Notebook 4: A complete account of the Loglan language as curated by Randall Holmes and others starting in 2013 (draft in progress, see version notes)

Randall Holmes (and maybe others)

May 20, 2023

Contents

0.1 Version Notes

5/10/2023: Starting.

5/20/2023: Added comments about the plan of the work to the introduction and embedded the current version of the PEG.

1 Introduction

I am sitting down in 2023, ten years after I (and others) started working on the project of wholesale revision of the Loglan language as it was handed down to us. I am setting out to write a self-contained account of where the language is and both say and show why it is interesting to study and perhaps use this intellectual construction.

Loglan is an artificial language, originally conceived by James Cooke Brown in 1955. It was designed for a specific purpose, which to my knowledge has never been carried out, and which is not really part of my aims for the language (if I have any beyond contemplation). The intention was to support an experimental test of the Sapir-Whorf hypothesis, that the language that a human being speaks constrains the way that they think.

The way that this purpose drove the design of the language is described by Brown (for example in the 1975 version of Loglan I) and in fact makes good sense. In order to be useful for an experimental test of the Sapir-Whorf hypothesis, the language needed to be small and easily learned (so that experimental subjects could learn it and try thinking in it) and extremely strange in some respect which could be engineered by the language designers (so that a Whorfian effect could be expected if there are such effects).

Brown claimed to have experimental evidence that the language was easily learned. I cannot evaluate this after the lapse of time and death of witnesses to these experiments. I can report that I have rather slowly learned the language: some aspects of it I think are indeed fairly easy to learn, and others are deep and complicated.

The way in which Brown chose to make it strange was to make it extremely logical. This again made sense, as progress in mathematical logic and computer programming meant that there was a good understanding of what a logical language might be like. The language now exists, and I have some rather dry remarks about the exact senses in which it is logical (though it should be clear because I am writing this and directing the project that I think the project has value and interest).

First of all, there is a pun inherent in the claim that Loglan is logical. It is logical in three different ways which are not necessarily related to each other.

It is to some extent designed to implement the machinery of first order predicate logic in a spoken language. As we will detail in one of our essays, it does not do this in any way that a logician would have chosen to do, but it does do it to a considerable extent. We will note briefly here, as an example (and a source of difficulty here and there) the structure of the Loglan sentence. An atomic sentence has the form $P(x_1, \ldots, x_n)$ in logic, where the P is the predicate (the verb, in this context) and the x_i's are the arguments (noun phrases playing the grammatical roles of subject and objects). In Loglan grammar, the default grammatical structure for such a sentence is SVO (a very common natural language approach), and the parse is $(x_1(P(x_2, \ldots, x_n)))$. This gives the first argument (the subject) a very special role (implicit in all Loglan work, more explicitly visible in our grammar) and further has the weird effect that the predicate (the verb) has the second and subsequent arguments (the objects) bound to it more tightly than the subject is bound to it. In logical notation, the roles of the arguments are exactly coordinate; not so in Loglan. This is an example of the way Loglan differs from logical notation; it isn't a criticism of the language as such. It is also an example of something so deeply embedded in the design of the language as it has come down to us that it would be hard to change it without ripping everything up and starting afresh; as I will say many times, my role is to curate an existing language, not to redesign it to make a better one.

The language is unambiguously parsable by computer. This is a characteristic of logical notations, but has nothing essential to do with the first sense in which Loglan is logical. It is, further, unambiguously parsable by computer (and one hopes by the human ear and brain) in two quite different senses. Its phonetics and lexicography are unambiguous: in principle samples of the the written and spoken forms of Loglan, if resolved successfully into phonemes and pauses, should be unambiguously resolvable into "words" in a general sense (with information about the lexical classes of these "words"). Its grammar is unambiguous: a written or spoken sample of Loglan should parse in just one way.

Loglan as it was handed down to us came with a parser (LIP) which would parse sentences using a BNF grammar which was provably unambiguous. The software used cues in words to classify them phonetically, but it did not implement the official phonetic structure of the language fully and moreover attempts to test the phonetics carefully revealed bugs. The sense in which the BNF grammar was unambiguous was qualified. A method of proving an LALR(1) grammar unambiguous was used. But Loglan grammar is not LALR(1): a preprocessing step was needed. Both the failure to fully implement the phonetics and the use of the preprocessor introduced ambiguity into the language (and it was unsatisfactory that strenuous tests of some language features led to parser crashes).

Loglan was actively developed from 1975 to 1983 (partly supported by a government grant). This grant was not renewed and the project fell on hard times. It was also decided to completely remodel the phonetics of the 1975 version, which occasioned considerable work. In 1987, Notebook 3, a full account of all aspects of the language, was released, and in 1989 a new version of the book Loglan I was released, giving a baseline for "the language as handed down to us" (along with computer based dictionaries: no new paper dictionary was released along with the 1989 Loglan 1). Sometime between 1983 and the release of Notebook 3, there was a schism in the project, and a separate, closely related but not mutually intelligible language came into being, which is called Lojban. I have no comments to make about the language split; I may occasionally have things to say about Lojban as a related language with related problems and opportunities. The Loglan Institute is currently on good terms with the Logical Language Group.

The sources on which my work (with some collaboration from others) is based are Loglan 1,4,5, the Loglan book and dictionaries released in 1975, Notebook 3, the complete language description released in 1987, the trial.85 formal grammar on which LIP was built, the 1989 version of Loglan I, and the computer dictionaries (online versions of Loglan 4 and 5). I have all issues of the Loglanist periodical which was produced at intervals from the late 1970s until 1983, and I have most issues of the periodical Lognet from the 80s and 90s, and Appendix H, which details changes to the Loglan described in 1989 which were made officially by the Loglan Academy of that time. I also have Alex Leith's novella *A Visit to Loglandia* and some other sample Loglan text.

I do not have a lot of things which a historian of the language would like to have. I have very little documentary evidence of what went on from 1983 to 1989. I would like to gather such documentation if anyone has it. As I said above, I am not interested in the politics of the language split. I am deeply interested in any documentation of the process of language building which still exists. My interest in this is historical rather than linguistic: I view Notebook 3 as the base language description from which the versions of 1989 and 2013+ spring, with the 1975 releases and the Loglanist volumes as partial history of the background.

James Cooke Brown died in 2000 and Alex Leith, who succeeded him as leader of the Loglan Institute, died shortly afterward, to be succeeded by Robert McIvor, the main architect of the Loglan formal grammar. In 2008, McIvor tapped me as “CEO” of the Loglan Institute (Brown’s choice of title, not mine).

I had been the house logician in the 1990’s but resigned (without acrimony) when Brown declined advice of mine about the role of sets in the language. I met Brown in person once and I corresponded with McIvor on many occasions. Unfortunately, a lot of relevant email disappeared in the collapse of Netscape (along with much of my early professional email).

In 2008 I became leader of the language (I certainly don’t regard myself as owner: I don’t think a language can be owned), without really being able to speak it. I did have a good understanding of the outlines of the formal grammar and implementation of logic, hampered by the unsatisfactory opacity of LIP. The old parser had unreadable internals (except for the BNF grammar) and its output was unreadable (human beings cannot make good use of parses which require matching many levels of nested parentheses).

From 2008 to the present I have met almost every week with a few people in the virtual world Second Life to talk in and about Loglan. In this way, I have acquired some ability to speak the language, mostly in text rather than voice.

In 2013, I heard a rumor that someone had used PEG (Parsing Expression Grammar) machinery to parse Lojban from the level of letters up. I decided to do this for Loglan. This required me to develop what I think was the first formal parser for TLI Loglan phonetics (I think something at least approximating this had already been done in Lojban), which at the same time incorporated the grammar of Loglan as expressed in trial.85 (wth some amendments).

This gave us a language definition which was for the first time fully accessible and testable at all levels.

This enabled me to address some major problems on the periphery of the language definition which were already recognized as problems in the 1990’s. Notable instances are the problem of serial names and the problem of acronyms. The original Loglan solution for strong quotation was not implementable in BNF or PEG; I installed a different solution (which owes something in its details to an earlier analysis of Linnaean names in the 90s by other workers). There was a systematic problem with tense-suffixed logical connectives (APA words) which required substantial work. A full solution to the problem of the left boundaries of names was installed, which could only be tested because the new grammar software handles phonetic, lexical, and grammatical levels.

The main testbed for the grammar software has been parsing the entire text of Alex Leith's novel (twice), revising it as necessary for the new parser to work. There are many changes in the text, mostly minor, and the whole process gives evidence for the claim that 2013 Loglan, though different, would be intelligible to a speaker of 1989 Loglan if there were such a being. And that is my aim: I am engaged in making an existing language work, not in redesigning it to be a better one. 2013 Loglan has some profound differences in deep structure from 1989 Loglan, but the intention is to keep existing text stable, and the differences would only come into play with much more complex and extensive writing in the language.

The plan of this work is to have three parts: this Introduction (hopefully succint), Part I, a collection of essays on aspects of the language on phonetic, lexicographic, and grammatical levels, and Part II, a point by point commentary on the PEG grammar (yes, I know this is a pleonasm). The plan for wriring it is to write Part II and allow it to motive the development of the essays in Part I. I already have a fairly clean account of the phonetics of the language, which I may write anew but which will surely closely resemble what appears in the existing reference grammar. I want to write a similar top down human accessible acount of the grammar: the current one in the existing reference grammar is more like a point by point comment on points in the PEG.

2 Part I: Essays

3 Part II: Commentary on the PEG

The text of the PEG follows, to be interspersed with formal commentary for this document. The original text contains comments, which we may edit or preserve as we work.

```
<H1>The Loglan PEG Grammar</H1>

<LI> This document and other related Loglan materials are
   intellectual property of the Loglan Institute, a Florida
   corporation which still exists, but the Institute freely permits
    and enourages the use of this and other Loglan materials for
   noncommercial purposes.

<p>

<TT>

<LI>In this file I will develop the entire Loglan grammar on top of
    the phonetic proposal

<H2> PEG notation </H2>

<LI>A PEG (Parsing Expression Grammar) is made up of lines of the
   form

<TT> class_name <- PEG notation</TT>

<LI>Each PEG notation describes a set of strings with conditions on
    the context in which they occur.<p>

<LI>Concrete strings: <TT>'string'</TT> or <TT>"string"</TT>
   literally denotes the 6 character string given.<p>

<LI>Classes of characters: <TT>[aeiou]</TT> describes the set of
   one character strings which are either

<LI>a, e, i, o, or u. Ranges can appear: <TT>[a-zA-z]</TT>
   describes the union of the sets of lower case letters and upper
   case letters, considered as one character strings.<p>
```

```
<LI>If <TT>A</TT> and <TT>B</TT> are PEG notations, <TT>(A B)</TT>
   denotes a string of class <TT>A</TT> followed by a string of
   class <TT>B</TT> (in which the string of class <TT>A</TT> is the
    preferred string of this class read from the beginning of the
   source string).<p>

<LI>If <TT>A</TT> and <TT>B</TT> are PEG notations, <TT>(A / B)</TT
   > denotes a string of either class <TT>A</TT> or a string of
   class <TT>B</TT>, with a string of class <TT>A</TT> being read
   by preference if possible. The fact that a preference is
   indicated in alternative lists makes PEG reading deterministic (
   in a sense, there are no ambiguities for a PEG grammar). The
   problem in a PEG corresponding to ambiguity in a BNF grammar is
   incorrectly ordered lists of alternatives.<p>

<LI>If <TT>A</TT> is a PEG notation, <TT>(A)?</TT> represents a
   string of class <TT>A</TT> (preferred) or an empty string if
   there is no string of class <TT>A</TT>: this represents optional
    appearance of <TT>A</TT>. <TT>(A)*</TT> represents zero or more
    consecutive strings of class <TT>A</TT> (as many as possible)
   and <TT>(A)+</TT> represents one or more consecutive such
   strings.<p>

<LI>If <TT>A</TT> is a PEG notation, <TT>&(A)</TT> represents a
   length 0 string which is followed by a string of class <TT>A</TT
   >, and <TT>!(A)</TT> represents a length 0 string which is <B>
   not</B> followed by a string of class <TT>A</TT>. This gives us
   powerful lookahead features: for example, <TT>((A)! B</TT>
   represents a string of class <TT>B</TT> whose beginning is not
   also the beginning of a string of class

<LI><TT>A</TT>: it is tempting but not accurate to say that it does
    not have an initial segment of class <TT>A</TT>, because
   detection of a string of class <TT>A</TT> longer than the string
    of class <TT>B</TT> read would cause reading of this class to
   fail.<p>

<LI>The period . represents the class of single characters (so !.
   is end of text).<p>
```

```
<LI>New notations are introduced by lines <p>

<LI><TT> class_name <- PEG notation</TT>:<p>

<LI>this is not just an abbreviation facility because such
   definitions may be mutually recursive.<p>

<LI>A PEG notation applied to a source string will give either
   failure or a uniquely determined initial string of the source (
   parsed suitably); in a sense PEG is unambiguous. What
   corresponds as an issue to ambiguity for a BNF grammar is

<LI>inappropriate choice of order of alternatives in PEG
   disjunctions <TT>(A / B)</TT>: what often represents a problem
   with a grammar is what I call <LI><LI><LI> <LI>"preemption",
   where an earlier alternative reads an initial segment of a
   string where a later alternative could have read more of it.<p>

<LI>It's possible to have a PEG go into an infinite loop and fail
   to produce a parse. My PEG generator has a termination checker,
   so the Loglan grammar does not have these problems. I have
   contemplated writing a preemption checker, but this is a rather
   difficult problem.<p>

<p>

<H2>Dated updates now to appear here</H2>

<p>

<LI> 2/22/2022 Eliminated all use of ICI and ICA as binary
   connectives between non-sentences. Non-sentence utterances
   starting with ICA connectives remain and seem to be adequate for
    all actual uses along these lines.

<LI> 2/20/2022 In addition, pulled the tightly binding ICI
   connectives into sentence instead of utterance structure. I
   doubt there is any practice in using them in their former role,
   so they will be experienced as a new feature.
```

<LI> 2/19/2022 Major restructuring of sentences. Sentences with head terms with gi and goi (uttAx) are moved into class sen1, so they can be afterthought logically

<LI>connected like any other sentences. Note that the fact this couldnt be done means that Loglan never really had OSV sentences in any practical sense. The real reason I need this <LI>is convenient isomorphism between Loglan and standard logical notation. This means that class uttA0 is no longer needed (removed in its one application in favor of sen1).

<LI> experimentally allowing terms before keksents, which are in effect headterms with [gio].

<LI> 2/15/2022 first public release with systematic renaming.

<LI> work order 2/10/2022 no action taken yet. To go with [pau], ago, add [fau] from now and [vau] distance away to class PA00. [cioru] is rejected for cause.

<LI> 2/9/2022 added the [bao] lambda quantifier for constructing abstract relations between more than two objects

<LI> 2/6/2022 removed kue from NI0 and added it (allowing digit suffixes) to NU0.

<p>

<p>

<LI> 2/5/2022 fixed problem with CI as name marker created in literal character cleanup; allowed [pi] to be prefixed to anything of class NI

<p>

<LI> 2/2/2022 Large project: renaming of classes in the grammar. Identify ones which ought to be known to speakers (targets for having associated Loglan predicates). Establish standard suffixes for phonemes, morphemes, lexemes. About done, 2/12/2022.

<p>

<LI> 2/2/2022 no modification yet. Small project: create class NUJI with nuji and nuja, and correct grammar so NUJI takes only arguments. At present, nuja is not supported and nuji can take predicates and modifiers. Later: class NUJI installed and properly restricted: nujiza and the like are not for the moment supported.

<p>

<LI> 1/31/2022 removes a lot of use of literal characters by defining rules sp and stress2. Corrects some minor errors in literal character lists.

<p>

<LI>a note, not reflecting a modification. I'm wondering whether the pause required in [fo tonira] ([fotonira] means something quite

<LI>different) should be a mandatory comma pause. It looks as if it might not be hard to implement.

<LI>

<p> No modification as yet: I am thinking of banning JIO followed by an imperative. JI should be used.

<p>

<LI> fixed a bug which broke the end of speech marker "#"

#p$

<LI> 1/29/2022 Created wrapper classes so that parses which mix letters and grammar classes are avoided.

<p>

```
<LI> 1/29/2022 Also fixed a small bug in class headterms. Fixed an
   apparent bug in class NameWord ([hue] was omitted) which has
   never caused any identifiable trouble.

<p>

<LI> 1/26/2022 complete elimination of the alternative parser and
   all constituent rules (including the word [gaa]) as I have
   abandoned this idea.

<LI>a note: there is a problem with interaction of quoted forms
   with alien text operators.

<p>

<LI> 1/23/2022 made imperative important and cloned sentence and
   uttAx in versions which will not be marked as imperative if they
    lack subjects. Considering simply forbidding jio clauses to
   contain no subject sentences, but this is not implemented (ji
   should be used). Fixing the restriction on <ci> as a name marker
    to allow it to be followed by whitespace and a name.

<p>

<LI>1/22/2022: provisionally removing the requirement that the
   terms before the predicate in an SVO sentence contain no more
   than one untagged argument.

<LI>The option of using the particle [gio] before any sutori
   untagged arguments before the predicate remains.<p>

<p>

<LI>1/21/2022 Starting a literate programming exercise: turn this
   document into HTML while preserving its performance as a PEG
   grammar.

#Also note that the alternative version is now turned off. The only
    component present is [gaa] and I do not see a reason for anyone
```

to use it.

#The alternative parser is readily turned back on by changing the line statement1x. This version labels the default stressed syllable in a predicate in the PhoneticComplex parse.

<p>

<LI>a serious problem with ICA, an actual ambiguity which has existed since the beginning of the language,

<LI>hopefully fixed: the fix is that an apparent ICA initial utterance which could without the period be

<LI>a continuation of a sentence is read as such. The important point is that there is no audible difference

<LI>between comma followed by ICA and period followed by ICA: we solve the problem by reading the latter

<LI>as the former where possible.

<p>

<LI>11/24/2021 KIA, the one "word" deletion operator, is installed. What it actually does is a bit subtle.

<p>

<LI>2/4/2021 Imposed the rule that two final consonants cannot be consonants from voiced/unvoiced pairs

<LI>with different voice. Also forbid second final consonant to be h.

<p>

<LI>I have further fine-tuning of djifoa gluing in mind.

<LI>Allow the -r glue to be expressed as

```
<LI>-rr after all mandatory monosyllables, removing the annoying
   pronunciation problem?

<LI>I was thinking of allowing -hy gluing in other contexts, but it
    is actually a bad idea.

<p>

<LI>9/15/2019 installed semantic case tags with order distinctions
   for use with predicates with more than one argument of the same
   case.

<LI>one solution is beucine, beucito... another is beuzi, beuza,
   beuzu.

<p>

<LI>4/28/2019 Various debugging of the new predicate algorithm.
   Added CVVhy as a glued form for CVV djifoa.

<LI>added capitalization of djifoa glue! Confirming my apparent
   earlier decision that a CVV(h)y djifoa must be followed

<LI>by a full predicate complex.

<p>

<LI>4/26/2019: this incorporates various revisions to the phonetics
    , correcting errors or clarifying rules,

<LI>motivated by my development of the phonetics section of a new
   grammar document. The one notable

<LI>change is that [ci] is now only a name marker if followed by an
    explicit pause. This only requires

<LI>changes in writing in serial names. In speech, it is
   recommended that one not pause after [ci]
```

<LI>except before a name word. The benefit is that non-serial-name related uses of [ci] no longer

<LI>threaten mysterious needs to add explicit pauses before following name words.

<p>

<LI>I want to add the [zao] proposal of John Cowan. Done, 4/15/2019. the imperative pronoun [koo] has been added though not officially. I should also add [dao] for the dummy argument, but not today (it is in as of 4/18)

<p>

#4/25 Making note of the idea that [ci] should not be a name marker unless followed

<LI>by a pause. This would require that one pause before ci-marked names and it would

<LI>remove some very confusing corrections for the false name marker problem. If we

<LI>required the pause to be explicit we would be imposing the expectation that whitespace

<LI>after [ci] is not a pause. Otherwise we could encourage writing a juncture after [ci]

<LI>to deny presence of a pause, which is reasonable considering the meanings of [ci].

