

A New Systematic Grammar of TLI Loglan

Randall Holmes

August 29, 2021

Chapter 1

Introduction

This is a new grammar of TLI Loglan. The intention is to present a coherent picture of my provisional adjustments of the language. The organizational principle is that I follow the structure of the Parsing Expression Grammar (PEG) which is used to generate the computer grammar.

There will be observations on points of difference with earlier versions of the language as necessary.

This document speaks authoritatively, but not all these proposals have been approved by the TLI Loglan community. I am writing this in hope of providing support for a consensus that this is the way to proceed. The membership is welcome to offer criticisms, whether general or of particular points.

The reader can safely ignore comments in footnotes unless already proficient in Loglan or interested in the history of the language.

This document now contains the current text of the file

`draft-grammar-with-comments-alternative,`

with word wrap and line numbers. This means that if you want to compile this file yourself, you need not only its source but the current PEG source.

Chapter 2

Phonetics and Orthography

In this chapter, we discuss the rules for writing and pronouncing Loglan, and the way in which a stream of speech sounds or letters is formally resolved into Loglan words (with the proviso that grammar rather than phonetics dictates word boundaries between structure words).

2.1 Phonetic and orthographic components

2.1.1 Loglan letters and punctuation

The letters of the Loglan alphabet are the 23 letters of the Roman alphabet excluding **q, w, x**.

The foreign letters **q, w, x** can only occur in “alien text” embedded in Loglan.¹

The vowels are **a, e, i, o, u, y**. The first five are the regular vowels.

The consonants are **b, c, d, f, g, h, j, k, l, m, n, p, r, s, t, v, z**

The Loglan name of a vowel **V** has two forms (legacy and modern²): legacy uppercase is **Vma** and lowercase is **Vfi**, and modern uppercase is **ziVma** and lowercase **ziV**. The legacy forms are fully supported, but they

¹These letters were originally not included in Loglan, then they were added with strange pronunciations in the 1980's and 90's, then they were largely eliminated from the dictionary in the late 90's; after 2013, we proceeded to eliminate them completely again. Names for these letters (usable as pronouns) will be presented later.

²The modern forms were suggested by us after 2013, but we have fully accommodated the phonetics to the original forms.

are phonetically irregular as Loglan words, and there are contexts where the modern forms must be used.

The Loglan name of a consonant C is **Cai** (uppercase) or **Cei** (lowercase).

There are some other series of letter names to be introduced below. The primary function of these words is not phonetic, but as variables (pronouns), as will be explained later.

The junctures, indicating syllable breaks or stress are -, ', *. The hyphen - is a simple syllable break.³ ' is a syllable stress marker, which may appear in place of (not in addition to) a hyphen after a stressed syllable, or in final position in a word after a stressed syllable. * is a symbol for emphatic syllable stress, with the same grammar as '. A juncture is never followed by another juncture; a hyphen can be followed only by a letter (the hyphen, unlike the stress marks, never appears in final position).

We define a phonetic block as a sequence of letters and junctures (referred to collectively as “characters”), with the junctures giving information about syllable breaks and stress. A phonetic block is always intended to be pronounced without pause.

The terminal punctuation marks of Loglan are .:?!;#.

The comma , is an especially important punctuation mark, with the phonetic meaning of a pause in the flow of speech. A comma is always followed by whitespace followed by a phonetic block or alien text (ignoring any initial parentheses or double quotes appearing before the block or alien text). A phonetic pause is always denoted either by whitespace or a comma followed by whitespace; in some contexts the comma-marked pause is mandatory. Whitespace may or may not denote a pause; there are some contexts where whitespace must denote a pause.⁴

The double hyphen – is an independent punctuation mark, not a syllable break: it represents a pause, probably longer than the pause represented by a comma, and in some cases may be used in place of a comma where a pause is required. The ellipsis ... is an independent punctuation mark, not terminal punctuation, similarly representing a pause and occasionally usable in place of a comma.

Parentheses and double quotes may enclose Loglan or alien text under some circumstances described in the grammar. These are generally ignored

³The use of the hyphen to abbreviate the phonetic hyphen **y** found in earlier sources is not accepted here; in general, we do not pronounce punctuation.

⁴Uses of a close comma without a following space in earlier versions of Loglan are entirely replaced with uses of the hyphen as a syllable break.

for phonetic purposes. They are not pronounced: punctuation marks are never intended to be pronounced in the version of Loglan described here, though they may dictate pauses.

The letters have lowercase forms and uppercase forms and there is a capitalization rule applying to phonetic blocks. The formal capitalization rule is quite complex: the basic idea is that an uppercase letter will not appear immediately following a lowercase letter unless it is the first letter of a phonetic copy of a letter name (class **TAIO** to be discussed below; the letter names given above are words of this class), and a vowel may appear capitalized after **z** (for malicious reasons to be explained later). There is no restriction on capitalization resuming after a juncture. This allows the usual sort of capitalization, and also allows all-caps, and where junctures are present individual blocks of letters may be capitalized in different styles independently. The special treatment of letter names will be motivated in examples when these words appear.⁵

2.1.2 Pronunciation of Loglan letters

The regular vowels have typical continental European (not English!) pronunciation. The irregular vowel **y** may be pronounced with the indistinct schwa sound. Unstressed regular vowels do **not** become schwa (English and Russian speakers note!) A more distinct pronunciation for **y** (the vowel in English “look”⁶ or the vowel written **Ы** in Cyrillic) might be preferred.

The pronunciations of **b, d, f, g, k, l, m, p, r, s, t, v, z** require no special comment (except to note that **g** is always hard).

The letters **c, j** have unusual pronunciations, **c** being English “sh” in “ship”, and **j** being the voiced version of the same sound, the “z” in “azure”. English “ch” and “j” are the diphthongs **tc** and **dj** respectively.

The letter **h** has the typical English pronunciation, except in syllable final position, where it has the sound of “ch” in Scottish “loch”. The latter pronunciation is actually permitted in other positions as well.⁷

⁵The approach to punctuation which we have taken has been driven partially by our own design decisions and partly by the punctuation and capitalization practices in the Visit to Loglandia.

⁶A suggestion of John Cowan.

⁷Syllable-final **h** did not exist in previous versions of Loglan. The pronunciation of **h** which is mandatory in final position might be preferred by speakers of some languages in other positions.

The letter **n** is pronounced as usual in English, and (as usual in English as well!) is pronounced as the “ng” in “sing” when it occurs before **g**, **k**, or the alternative pronunciation of **h**.

The vowels **i** and **u** are sometimes pronounced as English “y” and “w”. as will be explained below.

The consonants **l**, **m**, **n**, **r** are sometimes syllabic (“vocalic”). When they are used syllabically, they are always doubled.⁸

We neither reject nor support complex alternative schemes of pronunciation of the language outlined in older sources; the alternatives we propose here seem sufficient.

2.1.3 Alien text

We describe the rules for embedding alien text in Loglan.⁹ For such text, rules of pronunciation are not supplied by Loglan. It is required that alien text (however it is pronounced) be preceded by a pause (regarded as part of the alien text) and followed by a pause or end of text or speech (not regarded as part of the alien text): a pause may be expressed either by whitespace or by a comma or terminal punctuation (terminal punctuation only after the alien text, of course) followed by whitespace, and end of text or speech simply by end of text or by terminal punctuation. The body of the alien text between the initial pause and the final pause or end may be text not containing double quotes enclosed in double quotes, or it may consist of one or more blocks of text excluding commas, spaces and terminal punctuation marks, separated by the word **y**, which must be preceded and followed by pauses in speech, which are independently expressible by whitespace or by a comma-marked pause. Examples are “**War and Peace**” and **War y and y Peace**. When alien text is enclosed in quotes, occurrences of **y** between pause-separated components of the alien text may be omitted in writing but

⁸The rule that syllabic continuants must be doubled forces changes of spelling in names in legacy Loglan text in many cases. Syllabic consonants in borrowed predicates were already doubled, and Brown suggested in Loglan 1 that this might be a good rule to adopt in general.

⁹The model for these rules is actually the final state of the rule for Linnaean names with **lao** (now foreign names in general) given in the late 90’s. We require that the occurrences of **y** there suggested merely for speech also be expressed in writing. The use of double quotes is a novelty but seems natural. The strong quotation scheme of 1989 Loglan is abandoned, essentially by giving **lie** the same phonetic grammar as **lao**.

must appear in speech: the two examples are pronounced in the same way. Some contexts require double quoted alien text in writing.

Alien text is always preceded by one of the alien text markers **hoi**, **hue**, **lie**, **lao**, **lio**, **sao**, **sue**, whose grammar and semantics will be discussed below. Alien text marked with **hoi** or **hue** must be double-quoted. The parser identifies blocks of alien text by looking for these markers (some of the markers have multiple functions and will not always be followed by alien text).

Examples of alien text in Loglan utterances will appear when we discuss the grammatical constructions that use them.

2.1.4 Vocalic diphthongs

In this section, we describe two-letter forms which may appear as the “vowel” component of a syllable.

The consonants **l**, **m**, **n**, **r** we call *continuants*. A doubled continuant **ll**, **mm**, **nn**, **rr** represents a syllabic continuant, which may serve as the vocalic component of a syllable. A syllabic continuant may not be followed or preceded by another instance of the same continuant without an intervening pause in speech.

We now consider how to pronounce sequences of regular vowels not separated by junctures. The issue is how to resolve such a sequence into syllables. The irregular vowel **y** is usually a single syllable, except for occurrences of syllables **iy** and **uy** in rare structure words, which will be discussed later.

