# explicitly stressed borrowed predicates

StressedBorrowing <- !ResolvedBorrowing &caprule StressedPreBorrowing !([ ]* &V1 Cmapua)
```

```
#This is the shape of non-final borrowing djifoa.  Notice that a final stress is allowed.
#The curious provision for explicitly stressing a borrowing djifoa and pausing is supported.

# borrowing djifoa without explicit stress (stressed ones are not of this class!)
# Note that one can pause after these (explicitly, with a comma, in which case the stress mu

BorrowingDjifoa <- !ResolvedBorrowing &caprule PreBorrowing2 (['*] [y] [,] [ ]+/juncture? [y

# stressed borrowing djifoa finally implemented!

StressedBorrowingDjifoa <- !ResolvedBorrowing &caprule PreBorrowing3 [y] [-]? ([,] [ ]+)?

# We resolve complexes twice, once into syllables and once into djifoa.  We again have to en
# we end up in the same place!  The syllable resolution is very similar to that of borrowing
# the unstressed middle syllable of the tail can be a SyllableY, and can also be a
# SyllableC if the final djifoa is a borrowing.

# A stressed borrowing djifoa with the property that the tail is still a phonetic complex is
# a unit for this analysis.

# note here that I specifically rule out a complex being followed without pause by y.  I do
# this out for the vowel final djifoa because they can be followed by y at the end of a borr
# djifoa.

PhoneticComplexTail1 <- !SyllableC !SyllableY &StressedSyllable Syllable (!StressedSyllable

PhoneticComplexTail2 <- !SyllableC !SyllableY Syllable (!StressedSyllable &(SyllableC/Syllab

PhoneticComplexTail <- PhoneticComplexTail1 / PhoneticComplexTail2

# note the explicit predstart test here.

PhoneticComplex <- &predstart!CCVV!Cmapua!SyllableC(StressedBorrowingDjifoa &PhoneticComplex

# the analysis of final djifoa and stressed djifoa differs only in details from
# what is above for resolution of borrowings.  The issues about CVV djifoa with doubled
# vowels are rather exciting.

# a stressed borrowing djifoa with the tail still a phonetic complex is a black box unit for
# this construction.

# My approach imposes the restriction on JCB's "pause after a borrowing djifoa" idea that wh
# the pause must itself contain a penultimate stress:  <igllu'ymao> is a predicate but <igll
# while <iglluy', gudmao> is a predicate.

# the analysis of the djifoa resolution process is the same as above, with additional remark
```

```
# about doubled vowel syllables:  notice that where the complex tail involved a doubled vowe
# without explicit stress, we insist on that djifoa or the single-syllable next djifoa endin
# a non-character:  in the absence of explicit stress, we always rely on whitespace or punct
# to indicate the end of the predicate.

# all sorts of subtleties about borrowings and borrowing djifoa are finessed by always looki
# them first.  There are no restrictions re fronts of borrowings or borrowing djifoa looking
# djifoa;  the fact that borrowing djifoa end in y and borrowings do not contain y makes it
# possible to tell when one is looking at the head of a borrowing djifoa.  Regular djifoa ju
# djifoa need to be y-hyphenated so as not to be absorbed into the front of the borrowing (I
# that I actually need to impose a formal rule to this effect, though I am not absolutely ce
# be difficult to formulate [and does appear in the previous version, where it is a truly un
# of PEG code]).

FinalDjifoa <- (Borrowing/CCVCV/CVCCV/CVVNoHyphen/CCVNOY) !character

MediallyStressed <- (StressedBorrowing/CCVCVStressed/CVCCVStressed/CVVNOYMedialStress)

FinallyStressed <-(StressedBorrowingDjifoa/CCVCYStressed/CVCCYStressed/CVVFinalStress/CCVStr

ComplexTail <- (CVVHiddenStress (&(C1 Mono) CVVNoHyphen/CCVNOY) !character/FinallyStressed (

PreComplex <-  (!CVVHiddenStress (!ComplexTail)(StressedBorrowingDjifoa &PhoneticComplex/Bor

# originally I had complicated tests here for the conditions under which an initial
# CVC cmapua has to be y-hyphenated:  I was being wrong headed, the predstart rules
# already enforce this (in the bad cases, the initial CV- falls off).  The user will
# simply find that they cannot put the word together otherwise.  The previous version
# did need this test because it actually used full lookahead to check for the start of a pre

Complex <- &caprule &PreComplex PhoneticComplex !([ ]* (connective))

# format for the LI quote and KIE parenthesis

LiQuote <- (&caprule [Ll][Ii]juncture? comma2? [\"] phoneticutterance [\"] comma2? &caprule

# the condition on Word that a Cmapua is not followed by another Cmapua
# with mere whitespace between was used by <liu> quotation, but is now redundant,
# because I have required that <liu> quotations be closed with explicit pauses in all cases.

Word <- (NameWord / Cmapua !([ ]*Cmapua)/ Complex/CCVNOY)

# it is an odd point that all borrowings parse as complexes -- so when I parsed all the word
# parsed as complexes.  A borrowing is a complex consisting of a single final borrowing djif
# I did redesign this so that borrowings are parsed as borrowings.  (This is the class
# I used to parse the dictionary).
```

```
# Yes, CVC djifoa do get parsed as names in the dictionary, so the CVC case here is redundar
# think that only the CCV djifoa actually get parsed as such.

SingleWord <- (Borrowing !./Complex !./ Word !./PreName !. /CCVNOY) !.

# name word appearing initially without leading spaces is important, because one type of Nar

phoneticutterance1 <- (NameWord /[ ]* LiQuote/[ ]* NameWord/[ ]* AlienWord/[ ]*Cmapua/[ ]* 

phoneticutterance <- (phoneticutterance1/[,][ ]+/terminal)+

# consonants and vowel groups in cmapua

# as noted above, !predstart stands in for the computationally disastrous &Cmapua

badstress <- ['*] [ ]* &C1 predstart

B <- (!predstart [Bb])

C <- (!predstart [Cc])

D <- (!predstart [Dd])

F <- (!predstart [Ff])

G <- (!predstart [Gg])

H <- (!predstart [Hh])

J <- (!predstart [Jj])

K <- (!predstart [Kk])

L <- (!predstart [Ll])

M <- (!predstart [Mm])

N <- (!predstart [Nn])

P <- (!predstart [Pp])

R <- (!predstart [Rr])

S <- (!predstart [Ss])
```

```
T <- (!predstart [Tt])

V <- (!predstart [Vv])

Z <- (!predstart [Zz])

# the monosyllabic classes may be followed by one vowel
# if they start a Cvv-V cmapua unit;  the others may never
# be followed by vowels.  Classes ending in -b are
# used in Cvv-V cmapua units.