<LI>I am implementing the version with explicit pauses between [ci] and names

<LI>and the directive not to pause after [ci] without explicit indication. This solution

```
<LI>involves rewriting existing text only in the rare instances
   where [ci] precedes a name.

<p>

<LI>4/25/2019 Corrected some instances of (expanded) badstress. Now
    forbidding (C)VVVV initial predicates. Probably I should use
   class badstress systematically in defining cmapua.

<p>

<LI>4/24/2019 Final consonants in syllables cannot be followed by
   syllabic continuants.

<LI>this rationalizes the definition of SyllableA.

<p>

<LI>4/22 I am thinking of explicitly flagging imperative sentences;
    not changing

<LI>the grammar but making this visible in the parse. This might
   also have some

<LI>effects on logical connections. 4/23 created an imperative
   class for atomic

<LI>imperative sentences; this has no actual effect on parses, just

<LI>organizes them in a more enlightening way.

<p>

<LI>4/17-18 2019: updates commented out which make sentpred
   linkable with forethought

<LI>and afterthought connectives (making some uses of [guu] to
   share arguments
<LI>unnecessary). There are subtleties. Basically, untensed predicates without

<LI>argument lists will be linked by A and KA series connectives. Such a linked

<LI>set can be tensed as a whole. Such a linked set will share a following termset.

<LI>This will probably change many parses in the Visit and other legacy sources.

<LI>This required some really subtle adjustments to work right, divinable from

<LI>the actual rules given. Definitely experimental.

<p>

<LI>3/9/2019 further, extended LIU1 to handle [ainoi] and its kin

<LI>(actual mod is to class Cmapua) Further, fixing mismatch

<LI>between connective and A classes. One does now have to pause

<LI>before [ha] and its compounds.

<p>

<LI>3/9/2019 repaired bugs in negative attitudinals. A pause

<LI>in a negative attitudinal of the [no, ui] form will not break

<LI>it. [ainoi] didnt work for two reasons: the clauses

<LI>in the definition of NOUI were in the wrong order, and

<LI>the connective class mistakenly included [noi] so the

<LI>phonetics checker was crashing! I had to move N and NOI
```

```
<LI>earlier to make this work. Not yet installed in the other

<LI>version.

<p>

<LI>1/26/2019 added [vie], JCB's "objective subjunctive" as a PA

<LI>class word. I should add this to the other file as well.

<p>

<LI>12/22/18: just a comment: one does not have to pause before [ha
   ] and its compounds.

<LI>I do not know whether to fix this. One did not have to in LIP
   either. For the moment I will

<LI>leave it as it is. As a matter of style, one probably should
   pause.

<p>

<LI>10/6/18 minor adjustments, made only in this file. Allow [sujo]
    (a wicked thing to say). Do not

<LI>allow [futo]: suffixed conversion operators must be nu + suffix
   .

<p>

<LI>6/2 fixed LIO + alien text. I also fixed some other glitches
   described in the reference grammar.

<p>

<LI>5/11 making version without "alternative parser" features. This
    version allows GAA but it doesn't
```

```
<LI>do anything: the definitions of argumentA and kin are the only
   point of difference. Master version:

<LI>becomes "alternative" by reinstating alternative definitions of
    argumentA and kin. Further, made changes

<LI>recommended in the reference grammar. ALTERNATIVE -- this is
   actually my master version. Edit

<LI>this and revise the argumentA and kin entries to make the
   original version.

<p>

<LI>4/24 discovered and repaired a bug re ci-marked names suffixed
   to descriptions. Discovered a bug in numerical

<LI>descriptions yet to be fixed: [lio] needs to be an alien text
   marker, maybe taking double quotes. The description-

<LI>with-suffixed-name bug was actually quite gruesome. I think it
   is repaired.

<p>

<LI>4/23 streamlined definition of descriptn. Shouldn't change
   anything. It was remarkably tricky though; preserving the old
   form

<LI>in case of further trouble.

<p>

<LI>4/22 I think this will be the master grammar file, with
   alternative lines to turn off the

<LI>GAA-related features. (1/21/2022, they are now turned off)

<p>
```

<LI>4/22 allowing general predicates in gasent1. This removes an extreme oddity in parsing of imperatives.

<LI>I do not see any new dangers from this.

<p>

<LI>4/22 I changed the final element of a keksent to be a sentence (new class uttA0), not a general sentence fragment.

<LI>several parse errors in the Visit were uncovered by this.

<p>

<LI>4/22: note that I still have the obligation to restore the [zao] construction.

<p>

<LI>4/9/2018 the large subject marker GAA can also be used to defend the beginnings of gasents and imperatives

<LI>from absorbing trailing arguments into an unintended statement. In this context [gaa] may be followed by [ga] ;-)

<p>

<LI>4/8/2018 this is an alternative version in which an argument which starts an SVO sentence will not be accepted

<LI>as a trailing argument of a previous sentence. This allows neat termination of [lepo] clauses preceding

<LI>a subject, for example. Unlike the previous alternative approach, this seems to involve a single fairly

<LI>tidy change: it is all an issue of avoiding needs for explicit closure. Further refinement: SVO sentences

<LI>can be marked with GAA (which is not a tense: it appears optionally just before the predicate, or just

<LI>before sutori arguments marked with GIO if there are any), the "large subject marker": an argument which

<LI>starts an SVO sentence *not marked with GAA* will not be accepted as a trailing argument of a previous

<LI>sentence. This is a sufficiently complex grammar change that it requires thought: it is not conservative

<LI>in my usual sense. The fact that GAA carries a mandatory stress is virtuous. Its resemblance to the

<LI>particle GA when used as a tense is not a bad thing: it would often be used instead of GA to close

<LI>a [lepo] clause appearing as a subject, and it is perhaps better for that purpose. Note that GAA can

<LI>and often will be followed by a tense. This grammar change depends strongly on the previous ruling that the O in

<LI>SOV(O) sentences must be marked with [gio]: S gio O^n V (O^m).

<p>

<LI>nuu is an atomic A core and there is no nu-affix to A connectives and their kin

<LI>1/20/2018 redefined CA cores to include a possible NU prefix. This allows more logically connected tenses, for example.

<p>

<LI>1/13/2018 reorganized the internals of class PA in a way which should allow more things and not forbid anything legal now.

<LI>this is pursuant on an analysis of the classes NI and PA as phrases, rather than words, as I start writing a global lexicography

<LI>proposal document. Enforced explicit pauses after PA phrases appearing as arguments with a following modifier with an argument.

<p>

<LI>12/30/2017 fixed a problem with name markers in the clas NameWord and made a slight change to the new option in NI (names

<LI>as dimensions).

<p>

<LI>12/27/2017 installing an alternative treatment of acronyms under which they are simply names (suffix -n to acronyms in all uses).

<LI>supporting this requires no change at all to acronymic name usage (just use the -n versions with the usual rules for names),

<LI>and for dimension usage requires [mue] to be a name marker and support for [mue] PreName as an alternative suffix to NI.

<p>

<LI>12/27/2017 Frivolously fooling with the capitalization conventions. They ought to work better now...but I could have broken something.

<LI>the main new idea was to require that a capitalized embedded letteral actually be followed by lowercase if it was preceded by lowercase

<LI>(with the obvious exception for a letteral followed by a letteral). Also changed the rules for diphthongs in cmapua to make all-caps

<LI>legal for cmapua. The general idea is that one can start with a capital letter and stay capitalized until one hits a lower case letter,

<LI>at which point one can jump back up to caps only at a juncture (after which you can remain capitalized) or temporarily for a vowel

<LI>after z- (after which lower case resumes) or an embedded literal (after which lowercase resumes). The total effect is that this allows

<LI>attested capitalization patterns in Loglan (including capitalization of embedded literals as in possessive articles and acronyms)

<LI>and also allows all-caps for individual words (attested in Leith but suppressed in my version) and supports capitalization of components

<LI>of names as in [la Beibi-Djein] (by artful use of syllable breaks: Leith just has BeibiDjein, which does not work for me).

<p>

<LI>12/26/2017 Installed [niu] (quotation of phonetically legal but so far non-Loglan words). I did not make [niu] a name marker, so if one were to

<LI>use it with names (where it isn't really appropriate), one would have to pause initially: [niu, Djan].

<p>

<LI>I note in this connection that quotation of names with li...lu remains limited, since names by themselves are not

<LI>utterances: one needs the [la]. I fixed this as an exception in the previous parser; I may do it here or I may

```
<LI>not, haven't decided. Single name words can be quoted with [liu
   ], of course, but not serial names.

<p>

<LI>12/24/2017 Refined treatment of vowel pairs for Cvv-V cmapua
   units. First 12/24 version rather disastrously

<LI>broken: this should be fixed!

<p>

<LI>12/23/2017 This is now completely commented, with minor local
   exceptions to which I will return later.

<LI>This document is the basis on which I will build all subsequent
    parsers, with due modifications to the comments.

<LI>The Python PEG engine and preamble files contain commands for
   constructinging a Python parser from it directly.

<p>

<LI>12/22/2017 major progress on commenting the grammar

<p>

<LI>yet later 12/20: no change in performance of the grammar,
   extensive commenting in the

<LI>grammar section. Considerable changes in arrangement: for
   example, vocatives, inverse vocatives,

<LI>and free modifiers are moved to a much earlier point. I'm
   hoping to get a genuinely almost readable

<LI>commented grammar...

<p>
```

```
<LI>later 12/20 starting the process of commenting and editing the
   grammar, starting

<LI>at basic sentence structures. Notably rewrote the class [
   keksent] more compactly,

<LI>one hopes with no actual effect on parses.

<p>

<LI>12/20/2017 Do not require expression of pause after finally
   stressed cmapua before

<LI>vowel initial predicate as a comma, since the initial vowel
   signals the pause anyway.

<LI>Allow final stress in names. Fixed bug in CVVHiddenStress.
   Prevented

<LI>broken monosyllables in finally stressed CVV djifoa. refinement
    of caprule

<p>

<LI>12/19/2017 seem to have had a versioning failure and lost the
   fix which requires

<LI>CVVy djifoa to be followed by complete complexes. Restored.

<p>

<LI>12/18/2017 fixed a bug in treatment of stressed syllables in
   recognizing predicate starts. Also

<LI>narrowed the generalized VCCV rule to allow more of the quite
   unlikely space of predicates with lots

<LI>of vowels before the CC pair. Probably they should be banned (
   and none have ever been proposed with
<LI>more than three) but that rule is not the context in which to
   arbitrarily ban half of them. Some cleanup

<LI>of the display of parses, for which updated version of
   logicpreamble.py should also be uploaded. A refinement

<LI>to class "connective" checking that apparent logical
   connectives are not initial segments of predicates.

<LI>This has the effect of delaying the declaration of "connective"
    until after the declaration of

<LI>"predstart".

<p>

<LI>12/17/2017 further refinement of the 12/16 version: a couple of
    bugs spotted.

<p>

<LI>12/16/2017 There should be no change in parsing behavior, but
   the predstart ruleset is shorter

<LI>and more intelligible, and I realized that Complex doesnt need
   a check for the anti-slinkui test

<LI>(the requirement that certain initial CVC cmapua be y hypenated
    which replaces the slinkui test))

<LI>at all: the way predstart works already ensures that initial CV
    cmapua fall off in the excluded

<LI>cases, the idea being that we test the front of a predicate
   without lookahead in all cases. Also

<LI>addressed the subtle point that one wasn’t forced to pause
   after a predicate before following y
```

```
<LI>(not likely to arise as a problem).

<p>

<LI>12/14/2017 Corrected vowel grouping to avoid paradoxical vowel triples which are default

<LI>grouped in a way which becomes illegal if made explicit. SyllableA really should contain a final

<LI>consonant: the previous form was messing up vowel grouping. Serious bug where end of djifoa

<LI>and syllable resolution of a predicate may fail to agree. I think I blocked this by ensuring that

<LI>final djifoa are not followed by vowels. Other fine tuning of the complex algorithm. Also had

<LI>to repair the check for CVCCCV and CVCCVV predicates.

<p>

<LI>12/13/2017: added kie ( utterance ) kiu to class LiQuote. Did fine tuning to ensure

<LI>that cmapua streams stop before [li] or [kie], that names can stop at double quotes or close

<LI>parentheses, and that the capitalization rule ignores opening parentheses as well as double

<LI>quotes. One can now adorn li lu with quotes (on the inside) in a reasonable way

<LI>and adorn kie kiu with parentheses (on the inside) in a reasonable way. One cannot

<LI>*replace* these words (or any words) with punctuation in my model of Loglan. Also,
<LI>updates to comments, and <LI>(end of utterance) added as a
   marker of terminal punctuation.

<p>

<H2>Comments on the initial release of this grammar</H2>

<LI>This is now done, in a first pass. That is, the grammar is
   adapted and appears to work, more or less.

<LI>What is needed is comments on the lexicography and the grammar
   ...Phonetics has now pretty clearly been sorted

<LI>from the grammar (there are some places where the phonetics
   accept grammar information with regard to punctuation).

<p>

<LI>Alien text is now handled somewhat differently. Some issues to
   do with quoting names are not finalized and have not been tested
   .

<p>

<LI>I added -iy and -uy as VV forms allowed in general in cmapua
   but not in other words; they are always monosyllabic. What this

<LI>immediately allows me to do is to give Y a name which is not
   phonetically irregular! [ziy] is supported: [yfi] is too, now.

<p>

<LI>capitalization is roughly back to where it was in the original,
    but all-caps are allowed.

<p>

<LI>acronyms are liable to be horrible.
```

<p>

<LI>Fixed the recursion problem in a way which will not be visible in ordinary parses. Streams of cmapua will always

<LI>be broken at name or alien text markers (instead of using lookahead to check that we do not stand at the beginning

<LI>of a name word or alien text word). The next cycle will then check for a name or alien text, and also check for

<LI>badnamemarkers; no lookahead is happening while a stream of cmapua is being read except checking for

<LI>the markers of names and alien text. This will change the way phonetic parses look (streams of cmapua will

<LI>break (and sometimes resume) at name markers or alien text markers, but it will not change any grammatical

<LI>parses.

<p>

<H2> Rule name conventions. These should now be enforced.</H2>

<p>

<LI> We define a way to sort rule names into layers, which an automated tool ought to be able to use. <p>

<LI> Phonemes: a rule name with one or two lower case letters optionally followed by digits. <p>

<LI> Intermediate phonetic and morphological groups: a rule name whose first two characters are an uppercase and a lowercase letter. This includes both classes of phonemes and classes of strings of phonemes.<p>

<LI> Lexemes (words, take this with a grain of salt): names made entirely of uppercase letters. Note that names and alien text constructions will tend to be treated as single words including their markers.

<LI> Lexeme-like: Names made of more than one uppercase followed by lowercase letters are of lexeme-like things (name and alien text markers).<p>

<LI> Lexeme precursors: Names made of uppercase letters followed by digits are precursors of lexeme classes. Cmapua components which are true affixes will be labelled with such a class and not with a lexeme class. The point here is that some of the lexeme classes have internal grammar.<p>

<LI> Grammar classes: Names starting with at least three lower case letters (anything may follow) are grammar classes. Ones the speaker should be aware of will not usually be followed by digits. <p>

<p>

<p>

<H2>Part I Phonetics</H2>

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

<p>

<LI>I have now parsed all the words in the dictionary, and all single words of appropriate classes parse successfully.

<LI>I have added alien text and quotation constructions which do not conform to these rules; so actually

<LI>all Loglan text should parse, mod some punctuation and capitalization issues. The conventions for

```
<LI>alien text here are not the same as those in the current
   provisional parser.

<p>

<LI>I believe the conventions for forcing comma pauses before vowel
    initial cmapua and after names

<LI>except in special contexts have been enforced. In a full
   grammar, one probably would want

<LI>to disable pauses before vowel initial letterals (done). This
   grammar also does not support the lingering

<LI>irregularities in acronyms (and won't).

<p>

<LI>This grammar (in Part I) is entirely about phonetics: all it
   does is parse text into names (with associated initial

<LI>pauses or name markers), cmapua (qua unanalyzed streams of
   cmapua units),

<LI>borrowings and complexes, along with interspersed comma pauses
   and marks

<LI>of terminal punctuation. It does support conventions about
   where commas are required

<LI>and a simple capitalization rule. Streams of cmapua break when
   markers initial

<LI>in other forms are encountered (and may in some cases resume
   when the markers

<LI>are a deception).

<p>
```

<LI>a likely locus for odd bugs is the group of predstartX rules which detect apparent cmapua which

<LI>are actually preambles to predicates. These are tricky! (and I did indeed find some lingering

<LI>problems when I parsed the dictionary). Another reason to watch this rule predstart

<LI>is that it carries a lot of weight: !predstart is used as a lightweight test

<LI>that what follows is a cmapua (a point discussed in more detail later).

<p>

<LI>In reviewing this, I think that very little is different from 1990's Loglan (the borrowing djifoa

<LI>are post-1989 L1, but not my creation). Some things add precision without making anything in 1990's Loglan incorrect.

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

<LI>The requirement that names resolve into syllables is new, and makes some 1990's Loglan names incorrect,

<LI> usually because they end in three consonants.

<LI>The rule restricting final consonant pairs from being noncontinuant/continuant is new, but

<LI> does not affect any actual predicate ever proposed.

<LI>Enhancing the VccV rule to also forbid CVVV...ccV caused one predicate to be changed

<LI> ([haiukre] became [haiukrre], and haiukre was a novelty anyway , using a new name for X in X-ray)

<LI>The exact definition of syllables and use of syllable breaks and stress marks is new (the close comma

<LI> was replaced with the hyphen, so Lo,is becomes Lo-is); but this does not make anything in 1990's Loglan

<LI> incorrect, it merely increases precision and makes phonetic transcript possible.

<LI>Forbidding doubled vowels in borrowings was new, was already approved, and caused us to change

<LI> [alkooli] to [alkoholi].

<LI>Formally allowing the CVccVV and CVcccV predicates without y-hyphens took a proposal in 2013 because

<LI> Appendix H was careless in describing their abandonment of the slinkui test, but the dictionary

<LI> makes it evident that this was their intent all along. The slinkui test had already been

<LI> abandoned in the 1990s.

<LI>Formally abandoning qwx was already something that the dictionary workers in the 1990's were working

<LI> on; we completed it.

<LI>Allowing glottal stop in vowel pairs and forbidding it as an allophone of pause is a new phonetic

<LI> feature in the proposal but not reflected in the parser, of course. Alternative pronunciations of

<LI> y and h and allowing h in final position are invisible or do not make any 1990's Loglan incorrect.

<LI>Permitting false name markers in names was already afoot in the 1990's and the basic outlines of our

<LI> approach were already in place. The rule requiring explicit pauses between a name marker not starting

<LI> a name word and the beginning of the next name word is new, but reflects something which was already

<LI> a fact about 1990's Loglan pronunciation: those pauses had to be made in speech

<LI>(and in the 1990's they had no tools to do relevant computer tests)! The requirement

<LI> that names resolve into syllables restricts which literal occurrences of name markers are actually

<LI> false name markers (the tail they induce in the name must itself resolve into syllables).