We first consider sequences of two regular vowels. Some of these sequences are mandatory monosyllables, some are optional monosyllables, and some cannot be read as monosyllables.

There are four mandatory diphthongs **ai**, **ei**, **oi**, **ao**. The diphthong **ao** has the irregular pronunciation of “ow” in English “cow”. The pairs of letters **ai**, **ei**, **oi** are not mandatory diphthongs when followed by **i** without an intervening juncture: **aii** is grouped **a-ii**.

The forms **a-i**, **e-i**, **o-i**, **a-o** (the hyphens may be replaced with other junctures), called broken monosyllables, can only occur in names (we do not regard **a-ii** and its kin as containing broken monosyllables, but here we are talking about more than two letters). One may write any other pair of regular vowels separated by a juncture, with the effect of enforcing the two syllable pronunciation (where it is optional).

There are six optional diphthongs, made up of **i** or **u** followed by a regular vowel. (forms **iy** and **uy** also occur in special contexts, to be discussed later). If these are pronounced as a single syllable, initial **i** is pronounced as English “y” and initial **u** is pronounced as English “w”. The disyllabic pronunciation can be compelled by writing a syllable break. Monosyllabic **iu** cannot be followed without a juncture by **u** and monosyllabic **ui** cannot be followed without a juncture by **i** (so, for example, if **iuu** is encountered it will be read **i-uu**). As a rule, the speaker has a choice when presented with an optional monosyllable of pronouncing it as one or two syllables; sometimes the context forces one of the pronunciations.

Other pairs of adjacent vowels are pronounced as two separate syllables; the use of a glottal stop to separate the components of a disyllabic vowel pair is permitted, but not expressed in the orthography. The glottal stop is **not** allowed as an allophone of the pause phoneme; all required pauses must be distinct, if sometimes brief.¹⁰

A pair of identical adjacent vowels not pronounced as a monosyllable has the characteristic that one of the vowels must be stressed and the other unstressed. This always holds for **aa**, **ee**, **oo** and sometimes holds for **ii**, **uu** (special rules stated above are designed to encourage pronunciation of the latter two pairs as monosyllables whenever possible!)

For a three-vowel sequence appearing in a predicate or name word, the general rule is that formation of monosyllabic **ii** or **uu** is the highest priority (so in **aii**, forming **ii** wins over forming **ai**, which in this context is not a mandatory monosyllable anyway, producing **a-ii**), followed by formation of a mandatory monosyllable (recalling that **i**-final mandatory monosyllables are not followed by **i**; **aoi** is grouped **ao-i** and is not considered to contain a broken monosyllable): e.g., **aiu** is grouped **ai-u**), followed by formation of an optional monosyllable (which is often an optional preference for the speaker; the parser does exercise this preference). An extreme example of speaker freedom is **iue**, which the parser will resolve into two syllables **iu-e** (choosing to group the first two when both pairs have the same precedence) but which the speaker can resolve into two or three syllables in any of the three possible ways.

¹⁰Previous versions of Loglan do not allow the glottal stop to appear medially in disyllables (we allow it but also allow the traditional Loglan pronunciation, a smooth glide from one vowel to the other); previous versions of Loglan allowed the glottal stop as an allophone of pause, and we do not. Lojban uses the h sound medially in disyllables, which would be allowed for a Loglan speaker who chose always to use the alternative pronunciation of **h**.

We present a formal rule¹¹ for reading the next syllable from a sequence of regular vowels of any length written without junctures, which is used in resolving predicates and names into syllables. A mandatory diphthong is read as the first syllable if it is present (recalling that if the pair of vowels ends in **i** and is followed by another **i** it is not a mandatory diphthong); a single vowel is read if it is not initial in a mandatory diphthong and the next two vowels form a mandatory diphthong; if neither of the previous two cases holds an optional diphthong is read by preference by the parser (though a disyllabic reading is permitted); as the final option a single vowel is chosen, subject to the rule that **i** or **u** (when not part of a diphthong) cannot be followed by an intervening juncture and a consonantal occurrence of the same vowel (this situation will cause parse failure). The process of resolution of the first syllable from a stream of vowels is repeated until the stream of vowels is completely resolved into syllables. This rule may look forbidding, but it should be noted that sequences of four or more vowels are quite rare in Loglan predicates or names, so the two and three vowel accounts will usually be quite enough.

There is a separate rule, used in resolving certain structure words, in which a sequence of vowels of even length is parsed into vowel pairs, each of which is read as monosyllable or disyllable as the rules require or permit. There is a further special rule for certain structure words with three-vowel sequences, which does not conform with the rule stated above for resolving vowel sequences in predicates and names, which will be stated when these structure words are described.

When the vowel component of a syllable is read, this will be either a syllabic continuant, or **y**, or a vowel or vowel diphthong chosen using the appropriate one of the rules above.

One should note that the rules presented here are not of interest to readers and writers, speakers and listeners, very directly; but they are certainly of interest to word makers, and might briefly be of interest to a dictionary reader encountering a word for the first time. Such phonetic rules exist in natural languages, whose speakers are not necessarily even aware of them; one could imagine that the native Loglander, though her speech will conform perfectly to the rules stated above, will not know much about them unless

¹¹The formal rule for reading long sequences of vowels in names appearing in Loglan 1 is incredible, as it requires indefinite lookahead; of course it was also really intended only for use with three or perhaps four vowels.

she is a grammarian!

2.1.5 Consonant grouping

There are different rules for syllable-initial and syllable-final consonant grouping. It is worth noting that consonant grouping only occurs in regular Loglan text in predicates and names. Syllables with final consonants also occur only in predicates and names.

These are governed by two sets of phonetic rules. There is a list of permitted initial pairs of consonants¹². The initial group of consonants in a syllable consists of a single consonant, or a permissible initial pair of consonants, or a triple of consonants in which each adjacent pair of consonants is an initial pair.¹³ We refer to a pair of consonants which would be a permissible initial pair if an intervening juncture were removed as a “broken initial pair”.

There is a list of forbidden medial pairs¹⁴ and a list of forbidden medial triples¹⁵. These cannot occur even if broken by a juncture.

There can be one or two final consonants in a syllable, which cannot be part of a forbidden medial pair or triple whether together (if there are two of them) or combined with consonants taken from the beginning of the following syllable. A pair of final consonants cannot be a non-continuant followed by a continuant (this appears to be pronounceable only as a separate syllable). A final consonant cannot be followed by a regular vowel or a syllabic continuant, even with an intervening juncture (in other words, such a consonant should be read as part of the following syllable).¹⁶

A new (2/13/2021) rule forbids a pair of final consonants to consist of one

¹²The initial pairs are **bl br ck cl cm cn cp cr ct dj dr dz fl fr gl gr jm kl kr mr pl pr sk sl sm sn sp sr st sv tc tr ts vl vr zb zl zv**

¹³The rule for initial consonant groups appears in Notebook 3.

¹⁴The impermissible medial pairs consist of all doubled consonants, any pair beginning with **h**, any pair both of which are taken from **cjsz**, **fv**, **kg**, **pb**, **td**, any of (**fkpt**) followed by either of (**jz**), **bj**, and **sb**.

¹⁵**cdz**, **cvl**, **ndj**, **ndz**, **dcm**, **dct**, **dts**, **pdz**, **gts**, **gzb**, **svl**, **jdj**, **jtc**, **jts**, **jvr**, **tv1**, **kdz**, **vts**, and **mzb**

¹⁶The rules forbidding final consonants from participating in illegal medial pairs or triples are found in our sources. The rule forbidding a pair of final consonants from being a non-continuant followed by a continuant seems quite natural but is ours; no word was proposed that violated it, in any case. Other rules that we state depend on a precise definition of the syllable, which appears nowhere in Loglan sources, although the notion of syllable is important in the definition of borrowed predicates in Notebook 3.

of **ptksfh** and one of **bdgzv**, in either order. This forbids a voiced and an unvoiced consonant to occur together in a pair of final consonants, if neither is a continuant.

A consonant in either of these sorts of groups which is a continuant cannot be adjacent to another copy of the same continuant, within or without the cluster, even if separated by a juncture. An initial consonant triple cannot be followed by a syllabic continuant at all.

2.1.6 The Loglan syllable

A Loglan syllable consists of three parts.

There is an optional initial group of one, two or three consonants governed by rules stated in the previous subsection.

This is followed by the mandatory vocalic component of the syllable, which is either a pair of identical continuants, a single regular vowel, a vowel diphthong, or **y** (**iy** or **uy** occur only in syllables (C)**iy** and (C)**uy** which are directly allowed as units in structure words but not supported in the formal syllable definition).

This is followed optionally by one or two final consonants, for which rules are stated above, with the additional remark that unless the syllable is of the shape CVC with the vowel regular, no final consonant in the syllable (neither of them, if there are two) may be readable as standing at the beginning of a following syllable (in other words, except in the case of CVC syllables, the automatic placement of syllable breaks where an explicit juncture is not present is as early as possible; but a CVC syllable is preferred to a CV syllable where possible). Explicit junctures will override the preferred syllable breaks, but there are subtle rules about where explicit junctures can be placed: sometimes they will simply cause parse errors.¹⁷

It is worth noting that previous versions of Loglan had no official formal definition of the syllable, though the syllable did play a role in the definition of some word classes.¹⁸

¹⁷The subtleties have to do with the fact that a borrowed predicate cannot resolve into djifoa (see below for these terms); an apparently legal borrowing predicate written with explicit junctures will be rejected if moving some of the junctures would create a legal complex predicate. These are issues which mostly affect the word designer. If you are trying to write a complex predicate from the dictionary with explicit syllable breaks, make sure that the breaks you supply conform with djifoa boundaries and these issues will not arise.