# the single vowel classes were moved before the class
# connective in the phonetics section.


V3 <- juncture? V2 !badstress

AA <- ([Aa] juncture? [Aa] !badstress juncture? !V1)

AE <- ([Aa] juncture? [Ee]  !badstress juncture? !V1)

AI <- ([Aa] [Ii]  !badstress juncture? !(V1))

AO <- ([Aa] [Oo]  !badstress juncture? !(V1))

AIb <- ([Aa] [Ii]  !badstress juncture? &(V2 juncture? !V1))

AOb <- ([Aa] [Oo]  !badstress juncture? &(V2 juncture? !V1))

AU <- ([Aa] juncture? [Uu]  !badstress juncture? !V1)

EA <- ([Ee] juncture? [Aa]  !badstress juncture? !V1)

EE <- ([Ee] juncture? [Ee]  !badstress juncture? !V1)

EI <- ([Ee] [Ii]  !badstress juncture? !(V1))

EIb <- ([Ee] [Ii]  !badstress juncture? &(V2 juncture? !V1))

EO <- ([Ee] juncture? [Oo]  !badstress juncture? !V1)

EU <- ([Ee] juncture? [Uu]  !badstress juncture? !V1)

IA <- ([Ii] juncture? [Aa]   !badstress juncture? !(V1))

IE <- ([Ii] juncture? [Ee]  !badstress juncture? !(V1))
```

```
II <- ([Ii] juncture? [Ii]  !badstress juncture? !(V1))

IO <- ([Ii] juncture? [Oo]  !badstress juncture? !(V1))

IU <- ([Ii] juncture? [Uu]   !badstress juncture? !(V1))

IAb <- ([Ii] juncture? [Aa]   !badstress juncture? &(V2 juncture? !V1))

IEb <- ([Ii] juncture? [Ee]  !badstress juncture? &(V2 juncture? !V1))

IIb <- ([Ii] juncture? [Ii]  !badstress juncture? &(V2 juncture? !V1))

IOb <- ([Ii] juncture? [Oo]  !badstress juncture? &(V2 juncture? !V1))

IUb <- ([Ii] juncture? [Uu]   !badstress juncture? &(V2 juncture? !V1))

OA <- ([Oo] juncture? [Aa]  !badstress juncture? !V1)

OE <- ([Oo] juncture? [Ee]  !badstress juncture? !V1)

OI <- ([Oo] [Ii]  !badstress juncture? !(V1))

OIb <- ([Oo] [Ii]  !badstress juncture? &(V2 juncture? !V1))

OO <- ([Oo] juncture? [Oo]  !badstress juncture? !V1)

OU <- ([Oo] juncture? [Uu]   !badstress juncture? !V1)

UA <- ([Uu] juncture? [Aa]   !badstress juncture? !(V1))

UE <- ([Uu] juncture? [Ee]  !badstress juncture? !(V1))

UI <- ([Uu] juncture? [Ii]   !badstress juncture? !(V1))

UO <- ([Uu] juncture? [Oo]  !badstress juncture? !(V1))

UU <- ([Uu] juncture? [Uu]  !badstress juncture? !(V1))

UAb <- ([Uu] juncture? [Aa]   !badstress juncture? &(V2 juncture? !V1))

UEb <- ([Uu] juncture? [Ee]  !badstress juncture? &(V2 juncture? !V1))

UIb <- ([Uu] juncture? [Ii]   !badstress juncture? &(V2 juncture? !V1))

UOb <- ([Uu] juncture? [Oo]  !badstress juncture? &(V2 juncture? !V1))
```

```
UUb <- ([Uu] juncture? [Uu]  !badstress juncture? &(V2 juncture? !V1))

# adding the new IY and UY, which might see use some time.
# they are mandatory monosyllables but do not take a possible additional
# following vowel as the regular ones do.  So far only used in <ziy>.

IY <- [Ii] [Yy] !badstress juncture? !V1

UY <- [Uu] [Yy] !badstress juncture? !V1

# this is a pause not required by the phonetics.  This is the only
# sort of pause which could in principle carry semantic freight (the
# pause/GU equivalence beloved of our Founder) but we have abandoned
# this.  There is one place, after initial <no> in an utterance, where
# a pause can have effect on the parse (but not on the meaning, I believe,
# unless a word break is involved).

# this class should NEVER be used in a context which might follow
# a name word.  In previous versions, pauses after name words were included
# in the name word;  this is not the case here, so a PAUSE
# after a name word would not be recognized as a mandatory pause.

# in any event, as long as we stay away from pause/GU equivalence, this
# is not a serious issue!

# this class does do some work in the handling of issues surrounding the legacy
# shape of APA connectives, concerning which the less said, the better.

PAUSE <- [,] [ ]+ !(V1/connective) &caprule

# more punctuation

comma <- [,] [ ]+ &caprule

comma2 <- [,]? [ ]+ &caprule

# Part II Lexicography

# In this section I develop the grammar of words in Loglan.  I'll work by editing the origi

# I place the start of this section exactly here, just before two final items of
# punctuation, because these items of punctuation look forward not only to lexicography
# but to the full grammar!

# the end of utterance symbol <#> should be added in the phonetics
```

```
# section as a species of terminal marker. Done.  We do *not* actually
# endorse use of this marker, but we can notionally support it and it is in
# our sources.

end <- (([ ]* '#' [ ]+ utterance)/([ ]+ !.)/!.)

# this rule allows terminal punctuation to be followed by an inverse vocative,
# a frequent occurrence in Leith's novel, and something which makes sense.

period <- (([!.:;?] (&end/([ ]+ &caprule))) (invvoc period?)?)

# Letters with y will be special cases
# idea:  allow IY and UY (always monosyllables) as vowel combinations in cmapua only.
# done:  Y has a name now.  <yfi> is also added.

# the classes in this section after this point are the cmapua word classes of Loglan (if the
# I suppose the alien text classes are not really word classes, but they are lexicographic i
# Paradoxically, the PA and NI classes admit internal explicit pauses.  So of course do prec

# Loglan does admit true multisyllable cmapua:  there are words made of cmapua units which h
# units at which one cannot pause without breaking the word.  Lojban, I am told, does not.

# this version has the general feature that the quotation and alien text constructions are n
# they are supported by the phonetic rules (as dire exceptions, of course) and the grammatic
# conform with the phonetic layer.  Alien text and utterances quoted with <li>...<lu> can be
# LI only supports full utterances, for the moment.  All alien text constructors take the sa
# the vocative and inverse vocative *require* quotes to avoid misreading ungrammatical expre
# as correct (inverse) vocatives.

# the names <yfi>, <ziy> for Y are supported.  The Ceo names are left as they are.  I decide
# of letteral pronouns is actually a reasonable use of short words, and the Ceio words are t

TAIO <- (V1 juncture? M a/V1 juncture? F i/V1 juncture? Z i/!predstart C1 AI/!predstart C1 I

# a negative suffix used in various contexts.  Always a suffix:  its use as a prefix in tens
# think still supported in LIP.  Ambiguities demonstrably followed from this usage (an examp
# of non-ambiguity of 1989 Loglan was compromised by the opaque lexicography).

NOI <- (N OI)

# the logical connectives.  [AO] is the class of core logical connectives.  [A] is the fully
# possible nu- (always in nuno- or nuu) and no- prefixes, possible -noi suffix, and possible
# with -fi (our new proposal) or an explicit pause.

AO <- &Cmapua (a/e/o/u/H a/N UU)
```

```
A <- [ ]* !predstart !TAI0 (N [o])? A0 NOI? !([ ]+ PANOPAUSES PAUSE) !(PANOPAUSES !PAUSE [ ,

# 4/18 in connected sentpreds, fi must be used to close, not a pause.

# A2 <- [ ]* !predstart !TAI0 (N [o])? A0 NOI? !([ ]+ PANOPAUSES PAUSE) !(PANOPAUSES !PAUSE


# A not closed with -fi or a pause

ANOFI <- [ ]* (!predstart !TAI0 ( (N [o])? A0 NOI? PANOPAUSES?))

A1 <- A

# versions of A with different binding strength

ACI <- (ANOFI C i)

AGE <- (ANOFI G e)

# a tightly binding series of logical connectives used to link predicates
# this also includes the fusion connective <ze> when used between predicates.

CA0 <- (( (N o)? ((C a)/(C e)/(C o)/(C u)/(Z e)/(C i H a)/N u C u)) NOI?)