<LI>Working out the full details of borrowing djifoa was interesting: I'm not sure that I've done anything

<LI> *new* there; explicitly noting the stress shift in borrowing djifoa might be viewed as something

<LI> new but it is a logical consequence of JCB's permission to pause after a borrowing djifoa, which contains

<LI> explicit language about how it is to be stressed, and the

<LI> final definition of a borrowing djifoa as simply a borrowing followed by -y. The shift strikes

<LI> me as a really good idea anyway, because it marks djifoa with a pause after it as phonetically different

```
<LI> in an additional way other than ending with the very
   indistinct vowel y. My rules as given here do not

<LI> directly enforce the rule that a borrowing djifoa must be
   preceded by y but I think they indirectly

<LI> enforce it in all or almost all cases: the parser tries to
   read a borrowing djifoa before reading

<LI> any other kind of djifoa, so it is hard to see how to deploy a
    short djifoa in such a way that it would

<LI> fall off the head of a borrowing without using y.

<LI>These phonetics do not support certain irregularities in
   acronyms. We note that

<LI>it is now allowed to insert [, mue] into an acronym, which
   would be necessary for example

<LI>between a Ceo letteral and a following VCV letteral.

<p>

<H3>Sounds</H3>

<p>

<LI> the sound of silence

<p>

sp <- [ ]+

<H4>Vowels</H4>

<p>

#all vowels
```

```
<p>

Vo1 <- [aeiouyAEIOUY]

<p>

#regular vowels

<p>

Vo2 <- [aeiouAEIOU]

<H3>Consonants</H3>

<p>

#consonants

<p>

Co1 <- [bcdfghjklmnprstvzBCDFGHJKLMNPRSTVZ]

<p>

#consonants in voiced/unvoiced pairs

<p>

Cvoiced <- [bdgjvzBDGJVZ]

<p>

Cunvoiced <- [ptkcfsPTKCFS]

<p>

<LI>bad voice pair (or pair second term of which is h)

<LI>forbidden as pairs of final consonants
```

```
<p>

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

<p>

<H4>Letters and capitalization</H4>

<LI>letters

<p>

Letter <- (![qwxQWX] [a-zA-Z])

<p>

<LI>a capitalization convention which allows what our current one
   allows and also allows all-caps.

<LI>if case goes down from upper case to lower case, it can only go
    back up in certain cases. This

<LI>does allow capitalization of initial segments of words. There
   is a forward reference to the grammar

<LI>in that free capitalization of embedded literals is permitted,
   and capitalization of vowels

<LI>guarded with z in literals as in DaiNaizA.

<p>

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

<p>

Uppercase <- (![QWX] [A-Z])

<p>
```

```
caprule <- ([\"(]? &(([z] Vo1 (!Uppercase/&TAI0))/(Lowercase TAI0
   (!Uppercase/&TAI0))/(!(Lowercase Uppercase) .)) Letter (&(([z]
   Vo1 (!Uppercase/&TAI0))/(Lowercase TAI0 (!Uppercase/&TAI0))/(!(
   Lowercase Uppercase) .)) (Letter/Juncture))* !(Letter/Juncture))

<p>

<H4>Junctures: syllable breaks and stresses</H4>

<LI>syllable markers: the hyphen is always medial so must be
   followed by a letter.

<LI>the stress marks can be syllable final and word final. A
   juncture is never followed

<LI>by another juncture.

<p>

Stress2 <- [\'*]

<p>

Juncture <- ((([-] &Letter)/Stress2) !Juncture)

<p>

Stress <- ([\'*] !Juncture)

<H4>Terminal punctuations and general characters</H4>

<LI>terminal punctuation

<p>

Terminal <- [.:?!;#]

<p>
```

```
<LI>characters which can occur in words

<p>

Character <- (Letter/Juncture)

<p>

<H3>Alien text</H3>

<LI>to really get all Loglan text, we should add the alien text
   constructions and the markers of alien text,

<LI>[lie], [lao], [sao], [sue] and certain quotations which violate
    the phonetic rules.

<p>

<LI>we adopt the convention that all alien text may be but does not
    have to be enclosed in quotes.

<LI>it needs to be understood that in quoted alien text, whitespace
    is understood as [, y,]; in the unquoted

<LI>version this is shown explicitly. This handling of alien text
   is taken from the final 1990's treatment

<LI>of Linnaeans = foreign names, and extended by us to replace the
    impossible treatment of strong

<LI>quotation in 1989 Loglan.

<p>

<LI>this is a little different from what is allowed in the previous
    provisional parser, but similar.

<LI>A difference is that all the alien text markers are allowed to
   be followed by the same sorts of alien text.
```

```
<p>

<LI>the forms with [hoi] and [hue] are required to have following
   quotes in written form to avoid

<LI>unintended parses, which otherwise become likely in case of
   typos in non-alien text cases.

<p>

AlienText <- (([,]? sp [\"] (![\"] .)+ [\"])/([,]? sp (![, ] !
   Terminal .)+ ([,]? sp [Yy] [,]? sp (![, ] !Terminal .)+)*))

<LI> adding wrapper classes for alien text markers

<p>

HOIalien <- ([Hh] [Oo] [Ii])

<p>

HUEalien <- ([Hh] [Uu] Juncture? [Ee])

<p>

LIEalien <- ([Ll] [Ii] Juncture? [Ee])

<p>

LAOalien <- ([Ll] [Aa] [Oo])

<p>

LIOalien <- ([Ll] [Ii] Juncture? [Oo])

<p>

SAOalien <- ([Ss] [Aa] [Oo])

<p>
```

```
SUEalien <- ([Ss] [Uu] Juncture? [Ee])

<p>

AlienWord <- (&caprule ((HOIalien Juncture? &([,]? sp [\"]))/(
   HUEalien Juncture? &([,]? sp [\"]))/(LIEalien Juncture?)/(
   LAOalien Juncture?)/(LIOalien Juncture?)/(SAOalien Juncture?)/(
   SUEalien Juncture?)) AlienText)

<p>

<LI>while reading streams of cmapua, the parser will watch for the
   markers of alien text.

<p>

Alienmarker <- ((([Hh] [Oo] [Ii] Juncture? &([,]? sp [\"]))/([Hh] [
   Uu] Juncture? [Ee] Juncture? &([,]? sp [\"]))/([Ll] [Ii]
   Juncture? [Ee] Juncture?)/([Ll] [Aa] [Oo] Juncture?)/([Ll] [Ii]
   Juncture? [Oo] Juncture?)/([Ss] [Aa] [Oo] Juncture?)/([Ss] [Uu]
   Juncture? [Ee] Juncture?)) !Vo1)

<p>

<LI>5/11/18 added [lio] as an alien text marker, to support
   numerals.

<p>

<LI>the continuant consonants and the syllabic pairs they can form

<H3>Complex Vowel Forms</H3>

Continuant <- [mnlrMNLR]

<p>

Syllabic <- (([mM] [mM] !(Juncture? [mM]))/([nN] [nN] !(Juncture? [
   nN]))/([rR] [rR] !(Juncture? [rR]))/([lL] [lL] !(Juncture? [lL])
   ))

<p>

<LI>the obligatory monosyllables, and these syllables when broken
   by a usually bad syllable juncture.

<LI>The i-final forms are not obligatory mono when followed by
   another i.

<p>

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

<p>

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

<p>

<LI>the obligatory and optional monosyllables. Sequences of three
   of the same letter

<LI>are averted. Avoid formation of doubled i or u after ui or ui.

<p>

Mono <- (MustMono/([iI] !([uU] [uU]) Vo2)/([uU] !([iI] [iI]) Vo2))

<p>

<LI>vowel pairs of the form found in cmapua and djifoa.

<LI>(other than the special IY, UY covered in the cmapua rules)

<p>

<LI>The mysterious prohibition controls a permitted phonetic
   exception in djifoa gluing.
```

```
<LI>compua are never followed directly by vocalic continuants in
   any case.

<p>

Vv <- (!(!MustMono Vo2 Juncture? Vo2 Juncture? [Rr] [Rr]) (!
   BrokenMono Vo2 Juncture? Vo2))

<p>

<LI>the next vocalic unit to be chosen from a stream of vowels

<LI>in a predicate or name. This is different than in our Sources

<LI>and formally described in the proposal.

<p>

NextVowels <- (MustMono/(Vo2 &MustMono)/Mono/(!([Ii] Juncture [Ii]
   Vo1) !([Uu] Juncture [Uu] Vo1) Vo2))

<p>

<LI>5/11/18 forbidding consonantal vowels to follow the same vowel.

<p>

<LI>the doubled vowels that trigger the rule that one of them must
   be stressed

<p>

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

<p>

<LI>the mandatory "vowel" component of a syllable
```

```
<p>

Vocalic <- (NextVowels/Syllabic/[Yy])

<p>

<H3>Complex Consonant Forms</H3>

<p>

<LI>the permissible initial pairs of consonants, and the same pairs
    possibly

<LI>broken by syllable junctures.

<p>

Initial <- (([Bb] [Ll])/([Bb] [Rr])/([Cc] [Kk])/([Cc] [Ll])/([Cc] [
   Mm])/([Cc] [Nn])/([Cc] [Pp])/([Cc] [Rr])/([Cc] [Tt])/([Dd] [Jj])
   /([Dd] [Rr])/([Dd] [Zz])/([Ff] [Ll])/([Ff] [Rr])/([Gg] [Ll])/([
   Gg] [Rr])/([Jj] [Mm])/([Kk] [Ll])/([Kk] [Rr])/([Mm] [Rr])/([Pp]
   [Ll])/([Pp] [Rr])/([Ss] [Kk])/([Ss] [Ll])/([Ss] [Mm])/([Ss] [Nn
   ])/([Ss] [Pp])/([Ss] [Rr])/([Ss] [Tt])/([Ss] [Vv])/([Tt] [Cc])
   /([Tt] [Rr])/([Tt] [Ss])/([Vv] [Ll])/([Vv] [Rr])/([Zz] [Bb])/([
   Zz] [Ll])/([Zz] [Vv]))

<p>

MaybeInitial <- (([Bb] Juncture? [Ll])/([Bb] Juncture? [Rr])/([Cc]
   Juncture? [Kk])/([Cc] Juncture? [Ll])/([Cc] Juncture? [Mm])/([Cc
   ] Juncture? [Nn])/([Cc] Juncture? [Pp])/([Cc] Juncture? [Rr])/([
   Cc] Juncture? [Tt])/([Dd] Juncture? [Jj])/([Dd] Juncture? [Rr])
   /([Dd] Juncture? [Zz])/([Ff] Juncture? [Ll])/([Ff] Juncture? [Rr
   ])/([Gg] Juncture? [Ll])/([Gg] Juncture? [Rr])/([Jj] Juncture? [
   Mm])/([Kk] Juncture? [Ll])/([Kk] Juncture? [Rr])/([Mm] Juncture?
    [Rr])/([Pp] Juncture? [Ll])/([Pp] Juncture? [Rr])/([Ss]
   Juncture? [Kk])/([Ss] Juncture? [Ll])/([Ss] Juncture? [Mm])/([Ss
   ] Juncture? [Nn])/([Ss] Juncture? [Pp])/([Ss] Juncture? [Rr])/([
   Ss] Juncture? [Tt])/([Ss] Juncture? [Vv])/([Tt] Juncture? [Cc])
   /([Tt] Juncture? [Rr])/([Tt] Juncture? [Ss])/([Vv] Juncture? [Ll
   ])/([Vv] Juncture? [Rr])/([Zz] Juncture? [Bb])/([Zz] Juncture? [
   Ll])/([Zz] Juncture? [Vv]))

<p>

<LI>the permissible initial consonant groups in a syllable.
   Adjacent consonants should be initial pairs.

<LI>The group should not overlap a syllabic pair. Such a group is
   of course followed by a vocalic unit.

<p>

<LI>this rule for initial consonant groups is stated in NB3.

<p>

<LI>I forbid a three-consonant initial group to be followed by a
   syllabic pair. This seems obvious.

<p>

InitialConsonants <- ((!Syllabic Co1 &Vocalic)/(!(Co1 Syllabic)
   Initial &Vocalic)/(&Initial Co1 !(Co1 Syllabic) Initial !
   Syllabic &Vocalic))

<p>

<LI>the forbidden medial pairs and triples. These are forbidden
   regardless of placement

<LI>of syllable breaks.

<p>

<LI>each of these is actually a single consonant followed by an
   initial, and the idea was to identify CVC-CCV junctions which

<LI>would be hard to pronounce. But the placement of the syllable
   break is not relevant to the exclusion of the sequence.
```

```
<LI>Notice that the continuant syllabic pairs are excluded: this
   prevents final consonants from being included in such pairs.

<p>

NoMedial2 <- (([Bb] Juncture? [Bb])/([Cc] Juncture? [Cc])/([Dd]
   Juncture? [Dd])/([Ff] Juncture? [Ff])/([Gg] Juncture? [Gg])/([Hh
   ] Juncture? Co1)/([Jj] Juncture? [Jj])/([Kk] Juncture? [Kk])/([
   Ll] Juncture? [Ll])/([Mm] Juncture? [Mm])/([Nn] Juncture? [Nn])
   /([Pp] Juncture? [Pp])/([Rr] Juncture? [Rr])/([Ss] Juncture? [Ss
   ])/([Tt] Juncture? [Tt])/([Vv] Juncture? [Vv])/([Zz] Juncture? [
   Zz])/([CJSZcjsz] Juncture? [CJSZcjsz])/([Ff] Juncture? [Vv])/([
   Kk] Juncture? [Gg])/([Pp] Juncture? [Bb])/([Tt] Juncture? [Dd])
   /([FKPTfkpt] Juncture? [JZjz])/([Bb] Juncture? [Jj])/([Ss]
   Juncture? [Bb]))

<p>

NoMedial3 <- (([Cc] Juncture? [Dd] Juncture? [Zz])/([Cc] Juncture?
   [Vv] Juncture? [Ll])/([Nn] Juncture? [Dd] Juncture? [Jj])/([Nn]
   Juncture? [Dd] Juncture? [Zz])/([Dd] Juncture? [Cc] Juncture? [
   Mm])/([Dd] Juncture? [Cc] Juncture? [Tt])/([Dd] Juncture? [Tt]
   Juncture? [Ss])/([Pp] Juncture? [Dd] Juncture? [Zz])/([Gg]
   Juncture? [Tt] Juncture? [Ss])/([Gg] Juncture? [Zz] Juncture? [
   Bb])/([Ss] Juncture? [Vv] Juncture? [Ll])/([Jj] Juncture? [Dd]
   Juncture? [Jj])/([Jj] Juncture? [Tt] Juncture? [Cc])/([Jj]
   Juncture? [Tt] Juncture? [Ss])/([Jj] Juncture? [Vv] Juncture? [
   Rr])/([Tt] Juncture? [Vv] Juncture? [Ll])/([Kk] Juncture? [Dd]
   Juncture? [Zz])/([Vv] Juncture? [Tt] Juncture? [Ss])/([Mm]
   Juncture? [Zz] Juncture? [Bb]))

<p>

<H3>The Syllable</H3>

<LI>there are no formal rules about syllables as such in our
   Sources, which is odd since
```

```
<LI>the definition of predicates depends on the placement of
   stresses on syllables.

<p>

<LI>The first rule enforces the special point needed in complexes
   that

<LI>a CVC syllable is preferred to a CV syllable where possible; we
    economically apply

<LI>the same rule for default placement of syllable breaks
   everywhere, which is, with

<LI>that exception, that the break comes as soon as possible.

<p>

<LI>the SyllableB approach is taken if the following syllable would
    otherwise start with a syllabic pair.

<p>

<LI>the reason for this approach is that if one syllabizes a well
   formed complex in this way...

<LI>the syllable breaks magically fall on the djifoa boundaries.
   This does mean that the

<LI>default break in [cabro] is [cab-ro], which feels funny but is
   harmless. Explicitly breaking

<LI>it [ca-bro] will also parse correctly.

<p>

SyllableA <- (Co1 Vo2 FinalConsonant (!Syllable FinalConsonant)?)

<p>
```

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

<p>

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

<p>

<LI>The final consonant in a syllable. There may be one or two
   final consonants. A pair of final

<LI>consonants may not be a non-continuant followed by a continuant
   . A final consonant may not

<LI>start a forbidden medial pair or triple.

<p>

<LI>The rule that a final consonant pair may not be a non-
   continuant followed by a continuant

<LI>is natural and obvious but not in our Sources. Such a pair of
   consonants would seem to

<LI>naturally form another syllable.

<p>

<LI>a pair of final consonants cannot be differently voiced

<p>

FinalConsonant <- (!Syllabic !(&Badvoice Co1 !Syllable) (!(!
   Continuant Co1 !Syllable Continuant) !NoMedial2 !NoMedial3 Co1
   !(Juncture? (Vo2/Syllabic))))

<p>
```

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

<H4>Varieties of Syllable</H4>

<LI>Here are various flavors of syllable we may need.

<p>

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

<p>

SyllableD <- (&(InitialConsonants? ([Yy]/DoubleVowel/BrokenMono/(&
   Mono Vo2 DoubleVowel)/(!MustMono &Mono Vo2 BrokenMono)))
   Syllable)

<p>

<LI>this (below) is the kind of syllable which can exist in a
   borrowed predicate:

<LI>it cannot start with a continuant pair, it cannot have a y as
   vocalic unit,

<LI>and its vocalic unit (whether it has one or two regular vowels)

<LI>cannot be involved in a double vowel or an explicitly broken

<LI>mandatory monosyllable.

<p>

BorrowingSyllable <- (!Syllabic !SyllableD Syllable)

<p>

<LI>this is the final syllable of a predicate. It cannot be
   followed
```

```
<LI>without pause by a regular vowel.

<p>

VowelFinal <- (InitialConsonants? Vocalic Juncture? !Vo2)

<p>

<LI>syllables with syllabic consonant vocalic units

<LI>this class is only used in borrowings, and we *could*
   reasonably

<LI>require it to be followed by a vowel. But I won't for now.

<LI>for gluing this restriction would work, but we might literally
   borrow predicates

<LI>with syllabic continuant pronunciations.

<p>

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

<p>

<LI>syllables with y

<p>

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

<p>

<LI>an explicitly stressed syllable.

<p>

StressedSyllable <- ((SyllableA/SyllableB) Stress2)
```

```
<p>

<H3>Name Words</H3>

<LI>a final syllable in a word, ending in a consonant.

<p>

NameEndSyllable <- (InitialConsonants? (Syllabic/(Vocalic &
   FinalConsonant)) FinalConsonant? FinalConsonant? Stress? !Letter
   )

<p>

<H4>The Pause</H4>

<LI>the pause classes actually hang on the letter before the pause.

<p>

<LI>whitespace which might or might not be a pause.

<p>

Maybepause <- (Vo1 Stress2? sp Co1)

<p>

<LI>explicit pauses: these are whitespace before a vowel or after a
    consonant, or comma marked pauses.

<p>

Explicitpause <- ((Co1 Stress2? sp &Letter)/(Letter Stress2? sp &
   Vo1)/(Letter Stress2? [,] sp &Letter))

<p>

<H4>The full analysis of names</H4>
```

```
<LI>these are final syllables in words followed by whitespace which
    might not be a pause.

<LI>the definition actually doesnt mention the maybepause class.

<p>

MaybePauseSyllable <- (InitialConsonants? Vocalic Stress2? &(sp &
   Co1))

<p>

<LI>a name word (without initial marking) is resolvable into
   syllables and ends with a consonant.

<p>

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

<p>

<LI>this is a busted name word with whitespace in it -- but not
   whitespace at which one has to pause.

<p>

BadPreName <- (((MaybePauseSyllable sp)/(Syllable &Syllable))*
   NameEndSyllable)

<p>

<LI>This is a name marker followed by a consonant initial name word
    without pause.

<p>

<LI>I deployed a minimal set of name marker words; I can add the
   others whenever.
```

<LI>I have decided (see below) to retain the social lubrication words as vocative markers

<LI>*without* making them name markers, so one must pause [Loi, Djan]. By not allowing

<LI>freemods right after vocative markers in the vocative rule, I make [Loi hoi Djan] work as well,

<LI>without pause.