¹⁸The lack of felt need for a formal definition of the syllable may have come from the fact

2.1.7 Pauses and whitespace: general principles

A pause is always expressed as either a comma followed by whitespace (which must be followed by a phonetic block) or simply whitespace, which must be followed by a phonetic block. The former is always a pause; the latter may sometimes not be a pause.

Whitespace at the beginning or the end of alien text must represent an actual pause.

Whitespace after a consonant and/or before a vowel must represent an actual pause.

Names are the only consonant-final words in regular Loglan text, and they must be followed by comma-marked pauses, terminal punctuation, or end of text, or by whitespace followed by another name word or the structure word **ci**.

Logical connective words of class A, some but not all of which are vowel-initial, and sentence connectives of classes I and ICA must be preceded by comma-marked pauses. The APA and IPA logical and sentence connectives, to be discussed below, and the ICA and ICAPA sentence connectives must be followed either by the suffix **fi** or a comma-marked pause. The issues in this paragraph are handled entirely in the grammar section.

Words quoted with **liu** must be followed by a comma-marked pause (or terminal punctuation or end of text).

If the final syllable of a structure word is stressed and it is followed by a predicate, it must be followed by a comma-marked pause. This rule is of course only enforced in our orthography if we actually write explicit stress.¹⁹

In general, certain lexicographic issues tend to force explicit comma-marked pauses. If a pause in a sequence of structure word syllables breaks a

that structure words and complex predicates resolve into units which are not themselves necessarily syllables, but which are expected to conform with syllable boundaries; it is with the introduction of borrowed predicates that a precise notion of the syllable became essential to someone who wanted to parse words, and once this notion was in hand, it became natural to require that names (which were just consonant final strings of phonemes in earlier versions of Loglan) be resolvable into syllables as well. The accuracy of our implementation can be gauged by the fact that almost all words in the dictionary parsed correctly when we ran a test, and the ones which did not parse had recognizable errors which needed to be fixed. It should be noted that we cannot have three final consonants in a syllable, and this is not uncommon in names. This can usually be fixed by doubling a continuant, as in **Hollmz**, **Marrks**, but some names may be found to be definitely foreign.

¹⁹This rule goes back to the beginnings of Loglan, but as no earlier parser had explicit indications of stress, there was never any occasion for an earlier parser to enforce it.

word, it must be explicitly comma-marked as a rule, since if it were written as mere whitespace, not pausing would cause a different interpretation of the utterance. There will be a discussion of multi-syllable structure words in the lexicography section which lays out the situations under which this issue can occur.²⁰

The “false name marker” problem creates further need of explicit pauses, which will be discussed below.

This version of Loglan supports a form of orthography known as “phonetic transcript” in which no whitespace appears but comma-marked pauses. This means that we require that in every place where we can or must pause, it must be possible to replace whitespace with a comma-marked pause. It is mostly but not entirely true that every place whitespace is written is a place where one *can* pause: it is possible to create situations with the APA and IPA connectives in their legacy form where a whitespace that one can write cannot represent a pause, and there is a rule that one should not pause after the structure word **ci** before a consonant unless the pause is comma-marked. This whitespace can, however, be omitted. Whitespace which does not represent pauses can always be omitted, though in the case of whitespace after predicates, this may require the writer to insert explicit indications of stress so that the reader can tell where the predicate ends. Whitespace which cannot be omitted can always be replaced with an explicit comma-marked pause.

Because we have phonetic transcript, we do not need a special notation for expressing pronunciation.²¹

2.2 Phonetic word forms

2.2.1 The four forms, and general principles

There are four basic word forms in Loglan:

1. Items of alien text (with their preceding alien text markers), already described above.

²⁰In Lojban, apparently all structure word syllables are separate words, but this is not the case in Loglan.

²¹Brown’s phonetic notation in the sources is *ad hoc* and reveals such things as very inconsistent notions about syllable breaks.

2. Phonetic names (name words accompanied with their required preceding pauses or name marker words with intervening optional pause).
3. Structure words
4. Predicates, further subdivided into complexes and borrowings.

These classes of words have general characteristics which allow us to distinguish them. We leave aside the case of alien text which we have already analyzed.

1. Name words are the only consonant-final words in Loglan (other than alien text). They are thus followed by pauses in speech (and usually by explicit pauses in writing). This makes the right boundary of a name word easy to recognize. One must also pause at the beginning of a name word, unless it is preceded by one of a limited class of name markers. There are few contexts in which a name word can appear without an immediately preceding name marker word, and if a name word happens to include a phonetic copy of a name marker word (a “false name marker”) it *must* be immediately preceded by a name marker word (an intervening pause being permitted). Where a name marker word occurs which is not immediately followed by a name word but followed by a name word starting later, a comma-marked explicit pause (or terminal punctuation) must appear somewhere between the name marker word not serving as such and the following name word: this prevents pronunciation of the text in a way which causes everything between the name marker word and the end of the later name word to be construed as a single longer name word.
2. Predicates end with a regular vowel (so they are not names), are penultimately stressed (with qualifications to be stated later); this allows the right boundary of a predicate word to be recognized in speech, or in phonetic transcript), and contain adjacent consonants (in some cases the pair of consonants may be separated by **y**). The left boundary of a predicate is determined by the fact that it must begin CC or (C)VⁿC(**y**)C. in the latter case with some conditions ensuring that the (C)Vⁿ cannot be construed as a structure word. Predicates can more rarely begin (CVV**y**)ⁿ((C)V^m)CC.

3. Structure words (Loglan **cmapua**) are not names or predicates (actually some are semantically names or predicates, but this is a matter for the grammar). In addition, we specify that they resolve into phonetic units of the shapes V, VV, CV, CVV (where the VV may be a monosyllable or a disyllable, and **iy**, **uy** are permitted), and the rare Cvv-V, where the vv is a monosyllable (mandatory or optional, but in any case pronounced as such). Further, a V unit may only occur initially, and any structure word which contains a VV unit consists entirely of VV units (except that we allow words of the shapes **no-VV** and **VV-noi**). A sequence of VV units is resolved into syllables by pronouncing each unit as one or two syllables as the grammar requires or permits. Note that the unit **cmapua** are not necessarily syllables, but their boundaries are syllable boundaries in a structure word. Where a structure word is followed by a predicate beginning with CC, stressing its last **cmapua** unit might create the possibility of reading the last **cmapua** unit and the first syllable of the intended predicate word as a predicate: to avert this, we require that a finally stressed structure word must be separated from a following predicate word (not just a CC-initial one) by a comma-marked pause.

It should be noted that the classes of words here should be qualified as phonetic names, phonetic predicates, and phonetic structure words, as there are cases where “words” which are phonetically of one of these shapes are used in a way associated with one of the others.

2.2.2 Phonetic Names

We distinguish between a name word, such as **Djan**, and a phonetic name, such as **la Djan Braon**, which comes equipped with the name marker word or initial pause that a name word requires in its context, and may contain more than one name word after the name marker.

A name word is a phonetic block which resolves into syllables, the last of which ends in a consonant (possibly with a final stress).

A possible name word is a name word, or a name word modified by insertion of whitespace at junctures preceded by a vowel and succeeded by a consonant (so that the whitespace does not necessarily represent a pause).

A marked name is a name marker word followed by a consonant initial name word, possibly with intervening whitespace between the two.

A falsely marked name is a name word with a proper final segment which is a marked name: that is, it is a name word with a false name marker in it. Notice that a phonetic occurrence of a name marker word is not a false name marker unless what follows it is a consonant-initial name word.²²

The name marker words are **la**, **hoi**, **hue**, **ci**, **liu**, **gao**, **mue**. A subtle point is that **ci** is only a name marker when followed by a pause (an explicit comma-marked pause or whitespace followed by a vowel): this allows us to avoid difficult-to-predict needs for pauses after the many uses of **ci**. It does mean that when whitespace is written after **ci** before a consonant, we presume that the speaker does not pause.

A phonetic name (including its name marker or preceding pause if there is one) is of one of the following kinds:

1. a marked name as described above (a name marker followed by possible whitespace followed by a consonant-initial name word).
2. a vowel initial name word which is not a falsely marked name, or a comma-marked pause followed by a name word which is not a falsely marked name.
3. a name marker followed by optional whitespace or explicit pause followed by a name word, with the additional proviso that the optional whitespace or pause must be present if the name word is vowel-initial.

To any of these, a series of name words marked with **ci** and unmarked name words which are not falsely marked, may be appended as part of the phonetic name, so **la Djan Braon** is a phonetic name, and so is **la Pierr ci, Laplas**. In the last example one pauses both before and after **ci**; the second comma must be written, and the use of **ci** is necessary because **Laplas** is a falsely marked name.

²²In early versions of Loglan, falsely marked names were simply forbidden, but **la** is very common. Later, they were admitted and some effort was made to avoid problems with them. The idea that a falsely marked name must be marked appeared in the context of implementation of serial names (falsely marked names in a serial name had to be marked with **ci**; we required after 2013 that predicate components of serial names be marked with **ci** as well to avoid the need for two pause phonemes to avoid confusion of serial names with sentences.) We extended the idea that falsely marked names must be marked to all contexts, and in addition reduced the distribution of unmarked names to very few contexts by forbidding unmarked vocatives.

It is then required that this be followed either by an explicit pause, terminal punctuation, end of text, or whitespace followed by **ci** followed by a predicate of class **predunit**, a peek forward at the grammar. Note that the following explicit pause or punctuation or **ci** phrase is not part of the phonetic name: this is information about the context in which a phonetic name can appear.