CA1 <- (CA0 !([ ]+ PANOPAUSES PAUSE) !(PANOPAUSES !PAUSE [ ,]) (PANOPAUSES ((F i)/&PAUSE))?)

CA1NOFI <- (CA0 PANOPAUSES?)

CA <- ([ ]* CA1)

# the fusion connective when used in arguments

ZE2 <- ([ ]* (Z e))

# sentence connectives.  [I] is the class of utterance initiators (no logical definition).
# the subsequent classes are inhabited by sentence logical connectives with various binding
# strengths.

I <- ([ ]* !predstart !TAI0 i !([ ]+ PANOPAUSES PAUSE) !(PANOPAUSES !PAUSE [ ,]) (PANOPAUSES

ICA <- ([ ]* i ((H a)/CA1))

ICI <- ([ ]* i CA1NOFI? C i)

IGE <- ([ ]* i CA1NOFI? G e)
```

```
# forethought logical connectives

KA0 <- ((K a)/(K e)/(K o)/(K u)/(K i H a)/(N u K u))

# causal and comparative modifiers

KOU <- ((K OU)/(M OI)/(R AU)/(S OA)/(M OU)/(C IU))

# negative and converse forms

KOU1 <- (((N u N o)/(N u)/(N o)) KOU)

# the full type of forethought connectives, adding the causal and comparative connectives

KA <- ([ ]* ((KA0)/((KOU1/KOU) K i)) NOI?)

# the last component of the KA...KI... structure of forethought connections

KI <- ([ ]* (K i) NOI?)

# causal and comparative modifiers which are *not* forethought connectives

KOU2 <- (KOU1 !KI)

# a test used to at least partially enforce the penultimate stress rule on quantifier predic

BadNIStress <- ((C1 V2 V2? stress (M a)? (M OA)? NI RA)/(C1 V2 stress V2 (M a)? (M OA)? NI R

# root quantity words, including the numerals

NI0 <- (!BadNIStress ((K UA)/(G IE)/(G IU)/(H IE)/(H IU)/(K UE)/(N EA)/(N IO)/(P EA)/(P IO)/

# the class of SA roots, which modify quantifiers

SA <- (!BadNIStress ((S a)/(S i)/(S u)/(IE (comma2? !IE SA)?)) NOI?)

# the family of quantifiers which double as suffixes for the quantifier predicates
# this class perhaps should also include some other quantifier words. <re> for example ought
# No action here, just a remark.

RA <- (!BadNIStress ((R a)/(R i)/(R o)/R e/R u))

# re and ru added to class RA 5/11/18

# quantifier units consisting of a NI or RA root with <ma> 00 or <moa> 000 appended; to <moa
```

```
# append a digit to iterate <moa>:  <fomoate> is four billion, for example.  <rimoa>, a few

# a NI1 or RA1 may be followed by a pause before another NI word other than a numerical prec
# one is allowed to breathe in the middle of long numerals.  I question whether the pause
# provision makes sense in RA1.

NI1 <- ((NI0 (!BadNIStress M a)? (!BadNIStress M OA NI0*)?) (comma2 !(NI RA) &NI)?)

RA1 <- ((RA (!BadNIStress M a)? (!BadNIStress M OA NI0*)?) (comma2 !(NI RA) &NI)?)

# a composite NI word, optional SA prefix before a sequence of NI words or a RA word,
# or a single SA word [which will modify a default quantifier not expressed],
# possibly negated, connected with CA0 roots to other such constructs.

NI2 <- (( (SA? (NI1+/RA1))/SA) NOI? (CA0 ((SA? (NI1+/RA1))/SA) NOI?)*)

# a full NI word with an acronymic dimension (starting with <mue>, ending with a pause) or <
# and figure out its semantics.  An arbitrary name word may now be used as a dimension, as w

NI <- ([ ]* NI2 (&(M UE) Acronym (comma/&end/&period) !(C u)/comma2? M UE comma2? PreName !(

# mex is now identical with NI, but it's in use in later rules.

mex <- ([ ]* NI)

# a word used for various tightly binding constructions:  a sort of verbal hyphen.
# also a name marker, which means phonetic care is needed (pause after constructions with <c

CI <- ([ ]* (C i))

# Acronyms, which are names (not predicates as in 1989 Loglan) or dimensions (in NI above).
# units in acronym are TAI0 letterals, zV short forms for vowels, the dummy unit <mue>, and
# quantity units.  NI1 quantity units may not be initial. <mue> units may be preceded by pau
# An acronym has at least two units.

# it is worth noting that acronyms, once viewed as names, could be entirely suppressed as a
# grammar by really making them names (terminate them with -n).  I suppose a similar approac
# for dimensions, allowing any name word to serve as a dimension.  <mue> would be a name mar
# with dimensions in this case.  <temuedain>, three dollars.  Now supported.

Acronym <- ([ ]* &caprule ((M UE)/TAI0/(Z V2 !V2)) ((comma &Acronym M UE)/NI1/TAI0/(Z V2 (!V

# the full class of letterals, including the <gao> construction whose details I should look

TAI <- ([ ]* (TAI0/((G AO) !V2 [ ]* (PreName/Predicate/CmapuaUnit))))
```

```
# atomic non-letteral pronouns.

#4/15/2019 reserved <koo> for a Lojban style imperative pronoun, though not officially adopt

DA0 <- ((T AO)/(T IO)/(T UA)/(M IO)/(M IU)/(M UO)/(M UU)/(T OA)/(T OI)/(T OO)/(T OU)/(T UO)/

# letterals (not including <gao> constructions and atomic pronouns optionally suffixed with
# suffixed forms, because <ci> is a name marker.

DA1 <- ((TAI0/DA0) (C i ![ ] NI0)?)

# general pronoun words.

DA <- ([ ]* DA1)

# roots for PA words:  tense and location words, prepositions building relative modifiers.

PA0 <- (NI2? (N u !KOU)? ((G IA)/(G UA)/(P AU)/(P IA)/(P UA)/(N IA)/(N UA)/(B IU)/(F EA)/(F

# the form used for actual prepositions and suffixes to A words, with minimal pauses allowed
# these are built by concatenating KOU2 and PA0 units, then linking these with CA0 roots (wh
# no- prefixes and -noi suffixes, and next to which one *can* pause), optionally suffixed wi

PANOPAUSES <- ((KOU2/PA0)+ ((comma2? CA0 comma2?) (KOU2/PA0)+)*)

# prepositional words

PA3 <- ([ ]* PANOPAUSES)

# class PA can appear as tense markers or as relative modifiers without arguments; here paus
# are allowed not only next to CA0 units but between KOU2/PA units.  Like NI words, PA
# words are a class of arbitrary length constructions, and we think breaths within them
# (especially complex ones) are natural.

PA <- ((KOU2/PA0)+ (((comma2? CA0 comma2?)/(comma2 !mod1a)) (KOU2/PA0)+)*) !modifier

PA2 <- ([ ]* PA)

GA <- ([ ]* (G a))

# the class of tense markers which can appear before predicates.

PA1 <- ((PA2/GA))

# suffixes which indicate extent or remoteness/proximity of the action of prepositions.
```

```
ZI <- ((Z i)/(Z a)/(Z u))

# the primitive description building "articles".  These include <la> which requires special
# care in its use because it is a name marker.

LE <- ([ ]* ((L EA)/(L EU)/(L OE)/(L EE)/(L AA)/(L e)/(L o)/(L a)))

# articles which can be used with abstract descriptions:  these include some quantity words.
# this means that some abstract descriptions are semantically indefinites:  I wonder if this
# could be improved by having a separate abstract indefinite construction.