<p>

<LI>MarkedName <- &caprule ((([Ll] !pause [Aa] juncture?)/ ([Hh] [Oo] !pause [Ii] juncture?) / ([Hh] [Uu] juncture? !pause [Ee] juncture?) / ([Cc] !pause [Ii] juncture?)/([Ll] [Ii] juncture? ! pause [Uu] juncture?)/[Gg][Aa] !pause [Oo] juncture?/[Mm][Uu] juncture? !pause [Ee] juncture?) sp? &C1 &caprule PreName)

<p>

<LI> adding wrapper classes for name markers

<p>

LAname <- ([Ll] [Aa])

<p>

HOIname <- ([Hh] [Oo] [Ii])

<p>

CIname <- ([Cc] [Ii])

<p>

LIUname <- ([Ll] [Ii] Juncture? [Uu])

<p>

```
MUEname <- ([Mm] [Uu] Juncture? [Ee])

<p>

GAOname <- ([Gg] [Aa] [Oo])

<p>

HUEname <- ([Hh] [Uu] Juncture? [Ee])

<p>

<LI> second series is for marked names, no pauses after them

<p>

LAname2 <- ([Ll] !Explicitpause [Aa])

<p>

HOIname2 <- ([Hh] [Oo] !Explicitpause [Ii])

<p>

LIUname2 <- ([Ll] [Ii] Juncture? !Explicitpause [Uu])

<p>

MUEname2 <- ([Mm] [Uu] Juncture? !Explicitpause [Ee])

<p>

GAOname2 <- ([Gg] [Aa] !Explicitpause [Oo])

<p>

HUEname2 <- ([Hh] [Uu] Juncture? !Explicitpause [Ee])

<p>
```

```
MarkedName <- (&caprule ((LAname2 Juncture?)/(HOIname2 Juncture?)/(
   HUEname2 Juncture?)/(LIUname2 Juncture?)/(GAOname2 Juncture?)/(
   MUEname2 Juncture?)) sp? &Co1 &caprule PRENAME)

<p>

<LI>This is an unmarked name word with a false name marker in it.

<p>

FalseMarked <- (&PRENAME (!MarkedName Character)* MarkedName)

<p>

<LI>This is the full definition of name words. These are either
   marked consonant initial names without pause defined above,

<LI>names without false name markers beginning with explicit pauses
    (either comma marked or vowel-initial)

<LI>and name markers followed, with or without pause, by name words
   . In the latter case there must be at least

<LI>whitespace before a vowel initial name.

<p>

<LI>a series of names without false name markers and names marked
   with ci, separated by spaces, may be appended.

<p>

<LI>there is a look ahead at the grammar: a NameWord can be
   followed without explicit pause (there is whitespace and

<LI>a pause in speech!) by another

<LI>kind of utterance only in a serial name when what follows is of
    the form [ci] predunit, to be included
```

```
<LI>in the name.

<p>

NAMEWORD <- (((&caprule MarkedName)/([,] sp !FalseMarked &caprule
   PRENAME)/(&Vo1 !FalseMarked &caprule PRENAME)/(&caprule (((
   LAname Juncture?)/(HOIname Juncture?)/(HUEname Juncture?)/(
   CIname Juncture? &([,]? sp))/(LIUname Juncture?)/(MUEname
   Juncture?)/(GAOname Juncture?)) !Vo1 [,]? sp? &caprule PRENAME))
   ) (([,]? sp !FalseMarked &caprule PRENAME)/([,]? sp &([Cc] [Ii])
    NAMEWORD))* &((sp? [Cc] [Ii] predunit)/(&(([,] sp)/Terminal
   /[\")]/!.) .)/!.))

<p>

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

<p>

<LI>I am contemplating adding the words of social lubrication as
   name markers, but in a more restricted

<LI>way that in the last provisional parser, in which I made them
   full-fledged vocative markers. [Actually,

<LI>I preserved their status as vocative markers without restoring
   their status as name markers, in the latest version].

<p>

<LI>adding [mue] as a name marker

<p>

Namemarker <- ((([Ll] [Aa] Juncture?)/([Hh] [Oo] [Ii] Juncture?)/([
   Hh] [Uu] Juncture? [Ee] Juncture?)/([Cc] &(Explicitpause/([Ii]
   Juncture? sp PRENAME)) [Ii] Juncture?)/([Ll] [Ii] Juncture? [Uu]
    Juncture?)/([Gg] [Aa] [Oo] Juncture?)/([Mm] [Uu] Juncture? [Ee]
    Juncture?)) !V01)

<p>

<LI>this is the bad name marker phenomenon that needs to be
   excluded. This captures the idea

<LI>that what follows the name could be pronounced without pause as
    a name word according to the

<LI>orthography, but the fact that whitespace is present shows that
    this is not the intention.

<p>

<LI>it is worth noting that name markers at heads of name words
   pass this test

<LI>(because I omitted the test that what follows is not a PreName
   in the interests

<LI>of minimizing lookahead);

<LI>but this test is only applied to strings that have already been
    determined not to

<LI>be of class NameWord.

<p>

Badnamemarker <- (Namemarker !V01 [, ]? sp? BadPreName)

<LI>we test for the bad name marker condition at the beginning of
   each stream of cmapua,

<LI>and streams of cmapua stop before name markers (and may resume
   at a name marker

<LI>if neither a NameWord nor the bad marker condition is found).
```

```
<p>

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

<p>

<H3>Predicate Start Test</H3>

<LI>predicate start tests: the idea is the same as class "
   connective" below, to recognize

<LI>the start of a predicate without recursive appeals to the whole
    nasty definition of predicate.

<LI>The reason to do it is to recognize when CV^n followed by CC
   cannot be a cmapua unit.

<p>

<LI>New implementation 4/28/2019. This allows only (C)V(V)(V)
   before the pair of vowels, for much less

<LI>potential lookahead.

<p>

Vthree <- ((Vo2 Juncture?) (Vo2 Juncture?) (Vo2 Juncture?))

<p>

Vfour <- ((Vo2 Juncture?) (Vo2 Juncture?) (Vo2 Juncture?) (Vo2
   Juncture?))

<p>

<LI>predicate starting with two or three consonants: rules out CC(C
   )V(V) forms. Junctures in

<LI>the initial consonant group ignored.
```

```
<p>

Predstart1 <- (((&MaybeInitial Co1 Juncture? MaybeInitial)/
   MaybeInitial) &Vo2 !(Vo2 Stress !Mono Vo2) !(Vo2 Juncture? Vo2 !
   Character) !(Vo2 Juncture? !Character))

<p>

<LI>an apparent cmapua unit followed by a consonant group which
   cannot start a predicate -- CV(V) case

<p>

Predstart2 <- (Co1 Vo2 Juncture? (Vo2 Juncture?)? !Predstart1 Co1
   Juncture? Co1)

<p>

<LI>a stressed CV^n before a consonant group (CV(V) case)

<p>

Predstart3 <- (Co1 !Vthree (!StressedSyllable Vo2 Juncture?)? &
   StressedSyllable Vo2 Vo2? Juncture? Co1 Juncture? Co1)

<p>

<LI>other (C)V^n followed by nonpredicate

<p>

Predstart4 <- (Co1? Vo2 Juncture? (Vo2 Juncture?)? (Vo2 Juncture?)?
    !Predstart1 !(MaybeInitial Vo2) Co1 Juncture? Co1)

<p>

<LI>other stressed (C)V^n followed by consonant group

<p>
```

```
Predstart5 <- (Co1? !Vfour (!StressedSyllable Vo2 Juncture?)? (!
   StressedSyllable Vo2 Juncture?)? &StressedSyllable Vo2 Vo2?
   Juncture? !(MaybeInitial Vo2) Co1 Juncture? Co1)

<p>

<LI>forms with y; implemented CVVhy alternative for CVV cmapua

<p>

Predstart6 <- (Co1 (Vo2 Juncture?) ((Vo2 Juncture? [Hh]?)/(Co1
   Juncture? (Co1 Juncture?)?)) [Yy])

<p>

Predstart <- (Predstart1/Predstart2/Predstart3/Predstart4/
   Predstart5/Predstart6)

<p>

<LI>it is worth noting that in the sequel we have systematically
   replaced tests &Cmapua

<LI>with !predstart. The former involves lots of lookahead and was
   causing recursion crashes

<LI>in Python. The phonetics and the grammar are both structured so
    that any string

<LI>starting with a name marker is tested for NameWord-hood before
   it is tested for

<LI>cmapua-hood; the only thing it is tested for later is predicate
   -hood, and predstart

<LI>is a rough and ready test that something might be a predicate (
   and at any rate

<LI>cannot be a cmapua).
```

```
<p>

<H3>Structure Word Phonetics</H3>

<LI>this class requires pauses before it, after all the phonetic
   word classes.

<LI>what is being recognized is the beginning of a logical
   connective.

<p>

<LI>To avoid horrible recursion problems, giving this a concrete
   phonetic definition

<LI>without much lookahead. This can go right up in the phonetics
   section if it works

<LI>(and here it is!).

<p>

<LI>single vowel cmapua syllables early for connectives

<p>

a <- ([Aa] !Badstress Juncture? !Vo1)

<p>

e <- ([Ee] !Badstress Juncture? !Vo1)

<p>

i <- ([Ii] !Badstress Juncture? !Vo1)

<p>

o <- ([Oo] !Badstress Juncture? !Vo1)
```

```
<p>

u <- ([Uu] !Badstress Juncture? !Vo1)

<p>

Hearly <- (!Predstart [Hh])

<p>

Nearly <- (!Predstart [Nn])

<p>

<LI>these appear here for historical reasons and could be moved
   later

<p>

Connective <- (sp? !Predstart ([Nn] [Oo] Juncture? !i)? (a/e/i/o/u
   /(Hearly a)/(Nearly uu)) Juncture? !Vo2 !(!Predstart [Ff] [Ii])
   !(!Predstart [Mm] [Aa]) !(!Predstart [Zz] [Ii]))

<p>

<LI>cmapua units starting with consonants. This is the exact
   description from NB3. The fancy tail in each of the

<LI>three cases is enforcing the rule about pausing before a
   following predicate if stressed.

<p>

<LI>consonant initial cmapua units may not be followed by vowels
   without pause.

<p>
```

<LI>I am adding [iy] and [uy] (always monosyllable, yuh and wuh) as vowel pairs permitted in VV and CVV cmapua units.

<LI>it is worth noting that the "yuh" and "wuh" pronunciations of these diphthongs

<LI>are surprising to the English-reading eye.

<LI>The use for this envisaged is that the name [ziy] of Y becomes easy to introduce. Adding word space

<LI>is always nice, and these words seem pronounceable. I also made [yfi] possible: Y now has phonetically

<LI>regular names.

<p>

```
CmapuaUnit <- ((Co1 Mono Juncture? Vo2 !(Stress2 sp? &Co1 Predstart
   ) Juncture? !Vo1)/(Co1 (Vv/([Ii] [Yy])/([Uu] [Yy])) !(Stress2 sp
   ? &Co1 Predstart) Juncture? !Vo1)/(Co1 Vo2 !(Stress2 sp? &Co1
   Predstart) Juncture? !Vo1))
```

<p>

<LI>A stream of cmapua is read until the start of a predicate or a name marker word or an alien text marker word or a quote or parenthesis marker word is encountered.

<LI>the stream might resume with a name marker word if it does not in fact start a name word and does not potentially start a name

<LI>word due to inexplicit whitespace (doesn’t satisfy the bad name marker condition).

<p>

<LI>we force explicit comma pauses before logical connectives, but not before vowel initial cmapua in general;

```
<LI>other conditions force at least whitespace, which does stand
   for a pause, before such words.

<p>

<LI>detect starts of quotes or parentheses with li or [kie]

<p>

Likie <- (([Ll] [Ii] Juncture? !Vo1)/([Ki] [Ii] Juncture? [Ee]
   Juncture? !Vo1))

<p>

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

<p>

Cmapua <- (&caprule !Badnamemarker ((!Predstart (Vv/([Ii] [Yy])/([
   Uu] [Yy])) !(Stress2 sp? &Co1 Predstart) Juncture? NOIO)/(!
   Predstart [Nn] [Oo] Juncture? !Predstart (Vv/([Ii] [Yy])/([Uu] [
   Yy])) !(Stress2 sp? &Co1 Predstart) Juncture?)/((!Predstart (Vv
   /([Ii] [Yy])/([Uu] [Yy])) !(Stress2 sp? &Co1 Predstart) Juncture
   ?)+/(((!Predstart Vo1 !(Stress2 sp? &Co1 Predstart) Juncture?)
   /(!Predstart CmapuaUnit)) (!Namemarker !Alienmarker !Likie !
   Predstart CmapuaUnit)*))/(!Predstart Vo2 !(Stress2 sp? &Co1
   Predstart) Juncture?)) !Vo1 !(Co1+ Juncture) !(sp? Connective))

<p>

<LI>I have apparently now completely solved the problem of parsing
   cmapua as well as name words.

<p>

<H3>Predicate Phonetics</H3>

<LI>Now for predicates.
```

```
<H4>Djifoa ("affixes")</H4>

<p>

<LI>the elementary djifoa (not borrowings)

<p>

<LI>various special flavors of these djifoa will be needed.

<LI>These are the general definitions.

<p>

<LI>The NOY and Bad forms are for use for testing candidate
   borrowings for resolution

<LI>with bad syllable break placements. Borrowings do not contain Y
   ...

<p>

<LI>CVV djifoa with phonetic hyphens.

<p>

<LI>added checks to all cmapua classes: the vowel final ones, when
   not phonetically hyphenated, cannot

<LI>be followed by a regular vowel. This is crucial for getting the
    syllable analysis and the djifoa

<LI>analysis to end at the same point.

<p>

<LI>allowing h to be inserted before y in CVVy djifoa for a CVVhy
   form.

<p>
```

```
<LI>allowing -r glue to be expressed as -rr

<LI> some classes just for djifoa glue

<p>

wy <- [Yy]

<p>

ar <- [Rr]

<p>

en <- [Nn]

<p>

hh <- [Hh]

<p>

Dash <- [-]

<p>

Cvv <- (Co1 Vv ((Juncture? hh? wy Dash? &Complex)/(Juncture? ar ar?
    Juncture? &Co1)/(en Juncture? &ar)/(Juncture? !Vo2)))

<p>

CvvNoHyphen <- (Co1 Vv Juncture? !Vo2)

<p>

CvvHiddenStress <- (Co1 &DoubleVowel Vo1 Dash? Vo1 ((Dash? hh? wy
   Dash? &Complex)/(ar Dash? &Co1)/(en Dash? &ar)/(Dash? !Vo2)))

<p>
```

```
CvvFinalStress <- (Co1 Vv ((Stress2 hh? wy Dash? &Complex)/(ar
    Stress2 &Co1)/(Stress2 ar ar Juncture? &Co1)/(en Stress2 &ar)/(
    Stress2 !Vo2)))

<p>

CvvNoY <- (Co1 Vv ((Juncture? ar ar? Juncture? &Co1)/(en Juncture?
    &ar)/(Juncture? !Vo2)))

<p>

CvvNoYFinalStress <- (Co1 Vv ((ar Stress2 &Co1)/(Stress2 ar ar
    Juncture? &Co1)/(en Stress2 &ar)/(Stress2 !Vo2)))

<p>

CvvNoYMedialStress <- (Co1 !BrokenMono Vo2 Stress2 Vo2 Dash? !Vo2)

<p>

<LI>CCV djifoa with phonetic hyphens.

<p>

Ccv <- (Initial Vo2 ((Juncture? wy Dash? &Letter)/(Juncture? !Vo2))
    )

<p>

CcvStressed <- (Initial Vo2 ((Stress2 wy Dash? &Letter)/(Stress2 !
    Vo2)))

<p>

CcvNoY <- (Initial Vo2 Juncture? !Vo2)

<p>

CcvBad <- (MaybeInitial Vo2 Juncture? !Vo2)
```

```
<p>

CCVBadStressed <- (MaybeInitial Vo2 Stress2 !Vo2)

<p>

<LI>CVC djifoa with phonetic hyphens. These cannot be final and are
     always followed by a consonant (well, the

<LI>-y form may be followed by a vowel...

<LI>an eccentric syllable break is supported if the CVC is y-
   hyphenated:

<LI>[me-ky-kiu] and [mek-y-kiu] are both legal. The default is the
   latter.

<p>

Cvc <- ((Co1 Vo2 !NoMedial2 !NoMedial3 Co1 ((Juncture? wy Dash? &
   Letter)/(Juncture? &Co1)))/(Co1 Vo2 Juncture Co1 wy Dash? &
   Letter))

<p>

CvcStressed <- ((Co1 Vo2 !NoMedial2 !NoMedial3 Co1 ((Stress2 wy
   Dash? &Letter)/(Stress2 &Letter)))/(Co1 Vo2 Stress2 Co1 wy Dash?
     &Letter))

<p>

CvcNoY <- (Co1 Vo2 !NoMedial2 !NoMedial3 Co1 Juncture? &Co1)

<p>

CvcBad <- (Co1 Vo2 !NoMedial2 !NoMedial3 Juncture? Co1 &Co1)

<p>
```

```
CvcNoYStressed <- (Co1 Vo2 !NoMedial2 !NoMedial3 Co1 Stress2 &Co1)

<p>

CvcBadStressed <- (Co1 Vo2 !NoMedial2 !NoMedial3 Stress2 Co1 &Co1)

<p>

<LI>the five letter forms (always final in complexes)

<p>

CcvCv <- (Initial Vo2 Juncture? Co1 Vo2 Dash? !Vo2)

<p>

CcvCvStreased <- (Initial Vo2 Stress2 Co1 Vo2 Dash? !Vo2)

<p>

CcvCvBad <- (MaybeInitial Vo2 Juncture? Co1 Vo2 Dash? !Vo2)

<p>

CcvCvBadStressed <- (MaybeInitial Vo2 Stress2 Co1 Vo2 Dash? !Vo2)

<p>

CvcCv <- ((Co1 Vo2 Juncture? Initial Vo2 Dash? !Vo2)/(Co1 Vo2 !
   NoMedial2 Co1 Juncture? Co1 Vo2 Dash? !Vo2))

<p>

CvcCvStressed <- ((Co1 Vo2 Stress2 Initial Vo2 Dash? !Vo2)/(Co1 Vo2
    !NoMedial2 Co1 Stress2 Co1 Vo2 Dash? !Vo2))

<p>

<LI>the medial five letter djifoa
```

```
<p>

CcvCy <- (Initial Vo2 Juncture? Co1 wy Dash?)

<p>

CvcCy <- ((Co1 Vo2 Juncture? Initial wy Dash?)/(Co1 Vo2 !NoMedial2
   Co1 Juncture? Co1 wy Dash?))

<p>

CcvCyStressed <- (Initial Vo2 Stress2 Co1 wy Dash?)

<p>

CvcCyStressed <- ((Co1 Vo2 Stress2 Initial wy Dash?)/(Co1 Vo2 !
   NoMedial2 Co1 Stress2 Co1 wy Dash?))

<p>

<H4>Borrowed Predicates</H4>

<LI>to reason about resolution of borrowings into both syllables
   and djifoa (we want to exclude the latter

<LI>but we need to define it adequately) we need to recognize where
    to stop. A predicate word ends either

<LI>at a non-character (not a letter or syllable mark: whitespace,
   comma or terminal punctuation) or it

<LI>has an explicit or deducible penultimate stress. Borrowings do
   not contain doubled vowels, so they

<LI>have to have explicit stress in the latter case.

<p>

<LI>analysis: the stressed tail consists of a stressed syllable
   followed by an unstressed syllable.
```

```
<LI>identifying an unstressed final syllable is complicated by
   recognizing which CVV combinations can

<LI>be one syllable. This will either be an explicitly stressed
   syllable followed by a single syllable

<LI>or a syllable suitable to be stressed followed by an explicitly
    final syllable. CVV djifoa can

<LI>contain both syllables in a tail and of course the five letter
   djifoa have to be tails. A never stressed

<LI>SyllableC (with a continuant) may intervene.

<p>

<LI>tail of a borrowing with an explicit stress

<p>

BorrowingTail1 <- (!SyllableC &StressedSyllable BorrowingSyllable
   (!StressedSyllable &SyllableC BorrowingSyllable)? !
   StressedSyllable &BorrowingSyllable VowelFinal)

<p>

<LI>tail of a borrowing or borrowing djifoa with no explicit stress

<p>

BorrowingTail2 <- (!SyllableC BorrowingSyllable (!StressedSyllable
   &SyllableC BorrowingSyllable)? !StressedSyllable &
   BorrowingSyllable VowelFinal (&wy/!Character))

<p>

<LI>tail of a stressed borrowing djifoa, different because stress
   is shifted to the end
```

```
<p>

BorrowingTail3 <- (!SyllableC !StressedSyllable BorrowingSyllable
   (!StressedSyllable &SyllableC BorrowingSyllable)? &
   BorrowingSyllable InitialConsonants? Vocalic Stress2 &wy)

<p>

BorrowingTail <- (BorrowingTail1/BorrowingTail2)

<p>

<LI>short forms that are ruled out: CCVV and CCCVV forms.

<p>

Ccvv <- ((InitialConsonants Vo2 Juncture? Vo2 Juncture? !Character)
   /(InitialConsonants Vo2 Stress2 !Mono Vo2 Juncture?))

<p>

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

<p>

<LI>a continuant syllable cannot be initial in a borrowing and
   there cannot be successive continuant

<LI>syllables. There really ought to be no more than one!

<p>

<LI>borrowing, before checking that it doesnt resolve into djifoa

<p>

PreBorrowing <- (&Predstart !Ccvv !Cmapua !SyllableC (!
   BorrowingTail !StressedSyllable !(SyllableC SyllableC)
   BorrowingSyllable)* BorrowingTail)
```

```
<p>

<LI>ditto for an explicitly stressed borrowing

<p>

StressedPreBorrowing <- (&Predstart !Ccvv !Cmapua !SyllableC (!
   BorrowingTail !StressedSyllable !(SyllableC SyllableC)
   BorrowingSyllable)* BorrowingTail1)

<p>

<LI>borrowing djifoa without explicit stress (before resolution
   check)

<p>

PreBorrowing2 <- (&Predstart !Ccvv !Cmapua !SyllableC (!
   BorrowingTail !StressedSyllable !(SyllableC SyllableC)
   BorrowingSyllable)* BorrowingTail2)

<p>

<LI>stressed borrowing djifoa (before resolution check).

<p>

PreBorrowing3 <- (&Predstart !Ccvv !Cmapua !SyllableC (!
   BorrowingTail3 !StressedSyllable !(SyllableC SyllableC)
   BorrowingSyllable)* BorrowingTail3)

<LI>Now comes the problem of trying to say that a preborrowing
   cannot resolve into cmapua. The difficulty is with

<LI>recognizing the tail, so making sure that the two resolutions
   stop in the same place.

<p>
```

```
<LI>we know because it is a borrowing that there is at most one
   explicit stress, and it has to fall

<LI>in one of the cmapua! This should make it doable.

<p>

<LI>borrowing djifoa are terminated with y, so the final djifoa
   needs to take this into account

<p>

<LI>the idea behind both djifoa analyses is the same. If we end
   with a final djifoa followed by

<LI>a non-character, we improve our chances of ending the syllable
   analysis at the same point. We control

<LI>this by identifying djifoa with stresses in them: a medially
   stressed djifoa must be the last one

<LI>(and the syllable analysis will find its stressed syllable and
   end at its final syllable, the fact

<LI>that djifoa cannot be followed by vowels ensuring that the
   syllable analysis cannot overrun its end.

<LI>When the djifoa is finally stressed, the complex analysis ends
   with a further djifoa guaranteed to have

<LI>just one syllable, and the syllable analysis again will stop in
    the same place. The medial five letter forms

<LI>and borrowing djifoa of course are finally stressed mod an
   additional unstressed syllable which is skipped

<LI>by the syllable analysis, because it allows one to ignore an
   actually penultimate syllable with y or
```

```
<LI>a syllabic consonant. In the case where we never find a stress and end up at a final djifoa, the syllable

<LI>analysis will carry right through to the same final point.

<p>

<LI>in the attempted resolution of borrowings, our life is easier because we do not have

<LI>borrowing djifoa or medial five letter forms to consider, or any forms with y-hyphens.

<p>

RfinalDjifoa <- ((CcvCvBad/CvcCv/CvvNoHyphen/CcvBad/CvcBad) (&wy/!Character))

<p>

RmediallyStressed <- (CcvCvBadStressed/CvcCvStressed/CvvNoYMedialStress)

<p>

RfinallyStressed <- (CvvNoYFinalStress/CCVBadStressed/CvcBadStressed/CvcNoYStressed)

<p>

BorrowingComplexTail <- (RmediallyStressed/(RfinallyStressed ((&(Co1 Mono) CvvNoHyphen)/CcvBad))/RfinalDjifoa)

<p>

ResolvedBorrowing <- ((!BorrowingComplexTail (CvvNoY/CcvBad/CvcBad))* BorrowingComplexTail)

<p>
```

```
<LI>borrowed predicates

<p>

Borrowing <- (!ResolvedBorrowing &caprule PreBorrowing !(sp?
   Connective))

<p>

<LI>explicitly stressed borrowed predicates

<p>

StressedBorrowing <- (!ResolvedBorrowing &caprule
   StressedPreBorrowing !(sp? &Vo1 Cmapua))

<p>

#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.

<p>

<LI>borrowing djifoa without explicit stress (stressed ones are not
    of this class!)

<LI>Note that one can pause after these (explicitly, with a comma,
   in which case the stress must be explicit too)

<p>

BorrowingDjifoa <- (!ResolvedBorrowing &caprule PreBorrowing2 ((
   Stress2 wy [,] sp)/(Juncture? wy Dash?)))

<p>

<LI>stressed borrowing djifoa finally implemented!
```

```
<p>

StressedBorrowingDjifoa <- (!ResolvedBorrowing &caprule
   PreBorrowing3 wy Dash? ([,] sp)?)

<p>

<H4>Complex Predicates</H4>

<LI>We resolve complexes twice, once into syllables and once into
   djifoa. We again have to ensure that

<LI>we end up in the same place! The syllable resolution is very
   similar to that of borrowings;

<LI>the unstressed middle syllable of the tail can be a SyllableY,
   and can also be a

<LI>SyllableC if the final djifoa is a borrowing.

<p>

<LI>A stressed borrowing djifoa with the property that the tail is
   still a phonetic complex is

<LI>a unit for this analysis.

<p>

<LI>note here that I specifically rule out a complex being followed
    without pause by y. I do not rule

<LI>this out for the vowel final djifoa because they can be
   followed by y at the end of a borrowing

<LI>djifoa.

<p>
```

```
DefaultStressedSyllable <- Syllable

<p>

PhoneticComplexTail1 <- (!SyllableC !SyllableY &StressedSyllable
   DefaultStressedSyllable (!StressedSyllable &(SyllableC/SyllableY
   ) Syllable)? !StressedSyllable !SyllableY VowelFinal !Vo1)

<p>

PhoneticComplexTail2 <- (!SyllableC !SyllableY
   DefaultStressedSyllable (!StressedSyllable &(SyllableC/SyllableY
   ) Syllable)? !StressedSyllable !SyllableY VowelFinal !Character)

<p>

PhoneticComplexTail <- (PhoneticComplexTail1/PhoneticComplexTail2)

<p>

<LI>note the explicit predstart test here.

<p>

PhoneticComplex <- (&Predstart !Ccvv !Cmapua !SyllableC ((
   StressedBorrowingDjifoa &PhoneticComplex)/(!PhoneticComplexTail
   !StressedSyllable !(SyllableC SyllableC) Syllable))*
   PhoneticComplexTail)

<p>

<LI>the analysis of final djifoa and stressed djifoa differs only
   in details from

<LI>what is above for resolution of borrowings. The issues about
   CVV djifoa with doubled

<LI>vowels are rather exciting.

<p>
```