Names may contain explicit junctures, including ones which form broken monosyllables, and junctures may be required features of name words: **Lo-is** and **Lois** are different names.²³

2.2.3 Phonetic structure words (cmapua)

Phonetic structure words are sequences of cmapua units as sketched above; we give more details.

Cmapua units are of the shapes V, VV, CV, CVV, Cvv-V, where vv stands for a monosyllable and VV (in VV and CVV units) includes **iy**, **uy**. **y** is also accepted as a V unit.²⁴

A phonetic structure word is a string of cmapua units. A cmapua unit not of VV form cannot be followed by a vowel, even with an intervening juncture: this helps to enforce the condition that vowel-initial words must be preceded by pauses in speech, represented at least by whitespace.

Each cmapua unit is restricted by lookahead tests for other classes. A cmapua unit cannot be an alien text marker actually followed by alien text. A cmapua unit cannot be an occurrence of **li** or **kie** which actually stands at the beginning of a quotation or parenthetical free modifier (for the uses of these words, see the grammar section). A cmapua unit cannot be a name marker followed with optional pause by a possible name word (this excludes both actual phonetic names and strings which could be misread as phonetic names by ignoring instances of whitespace one of which should be made an explicit pause).²⁵ A cmapua unit cannot stand at the beginning of a legal

²³This goes back to previous versions of Loglan, but we use hyphens instead of close commas.

²⁴The practical reason for allowing **y** to occur above is to support names of the letter **y**, legacy **yfi** and modern **ziy** (pronounced “zyuh”!). It seemed more principled to install general phonetic conditions that allowed these forms than to allow them individually by fiat.

²⁵This is our definitive solution to the false name marker problem. Difficulties created by the markers other than **ci** should generally be easy to anticipate, by following style rules

predicate (the parser does a lookahead test which identifies strings which can only be predicates if they are grammatical and not possible phonetic names; we describe this test below at the beginning of the discussion of predicates).

A *cmapua* unit cannot be stressed and then followed by optional whitespace and the start of a consonant-initial predicate (as detected by the test above).

We then provide a phonetic test for the logical and sentence connective classes which must be preceded by a pause. A phonetic connective starts possibly with whitespace followed by possibly by an occurrence of **no** (not starting a predicate) followed definitely by a regular V syllable or **ha**²⁶, **nuu** (not starting a predicate), not followed by a vowel, and not followed by **fi**, **ma**, or **zi**, which would make a V unit into a legacy letteral (none of these starting a predicate).

We can now describe a phonetic structure word. It takes one of five forms.

1. a VV unit (here and in all clauses here including **iy**, **uy**) followed by **noi** (**noi** not starting a predicate).
2. **no** (not starting a predicate) followed by a VV unit
3. a sequence of VV units
4. a regular or irregular V unit
5. an optional regular or irregular V unit followed by a sequence of one or more consonant-initial *cmapua* units

A *cmapua* unit absorbs a following juncture.

Each *cmapua* unit is blocked from being followed by a vowel without intervening whitespace or by optional whitespace then a phonetic connective; this forces explicit pauses before the logical and sentence connectives.

The phonetic structure words defined here have boundaries dictated entirely by phonetic convenience; the actual boundaries of *cmapua* words in the proper sense are dictated by rules stated in the lexicography chapter. Some words which appear in other structures, such as the name markers, alien

such as “always pause after a predicate name”. The word **ci** presented special difficulties as a name marker because it has a wide variety of uses some of which have nothing to do with names. Viewing it as a name marker only when followed by a pause seems to be the final refinement of our solution.

²⁶Note that we are ruling here that **ha** and its derivatives must be preceded by a pause.

text markers, and **y**, and some others, are from a lexicographic standpoint structure words and do look like them phonetically.

2.2.4 Primitive Predicates and Combining Forms (djifoa)

The basic “native” predicates of Loglan are of the five letter forms CCVCV and CVCCV. The original stock of native predicates was generated by a rather *ad hoc* statistical comparison with words in major natural languages on which we have no intention of commenting, as we expect it never to be used again.

Each of the native predicates has one or more combining forms (originally called “affixes”, a deprecated usage; now usually called *djifoa*, the Loglan word for these forms).

Each five-letter native predicate has a djifoa formed by replacing its final vowel with **y**. This does mean that five letter predicates which have the same final vowel must be semantically very closely related (words for animals and languages can be given fine shades of meaning by adjusting the final letter; we do not intend to create further declensions of this kind, but we see nothing wrong with the ones we have).

In addition, many five-letter djifoa have one or more than one associated three letter djifoa, of one of the forms CVV, CVC, or CCV, which is formed by choosing three letters from the five letter djifoa in order of their occurrence. The process of choosing these djifoa is not likely to be modified or extended at this point, though there is some tension about the ones with doubled vowels which force stress.

There is also a short list of three-letter djifoa built from CV *cmapua*, by appending **r**, which we supply:

fer: from **fe**, five

for: from **fo**, four

fur: from **fu**, 3d place passive

jur: from **ju**, 4th place passive

ner: from **ne**, one

nir: from **ni**, zero

nor: from **no**, logical negation
nun: from **nu**, 2nd place passive, before **r**
nur: from **nu**, 2nd place passive, not before **r**
por: from **po**, state particle
rar: from **ra**, all
rer: from **re**, most of
ror: from **ro**, many of
ser: from **se**, seven
sor: from **so**, six
sur: from **su**, at least one of, some
ter: from **te**, three
tor: from **to**, two
ver: from **ve**, nine
vor: from **vo**, eight

Further, every CV cmapua unit has a corresponding CV**h** djifoa. CVV cmapua may be extended with (**h**)**y** (not with **n** or **r**) and used as djifoa: thus **zaiytrena**, A-train. This must be done with care as there may be djifoa derived from cmapua of the same shape.

Loglan complex predicates (native compound predicates) are built from sequences of these djifoa (in which the last item may be a full primitive or borrowed predicate). There are also borrowing djifoa built from borrowed predicates, which we discuss in the next section.

It is necessary to supply additional phonetic glue so that sequences of these djifoa actually can produce predicate words. Each three-letter djifoa has an alternative form with suffixed **y**. CVC**y** djifoa can be broken into syllables either as CVC-**y** or as CV-C**y**. CVV djifoa in initial position will “fall off”: so CVV**r** is available as an alternative form (which will form a consonant pair with the following djifoa or predicate word), and CVV**n** is

available as an alternative form when it is followed by **r**. Other problems are that CVC djifoa may not occur in final position, and CVV djifoa which have a doubled vowel can only occur in final or in penultimate position, because a predicate word can only contain one stress in a penultimate position. We propose for CVV**y** the alternative form CVV**hy**: a predicate starting this way will not be confused with any other kind of word, and this should be easier to pronounce distinctly.

A pronunciation difficulty with CVV**r** extended djifoa (at least for English speakers) is fixed by a permission which will probably seldom be expressed in writing, but can be: CVV**r** can take the alternative form CVV-**rr** when the VV is a mandatory monosyllable. If this is stressed, the stress falls on the VV and the vocalic continuant is an additional unstressed syllable before the final unstressed regular syllable.

Broken djifoa forms are also available to the parser, obtained from legal djifoa by inserting junctures (C-CV or C-VV or CV-C, for example). These are used to enforce the condition that borrowed predicates cannot decompose into djifoa, and it should not be possible to convert a complex (especially one which is illegal for reasons peculiar to complexes) to a legal borrowed predicate by moving junctures around.

The general point about syllable breaks is that while djifoa are not syllables, the boundaries between djifoa will be syllable breaks in a legal complex. Internal breaks are sometimes optional: a CVV djifoa with an optional monosyllable has two possible forms. The CVCCV five letter predicates may admit two forms CVC-CV and CV-CCV if the medial CC is an initial pair. The parser prefers the first form for technical reasons; it can be coerced by writing an explicit syllable break, and the latter version is often easier to pronounce.

2.2.5 Phonetic predicate words: general principles, and recognizing the beginning of a predicate

A phonetic predicate word ends with a regular vowel (so it is not a phonetic name), contains an adjacent pair of consonants (so it is not a structure word), and has penultimate stress (with the exception that an additional unstressed syllable with **y** or a vocalic continuant may intervene between the stressed and the final syllable), so that one can tell where it ends.

The part of the predicate before the consonant pair can be null (the

predicate may start with an initial pair or triple of consonants). This will be followed by a regular vowel, and there are no CC(C)V(V) predicates, in which the initial consonant group is followed just by one or two vowels.

The initial segment before the consonant pair can be an optional consonant followed by one to three regular vowels²⁷. This is the only alternative which can occur in a borrowed predicate. This can be ensured if the string starting with the consonant group doesn't satisfy the conditions given above to be the start of a predicate: the predicate must then begin with the initial (C)V(V)(V) (it cannot begin with part of the vowel sequence because one must pause before a vowel-initial word). It can also be ensured if the final syllable of the initial (C)V(V)(V) is stressed: if it were the end of a *cmapua*, it could only be followed by a predicate, and one cannot have a stressed *cmapua* followed by a predicate without pause. There is a further technical issue, which is explained below in the discussion of borrowing *djifoa*: for technical reasons, if the (C)V(V)(V) is not of one of the forms CV or CVV, the consonant group cannot be an initial pair followed by a regular vowel.

In a complex, there are other possibilities. A complex might start with one or more CVVy *djifoa*; no other sort of word can start this way. It might start with a CVCCy *djifoa*; no other sort of word can start this way. It might start with an extended CVC *djifoa*: CVCy. Again, no other Loglan word can start in this way.