LEFORPO <- ([ ]* ((L e)/(L o)/NI2))

# the numerical/quantity article.

LIO <- ([ ]* (L IO))

# structure words for the ordered and unordered list constructions.

LAU <- ([ ]* (L AU))

LOU <- ([ ]* (L OU))

LUA <- ([ ]* (L UA))

LUO <- ([ ]* (L UO))

ZEIA <- ([ ]* Z EIb a)

ZEIO <- ([ ]* Z EIb o)

# initial and final words for quoting Loglan utterances.

LI1 <- (L i)

LU1 <- (L u)

# quoting Loglan utterances, with or without explicit double quotes (if they appear, they mu
# appear on both sides).  The previous version allowed quotation of names;  likely this shou
# be restored.

LI <- ([ ]* LI1 comma2? utterance0 comma2? LU1/[ ]* LI1 comma2? [\"] utterance0 [\"] comma2?

# the foreign name construction.  This is an alien text construction

LAO <- ([ ]* &([Ll] [Aa] [Oo]juncture?) AlienWord)
```

```
# the strong quotation construction.  This is an alien text construction.

LIE <- ([ ]* &([Ll] [Ii] juncture? [Ee]juncture?) AlienWord)

LIO1 <- ([ ]* &([Ll] [Ii] juncture? [Oo]juncture?) AlienWord)



# I am not sure this class is used at all.

LW <- Cmapua

# articles for quotation of words

LIU0 <- ((L IU)/(N IU))

# this now imposes the condition that an explicit comma pause (or terminal punctuation, or e
# Word or PreName quoted with <liu>.  This seems like a good idea, anyway.

# this class appeals to the phonetics.  Words and PreNames can be quoted.  The ability to qu
# here may remove the need to quote them with <li>...<lu>.  Of course, some Words are in fac
# than single words:  we will see whether the privileges afforded are used.  The final claus
# use of letterals as actual names of letters.

# added <niu>:  didn't make it a name marker.

LIU1 <- ([ ]* ([Ll]/[Nn])[iI] juncture? [Uu] juncture? !V1 comma2? (PreName/Word) &(comma/te

# the construction of foreign and onomatopoeic predicates.  These are alien text constructio

SUE <- ([ ]* &([Ss] [Uu] juncture? [Ee] juncture?/[Ss] [Aa] [Oo] juncture?) AlienWord)

# left marker in a predicate metaphor construction

CUI <- ([ ]* (C UI) )

# other uses of GA

GA2 <- ([ ]* (G a) )

# ge/geu act as "parentheses" to make an atomic predicate from a complex metaphorically
# and logically connected predicates;  <ge> has other left marking uses.

GE <- ([ ]* (G e) )
```

```
GEU <- ([ ]* ((C UE)/(G EU)) )

# final marker of a list of head terms

GI <- ([ ]* ((G i)/(G OI)) )

# used to move a normally prefixed metaphorical modifier after what it modifies.

GO <- ([ ]* (G o) )

# marker for second and subsequent arguments before the predicate; NEW

GIO <- ([ ]* (G IO) )

# the generic right marker of many constructions.

GU <- ([ ]* (G u) )

# various flavors of right markers.

# It should be noted that at one point I executed a program of simplifying these to
# reduce the likelihood that multiple <gu>'s would ever be needed to close an utterance.
# first of all, I made the closures leaner, moving them out of the classes closed
# to their clients so that they generally can be used only when needed.
# Notably, the grammar of <guu> is quite different.   Second,
# I introduced some new flavors of right marker.  All can be realized with <gu>,
# but if one knows the right flavor one can close the right structure with a single
# right closure.

# right markers of subordinate clauses (argument modifiers).
# <gui> closes a different class than in the trial.85 grammar, with
# similar but on the whole better results.

GUIZA <- ([ ]* (G UI) (Z a) )

GUIZI <- ([ ]* (G UI) (Z i) )

GUIZU <- ([ ]* (G UI) (Z u) )

GUI <- (!GUIZA !GUIZI !GUIZU ([ ]* (G UI) ))

# right markers of abstract predicates and descriptions.
# probably the forms with z are to be preferred (and the other
# two are not needed) but I preserve all five classes for now.

GUO <- ([ ]* (G UO) )
```

```
GUOA <- ([ ]* (G UOb a/G UO Z a) )

GUOE <- ([ ]* (G UOb e) )

GUOI <- ([ ]* (G UOb i/G UO Z i) )

GUOO <- ([ ]* (G UOb o) )

GUOU <- ([ ]* (G UOb u/G UO Z u) )

# right marker used to close term (argument/predicate modifier) lists.
# it is important to note that in our grammar GUU is not a component of
# the class termset, nor is it a null termset:  it appears in other classes
# which include termsets as an option to close them.  The effects are similar
# to those in the trial.85 grammar, but there is less of a danger that
# extra unexpected closures will be needed.

GUU <- ([ ]* (G UU) )

# a new closure for arguments in various contexts

GUUA <- ([ ]* (G UUb a) )

# a new closure for sentences.  In particular, it
# may have real use in closing up the scope of a list of
# fronted terms before a series of logically connected sentences.

GIUO <- ([ ]* (G IUb o) )

# right marker used to close arguments tightly linked with JE/JUE.

GUE <- ([ ]* (G UE) )

# a new closure for descpreds

GUEA <- ([ ]* (G UEb a) )


# used to build tightly linked term lists.

JE <- ([ ]* (J e) )

JUE <- ([ ]* (J UE) )

# used to build subordinate clauses (argument modifiers).
```

```
JIZA <- ([ ]* ((J IE)/(J AE)/(P e)/(J i)/(J a)/(N u J i)) (Z a) )

JIOZA <- ([ ]* ((J IO)/(J AO)) (Z a) )

JIZI <- ([ ]* ((J IE)/(J AE)/(P e)/(J i)/(J a)/(N u J i)) (Z i) )

JIOZI <- ([ ]* ((J IO)/(J AO)) (Z i) )

JIZU <- ([ ]* ((J IE)/(J AE)/(P e)/(J i)/(J a)/(N u J i)) (Z u) )

JIOZU <- ([ ]* ((J IO)/(J AO)) (Z u) )

JI <- (!JIZA !JIZI !JIZU ([ ]* ((J IE)/(J AE)/(P e)/(J i)/(J a)/(N u J i)) ))

JIO <- (!JIOZA !JIOZI !JIOZU ([ ]* ((J IO)/(J AO)) ))

# case tags, both numerical position tags and the optional semantic case tags.

DIO <- ([ ]* ((B EU)/(C AU)/(D IO)/(F OA)/(K AO)/(J UI)/(N EU)/(P OU)/(G OA)/(S AU)/(V EU)/(

# markers of indirect reference.  Originally these had the same grammar as case tags,
# but they are now different.

LAE <- ([ ]* ((L AE)/(L UE)) )

# <me> turns arguments into predicates, <meu> closes this construction.

ME <- ([ ]* ((M EA)/(M e)) )

MEU <- ([ ]* M EU )

# reflexive and conversion operators:  first the root forms, then those with
# optional numerical suffixes.

NUO <- ((N UO)/(F UO)/(J UO)/(N u)/(F u)/(J u))

NU <- [ ]* (((N u/N UO) !([ ]+ (NIO/RA)) (NIO/RA)?)/NUO)+ freemod?