<LI>a stressed borrowing djifoa with the tail still a phonetic complex is a black box unit for

<LI>this construction.

<p>

<LI>My approach imposes the restriction on JCB’s "pause after a borrowing djifoa" idea that what follows

<LI>the pause must itself contain a penultimate stress: [igllu’ymao] is a predicate but [igllu’y, mao] is not.

<LI>while [iglluy’, gudmao] is a predicate.

<p>

<LI>the analysis of the djifoa resolution process is the same as above, with additional remarks

<LI>about doubled vowel syllables: notice that where the complex tail involved a doubled vowel syllable

<LI>without explicit stress, we insist on that djifoa or the single-syllable next djifoa ending in

<LI>a non-character: in the absence of explicit stress, we always rely on whitespace or punctuation

<LI>to indicate the end of the predicate.

<p>

<LI>all sorts of subtleties about borrowings and borrowing djifoa are finessed by always looking for

<LI>them first. There are no restrictions re fronts of borrowings or borrowing djifoa looking like regular

```
<LI>djifoa; the fact that borrowing djifoa end in y and borrowings
   do not contain y makes it always

<LI>possible to tell when one is looking at the head of a borrowing
    djifoa. Regular djifoa just before a borrowing

<LI>djifoa need to be y-hyphenated so as not to be absorbed into
   the front of the borrowing (I don’t believe

<LI>that I actually need to impose a formal rule to this effect,
   though I am not absolutely certain; it would

<LI>be difficult to formulate [and does appear in the previous
   version, where it is a truly unintelligible piece

<LI>of PEG code]).

<p>

FinalDjifoa <- ((Borrowing/CcvCv/CvcCv/CvvNoHyphen/CcvNoY) !
   Character)

<p>

MediallyStressed <- (StressedBorrowing/CcvCvStreased/CvcCvStressed/
   CvvNoYMedialStress)

<p>

FinallyStressed <- (StressedBorrowingDjifoa/CcvCyStressed/
   CvcCyStressed/CvvFinalStress/CcvStressed/CvcStressed)

<p>

ComplexTail <- ((CvvHiddenStress ((&(Co1 Mono) CvvNoHyphen)/CcvNoY)
    !Character)/(FinallyStressed ((&(Co1 Mono) CvvNoHyphen)/CcvNoY)
   )/MediallyStressed/FinalDjifoa)

<p>
```

```
PreComplex <- ((!CvvHiddenStress !ComplexTail ((
    StressedBorrowingDjifoa &PhoneticComplex)/BorrowingDjifoa/CvcCy/
    CcvCy/Cvv/Ccv/Cvc))* ComplexTail)

<p>

<LI>originally I had complicated tests here for the conditions
    under which an initial

<LI>CVC cmapua has to be y-hyphenated: I was being wrong headed,
    the predstart rules

<LI>already enforce this (in the bad cases, the initial CV- falls
    off). The user will

<LI>simply find that they cannot put the word together otherwise.
    The previous version

<LI>did need this test because it actually used full lookahead to
    check for the start of a predicate.

<p>

Complex <- (&caprule &PreComplex PhoneticComplex !(sp? Connective))

<p>

<H3>Quotation and Parenthesis of well-formed Loglan utterances;
    word classes</H3>

<LI>format for the LI quote and KIE parenthesis

<p>

LiQuote <- ((&caprule [Ll] [Ii] Juncture? Comma2? [\"]
    PhoneticUtterance [\"] Comma2? &caprule [Ll] [Uu] Juncture? !(sp
    ? Connective))/(&caprule [Kk] [Ii] Juncture? [Ee] Juncture?
    Comma2? [(] PhoneticUtterance [)] Comma2? &caprule [Kk] [Ii]
    Juncture? [Uu] Juncture? !(sp? Connective)))
```

```
<p>

<LI>the condition on Word that a Cmapua is not followed by another
   Cmapua

<LI>with mere whitespace between was used by [liu] quotation, but
   is now redundant,

<LI>because I have required that [liu] quotations be closed with
   explicit pauses in all cases.

<p>

Word <- (NAMEWORD/Cmapua/Complex/CcvNoY)

<p>

<LI>it is an odd point that all borrowings parse as complexes -- so
    when I parsed all the words the first time they all

<LI>parsed as complexes. A borrowing is a complex consisting of a
   single final borrowing djifoa!

<LI>I did redesign this so that borrowings are parsed as borrowings
   . (This is the class

<LI>I used to parse the dictionary).

<p>

<LI>Yes, CVC djifoa do get parsed as names in the dictionary, so
   the CVC case here is redundant. I actually

<LI>think that only the CCV djifoa actually get parsed as such.

<p>

SingleWord <- (((Borrowing !.)/(Complex !.)/(Word !.)/(PRENAME !.)/
   CcvNoY) !.)
```

```
<p>

<LI>name word appearing initially without leading spaces is
    important, because one type of NameWord includes a leading comma
    .

<H3>The full phonetic utterance classes</H3>

<p>

PhoneticUtterance1 <- (NAMEWORD/(sp? LiQuote)/(sp? NAMEWORD)/(sp?
    AlienWord)/(sp? Cmapua)/(sp? '--')/(sp? '...')/(sp? Borrowing !
    wy)/(sp? Complex)/(sp? CcvNoY))+

<p>

PhoneticUtterance <- (PhoneticUtterance1/([,] sp)/Terminal)+

<p>

<H2>Interlude: Phonemes and Pauses</H2>

<H3>Consonants and vowel groups in cmapua</H3>

<LI>as noted above, !predstart stands in for the computationally
    disastrous &Cmapua

<p>

Badstress <- (Stress2 sp? &Co1 Predstart)

<p>

b <- (!Predstart [Bb])

<p>

c <- (!Predstart [Cc])

<p>
```

```
d <- (!Predstart [Dd])
<p>
f <- (!Predstart [Ff])
<p>
g <- (!Predstart [Gg])
<p>
h <- (!Predstart [Hh])
<p>
j <- (!Predstart [Jj])
<p>
k <- (!Predstart [Kk])
<p>
l <- (!Predstart [Ll])
<p>
m <- (!Predstart [Mm])
<p>
n <- (!Predstart [Nn])
<p>
p <- (!Predstart [Pp])
<p>
```

```
r <- (!Predstart [Rr])

<p>

s <- (!Predstart [Ss])

<p>

t <- (!Predstart [Tt])

<p>

v <- (!Predstart [Vv])

<p>

z <- (!Predstart [Zz])

<p>

<LI>the monosyllabic classes may be followed by one vowel

<LI>if they start a Cvv-V cmapua unit; the others may never

<LI>be followed by vowels. Classes ending in -b are

<LI>used in Cvv-V cmapua units.

<p>

<LI>the single vowel classes were moved before the class

<LI>connective in the phonetics section.

<p>

Vo3 <- (Juncture? Vo2 !Badstress)

<p>
```

```
aa <- ([Aa] Juncture? [Aa] !Badstress Juncture? !Vo1)
<p>
ae <- ([Aa] Juncture? [Ee] !Badstress Juncture? !Vo1)
<p>
ai <- ([Aa] [Ii] !Badstress Juncture? !Vo1)
<p>
ao <- ([Aa] [Oo] !Badstress Juncture? !Vo1)
<p>
ai2 <- ([Aa] [Ii] !Badstress Juncture? &(Vo2 Juncture? !Vo1))
<p>
ao2 <- ([Aa] [Oo] !Badstress Juncture? &(Vo2 Juncture? !Vo1))
<p>
au <- ([Aa] Juncture? [Uu] !Badstress Juncture? !Vo1)
<p>
ea <- ([Ee] Juncture? [Aa] !Badstress Juncture? !Vo1)
<p>
ee <- ([Ee] Juncture? [Ee] !Badstress Juncture? !Vo1)
<p>
ei <- ([Ee] [Ii] !Badstress Juncture? !Vo1)
<p>
```

```
ei2 <- ([Ee] [Ii] !Badstress Juncture? &(Vo2 Juncture? !Vo1))

<p>

eo <- ([Ee] Juncture? [Oo] !Badstress Juncture? !Vo1)

<p>

eu <- ([Ee] Juncture? [Uu] !Badstress Juncture? !Vo1)

<p>

ia <- ([Ii] Juncture? [Aa] !Badstress Juncture? !Vo1)

<p>

ie <- ([Ii] Juncture? [Ee] !Badstress Juncture? !Vo1)

<p>

ii <- ([Ii] Juncture? [Ii] !Badstress Juncture? !Vo1)

<p>

io <- ([Ii] Juncture? [Oo] !Badstress Juncture? !Vo1)

<p>

iu <- ([Ii] Juncture? [Uu] !Badstress Juncture? !Vo1)

<p>

ia2 <- ([Ii] Juncture? [Aa] !Badstress Juncture? &(Vo2 Juncture? !
   Vo1))

<p>

ie2 <- ([Ii] Juncture? [Ee] !Badstress Juncture? &(Vo2 Juncture? !
   Vo1))
```

```
<p>

ii2 <- ([Ii] Juncture? [Ii] !Badstress Juncture? &(Vo2 Juncture? !
   Vo1))

<p>

io2 <- ([Ii] Juncture? [Oo] !Badstress Juncture? &(Vo2 Juncture? !
   Vo1))

<p>

iu2 <- ([Ii] Juncture? [Uu] !Badstress Juncture? &(Vo2 Juncture? !
   Vo1))

<p>

oa <- ([Oo] Juncture? [Aa] !Badstress Juncture? !Vo1)

<p>

oe <- ([Oo] Juncture? [Ee] !Badstress Juncture? !Vo1)

<p>

oi <- ([Oo] [Ii] !Badstress Juncture? !Vo1)

<p>

oi2 <- ([Oo] [Ii] !Badstress Juncture? &(Vo2 Juncture? !Vo1))

<p>

oo <- ([Oo] Juncture? [Oo] !Badstress Juncture? !Vo1)

<p>

ou <- ([Oo] Juncture? [Uu] !Badstress Juncture? !Vo1)
```

```
<p>

ua <- ([Uu] Juncture? [Aa] !Badstress Juncture? !Vo1)

<p>

ue <- ([Uu] Juncture? [Ee] !Badstress Juncture? !Vo1)

<p>

ui <- ([Uu] Juncture? [Ii] !Badstress Juncture? !Vo1)

<p>

uo <- ([Uu] Juncture? [Oo] !Badstress Juncture? !Vo1)

<p>

uu <- ([Uu] Juncture? [Uu] !Badstress Juncture? !Vo1)

<p>

ua2 <- ([Uu] Juncture? [Aa] !Badstress Juncture? &(Vo2 Juncture? !
   Vo1))

<p>

ue2 <- ([Uu] Juncture? [Ee] !Badstress Juncture? &(Vo2 Juncture? !
   Vo1))

<p>

ui2 <- ([Uu] Juncture? [Ii] !Badstress Juncture? &(Vo2 Juncture? !
   Vo1))

<p>

uo2 <- ([Uu] Juncture? [Oo] !Badstress Juncture? &(Vo2 Juncture? !
   Vo1))
```

```
<p>

uu2 <- ([Uu] Juncture? [Uu] !Badstress Juncture? &(Vo2 Juncture? !
   Vo1))

<p>

<LI>adding the new IY and UY, which might see use some time.

<LI>they are mandatory monosyllables but do not take a possible
   additional

<LI>following vowel as the regular ones do. So far only used in [
   ziy].

<p>

iy <- ([Ii] [Yy] !Badstress Juncture? !Vo1)

<p>

uy <- ([Uu] [Yy] !Badstress Juncture? !Vo1)

<p>

<H3>The optional pause and commas</H3>

<p>

<LI>this is a pause not required by the phonetics. This is the only

<LI>sort of pause which could in principle carry semantic freight (
   the

<LI>pause/GU equivalence beloved of our Founder) but we have
   abandoned

<LI>this. There is one place, after initial no in an utterance,
   where
```

```
<LI>a pause can have effect on the parse (but not on the meaning, I
    believe,

<LI>unless a word break is involved).

<p>

<LI>this class should NEVER be used in a context which might follow

<LI>a name word. In previous versions, pauses after name words were
    included

<LI>in the name word; this is not the case here, so a PAUSE

<LI>after a name word would not be recognized as a mandatory pause.

<p>

<LI>in any event, as long as we stay away from pause/GU equivalence
   , this

<LI>is not a serious issue!

<p>

<LI>this class does do some work in the handling of issues
   surrounding the legacy

<LI>shape of APA connectives, concerning which the less said, the
   better.

<p>

OptPause <- ([,] sp !(Vo1/Connective) &caprule)

<p>

<LI>more punctuation

<p>
```

```
Comma <- ([,] sp &caprule)

<p>

Comma2 <- ([,]? sp &caprule)

<p>

<H2>Part II: Lexicography</H2>

<LI>In this section I develop the grammar of words in Loglan. I'll
   work by editing the original provisional PEG grammar.

<p>

<LI>I place the start of this section exactly here, just before two
    final items of

<LI>punctuation, because these items of punctuation look forward
   not only to lexicography

<LI>but to the full grammar!

<H3>Period and end of utterance</H3>

<p>

<LI>the end of utterance symbol [#] should be added in the
   phonetics

<LI>section as a species of terminal marker. Done. We do *not*
   actually

<LI>endorse use of this marker, but we can notionally support it
   and it is in

<LI>our sources.

<p>
```

```
End <- ((sp? ’#’ sp utterance)/(sp !.)/!.)

<p>

<LI>this rule allows terminal punctuation to be followed by an
   inverse vocative,

<LI>a frequent occurrence in Leith’s novel, and something which
   makes sense.

<p>

Period <- (([!.:;?] (&End/(sp &caprule))) (invvoc Period?)?)

<p>

<LI>Letters with y will be special cases

<LI>idea: allow IY and UY (always monosyllables) as vowel
   combinations in cmapua only.

<LI>done: Y has a name now. [yfi] is also added.

<p>

<H3>The cmapua word classes</H3>

<LI>the classes in this section after this point are the cmapua
   word classes of Loglan (if they begin with sp? or a word class).

<LI>I suppose the alien text classes are not really word classes,
   but they are lexicographic items, as it were.

<LI>Paradoxically, the PA and NI classes admit internal explicit
   pauses. So of course do predicate words!

<p>
```

```
<LI>Loglan does admit true multisyllable cmapua: there are words
   made of cmapua units which have joints between

<LI>units at which one cannot pause without breaking the word.
   Lojban, I am told, does not.

<p>

<LI>this version has the general feature that the quotation and
   alien text constructions are not hacked:

<LI>they are supported by the phonetic rules (as dire exceptions,
   of course) and the grammatical constructions

<LI>conform with the phonetic layer. Alien text and utterances
   quoted with [li]...[lu] can be enclosed in double quotes.

<LI>LI only supports full utterances, for the moment. All alien
   text constructors take the same class as argument:

<LI>the vocative and inverse vocative *require* quotes to avoid
   misreading ungrammatical expressions with typos

<LI>as correct (inverse) vocatives.

<p>

<H4>Letterals (first approximation) </H4>

<LI>the names [yfi], [ziy] for Y are supported. The Ceo names are
   left as they are. I decided that a second short series

<LI>of letteral pronouns is actually a reasonable use of short
   words, and the Ceio words are there for other uses.

<p>

TAI0 <- ((Vo1 Juncture? m a)/(Vo1 Juncture? f i)/(Vo1 Juncture? z i
   )/(!Predstart Co1 ai)/(!Predstart Co1 ei)/(!Predstart Co1 ai2 u)
   /(!Predstart Co1 ei2 u)/(!Predstart Co1 eo)/(z [Ii] Vo1 !
   Badstress Juncture? !Vo1 (m a)?))

<p>

<H4>Logical and causal connectives</H4>

<LI>a negative suffix used in various contexts. Always a suffix:
   its use as a prefix in tenses was a mistake in NB3 and I

<LI>think still supported in LIP. Ambiguities demonstrably followed
    from this usage (an example of how the demonstration

<LI>of non-ambiguity of 1989 Loglan was compromised by the opaque
   lexicography).

<p>

NOI0 <- (n oi)

<p>

<LI>the logical connectives. [A0] is the class of core logical
   connectives. [A] is the fully decorated logical connective with

<LI>possible nu- (always in nuno- or nuu) and no- prefixes,
   possible -noi suffix, and possible (problematic) PA suffix,
   closed

<LI>with -fi (our new proposal) or an explicit pause.

<p>

A0 <- (&Cmapua (a/e/o/u/(h a)/(n uu)))

<p>

A <- (sp? !Predstart !TAI0 (n [o])? A0 NOI0? !(sp PAWORD0 OptPause)
    !(PAWORD0 !OptPause [ ,]) (PAWORD0 ((f i)/&OptPause))?)

<p>
```

```
<LI>4/18 in connected sentpreds, fi must be used to close, not a
   pause.

<p>

<LI>A2 <- sp? !predstart !TAI0 (N [o])? A0 NOI? !(sp PANOPAUSES
   PAUSE) !(PANOPAUSES !PAUSE [ ,]) (PANOPAUSES (F i))?

<p>

<LI>A not closed with -fi or a pause

<p>

ANOFI <- (sp? (!Predstart !TAI0 ((n [o])? A0 NOI0? PAWORD0?)))

<p>

AONE <- A

<p>

<LI>versions of A with different binding strength

<p>

ACI <- (ANOFI c i)

<p>

AGE <- (ANOFI g e)

<p>

<LI>a tightly binding series of logical connectives used to link
   predicates

<LI>this also includes the fusion connective [ze] when used between
    predicates.
```

```
<p>

CA0 <- (((n o)? ((c a)/(c e)/(c o)/(c u)/(z e)/(c i h a)/(n u c u))
   ) NOI0?)

<p>

CA1 <- (CA0 !(sp PAWORD0 OptPause) !(PAWORD0 !OptPause [ ,]) (
   PAWORD0 ((f i)/&OptPause))?)

<p>

CANOFI1 <- (CA0 PAWORD0?)

<p>

CA <- (sp? CA1)

<p>

<LI>the fusion connective when used in arguments

<p>

ZE <- (sp? (z e))

<p>

<LI>sentence connectives. [I] is the class of utterance initiators
   (no logical definition).

<LI>the subsequent classes are inhabited by sentence logical
   connectives with various binding

<LI>strengths.

<p>
```

```
I <- (sp? !Predstart !TAI0 i !(sp PAWORD0 OptPause) !(PAWORD0 !
   OptPause [ ,]) (PAWORD0 ((f i)/&OptPause))?)

<p>

ICA <- (sp? i ((h a)/CA1))

<p>

ICI <- (sp? i CANOFI1? c i)

<p>

IGE <- (sp? i CANOFI1? g e)

<p>

<LI>forethought logical connectives

<p>

KA0 <- ((k a)/(k e)/(k o)/(k u)/(k i h a)/(n u k u))

<p>

<LI>causal and comparative modifiers

<p>

KOU0 <- ((k ou)/(m oi)/(r au)/(s oa)/(m ou)/(c iu))

<p>

<LI>negative and converse forms

<p>

KOU1 <- (((n u n o)/(n u)/(n o)) KOU0)

<p>
```

```
<LI>the full type of forethought connectives, adding the causal and
     comparative connectives

<p>

KA <- (sp? (KA0/((KOU1/KOU0) k i)) NOI0?)

<p>

<LI>the last component of the KA...KI... structure of forethought
   connections

<p>

KI <- (sp? (k i) NOI0?)

<p>

<LI>causal and comparative modifiers which are *not* forethought
   connectives

KOU2 <- (KOU1 !KI)

<p>

<H4>Quantity words</H4>

<p>

<LI>a test used to at least partially enforce the penultimate
   stress rule on quantifier predicates

<p>

BadNIStress <- ((Co1 Vo2 Vo2? Stress (m a)? (m oa)? NI RA0)/(Co1
   Vo2 Stress Vo2 (m a)? (m oa)? NI RA0))

<p>
```

```
<LI>root quantity words, including the numerals (removing [kue] for
     another use)

<p>

NIO <- (!BadNIStress ((k ua)/(g ie)/(g iu)/(h ie)/(h iu)/(n ea)/(n
    io)/(p ea)/(p io)/(s uu)/(s ua)/(t ia)/(z oa)/(z oo)/(h o)/(n i)
    /(n e)/(t o)/(t e)/(f o)/(f e)/(v o)/(v e)/(p i)/(r e)/(r u)/(s
    e)/(s o)/(h i)))

<p>

<LI>the class of SA roots, which modify quantifiers

<p>

SA0 <- (!BadNIStress ((s a)/(s i)/(s u)/(ie (Comma2? !ie SA0)?))
    NOI0?)

<p>

<LI>the family of quantifiers which double as suffixes for the
    quantifier predicates

<LI>this class perhaps should also include some other quantifier
    words. [re] for example ought to be handled in the same way as [
    ra,ri,ro].

<LI>No action here, just a remark. Added [bao], which forms lambda
    abstractions (relations) not statements, to be used in lepu
    clauses.

<p>

RA0 <- (!BadNIStress ((r a)/(r i)/(r e)/(r u)/(r o)/(b ao)))

<p>

<LI>re and ru added to class RA 5/11/18
```

```
<p>

<LI>quantifier units consisting of a NI or RA root with [ma] 00 or
    [moa] 000 appended; to [moa] one can further

<LI>append a digit to iterate [moa]: [fomoate] is four billion, for
     example. [rimoa], a few thousand.

<p>

<LI>a NI1 or RA1 may be followed by a pause before another NI word
    other than a numerical predicate;

<LI>one is allowed to breathe in the middle of long numerals. I
    question whether the pause

<LI>provision makes sense in RA1.

<p>

NI1 <- ((NI0 (!BadNIStress m a)? (!BadNIStress m oa NI0*)?) (Comma2
     !(NI RA0) &NI)?)

<p>

RA1 <- ((RA0 (!BadNIStress m a)? (!BadNIStress m oa NI0*)?) (Comma2
     !(NI RA0) &NI)?)

<p>

<LI>a composite NI word, optional SA prefix before a sequence of NI
     words or a RA word,

<LI>or a single SA word [which will modify a default quantifier not
     expressed],

<LI>possibly negated, connected with CA0 roots to other such
    constructs.

<p>
```

```
NI2 <- (((SA0? (NI1+/RA1))/SA0) NOI0? (CA0 ((SA0? (NI1+/RA1))/SA0)
   NOI0?)*)
```