We describe a lookahead test: a string which is already known not to be a possible phonetic name cannot be anything but a predicate (or ill-formed) if one of the following things are true (and one of these things will be true of any actual predicate):

1. It begins with a permissible initial group of two or three consonants and is followed by a regular vowel, but not by one or two vowels (with possible junctures) followed by a non-character, nor by a stressed vowel followed by a vowel not in a diphthong (short words of the shapes CC(C)V(V) are not predicates). Junctures may appear after the vowels. This clause of the test ignores any junctures which may appear in the initial group of consonants (to support its use in following clauses).
2. It begins with CV(V) followed by a consonant group, with either the final syllable in the CV(V) [which might extend to a juncture in the

²⁷Previous versions of Loglan have allowed arbitrarily long sequences of vowels after the initial consonant in this case, but these have never been used and I like the bound on lookahead in this test obtained by forbidding more than three vowels.

consonant group] stressed or the part of the word beginning at the consonant group not meeting the test to start a predicate above. Junctures may appear after the vowels, and there might be a juncture in the consonant group, which will be ignored in testing whether it begins a predicate.

3. It begins with (C)V(V)(V) followed by a consonant group which is not an initial pair (even one broken by a juncture) followed by a regular vowel, with either the final syllable in the (C)V(V)(V) [which might extend to a juncture in the consonant group] stressed or the part of the word beginning at the consonant group not meeting the test to start a predicate above. Junctures may be inserted after the vowels or there might be one in the consonant group, which will be ignored in testing whether it begins a predicate.
4. It begins with a consonant and a regular vowel, followed by a regular vowel followed by **y** [or **hy**], or a consonant or pair of consonants followed by **y** (with possible intervening junctures). Both of these initial sequences are possible beginnings for a predicate complex, and what follows the CVⁿ could not start any legal word in that context.

The point of including this list is making it clear that it is fairly easy to see or hear the beginning of a predicate word (though the detailed description of the cases is admittedly annoying!)

2.2.6 Borrowed Predicates and Borrowing Djifoa

After all that about djifoa, we discuss borrowings first!

A borrowing must resolve into syllables. It must end in a regular vowel. Any explicit stress must be on the second-to-last syllable, not counting syllables with vocalic continuants (one such unstressed syllable may intervene between the stressed syllable and the unstressed final syllable). The end of a predicate is determined by either non-characters such as whitespace or punctuation, or by an explicit stress. A borrowing cannot be followed without intervening whitespace or explicit pause by a vowel, nor by optional whitespace followed by a connective. An explicit stress may force monosyllabic pronunciation on a last syllable which could otherwise be pronounced as a disyllable. There can be only one explicit stress in a borrowing. The deduced

stresses in doubled vowels do not play a role in parsing borrowings, as disyllabic doubled vowels are forbidden in borrowings. It is permissible to write a borrowing with explicit junctures, but this cannot change the meaning of the word, and broken monosyllables are not permitted. A borrowing must parse correctly in the absence of explicit junctures.

The beginning of a borrowed predicate must pass the test for beginnings of predicates given above. Since borrowed predicates cannot contain **y**, this enforces the condition that there be a pair of adjacent consonants in a borrowed predicate. We reiterate that a borrowed predicate cannot have the shape CC(C)VV (which is of course not resolvable into djifoa, so could not be the shape of a complex predicate).

There are some restrictions on the phonetics of borrowed predicates. Borrowed predicates may not contain **y** or any doubled vowel other than monosyllabic **ii** and **uu**.²⁸ Borrowed predicates may contain syllables with vocalic continuants. These never follow a vowel and so far never precede a consonant, and such a syllable is always medial (not first or last). There may not be two such syllables in succession, and such a syllable cannot be stressed. The point of such syllables is that they cannot occur in complexes (with one minor exception which cannot be confused with occurrences of syllabic continuants in borrowings: a CVV**r** djifoa may be expressed as CVV**rr**, in which the syllabic continuant follows a vowel). A borrowed predicate may not resolve into djifoa, including resolutions involving broken forms in which junctures are misplaced.

Borrowing djifoa are formed from borrowings by adding final **y** and moving the stress to the final syllable of the borrowing (still penultimate in the borrowing djifoa). It is permitted to stress the penultimate syllable in a borrowing djifoa and pause after the **y**, if what follows the borrowing djifoa contains a penultimate stress. Thus one may pronounce **bakteriyrodhopsini** as **bakteri'y, rodhopsi'ni**, but one may not pronounce **iglluymao** with a pause. The stress shift is a strong signal that one is not saying **bakte'ri** but its djifoa.²⁹

²⁸Forbidding doubled vowels in borrowings was an action of ours: admitting them made reasoning about the penultimate stress in a borrowing more difficult. Exactly one predicate **alkooli** had to be changed to the better **alkoholi**.

²⁹Nothing in this situation is due to me! The strange provision to pause after a borrowing djifoa is in Loglan 1. The definition of borrowing djifoa we use was given in the 1990's. I do not know if anyone noticed the stress shift caused by the change in the definition of borrowing djifoa, but it is all a logical consequence of the way things stood at Brown's

The reason for the difference between the treatment of CV(V) and other (C)V(V)(V) prefixes in the predicate start rules is caused by the danger that a borrowing djifoa for a (C)VⁿCCV predicate might turn into a cmapua followed by a stressed CCV \mathbf{y} djifoa when a borrowing djifoa was formed (this requires the CC to be an initial pair, of course). This is not a danger when the consonant is present and $n = 1$ or 2 because a borrowing will not be of the shape CV(V)CCV with the CC initial. Thus a string (C)VⁿCCV with the CC an initial pair is not allowed to start a predicate unless the initial consonant is present and n is 1 or 2.

2.2.7 Complex Predicates

A complex predicate is formed from a sequence of djifoa in which the final element may be a full primitive or borrowed predicate (and if it is a djifoa may not be CVC, nor may it be extended with \mathbf{r} , \mathbf{n} or \mathbf{y} , nor may it contain \mathbf{y} at all). A complex may not be formed entirely from CVV \mathbf{y} djifoa and a final CVV (this prevents forms without adjacent consonants; adjacent consonants may be separated by \mathbf{y} as in **mekykiu**). A complex must pass the predicate start test. CVV djifoa may need to be extended with \mathbf{r} , \mathbf{n} , or \mathbf{y} when they appear in initial position to keep from being read as cmapua. CVC djifoa in initial position of a complex not of one of the six-letter forms CVCCVV or CVCCCV must be extended with \mathbf{y} if the final consonant of the djifoa and the following consonant would form an initial pair (a juncture does not affect this): this prevents the complex from being read as a CV djifoa followed by a borrowing.³⁰ Thus **ficynirli**, mermaid. A CVC djifoa may need to be extended with \mathbf{y} to prevent formation of an illegal consonant group with the following consonant. Thus **mekykiu**, eye doctor. Regular djifoa must be extended with \mathbf{y} when they appear before borrowing djifoa: since borrowings cannot contain \mathbf{y} , this gives us precise information about their boundaries.

Complexes must have penultimate stress among those syllables not containing \mathbf{y} : an unstressed syllable containing \mathbf{y} (or a syllable **rr** serving as glue to a stressed CVV) may intervene between the stressed syllable and the un-

death.

³⁰This rule superseded the historical **slinkui** test, which prevented the CV from falling off the front of a CVCC... complex with the CC an initial pair by forbidding the formation of borrowed predicates obtained by extending a complex initially with a single consonant forming an admissible pair: instead of forbidding **paslinkui** in favor of **pasylinkui**, **paslinkui** was permitted and **slinkui** was forbidden to be a borrowing.

stressed final syllable. The parser does know about the doubled vowel stress rule, and will not accept a complex with a CVV with a doubled vowel unless one of the vowels can carry the penultimate stress. If a CVV with doubled vowel is followed by a CVV with optional monosyllable, the monosyllabic pronunciation is forced; otherwise a CVV with optional monosyllable gives the speaker a choice about where to place the stress. The parser determines where a predicate ends either by the occurrence of explicit stress or by the occurrence of a non-character from which it back-figures the location of the stress. The only stresses in a predicate are its penultimate stress and optionally stresses in its borrowing djifoa (mandatory if the borrowing djifoa is followed by a pause).

A complex may not be followed without intervening space (a juncture doesn't help) by a vowel. A complex may not be followed by whitespace followed by a connective.

An alternative formulation allows the formation of a complex from a sequence of *emapua* units and predicate words in which the last item is a predicate word, with successive items separated by the “word” **zao**, optionally flanked on either side by whitespace or comma-marked pauses. This form might be used to avoid borrowing djifoa. I can also imagine its use to clarify the meaning of a complex by replacing its constituent djifoa with the corresponding predicates in full.³¹

Our general view is that the replacement of a djifoa in a complex by another djifoa for the same predicate or even by the full predicate linked with **zao** gives another form for the same predicate word: such forms should not appear as separate dictionary items.³²

2.2.8 Phonetic Quotes and Parenthetical Expressions

A phonetically valid quoted utterance begins with **li** and ends with **lu** with optional pauses after **li** and before **lu** and with the intervening text, which must be a phonetic utterance, optionally enclosed in double quotes (the quotes being between **li** and **lu**). Replace **li** and **lu** with **kie** and **kiu** and optional use of quotes with optional use of parentheses (opening with an open

³¹This is a proposal of John Cowan.

³²For example, we think the word **heirslicui**, molasses, presents difficulties for an English speaker and she might want to say **hekrysllicui** instead; this is precisely the same word.

parenthesis, closing with a closing parenthesis, and you have the rule for spoken parenthetical expressions.³³

We give examples: **li**, **la Djan**, **lu**; **li** “**la Djan**”, **lu**; **kie** (**ji cluva mi**) **kiu**.