# abstract predicate constructors (from sentences)

# I do *not* think
# that <poia> will really be confused with <po ia>, particularly
# since we do require an explicit pause before <ia> in the latter case,
# but I record this concern:  the forms with z might be preferable.
```

```
PO1 <- ([ ]* ((P o)/(P u)/(Z o)))

PO1A <- ([ ]* ((P OIb a)/(P UIb a)/(Z OIb a)/(P o Z a)/(P u Z a)/(Z o Z a)))

PO1E <- ([ ]* ((P OIb e)/(P UIb e)/(Z OIb e)))

PO1I <- ([ ]* ((P OIb i)/(P UIb i)/(Z OIb i)/(P o Z i)/(P u Z i)/(Z o Z i)))

PO1O <- ([ ]* ((P OIb o)/(P UIb o)/(Z OIb o)))

PO1U <- ([ ]* ((P OIb u)/(P UIb u)/(Z OIb u)/(P o Z u)/(P u Z u)/(Z o Z u)))

# abstract predicate constructor from simple predicates

POSHORT1 <- ([ ]* ((P OI)/(P UI)/(Z OI)))

# word forms associated with the above abstract predicate root forms

PO <- ([ ]* PO1 )

POA <- ([ ]* PO1A )

POE <- ([ ]* PO1E )

POI <- ([ ]* PO1E )

POO <- ([ ]* PO1O )

POU <- ([ ]* PO1U )

POSHORT <- ([ ]* POSHORT1 )

# register markers

DIE <- ([ ]* ((D IE)/(F IE)/(K AE)/(N UE)/(R IE)) )

# vocative forms:  I still have the words of social lubrication as
# vocative markers.

HOI <- ([ ]* ((H OI)/(L OI)/(L OA)/(S IA)/(S IE)/(S IU)) )

# the verbal scare quote.  The quantifier suffix indicates how many preceding words are affe
# this is an odd mechanism.

JO <- ([ ]* (NIO/RA/SA)? (J o) )
```

```
# markers for forming parenthetical utterances as free modifiers.

KIE <- ([ ]* (K IE) )

KIU <- ([ ]* (K IU) )

KIE2 <- [ ]* K IE comma2? [(]

KIU2 <- [ ]* [)] comma2? K IU

# marker for forming smilies.

SOI <- ([ ]* (S OI) )

# a grab bag of attitudinal words, including but not restricted to the VV forms.

UI0 <- (!predstart (!([Ii] juncture? [Ee]) VV juncture?/(B EA)/(B UO)/(C EA)/(C IA)/(C OA)/(

# negative forms of the attitudinals.  The ones with <no> before the two vowel forms are a p
# should also be (though they present no pronunciation problem) so that they are resolved as

NOUI <- (([ ]* UI0 NOI)/([ ]* N [o] juncture? comma? [ ]* UI0 ))

# all attitudinals (adding the discursives nefi, tofi... etc)
# there is a technical problem with mixing UI0 roots of VV and CVV shapes.

UI1 <- ([ ]* (UI0+/(NI F i)) )

# the inverse vocative marker

HUE <- ([ ]* (H UE))

# occurrences of <no> as a word rather than an affix.

NO1 <- ([ ]* !KOU1 !NOUI (N o) !(comma2? Z AO comma2? Predicate) !([ ]* KOU) !([ ]* (JIO/JI/

# a technical closure for the alternative parser approach:  the "large subject marker"

GAA <- (NO1 freemod?)* ([ ]* (G AA))


# Names, acronyms and PreNames from above.

AcronymicName <- Acronym &(comma/period/end)

DJAN <- (PreName/AcronymicName)
```

```
# predicate words which are phonetically cmapua

# "identity predicates".  Converses are provided as a new proposal.

BI <- ([ ]* (N u)? ((B IA)/(B IE)/(C IE)/(C IO)/(B IA)/(B [i])) )

# interrogative and pronoun predicates

LWPREDA <- ((H e)/(D UA)/(D UI)/(B UA)/(B UI))

# here I should reinstall the <zao> proposal.

# the predicate words defined above in the phonetics section

Predicate <- (CmapuaUnit comma2?  Z AO comma2?)* Complex (comma2? Z AO comma2? Predicate)?

# predicate words, other than the "identity predicates" of class [BI]
# these include the numerical predicates (NI RA), also cmapua phonetically.

# we are installing John Cowan's <zao> proposal here, experimentally, 4/15/2019

PREDA <- ([ ]* &caprule (Predicate/LWPREDA/(![ ] NI RA)) )

# Part 3:  The Grammar Proper

# right markers turned into classes.

guoa <- (PAUSE? (GUOA/GU) freemod?)

guoe <- (PAUSE? (GUOE/GU) freemod?)

guoi <- (PAUSE? (GUOI/GU) freemod?)

guoo <- (PAUSE? (GUOO/GU) freemod?)

guou <- (PAUSE? (GUOU/GU) freemod?)

guo <- (!guoa !guoe !guoi !guoo !guou (PAUSE? (GUO/GU) freemod?))

guiza <- (PAUSE? (GUIZA/GU) freemod?)

guizi <- (PAUSE? (GUIZI/GU) freemod?)

guizu <- (PAUSE? (GUIZU/GU) freemod?)
```

```
gui <- (PAUSE? (GUI/GU) freemod?)

gue <- (PAUSE? (GUE/GU) freemod?)

guea <- (PAUSE? (GUEA/GU) freemod?)

guu <- (PAUSE? (GUU/GU) freemod?)

guua <- (PAUSE? (GUUA/GU) freemod?)

giuo <- (PAUSE? (GIUO/GU) freemod?)

meu <- (PAUSE? (MEU/GU) freemod?)

geu <- GEU

# Here note the absence of pause/GU equivalence.

gap <- (PAUSE? GU freemod?)

# this is the vocative construction.  It can appear early because all of its components are

# the intention is to indicate who is being addressed.  This can be handled via a name, a de
# alien text name (the last must be quoted).  The complexities of these grammatical construc
# introduced.

# HOI0 <- [ ]* [Hh] [Oo] [Ii] juncture?

# restore words of social lubrication as vocative markers but not as name markers:  <loi, Dj

# I do not allow a freemod to intervene between a vocative marker and the associated
# utterance, to avoid unintended grabbing of subjects by the words of social lubrication whe
# as vocative markers.  This lets <Loi, Djan> and <Loi hoi Djan> be equivalent.  The comma r
# first because the social lubrication words are in this version not name markers.

HOI0 <- ([ ]* ((([Hh] OI)/([Ll] OI)/([Ll] OA)/([Ss] IA)/([Ss] IE)/([Ss] IU)))) juncture? !V1

voc <- (HOI0 comma2? name /(HOI comma2? descpred guea? namesuffix?)/(HOI comma2? argument1 g

# this is the inverse vocative.  It can appear early because all of its components are marke

# the intention is to indicate who is speaking.  The range of ways this can be handled is si
# handled for the vocative;  there is the further option of a sentence (the [statement] clas
# for the case where an argument is used (to avoid it inadvertantly expanding to a sentence)

HUE0 <- [ ]* &caprule [Hh] [Uu] juncture? [Ee] juncture? !V1
```

```
invvoc <- (HUE0 comma2? name/HUE freemod? descpred guea? namesuffix?/(HUE freemod? statement


# this is the class of free modifiers.  Most of its components are head marked (those that a
# and it is useful for it to appear early because these things appear everywhere in subseque
# of whatever sort, is a freely insertable gadget which modifies the immediately preceding c
# if it is initial.