<p>

<LI>a full NI word with an acronymic dimension (starting with [mue], ending with a pause) or [cu] appended. I need to look up [cu]

<LI>and figure out its semantics. An arbitrary name word may now be used as a dimension, as well.

<p>

```
NI <- (sp? (p i)? NI2 ((&(m ue) ACRONYM (Comma/&End/&Period) !(c u)
   )/(Comma2? m ue Comma2? PRENAME !(c u)))? (c u)?)
```

<p>

<LI>mex is now identical with NI, but it’s in use in later rules.

<p>

```
mex <- (sp? NI)
```

<p>

<H4> The overused CI</H4>

<LI>a word used for various tightly binding constructions: a sort of verbal hyphen.

<LI>also a name marker, which means phonetic care is needed (pause after constructions with [ci]).

<p>

```
CI <- (sp? (c i))
```

<H4>Acronyms</H4>

<LI>Acronyms, which are names (not predicates as in 1989 Loglan) or dimensions (in NI above).

<LI>units in acronym are TAI0 letterals, zV short forms for vowels, the dummy unit [mue], and NI1

<LI>quantity units. NI1 quantity units may not be initial. [mue] units may be preceded by pauses.

<LI>An acronym has at least two units.

<p>

<LI>it is worth noting that acronyms, once viewed as names, could be entirely suppressed as a feature of the

<LI>grammar by really making them names (terminate them with -n). I suppose a similar approach would work

<LI>for dimensions, allowing any name word to serve as a dimension. [mue] would be a name marker for use

<LI>with dimensions in this case. [temuedain], three dollars. Now supported.

<p>

```
ACRONYM <- (sp? &caprule ((m ue)/TAI0/(z Vo2 !Vo2)) ((Comma &
   ACRONYM m ue)/NI1/TAI0/(z Vo2 (!Vo2/(z &Vo2))))+)
```

<p>

<H4>Letterals and other pronouns</H4>

<LI>the full class of letterals, including the [gao] construction whose details I should look at.

<p>

```
TAI <- (sp? (TAI0/((g ao) !Vo2 sp? (PRENAME/Predicate/CmapuaUnit)))
   )

<p>

<LI>atomic non-letteral pronouns.

<p>

#4/15/2019 reserved [koo] for a Lojban style imperative pronoun,
   though not officially adopting it. Also adding [dao] for a
   default, don't care argument, another Lojban feature.

<p>

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)/(t uu)/(s uo)/(h u)/(b a)/(b e)/(b o)/(
   b u)/(d a)/(d e)/(d i)/(d o)/(d u)/(m i)/(t u)/(m u)/(t i)/(t a)
   /(m o)/(k oo)/(d ao))

<p>

<LI>letterals (not including [gao] constructions and atomic
   pronouns optionally suffixed with a digit. One should pause
   after the

<LI>suffixed forms, because [ci] is a name marker.

<p>

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

<p>

<LI>general pronoun words.

<p>

DA <- (sp? DA1)
```

<p>

<H4>Tenses, locatives and modals</H4>

<LI>roots for PA words: tense and location words, prepositions building relative modifiers. All can optionally be negated with -noi. They may also be quantified. They may also be closed with ZI class affixes. PA cores.

<p>

<LI> put the long list of atomic PA words in a wrapper

<p>

```
PA00 <- ((g ia)/(g ua)/(p au)/(v au)/(f au)/(p ia)/(p ua)/(n ia)/(n
     ua)/(b iu)/(f ea)/(f ia)/(f ua)/(v ia)/(v ii)/(v iu)/(c oi)/(d
    au)/(d ii)/(d uo)/(f oi)/(f ui)/(g au)/(h ea)/(k au)/(k ii)/(k
    ui)/(l ia)/(l ui)/(m ia)/(n ui)/(p eu)/(r oi)/(r ui)/(s ea)/(s
    io)/(t ie)/(v ie)/(v a)/(v i)/(v u)/(p a)/(n a)/(f a)/(v a)/(
    KOU0 !(n oi) !KI))
```

<p>

```
PA0 <- (NI2? (n u !KOU0)? PA00 (n oi)? ZI?)
```

<p>

<LI>the form used for actual prepositions and suffixes to A words, with minimal pauses allowed.

<LI>these are built by concatenating KOU2 and PA0 units, then linking these with CA0 roots (which can take

<LI>no- prefixes and -noi suffixes, and next to which one *can* pause), optionally suffixed with a class ZI suffix.

<p>

```
PAWORD0 <- ((KOU2/PA0)+ ((Comma2? CA0 Comma2?) (KOU2/PA0)+)*)
```

```
<p>

<LI>prepositional words

<p>

PAWORD <- (sp? PAWORD0)

<p>

<LI>class PA can appear as tense markers or as relative modifiers
   without arguments; here pauses

<LI>are allowed not only next to CA0 units but between KOU2/PA
   units. Like NI words, PA

<LI>words are a class of arbitrary length constructions, and we
   think breaths within them

<LI>(especially complex ones) are natural.

<p>

PAPHRASE0 <- (((KOU2/PA0)+ (((Comma2? CA0 Comma2?)/(Comma2 !mod1a))
    (KOU2/PA0)+)*) !modifier)

<p>

PAPHRASE <- (sp? PAPHRASE0)

<p>

GA <- (sp? (g a))

<p>

<LI>the class of tense markers which can appear before predicates.

<p>
```

```
TENSE <- (PAPHRASE/GA)

<p>

<LI>suffixes which indicate extent or remoteness/proximity of the
   action of prepositions.

<p>

ZI <- ((z i)/(z a)/(z u))

<p>

<H4> Articles and other descriptors</H4>

<LI>the primitive description building "articles". These include [
   la] which requires special

<LI>care in its use because it is a name marker.

<p>

LE <- (sp? ((l ea)/(l eu)/(l oe)/(l ee)/(l aa)/(l e)/(l o)/(l a)))

<p>

<LI>articles which can be used with abstract descriptions: these
   include some quantity words.

<LI>this means that some abstract descriptions are semantically
   indefinites: I wonder if this

<LI>could be improved by having a separate abstract indefinite
   construction.

<p>

LEFORPO <- (sp? ((l e)/(l o)/NI2))
```

```
<p>

<LI>the numerical/quantity article.

<p>

LIO <- (sp? (l io))

<p>

<LI>structure words for the ordered and unordered list
   constructions.

<p>

LAU <- (sp? (l au))

<p>

LOU <- (sp? (l ou))

<p>

LUA <- (sp? (l ua))

<p>

LUO <- (sp? (l uo))

<p>

ZEIA <- (sp? z ei2 a)

<p>

ZEIO <- (sp? z ei2 o)

<p>

<LI>initial and final words for quoting Loglan utterances.
```

```
<p>

LIWORD <- (l i)

<p>

LUWORD <- (l u)

<p>

<H4>Quotations and other alien text constructions</H4>

<LI>quoting Loglan utterances, with or without explicit double
   quotes (if they appear, they must

<LI>appear on both sides). The previous version allowed quotation
   of names; likely this should

<LI>be restored.

<p>

liquote <- ((sp? LIWORD Comma2? utterance0 Comma2? LUWORD)/(sp?
   LIWORD Comma2? [\"] utterance0 [\"] Comma2? LUWORD))

<p>

<LI>the foreign name construction. This is an alien text
   construction

<p>

LAO <- (sp? &(LAOalien Juncture?) AlienWord)

<p>

<LI>the strong quotation construction. This is an alien text
   construction.
```

```
<p>

LIE <- (sp? &(LIEalien Juncture?) AlienWord)

<p>

LIOALIEN <- (sp? &(LIOalien Juncture?) AlienWord)

<p>

<LI>I am not sure this class is used at all.

<p>

Lw <- Cmapua

<p>

<LI>articles for quotation of words

<p>

LIUO <- ((l iu)/(n iu))

<p>

<LI>this now imposes the condition that an explicit comma pause (or
     terminal punctuation, or end) must appear at the end of the

<LI>Word or PreName quoted with [liu]. This seems like a good idea,
     anyway.

<p>

<LI>this class appeals to the phonetics. Words and PreNames can be
   quoted. The ability to quote names

<LI>here may remove the need to quote them with [li]...[lu]. Of
   course, some Words are in fact phrases rather
```

```
<LI>than single words: we will see whether the privileges afforded
    are used. The final clause allows

<LI>use of letterals as actual names of letters.

<p>

<LI>added [niu]: didn’t make it a name marker.

<p>

LNIU <- (([Ll]/[Nn]) [iI] Juncture? [Uu])

<p>

<p>

LIU <- ((sp? LNIU Juncture? !Vo1 Comma2? (PRENAME/Word) &(Comma/
    Terminal/End))/(sp? (l ii TAI)))

<p>

<LI>the construction of foreign and onomatopoeic predicates. These
    are alien text constructions.

<p>

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

<p>

<H4>Assorted left and right closers</H4>

<LI>left marker in a predicate metaphor construction

<p>

CUI <- (sp? (c ui))
```

```
<p>

<LI>other uses of GA

<p>

GATWO <- (sp? (g a))

<p>

<LI>ge/geu act as "parentheses" to make an atomic predicate from a complex metaphorically

<LI>and logically connected predicates; [ge] has other left marking uses.

<p>

GE <- (sp? (g e))

<p>

GEU <- (sp? ((c ue)/(g eu)))

<p>

<LI>final marker of a list of head terms

<p>

GI <- (sp? ((g i)/(g oi)))

<p>

<LI>used to move a normally prefixed metaphorical modifier after what it modifies.

<p>

GO <- (sp? (g o))
```

<p>

<LI>marker for second and subsequent arguments before the predicate ; NEW

<p>

GIO <- (sp? (g io))

<p>

<LI>the generic right marker of many constructions.

<p>

GU <- (sp? (g u))

<LI>various flavors of right markers.

<LI>It should be noted that at one point I executed a program of simplifying these to

<LI>reduce the likelihood that multiple [gu]'s would ever be needed to close an utterance.

<LI>first of all, I made the closures leaner, moving them out of the classes closed

<LI>to their clients so that they generally can be used only when needed.

<LI>Notably, the grammar of [guu] is quite different. Second,

<LI>I introduced some new flavors of right marker. All can be realized with [gu],

<LI>but if one knows the right flavor one can close the right structure with a single

```
<LI>right closure.

<LI>right markers of subordinate clauses (argument modifiers).

<LI>[gui] closes a different class than in the trial.85 grammar,
   with

<LI>similar but on the whole better results.

<p>

GUIZA <- (sp? (g ui) (z a))

<p>

GUIZI <- (sp? (g ui) (z i))

<p>

GUIZU <- (sp? (g ui) (z u))

<p>

GUI <- (!GUIZA !GUIZI !GUIZU (sp? (g ui)))

<p>

<LI>right markers of abstract predicates and descriptions.

<LI>probably the forms with z are to be preferred (and the other

<LI>two are not needed) but I preserve all five classes for now.

<p>

GUO <- (sp? (g uo))

<p>

GUOA <- (sp? ((g uo2 a)/(g uo z a)))
```

```
<p>

GUOE <- (sp? (g uo2 e))

<p>

GUOI <- (sp? ((g uo2 i)/(g uo z i)))

<p>

GUOO <- (sp? (g uo2 o))

<p>

GUOU <- (sp? ((g uo2 u)/(g uo z u)))

<p>

<LI>right marker used to close term (argument/predicate modifier) lists.

<LI>it is important to note that in our grammar GUU is not a component of

<LI>the class termset, nor is it a null termset: it appears in other classes

<LI>which include termsets as an option to close them. The effects are similar

<LI>to those in the trial.85 grammar, but there is less of a danger that

<LI>extra unexpected closures will be needed.

<p>

GUU <- (sp? (g uu))
```

```
<p>

<LI>a new closure for arguments in various contexts

<p>

GUUA <- (sp? (g uu2 a))

<p>

<LI>a new closure for sentences. In particular, it

<LI>may have real use in closing up the scope of a list of

<LI>fronted terms before a series of logically connected sentences.

<p>

GIUO <- (sp? (g iu2 o))

<p>

<LI>right marker used to close arguments tightly linked with JE/JUE
    .

<p>

GUE <- (sp? (g ue))

<p>

<LI>a new closure for descpreds

<p>

GUEA <- (sp? (g ue2 a))

<p>

<H4>Miscellaneous clause constructors</H4>
```

```
<LI>used to build tightly linked term lists.

<p>

JE <- (sp? (j e))

<p>

JUE <- (sp? (j ue))

<p>

<LI>used to build subordinate clauses (argument modifiers).

<p>

JIZA <- (sp? ((j ie)/(j ae)/(p e)/(j i)/(j a)) (z a))

<p>

JIOZA <- (sp? ((j io)/(j ao)) (z a))

<p>

JIZI <- (sp? ((j ie)/(j ae)/(p e)/(j i)/(j a)) (z i))

<p>

JIOZI <- (sp? ((j io)/(j ao)) (z i))

<p>

JIZU <- (sp? ((j ie)/(j ae)/(p e)/(j i)/(j a)) (z u))

<p>

JIOZU <- (sp? ((j io)/(j ao)) (z u))

<p>
```

```
JI <- (!JIZA !JIZI !JIZU (sp? ((j ie)/(j ae)/(p e)/(j i)/(j a))))

<p>

NUJI <- (sp? n u !sp JI)

<p>

JIO <- (!JIOZA !JIOZI !JIOZU (sp? ((j io)/(j ao))))

<p>

<H4>Case tags, semantic and positional</H4>

<LI>case tags, both numerical position tags and the optional
   semantic case tags.

<p>

DIO <- ((sp? ((b eu)/(c au)/(d io)/(f oa)/(k ao)/(j ui)/(n eu)/(p
   ou)/(g oa)/(s au)/(v eu)/(z ua)/(z ue)/(z ui)/(z uo)/(z uu))) ((
   c i ![ ] NIO)/ZI)?)

<p>

<LI>markers of indirect reference. Originally these had the same
   grammar as case tags,

<LI>but they are now different.

<p>

LAE <- (sp? ((l ae)/(l ue)))

<p>

<H4> The predicate constructor me</H4>
```

```
<LI>[me] turns arguments into predicates, [meu] closes this
   construction.

<p>

ME <- (sp? ((m ea)/(m e)))

<p>

MEU <- (sp? m eu)

<p>

<H4> Reflexive and conversion operators</H4>

<p>

<LI>reflexive and conversion operators: first the root forms, then
   those with

<LI>optional numerical suffixes.

<p>

NUO <- ((n uo)/(f uo)/(j uo)/(n u)/(f u)/(j u)/(k ue))

<p>

NU <- (sp? ((((n u)/(n uo)/(k ue)) !(sp (NIO/RAO)) (NIO/RAO)?)/NUO)
   + freemod?)

<p>

<H4>Abstract predicate constructors</H4>

<LI>I do *not* think

<LI>that [poia] will really be confused with [po ia], particularly
```

<LI>since we do require an explicit pause before [ia] in the latter case,

<LI>but I record this concern: the forms with z might be preferable .