2.2.9 Phonetically valid utterances

A phonetically valid unit phonetic utterance is a phonetic name, phonetic structure word, marked alien text item, phonetic predicate word, quoted or parenthesized expression, hyphen, or ellipsis, possibly with initial whitespace. A phonetic utterance is a sequence of unit phonetic utterances, explicit comma pauses, and items of terminal punctuation. Any Loglan utterance must be a phonetic utterance: of course it must also be grammatically correct, a matter for subsequent chapters.³⁴

It is useful to be aware that the parser proceeds in effect in two passes: it checks an entire utterance for phonetic validity, then checks whether it is grammatical in another pass.

2.3 Historical and philosophical note

Loglan phonetics is to our mind rather weird and wonderful.

James Jennings has commented that the choice to recognize word classes by patterns of consonants and vowels in the first instance was the “original sin of the language”. Perhaps so, but once this sin was committed, it could not be undone without discarding everything and creating a different language.

1975 Loglan had a very simple procedure for resolution of words, but a method for construction of new predicates which was ultimately found unsatisfactory. The great morphological revolution which consisted in introducing complex predicates as predicates built from djifoa and borrowings as all the phonetically acceptable predicates which were not complexes made the definition of the predicate much more complicated, and more of a challenge for the parser builder.

³³The use of punctuation here is a proposal of ours: it is quite natural but did necessitate the phonetic parser knowing about these forms.

³⁴We note that CCV djifoa are also unit phonetic utterances, strictly so that they may be quoted with **liu**. CVV djifoa are also *cmapua*, and CVC djifoa are also names, so they can be **liu**-quoted without special ceremony.

The problem of false name markers also led to a certain amount of excitement, once it was decided that **la** was too common to ban from names.

We find Loglan phonetics charming as well as weird; the language has a definite phonetic flavor and avoids monotonous regularity. We would not say that it is always easy to pronounce, and its resemblance to a Romance language can be overstated: it does allow quite a lot of consonant clustering. A further charm is that the baroque rules, however arbitrary they may seem, actually work out as almost inevitable consequences of a fairly small number of design decisions. We hope that we have given some hint in our discussion of what these design decisions were.

Chapter 3

The Grammar Proper

In this section, we will present the grammar, pausing now and then to introduce word classes that are needed; we will give short lists of commonly used words in the grammar text and full word lists in an appendix.

3.1 Simple sentence shapes

It is an interesting question where to start. We will begin with the basic Loglan sentence, and work upward to more complicated utterances and other kinds of utterance fragments, and downward to sentence components.

The simplest kind of Loglan sentence is exemplified by **La Djan, kamla** (John comes) and **La Djan, donsu le bakso, la Meris** (John gave Mary the box). This is an S(VO) sentence: each of these consists of a subject (**la Djan** in both cases), a species of noun phrase, followed by a verb phrase, which is a verb (**kamla** in the first, **donsu** in the other), followed optionally by a list of “objects” (noun phrases, none in the first example, two, **la Meris** (Mary) and **le bakso** (the box) in the second.

We pause here to discuss grammatical terminology. We have used the words “noun” and “verb” though Loglan actually has no such word classes. It does however have those functional roles, and we will use (hopefully with care) these words to help communicate what is going on in the grammar. The words **kamla**, **donsu**, and **bakso** all belong to the same word class, Loglan predicates, and can appear in any of the roles. **Ti bakso** (this is a box) is a sentence in which the very nounish (to the English mind) word **bakso** appears...as the verb! What we call a “noun phrase” can also be

called an “argument” (terminology taken from logic and also already used in Loglan grammar). Loglan grammarians have up until now used “predicate” indifferently for predicate words and for what we call “verbs” and “verb phrases”; we will continue to use “verb” and “verb phrase” as grammatical terms.

We discuss the difference between this kind of basic sentence and atomic sentences of predicate logic. An atomic sentence of predicate logic is of a form Mx (x is a man), Bxy (x is bluer than y), $Gxyz$ (x gives y to z). There is a predicate (M , B , G) and argument lists (x , xy , xyz). A sentence like Bxy might parse $(B)(x)(y)$ or perhaps $(B)((x)(y))$; the predicate and the individual arguments might be components at the same level or the predicate and the argument list might be components (the list further resolving into individual arguments).

The Loglan parse is different, in a way which brings Loglan closer to natural languages: **da mrenu**: xM , **da blanu de**: $x((B)(y))$, **da donsu de di**: $(x)G((y)(z))$. The weird thing here, from the standpoint of a logician, is the very special role of the subject: the whole sentence breaks at the top into the noun phrase subject and the verb phrase containing the verb and a list of the second and subsequent arguments. The oddities of the way this breaks down become clearly important later when we discuss logically connected predicates.

3.1.1 A brief review of components we use in example sentences

To support examples, we should say something about the components of this sentence and what we are currently putting in for these components.

Any Loglan predicate word can play the role of the “verb”. There are more complicated verbs than single predicate words, and we will see some possible additional complexities quite soon. A name such as **la Meris, la Djan Braon** is eligible to be a noun phrase (either a subject or an object). Pronouns such as **ti, ta** (this, that), or **da, de, di, do du** (pronouns referring to recently mentioned noun phrases by a scheme we will discuss below), or letter names (referring to recently mentioned noun phrases with the given initial), or noun phrases built from predicates such as **le mrenu**, (the man), are other possible arguments we may use before we have fully explained the range of possibilities for noun phrases.

3.1.2 Changes of argument order and omission of arguments

The sentence “I am better than you” is expressed **Mi gudbi tu**. The sentence “You are better than me” can of course be expressed **Tu gudbi mi**, but it can also be expressed **Mi nu gudbi tu**. **nu gudbi** is a verb, just as **gudbi** is. The effect of the particle **nu** is to reverse the first and second arguments. The particle **fu** interchanges the first and third arguments; the particle **ju** interchanges the first and fourth arguments. Compounds are possible: **nufunu** interchanges the second and third arguments, for example: **La Meris, nufunu donsu la Djan, le bakso** (Mary gave John the box). No one is going to carry out the transformation expressed by **nufunu** in three separate steps in their head during a conversation: it should be learned as a separate dictionary word. But if you work it out step by step, you will find that that is what it does. These constructions implement what would be “passives” in other languages: the Loglan term is “conversion”.

A variation implements “reflexives”. **nuo**, **fu**, **ju** have the effect of eliminating the second, third, fourth argument, respectively, by supplying the subject as that argument. **La Meris, nuo donsu la Djan** (Mary gave herself to John) or **la Meris, fu donsu le bakso** (Mary gave herself the box) exemplify this transformation. Compounds can be formed using the reflexives: for example **nufunu** eliminates the third argument by identifying it with the second.

Any Loglan predicate word has a certain number of arguments, which have a certain order in the situation it represents, which can be seen in its dictionary entry. Without special contrivances, you cannot supply the predicate with **more** arguments. But you can supply it with fewer arguments. **Mi gudbi tu** means “I am better than you”. Just **Mi gudbi** means “I am good”, with the underlying assertion being “I am better than someone”. Another example **La Meris, donsu le bakso**: “Mary gave the box away (gave it to someone)”.

This can be combined with changes of argument order: **Mi nu gudbi**, “I am bad” (I am worse than someone); **La Meris, nufunu donsu la Djan**: “Mary gave (something) to John.”. The argument omitted is always the last one, but if one changes the order of the arguments, one can put an argument one wishes to omit in the last position.

If the reader wonders why we introduce this transformation of verbs here, they should note two things: as we will see later, this is one of the most

tightly binding operations on verbs, and further, it acts exactly on the very simplest features of the structure of the simple Loglan sentence.

3.1.3 Tenses and variations

In this section we introduce tenses of the Loglan verb, which do not necessarily have anything to do with time. Tense is achieved using a structure word (either **ga** or a word of the PA class): the simplest examples are **na**, **pa**, **fa**. the present, past, and future tenses. A nontemporal examples is **vi** (here). There is also a null tense **ga**, which is used in situations where it is grammatically useful to have a tense but we do not actually want to say anything about the time, place or conditions of the assertion.

la Meris, pa cluva la Djan Mary loved John

Mi fa nufunu dons tu I will give you something

La Ailin, vi danse Eileen dances here

Le mrenu ga sadji The man is wise (in general, no commitment to a particular time). Here the **ga** is grammatically a tense but it doesn't add anything to the semantics. It is needed, because as we will see later, **le mrenu sadji** is not a sentence, but a noun phrase, "the wise man".

We aren't introducing tenses at length: we actually need to introduce them in order to describe a further manipulation of basic sentences. Notice that **nufunu dons** is tensed, rather than **fa dons** being converted: the tense is much more loosely attached than the conversion operator. In fact, the tense attaches to the verb phrase as a whole rather than to the verb.¹

3.1.4 Variations in sentence order

We can put the subject in a sentence after the verb in two ways.

The first kind of sentence we can produce has a tensed verb phrase with its objects (it might be tensed with **ga** strictly for grammatical purposes) followed optionally by **ga** then the subject:

Ga gudbi tu ga mi I am better than you

Ga gudbi tu There are better than you (here the subject is omitted!)

Ga dons le bakso la Djan, ga la Meris Mary gave the box to John

¹It is actually possible to convert a sort-of-tensed verb but it is tricky: **Mi nufunu ge dons je fa gue tu**, in which quite a lot is going on which we will not explain yet!