# NOUI is a negated attitudinal word.  UI1 is an attitudinal word:  these express an emotion
# assertion (noting that EI marks questions (yes or no answer expected) and SEU marks uttera

# SOI creates smilies in a general sense:  <soi crano> indicates that the listener should in
# similarly for other predicates.

# DIE and NO DIE are register markers, communicating the social attitude of the speaker towa
# example is "dear"

# KIE...KIU constructs a full parenthetical utterance as a comment, which can be enclosed in
# the marker words.

# JO is a scare quote device.

# the comma is a freemod with no semantic content:  this is a device for discarding phonetic
# and the speaker's optional pauses alike.   The pause before a non-pause marked prename is
# is excluded.  Ellipses and dashes are fancy pauses supported as freemods.

freemod <- ((NOUI/(SOI freemod? descpred guea?)/DIE/(NO1 DIE)/(KIE comma? utterance0 comma?

# the classes juelink to linkargs describe very tightly bound arguments which can be firmly
# the context of metaphorical modifications and the use of predicates in descriptive argumen

# note that we allow predicate modifiers (prepositional phrases) to be bound with <je/jue> w
# allowed in 1989 Loglan, but which we believe is supported in Lojban.

juelink <- (JUE freemod? (term/(PA2 freemod? gap?)))

links1 <- (juelink (freemod? juelink)* gue?)

links <- ((links1/(KA freemod? links freemod? KI freemod? links1)) (freemod? A1 freemod? lin

jelink <- (JE freemod? (term/(PA2 freemod? gap?)))

linkargs1 <- (jelink freemod? (links/gue)?)

linkargs <- ((linkargs1/(KA freemod? linkargs freemod? KI freemod? linkargs1)) (freemod? A1
```

```
# class abstractpred supports the construction of event, property, and quantity predicates f
# closable with <guo> if introduced with <po,pu,zo> and closable with suffixed variants of <
# variants of <po,pu,zo> (a NEW idea but it is clear that closure of these predicates (and c
# used associated descriptions) is an important issue).

abstractpred <- ((POA freemod? uttAx guoa?)/(POA freemod? sentence guoa?)/(POE freemod? uttA

# predunit1 describes the truly atomic forms of predicate.

# PREDA is the class of predicate words (the phonetic predicate words along with the special
# above under the PREDA rule.  NU PREDA handles permutations and identifications of argument

# SUE contains the alien text constructions with <sao> and <sue>, semantically quite differe
# in the same way.

# <ge>...<geu/cue> (the closing optional) can parenthesize a fairly complex predicate phrase
# forms can have conversion or reflexive operators (NU) applied.  I should look into why the
# is different.  An important use of this is in metaphor constructions, but it has other pot

# abstractpred is the class of abstraction predicates just introduced above.  These are trea
# be noted that their privileges in the trial.85 grammar are (absurdly) limited.

# <me>...<meu> (the closing optional, but important to have available) forms predicates from
# objects to which the argument refers.  <Ti me le mrenu> :  this is one of the men we are t

predunit1 <- ((SUE/(NU freemod? GE freemod? despredE (freemod? geu comma?)?)/(NU freemod? PR

# <no> binds very tightly to predunit1:  a possibly multiply negated predunit1 (or an unador

predunit2 <- ((NO1 freemod?)* predunit1)

# an instance of NO2 is one not absorbed by a predunit.  Example:  <Da no kukra prano>  X is
# <Da no ga kukra prano>  (X is not a fast runner, and in fact may not run at all).

NO2 <- (!predunit2 NO1)

# a predunit3 is a predunit2 with tightly attached arguments.

predunit3 <- ((predunit2 freemod? linkargs)/predunit2)

# a predunit is a predunit3 or a predunit3 converted by the short-scope abstraction operator
# <poi/pui/zoi> to an abstraction predicate.  This is the kind of predicate which can appear
# a component in a serial name.

predunit <- ((POSHORT freemod?)? predunit3)
```

```
# a further "atomic" (because tightly packaged) form is a forethought connected pair
# of predicates (this being the full predicate class defined at the end of the process)
# possibly closed with <guu>, possibly multiply negated as well.

# the closure with guu eliminated the historic rule against kekked heads of metaphors.

kekpredunit <- ((NO1 freemod?)* KA freemod? predicate freemod? KI freemod? predicate guu?)

# there follows the construction of metaphorically modified predicates,
# along with tightly logically linked predicates.

# CI and simple juxtaposition of predicates both represent modification of the second
# predicate by the first.  We impose no semantic conditions on this modification,
# except in the case of modification by predicates logically linked with CA,
# which do distribute logically in the expected way both as modifiers and as modified.
# We do not regard <preda1 preda2> as necessarily implying preda2:  we do regard
# it as having the same place structure as preda2.  It is very often but not always
# a qualification or kind of preda2;  in any case it is a relation analogous to preda2.

# modification with CI binds most tightly.

# we eliminated the distinction between the series of sentence and description
# predicate preliminary classes:  there seems to be no need for it even in the
# trial.85 grammar.

despredA <- ((predunit/kekpredunit) (freemod? CI freemod? (predunit/kekpredunit))*)

# this is logical connection of predicates with the tightly binding CA
# series of logical connectives.  CUI can be used to expand the scope of
# a CA connective over a metaphor on the left.  <ge>...<geu> is used to expand
# scope on the right (and could also be used on the left, it should be noted).
# descpredC is an internal of despredB assisting the function of CUI.
# the !PREDA in front of CUI is probably not needed.

despredB <- ((!PREDA CUI freemod? despredC freemod? CA freemod? despredB)/despredA)

despredC <- (despredB (freemod? despredB)*)

# tight logical linkage of despredB's

despredD <- (despredB (freemod? CA freemod? despredB)*)

# chain of modifications of despredD's (grouping to the left)

despredE <- (despredD (freemod? despredD)*)
```

```
# the GO construction allows inverse modification:  <preda1 GO preda2> is <preda2 preda1> as
#  there are profound effects on grouping.

descpred <- ((despredE freemod? GO freemod? descpred)/despredE)

# this version which appears in sentence predicates as opposed to descriptions differs
# in allowing loosely linked arguments (termsets) instead of those linked with <je/jue> for
# moved to the end by GO.

# 4/17/2019 shared argument experiment

# sentpred <- (( (KA freemod? sentpred freemod? KI freemod? sentpred !guu)/ (despredE freemo

sentpred <- ((despredE freemod? GO freemod? barepred)/despredE)

# sentpred <- ((despredE freemod? GO freemod? barepred)/despredE) (A1 ((despredE freemod? GO

# the construction of predicate modifiers (prepositional phrases usable as terms along with

mod1a <- (PA3 freemod? argument1 guua?)

# note special treatment of predicate modifiers without actual arguments.
# the !barepred serves to distinguish these predicate modifiers from actual
# "tenses" (predicate markers).

mod1 <- ((PA3 freemod? argument1 guua?)/(PA2 freemod? !barepred gap?))

# forethought connection of modifiers.  There is some subtlety in
# how this is handled.

kekmod <- ((NO1 freemod?)* (KA freemod? modifier freemod? KI freemod? mod))

mod <- (mod1/((NO1 freemod?)* mod1)/kekmod)

# afterthought connection of modifiers

modifier <- (mod (A1 freemod? mod)*)

# the serial name is a horrid heterogenous construction!  It can involve
# components of all three of the major phonetic classes essentially!

# However, I believe I have the definition right, with all the components
# correctly guarded :-)

name <- (PreName/AcronymicName) (comma2? !FalseMarked PreName/comma2? &([Cc] [Ii]) NameWord/
```

```
LA0 <- [ ]* [Ll] [Aa] juncture?