<p>

#constructions from sentences

<p>

```
PO1 <- (sp? ((p o)/(p u)/(z o)))
```

<p>

```
PO1A <- (sp? ((p oi2 a)/(p ui2 a)/(z oi2 a)/(p o z a)/(p u z a)/(z o z a)))
```

<p>

```
PO1E <- (sp? ((p oi2 e)/(p ui2 e)/(z oi2 e)))
```

<p>

```
PO1I <- (sp? ((p oi2 i)/(p ui2 i)/(z oi2 i)/(p o z i)/(p u z i)/(z o z i)))
```

<p>

```
PO1O <- (sp? ((p oi2 o)/(p ui2 o)/(z oi2 o)))
```

<p>

```
PO1U <- (sp? ((p oi2 u)/(p ui2 u)/(z oi2 u)/(p o z u)/(p u z u)/(z o z u)))
```

<p>

<LI>abstract predicate constructor from simple predicates

```
<p>

POSHORT1 <- (sp? ((p oi)/(p ui)/(z oi)))

<p>

<LI>word forms associated with the above abstract predicate root
   forms

<p>

PO <- (sp? PO1)

<p>

POA <- (sp? PO1A)

<p>

POE <- (sp? PO1E)

<p>

POI <- (sp? PO1E)

<p>

POO <- (sp? PO1O)

<p>

POU <- (sp? PO1U)

<p>

POSHORT <- (sp? POSHORT1)

<p>
```

```
<H4> register markers </H4>

<p>

DIE <- (sp? ((d ie)/(f ie)/(k ae)/(n ue)/(r ie)))

<p>

<H4> freemods and freemod builders </H4>

<p>

<LI>vocative forms: I still have the words of social lubrication as

<LI>vocative markers.

<p>

HOI <- (sp? ((h oi)/(l oi)/(l oa)/(s ia)/(s ie)/(s iu)))

<p>

<LI>the verbal scare quote. The quantifier suffix indicates how
   many preceding words are affected;

<LI>this is an odd mechanism.

<p>

JO <- (sp? (NIO/RAO/SAO)? (j o))

<p>

<LI>markers for forming parenthetical utterances as free modifiers.

<p>

KIE <- (sp? (k ie))

<p>
```

```
KIU <- (sp? (k iu))

<p>

KIE2 <- (sp? k ie Comma2? [(])

<p>

KIU2 <- (sp? [)] Comma2? k iu)

<p>

<LI>marker for forming smilies.

<p>

SOI <- (sp? (s oi))

<p>

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

<p>

UI0 <- (!Predstart ((!([Ii] Juncture? [Ee]) Vv Juncture?)/(b ea)/(b
    uo)/(c ea)/(c ia)/(c oa)/(d ou)/(f ae)/(f ao)/(f eu)/(g ea)/(k
   uo)/(k uu)/(r ea)/(n ao)/(n ie)/(p ae)/(p iu)/(s aa)/(s ui)/(t
   aa)/(t oe)/(v oi)/(z ou)/(l oi)/(l oa)/(s ia)/(s ii)/(t oe)/(s
   iu)/(c ao)/(c eu)/(s ie)/(s eu)/(s ie2 i)))

<p>

<LI>negative forms of the attitudinals. The ones with [no] before
   the two vowel forms are a phonetic exception. The others

<LI>should also be (though they present no pronunciation problem)
   so that they are resolved as single words.
```

```
<p>

<LI> There is a strong reason for [o] here.

<p>

NOUI <- ((sp? UI0 NOI0)/(sp? n [o] Juncture? Comma? sp? UI0))

<p>

<LI>all attitudinals (adding the discursives nefi, tofi... etc)

<LI>there is a technical problem with mixing UI0 roots of VV and
   CVV shapes.

<p>

UI <- (sp? (UI0+/(NI f i)))

<p>

<LI>the inverse vocative marker

<p>

HUE <- (sp? (h ue))

<p>

<H4>Negation</H4>

<LI>occurrences of [no] as a word rather than an affix.

<p>

NOWORD <- (sp? !KOU1 !NOUI (n o) !(Comma2? z ao Comma2? Predicate)
   !(sp? KOU0) !(sp? (JIO/JI/JIZA/JIOZA/JIZI/JIOZI/JIZU/JIOZU)))

<p>
```

```
<H3> The large word classes (names and predicates)</H3>

<LI>Names, acronyms and PreNames from above.

<p>

ACRONYMICNAME <- (ACRONYM &(Comma/Period/End))

<p>

DJAN <- (PRENAME/ACRONYMICNAME)

<p>

<LI>predicate words which are phonetically cmapua

<p>

<LI>"identity predicates". Converses are provided as a new proposal
    .

<p>

BI <- (sp? (n u)? ((b ia)/(b ie)/(c ie)/(c io)/(b ia)/(b i)/(b ii))
    )

<p>

<LI>interrogative and pronoun predicates

<p>

LWPREDA0 <- ((h e)/(d ua)/(d ui)/(b ua)/(b ui))

<p>

<LI>here I should reinstall the [zao] proposal.

<p>
```

```
<LI>the predicate words defined above in the phonetics section

<p>

Predicate <- ((CmapuaUnit Comma2? z ao Comma2?)* Complex (Comma2? z
     ao Comma2? Predicate)?)

<p>

<LI>predicate words, other than the "identity predicates" of class
    [BI]

<LI>these include the numerical predicates (NI RA), also cmapua
    phonetically.

<p>

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

<p>

PREDA <- (sp? &caprule (Predicate/LWPREDA0/(![ ] NI RA0)))

<p>

<H2>Part 3: The Grammar Proper</H2>

<H3>Right markers turned into classes</H3>

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

<p>

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

<p>

guoi <- (OptPause? (GUOI/GU) freemod?)
```

```
<p>

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

<p>

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

<p>

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

<p>

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

<p>

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

<p>

guizu <- (OptPause? (GUIZU/GU) freemod?)

<p>

gui <- (OptPause? (GUI/GU) freemod?)

<p>

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

<p>

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

<p>

guu <- (OptPause? (GUU/GU) freemod?)
```

```
<p>

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

<p>

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

<p>

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

<p>

geu <- GEU

<p>

<LI>Here note the absence of pause/GU equivalence.

<p>

gap <- (OptPause? GU freemod?)

<p>

<H3>The vocative and inverse vocative</H3>

<LI>this is the vocative construction. It can appear early because
   all of its components are marked.

<p>

<LI>the intention is to indicate who is being addressed. This can
   be handled via a name, a descriptive argument, a predicate or an

<LI>alien text name (the last must be quoted). The complexities of
   these grammatical constructions can be deferred until they are
```

```
<LI>introduced.

<p>

<LI>HOI0 <- sp? [Hh] [Oo] [Ii] juncture?

<p>

<LI>restore words of social lubrication as vocative markers but not
     as name markers: [loi, Djan]

<p>

<LI>I do not allow a freemod to intervene between a vocative marker
     and the associated

<LI>utterance, to avoid unintended grabbing of subjects by the
   words of social lubrication when they are used

<LI>as vocative markers. This lets [Loi, Djan] and [Loi hoi Djan]
   be equivalent. The comma needed in the

<LI>first because the social lubrication words are in this version
   not name markers.

<p>

HOI0 <- ((sp? (([Hh] oi)/([Ll] oi)/([Ll] oa)/([Ss] ia)/([Ss] ie)/([
   Ss] iu))) Juncture? !Vo1)

<p>

voc <- ((HOI0 Comma2? name)/(HOI Comma2? descpred guea? namesuffix
   ?)/(HOI Comma2? argument1 guua?)/(sp? &([Hh] [Oo] [Ii] Juncture
   ?) AlienWord))

<p>

<LI>this is the inverse vocative. It can appear early because all
   of its components are marked.
```

```
<p>

<LI>the intention is to indicate who is speaking. The range of ways
    this can be handled is similar to the range of ways it can be

<LI>handled for the vocative; there is the further option of a
   sentence (the [statement] class) and there is a strong closure
   option

<LI>for the case where an argument is used (to avoid it
   inadvertantly expanding to a sentence).

<p>

HUE0 <- (sp? &caprule [Hh] [Uu] Juncture? [Ee] Juncture? !Vo1)

<p>

invvoc <- ((HUE0 Comma2? name)/(HUE freemod? descpred guea?
   namesuffix?)/(HUE freemod? statement giuo?)/(HUE freemod?
   argument1 guu?)/(sp? &([Hh] [Uu] Juncture? [Ee] Juncture?)
   AlienWord))

<p>

<H3>Free modifiers</H3>

<LI>this is the class of free modifiers. Most of its components are
    head marked (those that aren't appear just above),

<LI>and it is useful for it to appear early because these things
   appear everywhere in subsequent constructions. A free modifier,

<LI>of whatever sort, is a freely insertable gadget which modifies
   the immediately preceding construction, or the entire utterance

<LI>if it is initial.

<p>
```

<LI>NOUI is a negated attitudinal word. UI1 is an attitudinal word: these express an emotional attitude toward the

<LI>assertion (noting that EI marks questions (yes or no answer expected) and SEU marks utterances as answers).

<p>

<LI>SOI creates smilies in a general sense: [soi crano] indicates that the listener should imagine the speaker smiling;

<LI>similarly for other predicates.

<p>

<LI>DIE and NO DIE are register markers, communicating the social attitude of the speaker toward the one addressed: [die] for

<LI>example is "dear"

<p>

<LI>KIE...KIU constructs a full parenthetical utterance as a comment, which can be enclosed in actual parentheses inside

<LI>the marker words.

<p>

<LI>JO is a scare quote device.

<p>

<LI>deletion of a previous word or wordlike unit (or more than one) using K IA

<p>

```
kiamod <- (Comma2? !(!PRENAME !Predstart k ia) ((PRENAME/LIU/
   AlienWord/(Cmapua (sp? !(k ia) !PRENAME !Predstart Cmapua)*)/
   Word) kiamod* Comma2? !PRENAME !Predstart k ia) Comma2?)
```

<p>

<LI>the comma is a freemod with no semantic content: this is a device for discarding phonetically required pauses

<LI>and the speaker’s optional pauses alike. The pause before a non -pause marked prename is part of the NameWord and so

<LI>is excluded. Ellipses and dashes are fancy pauses supported as freemods.

<p>

```
freemod <- ((kiamod/NOUI/(SOI freemod? descpred guea?)/DIE/(NOWORD
   DIE)/(KIE Comma? utterance0 Comma? KIU)/(KIE2 Comma? utterance0
   Comma? KIU2)/invvoc/voc/(Comma !(!FalseMarked PRENAME))/JO/UI/(
   sp? ’...’ (sp? &Letter)?)/(sp? ’--’ (sp? &Letter)?)) freemod?)
```

<p>

<H3>Tightly bound arguments and lists thereof</H3>

<LI>the classes juelink to linkargs describe very tightly bound arguments which can be firmly attached to predicates in

<LI>the context of metaphorical modifications and the use of predicates in descriptive arguments.

<p>

<LI>note that we allow predicate modifiers (prepositional phrases) to be bound with [je/jue] which is not

<LI>allowed in 1989 Loglan, but which we believe is supported in Lojban.

```
<p>

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

<p>

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

<p>

links <- ((links1/(KA freemod? links freemod? KI freemod? links1))
    (freemod? AONE freemod? links1)*)

<p>

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

<p>

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

<p>

linkargs <- ((linkargs1/(KA freemod? linkargs freemod? KI freemod?
    linkargs1)) (freemod? AONE freemod? linkargs1)*)

<p>

<H3>Abstract argument constructions</H3>

<LI>class abstractpred supports the construction of event, property
    , and quantity predicates from sentences. These are

<LI>closable with [guo] if introduced with [po,pu,zo] and closable
    with suffixed variants of [guo] if introduced with suffixed

<LI>variants of [po,pu,zo] (a NEW idea but it is clear that closure
     of these predicates (and of the more commonly

<LI>used associated descriptions) is an important issue).
```

```
<p>

<LI> using sentenceclone so that subject free sentences will not be
     marked as imperative

<p>

abstractpred <- ((POA freemod? uttAxclone guoa?)/(POA freemod?
   sentenceclone guoa?)/(POE freemod? uttAxclone guoe?)/(POE
   freemod? sentenceclone guoe?)/(POI freemod? uttAxclone guoi?)/(
   POI freemod? sentenceclone guoi?)/(POO freemod? uttAxclone guoo
   ?)/(POO freemod? sentenceclone guoo?)/(POU freemod? uttAxclone
   guou?)/(POU freemod? sentenceclone guou?)/(PO freemod?
   uttAxclone guo?)/(PO freemod? sentenceclone guo?))

<H3>Atomic predicates (predunit)</H3>

<LI>predunit1 describes the truly atomic forms of predicate.

<p>

<LI>PREDA is the class of predicate words (the phonetic predicate
   words along with the special phonetic cmapua which are
   predicates, listed

<LI>above under the PREDA rule. NU PREDA handles permutations and
   identifications of arguments of PREDAs.

<p>

<LI>SUE contains the alien text constructions with [sao] and [sue],
    semantically quite different but syntactically handled

<LI>in the same way.

<p>

<LI>[ge]...[geu/cue] (the closing optional) can parenthesize a
   fairly complex predicate phrase and turn it into an atomic form.
```

These

<LI>forms can have conversion or reflexive operators (NU) applied. I should look into why the class handled in the conversion case

<LI>is different. An important use of this is in metaphor constructions, but it has other potential uses.

<p>

<LI>abstractpred is the class of abstraction predicates just introduced above. These are treated as atomic in this grammar: it should

<LI>be noted that their privileges in the trial.85 grammar are (absurdly) limited.

<p>

<LI>[me]...[meu] (the closing optional, but important to have available) forms predicates from arguments, the predicate being true of the

<LI>objects to which the argument refers. [Ti me le mrenu] : this is one of the men we are talking about.

<p>

```
predunit1 <- ((SUE/(NU freemod? GE freemod? despredE (freemod? geu
   Comma?)?)/(NU freemod? PREDA)/(Comma? GE freemod? descpred (
   freemod? geu Comma?)?)/abstractpred/(ME freemod? argument1 meu?)
   /PREDA) freemod?)
```

<p>

<LI>[no] binds very tightly to predunit1: a possibly multiply negated predunit1 (or an unadorned predunit1) is a predunit2.

<p>

```
predunit2 <- ((NOWORD freemod?)* predunit1)

<p>

<LI>an instance of NO2 is one not absorbed by a predunit. Example:
   [Da no kukra prano] X is a slow (not-fast) runner vs

<LI>[Da no ga kukra prano] (X is not a fast runner, and in fact may
    not run at all).

<p>

neg2 <- (!predunit2 NOWORD)

<p>

<LI>a predunit3 is a predunit2 with tightly attached arguments.

<p>

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

<p>

<LI>a predunit is a predunit3 or a predunit3 converted by the short
   -scope abstraction operators

<LI>[poi/pui/zoi] to an abstraction predicate. This is the kind of
   predicate which can appear as

<LI>a component in a serial name.

<p>

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

<p>

<LI>a further "atomic" (because tightly packaged) form is a
   forethought connected pair
```

<LI>of predicates (this being the full predicate class defined at the end of the process)

<LI>possibly closed with [guu], possibly multiply negated as well.

<p>

<LI>the closure with guu eliminated the historic rule against kekked heads of metaphors.

<p>

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

<p>

<H4>The construction of metaphors</H4>

<LI>there follows the construction of metaphorically modified predicates,

<LI>along with tightly logically linked predicates.

<p>

<LI>CI and simple juxtaposition of predicates both represent modification of the second

<LI>predicate by the first. We impose no semantic conditions on this modification,

<LI>except in the case of modification by predicates logically linked with CA,

<LI>which do distribute logically in the expected way both as modifiers and as modified.

<LI>We do not regard [preda1 preda2] as necessarily implying preda2 : we do regard

<LI>it as having the same place structure as preda2. It is very often but not always

<LI>a qualification or kind of preda2; in any case it is a relation analogous to preda2.

<p>

<LI>modification with CI binds most tightly.

<p>

<LI>we eliminated the distinction between the series of sentence and description

<LI>predicate preliminary classes: there seems to be no need for it even in the

<LI>trial.85 grammar.

<p>

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

<p>

<LI>this is logical connection of predicates with the tightly binding CA

<LI>series of logical connectives. CUI can be used to expand the scope of

<LI>a CA connective over a metaphor on the left. [ge]...[geu] is used to expand

<LI>scope on the right (and could also be used on the left, it should be noted).

<LI>descpredC is an internal of despredB assisting the function of CUI.

<LI>the !PREDA in front of CUI is probably not needed.

<p>

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

<p>

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

<p>

<LI>tight logical linkage of despredB's

<p>

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

<p>

<LI>chain of modifications of despredD's (grouping to the left)

<p>

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

<p>

<LI>the GO construction allows inverse modification: [preda1 GO preda2] is [preda2 preda1] as it were.