Nia nu gudbi tu ga mi You are being better than me (combining conversion of the predicate with reordering of sentence components!) The tense word **nia** is the present progressive.

The second kind of sentence with subject delay consists of a tensed verb phrase with no objects followed by **ga** then all the arguments in the sentence.

Ga donsu ga la Meris, le bakso la Djan.

This allows us to achieve VOS and VSO word orders.²

We can put some objects before the verb, if we separate those objects from the subject with the particle **gio**.

Mi gio le bakso ga donsu tu I give the box to you (the tense **ga** is actually needed to keep from saying **le bakso donsu**.the boxy giver), or

Mi gio le bakso tu ga donsu

This supports SOV(O) sentence order.

The particle **gio** may optionally be used to set the subject apart from the objects in a VSO sentence:

Ga donsu ga la Meris, gio le bakso la Djan.

There is another device for modifying sentence order by bringing objects to the front, but this device cannot be properly introduced until after we discuss logically connected sentences.

3.1.5 Modifiers and tagged arguments

Tense, location and modal operators (the same words which can decorate verb phrases as tenses) can form *sentence modifiers* which are rather like additional arguments which can be supplied with any verb. In English grammar, these would be “prepositional phrases”.

Relative modifiers and arguments (including the tagged arguments introduced below) are called terms. The subject and the list of objects in a verb phrase are both term lists (of slightly different kinds, as we will see). The subject is an arbitrary list of terms containing at least one argument and no

²It is a reform of ours to require that gasents (the Loglan jargon for subject-delayed sentences) must have either exactly one argument delayed or all arguments delayed. We want to avert listeners being forced to retroactively change their understanding of the meanings of arguments appearing earlier as in ***Ga donsu le bakso, ga mi tu** in which one would reasonably start out thinking that **le bakso** was the object being given (the second argument) but the presence of two arguments after **ga** (permitted in 1989 Loglan) forces the listener to revise this: the actual meaning of the sentence would be “I gave you to the box”.

more than one untagged argument. It is important to notice that if the list of terms before the verb in a sentence does not contain any arguments, the sentence will be either a gasent (if tensed) or an imperative (if not tensed). The object list can contain no more than four untagged arguments (since there is no predicate taking more than five arguments).

Formally, a relative modifier is either a word of class PA (a tense, location or modal operator) followed by a noun phrase, followed optionally by the particle **guua**³ or the general right closer particle **gu**, or simply a PA word followed optionally by **gu**. In the last kind of sentence modifier, you should suppose that the omitted argument of the PA word is the present situation in which the speaker is delivering their speech (the referent of the pronoun **tio**).

Lists of PA words will appear in the next section.

Examples of such modifiers:

Vi la Djan, mi bleka le nirda Near John I watched the bird
mi bleka le nirda, vi la Djan

Mi godzi na I go now (here the “now” is not a tense but a sentence modifier).

The relationship of a sentence modifier to the sentence is exactly the same no matter where it appears in the sentence. It modifies the verb phrase, or equivalently, the entire situation represented by the sentence, not an argument it happens to be near.

Mi bleka le nirda vi la Djan means that I was near John when I was watching the bird.

Mi bleka le nirda ji vi la Djan means that I was watching the bird which was near John (a different grammatical construction, the subordinate clause, which we have not seen yet).

A modifier or modifiers may appear before the **ga** or tense in a subject-delayed sentence.

Na la Ven, pa kamla ga la Djan John will come at nine

Tagged arguments are arguments which are allowed to float free in a sentence in the same way that relative modifiers do. This can be done with numerical place tags or case tags.

The numerical place tags **zua**, **zue**, **zui**, **zuo**, **zuu** are signs of the first, second, third, fourth and fifth argument of a predicate. This allows argu-

³**guua** and some other right closers are new; we did a survey of grammatical constructions closable with **gu** and provided special forms for most of them.

ments to be freely reordered and moreover allows medial arguments to be omitted.

Zui la Djan, donsu zua la Meris Mary gave (something) to John.

It also allows arguments to be placed with the subject, as long as at most one argument in the subject is untagged:

La Meris, zui la Djan, pa donsu le bakso

It also allows more than one argument to be supplied for the same place.

Zua la Meris, zua la Djan, cluva la Ailin Mary and John love Eileen.

Untagged arguments are taken as usual to represent the places of the argument in order, skipping places corresponding to numerical place tags (or case tags) which have already appeared earlier in the sentence. We do not require a listener or reader to displace a sequence of arguments when a **zua** is encountered at the end of a sentence. Numerical place tags have a special effect in term lists appearing before the particle **gi**, which we will describe below.

The case tags are a bizarre idea which *we* would not have installed in this language. This is not to say that similar ideas do not occur in natural languages. With each place of a Loglan predicate, a *case* is associated in the dictionary, and that case tag may be used to reference that argument of that particular predicate in the same way the appropriate numerical case tag would reference it. Another possible use of a case tag is to suggest an argument whose place the speaker has forgotten, or perhaps an argument of the predicate which does not appear in the dictionary!

There is a further issue that the dictionary includes words in which distinct arguments have the same case. To support this, we have recently provided forms which reference the first, second, third, etc. argument of a given case.

The list of case tags will appear in the next section.

We give examples of use of these tags.

La Djan, dio la Meris, cluva John loves Mary. This is another way to get SOV order, and notice that **gio** is not needed.

Dio la Meris, (kao) la Djan, cluva, with the same meaning. The nominative case tag **kao** is optional: it can be used, which illustrates the fact that the subject needs to contain at least one argument and at most one untagged argument (two tagged arguments are all right).

La Djan, pa donsu dio la Meris, le bakso John gave the box to Mary. The **dio** indicates which argument **la Meris** is (she is not being given as a gift) and **le bakso** falls tidily into the first unused argument place.

Dio la Meris, beu le bakso, donsu la Djan means the same thing as the previous sentence.

3.1.6 Imperatives (and observatives)

An untensed sentence consisting of a verb followed by an object list, with possibly some modifiers before the verb, is an imperative.

Donsu le bakso la Djan! Give the box to John.

Na la Ven, donsu le bakso la Djan! At nine, give the box to John.

If a tensed verb followed by an object list (possibly preceded by modifiers) is given, this is actually a subject-delayed sentence with the subject omitted. We call this an *observative*: we note it as a special form mostly to indicate that such sentences are not imperatives.

Na crina It is raining (literally, someone is being rained on). This is a shortening of **Na crina ga ba**

Na donsu le bakso la Djan Someone is giving the box to John.

As an experiment, Loglan has borrowed a concept from Lojban and installed the imperative pronoun **koo**. This is used just like **tu** (you) with the extra force that the usual referent of **tu** is commanded to make the statement true.

Koo donsu le bakso la Djan! Give the box to John!

but also

La Meris, cluva koo! Make Mary love you!

Mi jupni lepo koo gudbi! Make me think well of you! (lit. Make me think you are good).

The imperatives with **koo** are sentences of perfectly general structure and do not belong to the imperative grammatical class which is the subject of this section, usually, though consider

Cluva koo! Love yourself!

3.2 Logically connected sentences

3.2.1 Forethought connected sentences

Chapter 4

Lexicography Appendix: full word lists

4.1 Case tags and indirect reference particles

The case tags, including the positional ones are listed:

beu: (patients/parts),

cau: (quantities/amounts/values),

dio: (destinations/receivers),

foa: (wholes/sets/collectives),

kao: (actors/agents/doers),

jui: (lessers),

neu: (conditions/circumstances/fields),

pou: (products/purposes),

goa: (greaterers),

sau: (sources/reasons/causes),

veu: (effects/states/effects/deeds/means/routes),

zua: (first argument),

zue: (second argument),

zui: (third argument),

zuo: (fourth argument),

zuu: (fifth argument),

lae: (lae X = what is referred to by X),

lue: (lue X = something which refers to X)

The operators of indirect reference **lae** and **lue** are a different sort of creature, which originally had the same grammar as case tags, but now have somewhat different behavior. The latter two operators can be iterated (and so can case tags, probably indicating that more than one applies to the same argument).

For each semantic case tag there are forms like **beuzi**, **beucine** to reference the first argument with that tag, **beuza**, **beucito** to reference the second argument with that tag, and **beuzu**, **beucite** to reference the third argument with that tag. Forms like **beucifo**, **beucife** are theoretically possible.

Chapter 5

The Formal Grammar in PEG Notation

This chapter contains the actual Parsing Expression Grammar (PEG) notation in which the formal grammar is represented: this is the source from which the computer parser is constructed. It is also intended to be the basis of the presentation of the grammar.