LANAME <- (LA0 comma2? name)

# general constructions of arguments with "articles".

# the rules here have the "possessive" construction as in <lemi hasfa; le la Djan, hasfa> em
# construction in 1989 Loglan, though speakers might think they are.  Here they are indeed t
# be "indefinite" (cannot start with a quantifier word);  the possessor can be followed by a
# <le la Djan, na hasfa>, "John's present house", by analogy with <lemina hasfa>, which is a
# LIP accepts <lemina> as a word).

# there are other subtleties to be reviewed.

#descriptn <- (!LANAME ((LAU wordset1)/(LOU wordset2)/(LE freemod? ((!mex arg1a freemod?)? (

descriptn <- (!LANAME ((LAU wordset1)/(LOU wordset2)/(LE freemod? ((!mex arg1a freemod?)? (P


# abstract descriptions.  Note that abstract descriptions are closed with <guo> entirely ind
# <le po preda guo> does not have a grammatical component <po preda guo>.  This avoids the c
# in Lojban.

abstractn <- ((LEFORPO freemod? POA freemod? uttAx guoa?)/(LEFORPO freemod? POA freemod? sen

# a wider class of basic argument constructions.  Notice that LANAME is always read by prefe

namesuffix <- (&(comma2 !FalseMarked PreName/[ ]* [Cc][Ii] juncture? comma2? (PreName/Acrony

arg1 <- (abstractn/(LIO freemod? descpred guea?)/(LIO freemod? argument1 guua?)/(LIO freemod

# this adds pronouns (incl. the fancy <gao> letterals) and the option of left marking an arg

arg1a <- ((DA/TAI/arg1/(GE freemod? arg1a)) freemod?)

# argument modifiers (subordinate clauses)

argmod1 <- ((([ ]* (N o) [ ]*)? ((JI freemod? predicate)/(JIO freemod? sentence)/(JIO freemo

# we improved the trial.85 grammar by closing not argmod1 but argmod with <gui>.  But the la
# when building an argmod1 have the argmod1 construction closed with the corresponding label
# gui and guiza actually have different grammar.

# trial.85 did not provide forethought connected argument modifiers, and we also see no need
# though they could readily be added.
```

```
argmod <- (argmod1 (A1 freemod? argmod1)* gui?)

# affix argument modifiers to a definite argument

arg2 <- (arg1a freemod? argmod*)

# build a possibly indefinite argument from an argument:  to le mrenu

arg3 <- (arg2/(mex freemod? arg2))

# build an indefinite argument from a predicate

indef1 <- (mex freemod? descpred)

# affix an argument modifier to an indefinite argument

indef2 <- (indef1 guua? argmod*)

indefinite <- indef2

# link arguments with the fusion connective <ze>

arg4 <- ((arg3/indefinite) (ZE2 freemod? (arg3/indefinite))*)

# forethought connection of arguments.  Note use of argx

arg5 <- (arg4/(KA freemod? argument1 freemod? KI freemod? argx))

# arguments with possible negations followed by possible indirect reference constructions.

argx <- ((NO1 freemod?)* (LAE freemod?)* arg5)

# afterthought connection with the tightly binding ACI connectives

arg7 <- (argx freemod? (ACI freemod? argx)?)

# afterthought connection with the usual A connectives.  Can't start with GE
# to avoid an ambiguity (to which 1989 Loglan is vulnerable) involving AGE connectives.

arg8 <- (!GE (arg7 freemod? (A1 freemod? arg7)*))

# afterthought connection (now right grouping, instead of the left grouping above)
# using the AGE connectives.  GUU can be used to affix an argument modifier at this top leve

argument1 <- (((arg8 freemod? AGE freemod? argument1)/arg8) (GUU freemod? argmod)*)
```

```
# possibly negated and case tagged arguments.  We (unlike 1989 Loglan) are careful
# to use argument only where case tags are appropriate.

argument <- ((NO1 freemod?)* (DIO freemod?)* argument1)




# an argument which is actually case tagged.

argxx <- (&((NO1 freemod?)* DIO) argument)

# arguments and predicate modifiers actually associated with predicates.

term <- (argument/modifier)

# a term list consisting entirely of modifiers.

modifiers <- (modifier (freemod? modifier)*)

# a term list consisting entirely of modifiers and tagged arguments.

modifiersx <- ((modifier/argxx) (freemod? (modifier/argxx))*)

# the subject class is a list of terms (arguments and predicate modifiers) in which all but
# of the arguments are tagged, and there is at least one argument, tagged or otherwise.

subject <- ((modifiers freemod?)? ((argxx subject)/(argument (modifiersx freemod?)?)))

# this case is identified as an aid to experimental termination of argument lists

statement1 <- (subject freemod? (GIO freemod? terms1)? predicate)

# these classes are exactly argument, but are used to signal
# which argument position after the predicate an argument occupies.
# I think the grammar is set up so that these will actually
# never be case tagged, though the grammar does not expressly forbid it.

# I am trying a simple version of the "alternative parser" approach:
# a term list will refuse to digest an argument which starts a new
# SVO sentence (statement1).

argumentA <- !statement1 argument
```

```
# argumentA <- argument

argumentB <- !statement1 argument

#  argumentB <- argument

argumentC <- !statement1 argument

#  argumentC <- argument

argumentD <- !statement1 argument

#  argumentD <- argument

# for argument lists not guarded against absorbing a following subject

argumentA1 <- argument

argumentB1 <- argument

argumentC1 <- argument

argumentD1 <- argument

# a general term list.  It cannot contain more than four untagged arguments (they will be la
# with the lettered subclasses given above).

terms <- ((modifiersx? argumentA (freemod? modifiersx)? argumentB? (freemod? modifiersx)? ar

# terms list not guarded against absorbing a following subject

terms1 <- ((modifiersx? argumentA1 (freemod? modifiersx)? argumentB1? (freemod? modifiersx)?

# innards of ordered and unordered list constructions.  These are something I totally rebuil
# unsatisfactory state in trial.85.  Note the use of comma words to separate items in lists.

word <- (arg1a/indef2)

words1 <- (word (ZEIA word)*)

words2 <- (word (ZEIO word)*)

wordset1 <- (words1? LUA)

wordset2 <- (words2? LUO)
```

```
# the full term set type to be affixed to predicates.

# forethought connection of term lists

termset1 <- (terms/(KA freemod? termset2 freemod? guu? KI freemod? termset1))

# afterthought connection of term lists.  There are cunning things going on here getting <gu
# to work correctly.  Note that <guu> is NOT a null term list as it was in trial.85.

termset2 <- (termset1 (guu &A1)? (A1 freemod? termset1 (guu &A1)?)*)

# there is an interesting option here of a list of terms followed by <go> followed by a prec
# intended to metaphorically modify the predicate to which the terms are affixed.  Is there
# why we cannot have a more complex construction in place of terms?

termset <- ((terms freemod? GO freemod? barepred)/termset2)

# this is the untensed predicate with arguments attached.  Here is the principal locus
# of closure with <guu>, but it is deceptive to say that <guu> merely closes barepred,
# as we have seen above, for example in [termset2].

# modified for 4/17/2019 shared argument experiment

barepred <- (sentpred freemod? ((termset guu?)/(guu (&termset)))?)

# barepred <- (sentpred freemod? ((termset guu?)/(guu (&termset/&A1)))?)

# tensed predicates

markpred <- (PA1 freemod? barepred)

# there follows an area in which my grammar looks different from trial.85.  Distinct paralle
# marked and unmarked predicates are demonstrably not needed even in trial.85.  The behavior
# connectives is plain weird in trial.85; here we treat ACI connectives in the same way as A
# binding more tightly.