<LI> there are profound effects on grouping.

```
<p>

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

<p>

<LI>this version which appears in sentence predicates as opposed to
    descriptions differs

<LI>in allowing loosely linked arguments (termsets) instead of
   those linked with [je/jue] for the predicate

<LI>moved to the end by GO.

<p>

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

<p>

<H3>Construction of sentence modifiers</H3>

<LI>the construction of predicate modifiers (prepositional phrases
   usable as terms along with arguments).

<p>

mod1a <- (PAWORD freemod? argument1 guua?)

<p>

<LI>note special treatment of predicate modifiers without actual
   arguments.

<LI>the !barepred serves to distinguish these predicate modifiers
   from actual

<LI>"tenses" (predicate markers).

<p>
```

```
mod1 <- ((PAWORD freemod? argument1 guua?)/(PAPHRASE freemod? ! barepred gap?))

<p>

<LI>forethought connection of modifiers. There is some subtlety in

<LI>how this is handled.

<p>

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

<p>

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

<p>

<LI>afterthought connection of modifiers

<p>

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

<p>

<H3>Serial names (a flash point)</H3>

<LI>the serial name is a horrid heterogenous construction! It can involve

<LI>components of all three of the major phonetic classes essentially!

<LI>However, I believe I have the definition right, with all the components
```

```
<LI>correctly guarded :-)

<p>

name <- ((PRENAME/ACRONYMICNAME) ((Comma2? !FalseMarked PRENAME)/(
   Comma2? &([Cc] [Ii]) NAMEWORD)/(Comma2? CI predunit !(Comma2? (!
   FalseMarked PRENAME)))/(Comma2? CI ACRONYMICNAME))* freemod?)

<p>

LAWORD <- (sp? [Ll] [Aa] Juncture?)

<p>

LANAME <- (LAWORD Comma2? name)

<p>

<H3>General construction of descriptive arguments</H3>

<LI>general constructions of arguments with "articles".

<p>

<LI>the rules here have the "possessive" construction as in [lemi
   hasfa; le la Djan, hasfa] embedded. These are not the same

<LI>construction in 1989 Loglan, though speakers might think they
   are. Here they are indeed the same. The "possessor" cannot

<LI>be "indefinite" (cannot start with a quantifier word); the
   possessor can be followed by a tense, as in

<LI>[le la Djan, na hasfa], "John's present house", by analogy with
    [lemina hasfa], which is accepted by LIP (because

<LI>LIP accepts [lemina] as a word).

<p>
```

```
<LI>there are other subtleties to be reviewed.

<p>

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

<p>

<LI>abstract descriptions. Note that abstract descriptions are
    closed with [guo] entirely independently of abstract predicates:

<LI>[le po preda guo] does not have a grammatical component [po
    preda guo]. This avoids the double closure often apparently
    necessary

<LI>in Lojban.

<p>

abstractn <- ((LEFORPO freemod? POA freemod? uttAxclone guoa?)/(
    LEFORPO freemod? POA freemod? sentenceclone guoa?)/(LEFORPO
    freemod? POE freemod? uttAxclone guoe?)/(LEFORPO freemod? POE
    freemod? sentenceclone guoe?)/(LEFORPO freemod? POI freemod?
    uttAxclone guoi?)/(LEFORPO freemod? POI freemod? sentenceclone
    guoi?)/(LEFORPO freemod? POO freemod? uttAxclone guoo?)/(LEFORPO
     freemod? POO freemod? sentenceclone guoo?)/(LEFORPO freemod?
    POU freemod? uttAxclone guou?)/(LEFORPO freemod? POU freemod?
    sentenceclone guou?)/(LEFORPO freemod? PO freemod? uttAxclone
    guo?)/(LEFORPO freemod? PO freemod? sentenceclone guo?))

<p>

<LI>a wider class of basic argument constructions. Notice that
    LANAME is always read by preference to descriptn.

<p>
```

```
Ciforsuffix <- ([Cc] [Ii])

<p>

namesuffix <- (&((Comma2 !FalseMarked PRENAME)/(sp? Ciforsuffix
    Juncture? Comma2? (PRENAME/ACRONYMICNAME))) ((sp? Ciforsuffix
    Juncture? Comma2?)/Comma2)? name)

<p>

arg1 <- (abstractn/(LIO freemod? descpred guea?)/(LIO freemod?
    argument1 guua?)/(LIO freemod? mex gap?)/LIOALIEN/LAO/LANAME/(
    descriptn guua? namesuffix?)/LIU/LIE/liquote)

<p>

<LI>this adds pronouns (incl. the fancy [gao] letterals) and the
    option of left marking an argument with [ge]

<p>

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

<p>

<H4>Argument modifiers (subordinate clauses)</H4>

argmod1 <- (((sp? (n o) sp?)? ((JI freemod? predicate)/(JIO freemod
    ? sentence)/(JIO freemod? uttAx)/(JI freemod? modifier)/((JI/
    NUJI) freemod? argument1)))/((sp? (n o) sp?)? (((JIZA freemod?
    predicate) guiza?)/((JIOZA freemod? sentence) guiza?)/((JIOZA
    freemod? uttAx) guiza?)/((JIZA freemod? modifier) guiza?)/(JIZA
    freemod? argument1 guiza?)))/((sp? (n o) sp?)? ((JIZI freemod?
    predicate guizi?)/(JIOZI freemod? sentence guizi?)/(JIOZI
    freemod? uttAx guizi?)/(JIZI freemod? modifier guizi?)/(JIZI
    freemod? argument1 guizi?)))/((sp? (n o) sp?)? ((JIZU freemod?
    predicate guizu?)/(JIOZU freemod? sentence guizu?)/(JIOZU
    freemod? uttAx guizu?)/(JIZU freemod? modifier guizu?)/(JIZU
    freemod? argument1 guizu?))))
```

<p>

<LI>we improved the trial.85 grammar by closing not argmod1 but argmod with [gui] . But the labelled argument modifier constructors

<LI>when building an argmod1 have the argmod1 construction closed with the corresponding labelled right marker, of course. Thus

<LI>gui and guiza actually have different grammar.

<p>

<LI>trial.85 did not provide forethought connected argument modifiers, and we also see no need for them,

<LI>though they could readily be added.

<p>

argmod <- (argmod1 (AONE freemod? argmod1)* gui?)

<p>

<H4>Arguments resume</H4>

<LI>affix argument modifiers to a definite argument

<p>

arg2 <- (arg1a freemod? argmod*)

<p>

<LI>build a possibly indefinite argument from an argument: to le mrenu

<p>

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

```
<p>

<LI>build an indefinite argument from a predicate

<p>

indef1 <- (mex freemod? descpred)

<LI>affix an argument modifier to an indefinite argument

<p>

indef2 <- (indef1 guua? argmod*)

<p>

indefinite <- indef2

<p>

<LI>link arguments with the fusion connective [ze]

<p>

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

<p>

<LI>forethought connection of arguments. Note use of argx

<p>

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

<p>

<LI>arguments with possible negations followed by possible indirect
    reference constructions.
```

<p>

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

<p>

<LI>afterthought connection with the tightly binding ACI connectives

<p>

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

<p>

<LI>afterthought connection with the usual A connectives. Can’t start with GE

<LI>to avoid an ambiguity (to which 1989 Loglan is vulnerable) involving AGE connectives.

<p>

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

<p>

<LI>afterthought connection (now right grouping, instead of the left grouping above)

<LI>using the AGE connectives. GUU can be used to affix an argument modifier at this top level.

<p>

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

<p>

```
<LI>possibly negated and case tagged arguments. We (unlike 1989 Loglan) are careful

<LI>to use argument only where case tags are appropriate.

<p>

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

<p>

<LI>an argument which is actually case tagged.

<p>

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

<p>

<H3>Term lists</H3>

<LI>arguments and predicate modifiers actually associated with predicates.

<p>

term <- (argument/modifier)

<p>

<LI>a term list consisting entirely of modifiers.

<p>

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

<p>

<LI>a term list consisting entirely of modifiers and tagged arguments.
```

```
<p>

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

<p>

<LI>the subject class is a list of terms (arguments and predicate
   modifiers) in which all but possibly one

<LI>of the arguments are tagged, and there is at least one argument
   , tagged or otherwise.

<p>

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

<p>

<LI>these classes are exactly argument, but are used to signal

<LI>which argument position after the predicate an argument
   occupies.

<LI>I think the grammar is set up so that these will actually

<LI>never be case tagged, though the grammar does not expressly
   forbid it.

<p>

argumentA <- argument

<p>

argumentB <- argument

<p>
```

```
argumentC <- argument

<p>

<LI> argumentC <- argument

<p>

argumentD <- argument

<p>

<LI>for argument lists not guarded against absorbing a following
   subject (now redundant)

<p>

argumentA1 <- argument

<p>

argumentB1 <- argument

<p>

argumentC1 <- argument

<p>

argumentD1 <- argument

<p>

<LI>a general term list. It cannot contain more than four untagged
   arguments (they will be labelled

<LI>with the lettered subclasses given above).

<p>
```

```
terms <- ((modifiersx? argumentA (freemod? modifiersx)? argumentB?
   (freemod? modifiersx)? argumentC? (freemod? modifiersx)?
   argumentD?)/modifiersx)

<p>

<LI>terms list not guarded against absorbing a following subject (
   now the same as terms)

<p>

terms1 <- ((modifiersx? argumentA1 (freemod? modifiersx)?
   argumentB1? (freemod? modifiersx)? argumentC1? (freemod?
   modifiersx)? argumentD1?)/modifiersx)

<p>

<LI>innards of ordered and unordered list constructions. These are
   something I totally rebuilt, as they were in a totally

<LI>unsatisfactory state in trial.85. Note the use of comma words
   to separate items in lists.

<p>

word <- (arg1a/indef2)

<p>

words1 <- (word (ZEIA? word)*)

<p>

words2 <- (word (ZEIO? word)*)

<p>

wordset1 <- (words1? LUA)

<p>
```

```
wordset2 <- (words2? LUO)

<p>

<LI>the full term set type to be affixed to predicates.

<p>

<LI>forethought connection of term lists

<p>

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

<p>

<LI>afterthought connection of term lists. There are cunning things
    going on here getting [guu]

<LI>to work correctly. Note that [guu] is NOT a null term list as
   it was in trial.85.

<p>

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

<p>

<LI>there is an interesting option here of a list of terms followed
    by [go] followed by a predicate

<LI>intended to metaphorically modify the predicate to which the
   terms are affixed. Is there a reason

<LI>why we cannot have a more complex construction in place of
   terms?
```

```
<p>

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

<p>

<H3>The general verb phrase construction</H3>

<LI>this is the untensed predicate with arguments attached. Here is
    the principal locus

<LI>of closure with [guu], but it is deceptive to say that [guu]
   merely closes barepred,

<LI>as we have seen above, for example in [termset2].

<p>

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

<p>

<LI>tensed predicates

<p>

markpred <- (TENSE freemod? barepred)

<p>

<LI>there follows an area in which my grammar looks different from
   trial.85. Distinct parallel forms for

<LI>marked and unmarked predicates are demonstrably not needed even
    in trial.85. The behavior of the ACI

<LI>connectives is plain weird in trial.85; here we treat ACI
   connectives in the same way as A connectives, but

<LI>binding more tightly.
```

```
<p>

<LI>units for the ACI construction following -- possibly multiply
   negated bare or marked predicates.

<p>

<LI>adding shared termsets to logically connected predicates are
   handled differently here than in trial.85,

<LI>which uses a very elegant but dreadfully left-grouping rule
   which a PEG cannot handle. Any realistic situation

<LI>should be manageable.

<p>

backpred1 <- ((neg2 freemod?)* (barepred/markpred))

<p>

<LI>ACI connected predicates. Shared termsets are added. Notice how
    we first group backpred1's then recursively

<LI>group backpreds.

<p>

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

<p>

<LI>A connected predicates; same comments as just above. Cannot
   start with GE to fix ambiguity with AGE connectives.

<p>
```

```
predicate2 <- (!GE (((backpred (AONE !GE freemod? backpred)+
   freemod? ((termset guu?)/(guu &termset))?) ((AONE freemod?
   predicate2)+ freemod? ((termset guu?)/(guu &termset))?)?)/
   backpred))

<p>

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

<p>

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

<p>

<LI>identity predicates from above, possibly negated

<p>

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

<p>

<LI>predicates in general. Note that identity predicates cannot be
   logically connected

<LI>except by using forethought connection (see above).

<p>

predicate <- (predicate1/identpred)

<p>

<H3>The sentence</H3>

<LI>The gasent is a basic form of the Loglan sentence in which the
   predicate leads.
```

<LI>The basic structure is [PA word (usually a tense) or [ga]) followed optionally by terms followed optionally by

<LI>[ga] followed by terms. The list of terms after [ga] (if present) will either contain

<LI>at least one argument and no more than one untagged argument

<LI>(a subject) [gasent1] or all the arguments of the predicate [gasent2]. We deprecate other arrangements possible in

<LI>1989 Loglan because they would cause unexpected reorientation of the arguments already given before [ga] as second

<LI>and further arguments were read after [ga]. [barepred] is an untensed predicate possibly with arguments; [sentpred]

<LI>is "simply a verb", i.e., a predicate without arguments.

<p>

<LI>there is a semantic change from 1989 Loglan reflected in a grammar change here:

<LI>in [gasent1] the final (ga subject) is optional. When it does not appear, the resulting

<LI>sentence is an observative (a sentence with subject omitted), not an imperative.

<LI>Imperatives for us are unmarked.

<p>

<LI>4/22 allowing general predicates in gasent. Otherwise the spaces of observatives and imperatives become quite confused.

<p>

```
gasent1 <- ((NOWORD freemod?)* (freemod? &markpred predicate (GATWO
     freemod? subject)?))

<p>

gasent2 <- ((NOWORD freemod?)* (TENSE freemod? sentpred modifiers?
    (GATWO freemod? subject freemod? GIO? freemod? terms?)))

<p>

gasent <- (gasent2/gasent1)

<p>

<LI>this is the simple Loglan sentence in various basic orders. The
     form "gasent" is discoussed just above.

<LI>Predicate modifiers

<LI>can be prefixed to the gasent. The final form given here is the
     basic SVO sentence. The "subject" class is a list of terms

#(arguments and predicate modifiers) containing at most one un-case
    -tagged argument. The most general SVO form is subject, followed
     optionally

#by [gio] followed by a list of terms (1989 Loglan allowed more
    than one untagged argument before the predicate, but this leads
    to practical problems

#in which preceding constructions with errors in them may supply
    extra unintended arguments. It should be noted in NB3 that JCB
    envisioned

#a single argument before the predicate, followed by the predicate,
     which may itself contain further arguments. A gasent nay
    optionally be negated

#(even multiple times).
```

<p>

<LI>re [gio] and some other changes, in his comments on the NB3 grammar JCB often notes restrictions on appearances of term lists which he

<LI>intends but which he thought were hard to implement in the machine grammar. The appearance of just one argument before the "verb"

<LI>in an SVO sentence was one of these (though later he takes it as a virtue that the actual machine grammar supports SOV: we did not

<LI>consider it a virtue to have unmarked SOV after observing unintended parses appearing in the Visit text). Another example of this

<LI>(which would not have been hard for JCB to implement, in fact) is our restriction of the form "terms gasent" to "modifiers gasent".

<LI>His comments make it clear that he does not want arguments among those terms.

<p>

```
# statement <- (gasent/(modifiers freemod? gasent)/(subject freemod
   ? freemod? (GIO freemod? terms1)? predicate))
```

<p>

```
statement <- (gasent/(modifiers freemod? gasent)/(subject freemod?
   freemod? (GIO freemod? terms1)? predicate))
```

<p>

<LI>this is a forethought connected basic sentence. It is odd (and actual odd results can be exhibited) that the final segment in both

```
<LI>of these rules is of the very general class uttA1, which
   includes some quite fragmentary utterances usually intended as
   answers.

<p>

<LI>12/20/2017 I rewrote the rule in a more compact form. This rule
    looks ahead to the class [sentence] which we now develop;

<LI>for the moment notice that [sentence] will include [statement].

<p>

<LI>4/14 tentatively allowing initial modifiers here and leaving
   this out of uttA0 which replaces uttA1 below.

<LI>The intention is to eliminate weird sentence fragments.

<p>

<LI> this is where I could reinstall permission to use headterms
   without [goi] before keksents. I do not think I want to.
<LI> I have done this experimentally.

<p>

keksent <- (terms? freemod? (NOWORD freemod?)* (KA freemod?
   headterms? freemod? sentence freemod? KI freemod? sen1))

<p>

<LI>cloned if not marked as imperative

<p>

keksentclone <- (terms? freemod? (NOWORD freemod?)* (KA freemod?
   headterms? freemod? sentenceclone freemod? KI freemod? sen1clone
   ))
```

```
<p>

<LI>sentence negation. We allow this to be set off from the main
   sentence with a mere pause, because generally

<LI>it does not differ in meaning from the result of negating the
   first argument or predicate modifier.

<p>

neghead <- ((NOWORD freemod? gap)/(neg2 OptPause))

<p>

<LI>this class includes [statement], predicate modifiers preceding
   a predicate (which may contain arguments), a statement,

<LI>a predicate, and a keksent. Of these, the first and third are
   imperatives.

<p>

<LI>4/23/2019 added actual rule for imperative sentences. This
   should not

<LI>affect the parse in any essential way.

<p>

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

<p>

<LI> clone of imperative for labelling which occurrences are
   actually imperative

<p>

nosubject <- ((modifiers freemod?)? !gasent predicate)
```

<p>

```
headterms <- (terms GI freemod?)+
```

<LI>[headterms] is a list of terms (arguments and predicate modifiers) ending in [gi]. Preceding a [sen1] with these

<LI>causes all predicates in the [sen1] to share these arguments. We propose either that the headterms arguments be directly

<LI>appended to the argument list of each component of the [sen1], or that there is an argument with a numbered case tag at the beginning

<LI>of the headterms list, and the list is inserted at the appropriate position in each component sentence. Neither of these is

<LI>the condition described in Loglan I, which presupposes that we always know what the last argument of each predicate used is.

<p>

<p>

<LI>this is the sentence class below prefixed with a list of fronted terms.

<LI>we think the [giuo] closure might prove useful.

<p>

```
uttAx <- (headterms freemod? sentence giuo?)
```

<p>

<LI> cloned if not to be marked as imperative

<p>

```
uttAxclone <- (headterms freemod? sentenceclone giuo?)

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

<p>

<LI> clone of sen1 with nosubject instead of imperative

<p>

sen1clone <- ((neghead freemod?)* (nosubject/statement/keksentclone
   /uttAxclone))

<p>

<LI>the class [sentence] consists of sen1's afterthought connected
   with A connectives

<LI> The logical structure of a [sentence] may not be transparent.
   The effect of appending another {ICA sen1) to a [sentence] is to
    connect the previous part of the sentence to the new [sen1]
   with the ICA connective. In other words, this groups to the left
   , logically.
<LI> Classes [sentence] and [uttAx] play an important role in my
   proposed definition of scope: the scope of a quantifier is the
   smallest item of one of these classes
<LI>which includes all instances of the variable it binds and all
   scopes of quantifiers which appear after it in its scope (in the
    peculiar SOV order we use). Notice
<LI>that an initial segment of a [sentence] is not a [sentence]:
   scopes are to be closed under ICA connectives.

<p>

<LI> adding another layer to sentences, afterthought connected with
    ICI connectives, more tightly binding (and removing class uttD
   below)

sentence1 <- (sen1 (ICI freemod? sen1)*)
```

```
<p>

sentence1clone <- (sen1clone (ICI freemod? sen1clone)*)

<p>

<LI> a sentence will continue through terminal punctuation; this
   averts ambiguity with the higher level utterance construction
   with ICA.

<p>

sentence <- (sentence1 ([!.:;?]? ICA freemod? sentence1)*)

<p>

sentenceclone <- (sentence1clone ([!.:;?]? ICA freemod?
   sentence1clone)*)

<p>


<H3>Utterances</H3>

<LI>weird answer fragments

<p>

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

<p>

<LI>a broad class of utterances, including various things one would
    usually only say as answers. Notice

<LI>that this utterance class can take terminal punctuation.

<p>
```

```
uttA1 <- ((links/linkargs/argmod/terms/uttA/NOWORD) freemod? Period
    ?)

<p>

<LI>possibly negated utterances of the previous class.

<p>

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

<p>

<LI> Higher level utterance construction with ICA connectives
    eliminated.

<p>

ICAUTT <- ICA

<p>

# uttE <- (uttC (ICAUTT freemod? uttC)*)

<p>

<LI>utterances of the previous class linked with I sentence
    connectives.

<p>

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

<p>

<LI>the utterance class for use in the context of parenthetical
    freemods or quotations, in which it does not go to end of text.

<p>
```

```
utterance0 <- (!GE ((ICAUTT freemod? uttF)/(!OptPause freemod
   Period? utterance0)/(!OptPause freemod Period?)/(uttF IGE
   utterance0)/uttF/(I freemod? uttF?)/(I freemod? Period?)) (&I
   utterance0)?)

<p>

<LI>Notice that there are two passes here: the parser first checks
   that the entire utterance

<LI>is phonetically valid, then returns and checks for grammatical
   validity.

<p>

<LI>the full utterance class. This goes to end of text, and
   incorporates the phonetics check. This incorporates the only
   situations

<LI>in which a freemod is initial. The IGE connectives bind even
   more loosely than the I connectives and right-group instead of

<LI>left grouping.

<p>

utterance <- (&(PhoneticUtterance End) (!GE ((ICAUTT freemod? uttF
   (&I utterance)? End)/(!OptPause freemod Period? utterance)/(!
   OptPause freemod Period? (&I utterance)? End)/(uttF IGE
   utterance)/(I freemod? Period? (&I utterance)? End)/(uttF (&I
   utterance)? End)/(I freemod? uttF (&I utterance)? End))))

</TT>
```