# units for the ACI construction following -- possibly multiply negated bare or marked predi

# adding shared termsets to logically connected predicates are handled differently here than
# which uses a very elegant but dreadfully left-grouping rule which a PEG cannot handle.  An
# should be manageable.

backpred1 <- ((NO2 freemod?)* (barepred/markpred))

# ACI connected predicates.  Shared termsets are added.  Notice how we first group backpred1
```

```
# group backpreds.

backpred <- (((backpred1 (ACI freemod? backpred1)+ freemod? ((termset guu?)/(guu &termset))?

# A connected predicates; same comments as just above.  Cannot start with GE to fix ambiguit

predicate2 <- (!GE (((backpred (A1 !GE freemod? backpred)+ freemod? ((termset guu?)/(guu &te

# predicate2's linked with right grouping AGE connectives (A and ACI are left grouping).

predicate1 <- ((predicate2 AGE freemod? predicate1)/predicate2)

# identity predicates from above, possibly negated

identpred <- ((NO1 freemod?)* (BI freemod? argument1 guu?))

# predicates in general.  Note that identity predicates cannot be logically connected
# except by using forethought connection (see above).

predicate <- (predicate1/identpred)


# The gasent is a basic form of the Loglan sentence in which the predicate leads.
# The basic structure is <PA word (usually a tense) or <ga>) followed optionally by terms fo
# <ga> followed by terms.  The list of terms after <ga> (if present) will either contain
# at least one argument and no more than one untagged argument
# (a subject) [gasent1] or all the arguments of the predicate [gasent2].  We deprecate other
# 1989 Loglan because they would cause unexpected reorientation of the arguments already giv
# and further arguments were read after <ga>.  [barepred] is an untensed predicate possibly
# is "simply a verb", i.e., a predicate without arguments.

# there is a semantic change from 1989 Loglan reflected in a grammar change here:
# in [gasent1] the final (ga subject) is optional.  When it does not appear, the resulting
# sentence is an observative (a sentence with subject omitted), not an imperative.
# Imperatives for us are unmarked.

# In the alternative version, the use of the large subject marker GAA can prevent inadvertar

# 4/22 allowing general predicates in gasent.  Otherwise the spaces of observatives and impe

#gasent1 <- ((NO1 freemod?)* (GAA? freemod? PA1 freemod? barepred (GA2 freemod? subject)?))

gasent1 <- ((NO1 freemod?)* (GAA? freemod? &markpred predicate (GA2 freemod? subject)?))

gasent2 <- ((NO1 freemod?)* (GAA? PA1 freemod? sentpred modifiers? (GA2 freemod? subject fre
```

```
gasent <- (gasent2/gasent1)

# this is the simple Loglan sentence in various basic orders.  The form "gasent" is discuss
# Predicate modifiers
# can be prefixed to the gasent.  The final form given here is the basic SVO sentence.  The
#(arguments and predicate modifiers) containing at most one un-case-tagged argument.  The mo
#by <gio> followed by a list of terms (1989 Loglan allowed more than one untagged argument b
#in which preceding constructions with errors in them may supply extra unintended arguments.
#a single argument before the predicate, followed by the predicate, which may itself contain
#(even multiple times).

# re <gio> and some other changes, in his comments on the NB3 grammar  JCB often notes restr
# intends but which he thought were hard to implement in the machine grammar.  The appearanc
# in an SVO sentence was one of these (though later he takes it as a virtue that the actual
# consider it a virtue to have unmarked SOV after observing unintended parses appearing in t
# (which would not have been hard for JCB to implement, in fact) is our restriction of the f
# His comments make it clear that he does not want arguments among those terms.

statement <- (gasent/(modifiers freemod? gasent)/(subject freemod? GAA? freemod? (GIO freemo

# this is a forethought connected basic sentence.  It is odd (and actual odd results can be
# of these rules is of the very general class uttA1, which includes some quite fragmentary u

# 12/20/2017 I rewrote the rule in a more compact form.  This rule looks ahead to the class
# for the moment notice that [sentence] will include [statement].

# 4/14 tentatively allowing initial modifiers here and leaving this out of uttA0 which repla
# The intention is to eliminate weird sentence fragments.

keksent <- modifiers? freemod? (NO1 freemod?)* (KA freemod? headterms? freemod? sentence fre

# sentence negation.  We allow this to be set off from the main sentence with a mere pause,
# it does not differ in meaning from the result of negating the first argument or predicate

neghead <- ((NO1 freemod? gap)/(NO2 PAUSE))

# this class includes [statement], predicate modifiers preceding a predicate (which may cont
# a predicate, and a keksent.  Of these, the first and third are imperatives.

# in the alternative version, the large subject marker GAA can prevent inadvertant absorptio

# 4/23/2019 added actual rule for imperative sentences.  This should not
# affect the parse in any essential way.

imperative <- ((modifiers freemod?)? GAA? !gasent predicate)
```

```
sen1 <- (neghead freemod?)* (imperative/statement/keksent)

# sen1 <- ((neghead freemod?)* ((modifiers freemod? GAA? !gasent predicate)/statement/GAA? p

# the class [sentence] consists of sen1's afterthought connected with A connectives

sentence <- (sen1 (ICA freemod? sen1)*)

# [headterms] is a list of terms (arguments and predicate modifiers) ending in <gi>.  Preced
# causes all predicates in the [sen1] to share these arguments.  We propose either that the
# appended to the argument list of each component of the [sen1], or that there is an argumen
# of the headterms list, and the list is inserted at the appropriate position in each compon
# the condition described in Loglan I, which presupposes that we always know what the last a

headterms <- (terms GI)+

# this is the previous class prefixed with a list of fronted terms.
# we think the <giuo> closure might prove useful.

uttAx <- (headterms freemod? sentence giuo?)

# weird answer fragments

uttA <- ((A1/mex) freemod?)

# a broad class of utterances, including various things one would usually only say as answer
# that this utterance class can take terminal punctuation.

uttA0 <- sen1/uttAx

uttA1 <- ((sen1/uttAx/links/linkargs/argmod/(modifiers freemod? keksent)/terms/uttA/NO1) fre

# possibly negated utterances of the previous class.

uttC <- ((neghead freemod? uttC)/uttA1)

# utterances linked with more tightly binding ICI sentence connectives.  Single sentences ar
# if not linked with ICI or ICA.

uttD <- ((sentence period? !ICI !ICA)/(uttC (ICI freemod? uttD)*))

# utterances of the previous class linked with ICA.  I went to some trouble to ensure that a
# [sentence] is actually parsed as a sentence, not a composite uttD, which was a deficiency
# LIP and of the trial.85 grammar.

uttE <- (uttD (ICA freemod? uttD)*)
```

```
# utterances of the previous class linked with I sentence connectives.

uttF <- (uttE (I freemod? uttE)*)

# the utterance class for use in the context of parenthetical freemods or quotations, in whi

utterance0 <- (!GE ((!PAUSE freemod period? utterance0)/(!PAUSE freemod period?)/(uttF IGE u

# Notice that there are two passes here:  the parser first checks that the entire utterance
# is phonetically valid, then returns and checks for grammatical validity.

# the full utterance class.  This goes to end of text, and incorporates the phonetics check.
# in which a freemod is initial.   The IGE connectives bind even more loosely than the I con
# left grouping.

utterance <- &(phoneticutterance !.) (!GE ((!PAUSE freemod period? utterance)/(!PAUSE freemo
```