I found a package which makes it so that this file can be included here in a way which does not run off the margins. Lines are numbered, but this should not be taken too seriously. Certainly any line number references in the text above should be checked whenever the grammar file source is modified. We do not have the text embedded in this file: the current contents of the file are read in.

```
1 # In this file I will develop the entire Loglan
   grammar on top of the phonetic proposal
2
3 # Dated updates now to appear here
4
5 # 2/4/2021 Imposed the rule that two final
   consonants cannot be consonants from voiced/
   unvoiced pairs
6 # with different voice. Also forbid second final
   consonant to be h.
7
8 # I have further fine-tuning of djifoa gluing in
```

```
mind.  
9 # Allow the -r glue to be expressed as  
10 # -rr after all mandatory monosyllables, removing  
    the annoying pronunciation problem?  
11 # I was thinking of allowing -hy gluing in other  
    contexts, but it is actually a bad idea.  
12  
13 # 9/15/2019 installed semantic case tags with order  
    distinctions for use with predicates with more  
    than one argument of the same case.  
14 # one solution is beucine, beucito... another is  
    beuzi, beuza, beuzu.  
15  
16  
17 # 4/28/2019 Various debugging of the new predicate  
    algorithm. Added CVVhy as a glued form for CVV  
    djifoa.  
18 # added capitalization of djifoa glue! Confirming  
    my apparent earlier decision that a CVV(h)y  
    djifoa must be followed  
19 # by a full predicate complex.  
20  
21 # 4/26/2019: this incorporates various revisions to  
    the phonetics, correcting errors or clarifying  
    rules,  
22 # motivated by my development of the phonetics  
    section of a new grammar document. The one  
    notable  
23 # change is that <ci> is now only a name marker if  
    followed by an explicit pause. This only  
    requires  
24 # changes in writing in serial names. In speech, it  
    is recommended that one not pause after <ci>  
25 # except before a name word. The benefit is that  
    non-serial-name related uses of <ci> no longer  
26 # threaten mysterious needs to add explicit pauses  
    before following name words.  
27
```

28 # 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)

29

30 #4/25 Making note of the idea that <ci> should not
be a name marker unless followed

31 # by a pause. This would require that one pause
before ci-marked names and it would

32 # remove some very confusing corrections for the
false name marker problem. If we

33 # required the pause to be explicit we would be
imposing the expectation that whitespace

34 # after <ci> is not a pause. Otherwise we could
encourage writing a juncture after <ci>

35 # to deny presence of a pause, which is reasonable
considering the meanings of <ci>.

36 # I am implementing the version with explicit pauses
between <ci> and names

37 # and the directive not to pause after <ci> without
explicit indication. This solution

38 # involves rewriting existing text only in the rare
instances where <ci> precedes a name.

39

40 # 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.

41

42 # 4/24/2019 Final consonants in syllables cannot be
followed by syllabic continuants.

43 # this rationalizes the definition of SyllableA.

44

45 # 4/22 I am thinking of explicitly flagging
imperative sentences; not changing

46 # the grammar but making this visible in the parse.
This might also have some

47 # effects on logical connections. 4/23 created an
imperative class for atomic
48 # imperative sentences; this has no actual effect
on parses, just
49 # organizes them in a more enlightening way.
50
51 # 4/17-18 2019: updates commented out which make
sentpred linkable with forethought
52 # and afterthought connectives (making some uses of
<guu> to share arguments
53 # unnecessary). There are subtleties. Basically,
untensed predicates without
54 # argument lists will be linked by A and KA series
connectives. Such a linked
55 # set can be tensed as a whole. Such a linked set
will share a following termset.
56 # This will probably change many parses in the Visit
and other legacy sources.
57 # This required some really subtle adjustments to
work right, divivable from
58 # the actual rules given. Definitely experimental.
59
60 # 3/9/2019 further, extended LIU1 to handle <ainoi>
and its kin
61 # (actual mod is to class Cmapua) Further, fixing
mismatch
62 # between connective and A classes. One does now
have to pause
63 # before <ha> and its compounds.
64
65 # 3/9/2019 repaired bugs in negative attitudinals.
A pause
66 # in a negative attitudinal of the <no, ui> form
will not break
67 # it. <ainoi> didnt work for two reasons: the
clauses
68 # in the definition of NOUI were in the wrong order,
and

69 # the connective class mistakenly included <noi> so
the
70 # phonetics checker was crashing! I had to move N
and NOI
71 # earlier to make this work. Not yet installed in
the other
72 # version.
73
74 # 1/26/2019 added <vie>, JCB's "objective
subjunctive" as a PA
75 # class word. I should add this to the other file
as well.
76
77 # 12/22/18: just a comment: one does not have to
pause before <ha> and its compounds.
78 # I do not know whether to fix this. One did not
have to in LIP either. For the moment I will
79 # leave it as it is. As a matter of style, one
probably should pause.
80
81 # 10/6/18 minor adjustments, made only in this file
. Allow <sujo> (a wicked thing to say). Do not
82 # allow <futo>: suffixed conversion operators must
be nu + suffix.
83
84 # 6/2 fixed LIO + alien text. I also fixed some
other glitches described in the reference grammar
. .
85
86 # 5/11 making version without "alternative parser"
features. This version allows GAA but it doesn't
87 # do anything: the definitions of argumentA and kin
are the only point of difference. Master
version:
88 # becomes "alternative" by reinstating alternative
definitions of argumentA and kin. Further, made
changes
89 # recommended in the reference grammar. ALTERNATIVE

```
-- this is actually my master version.  Edit
90 # this and revise the argumentA and kin entries to
    make the original version.
91
92 # 4/24 discovered and repaired a bug re ci-marked
    names suffixed to descriptions.  Discovered a bug
    in numerical
93 # descriptions yet to be fixed: <lio> needs to be
    an alien text marker, maybe taking double quotes.
    The description-
94 # with-suffixed-name bug was actually quite gruesome
    .  I think it is repaired.
95
96 # 4/23 streamlined definition of descriptn.  Shouldn
    't change anything.  It was remarkably tricky
    though; preserving the old form
97 # in case of further trouble.
98
99 # 4/22 I think this will be the master grammar file
    , with alternative lines to turn off the
100 # GAA-related features.
101
102 # 4/22 allowing general predicates in gasent1.  This
    removes an extreme oddity in parsing of
    imperatives.
103 # I do not see any new dangers from this.
104
105 # 4/22 I changed the final element of a keksent to
    be a sentence (new class uttA0), not a general
    sentence fragment.
106 # several parse errors in the Visit were uncovered
    by this.
107
108 # 4/22: note that I still have the obligation to
    restore the <zao> construction.
109
110 # 4/9/2018 the large subject marker GAA can also be
    used to defend the beginnings of gasents and
```

imperatives

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

112

113 # 4/8/2018 this is an alternative version in which
an argument which starts an SVO sentence will not
be accepted

114 # as a trailing argument of a previous sentence.
This allows neat termination of <lepo> clauses
preceding

115 # a subject, for example. Unlike the previous
alternative approach, this seems to involve a
single fairly

116 # tidy change: it is all an issue of avoiding needs
for explicit closure. Further refinement: SVO
sentences

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

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

119 # starts an SVO sentence *not marked with GAA* will
not be accepted as a trailing argument of a
previous

120 # sentence. This is a sufficiently complex grammar
change that it requires thought: it is not
conservative

121 # in my usual sense. The fact that GAA carries a
mandatory stress is virtuous. Its resemblance to
the

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

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

124 # and often will be followed by a tense. This
 grammar change depends strongly on the previous
 ruling that the 0 in
125 # SOV(0) sentences must be marked with <gio>: S gio
 Oⁿ V (O^m).

126
127 # nuu is an atomic A core and there is no nu-affix
 to A connectives and their kin

128
129 # 1/20/2018 redefined CA cores to include a possible
 NU prefix. This allows more logically connected
 tenses, for example.

130
131 # 1/13/2018 reorganized the internals of class PA in
 a way which should allow more things and not
 forbid anything legal now.

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

133 # proposal document. Enforced explicit pauses after
 PA phrases appearing as arguments with a
 following modifier with an argument.

134
135 # 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
136 # as dimensions).

137
138 # 12/27/2017 installing an alternative treatment of
 acronyms under which they are simply names (
 suffix -n to acronyms in all uses).

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

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

141

142 # 12/27/2017 Frivolously fooling with the
capitalization conventions. They ought to work
better now...but I could have broken something.

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

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

145 # 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,
146 # 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
147 # after z- (after which lower case resumes) or an
embedded literal (after which lowercase resumes).

The total effect is that this allows

148 # attested capitalization patterns in Loglan (
including capitalization of embedded literals as
in possessive articles and acronyms)

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

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

151

152 # 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

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

154

155 # I note in this connection that quotation of names

```
with li...lu remains limited, since names by
themselves are not
156 # utterances: one needs the <la>. I fixed this as
an exception in the previous parser; I may do it
here or I may
157 # not, haven't decided. Single name words can be
quoted with <liu>, of course, but not serial
names.
158
159 # 12/24/2017 Refined treatment of vowel pairs for
Cvv-V cmapua units. First 12/24 version rather
disastrously
160 # broken: this should be fixed!
161
162 # 12/23/2017 This is now completely commented, with
minor local exceptions to which I will return
later.
163 # This document is the basis on which I will build
all subsequent parsers, with due modifications to
the comments.
164 # The Python PEG engine and preamble files contain
commands for constructing a Python parser from
it directly.
165
166 # 12/22/2017 major progress on commenting the
grammar
167
168 # yet later 12/20: no change in performance of the
grammar, extensive commenting in the
169 # grammar section. Considerable changes in
arrangement: for example, vocatives, inverse
vocatives,
170 # and free modifiers are moved to a much earlier
point. I'm hoping to get a genuinely almost
readable
171 # commented grammar...
172
173 # later 12/20 starting the process of commenting
```

and editing the grammar, starting
174 # at basic sentence structures. Notably rewrote the
class [keksent] more compactly,
175 # one hopes with no actual effect on parses.
176
177 # 12/20/2017 Do not require expression of pause
after finally stressed cmapua before
178 # vowel initial predicate as a comma, since the
initial vowel signals the pause anyway.
179 # Allow final stress in names. Fixed bug in
CVVHiddenStress. Prevented
180 # broken monosyllables in finally stressed CVV
djifoa. refinement of caprule
181
182 # 12/19/2017 seem to have had a versioning failure
and lost the fix which requires
183 # CVVy djifoa to be followed by complete complexes.
Restored.
184
185 # 12/18/2017 fixed a bug in treatment of stressed
syllables in recognizing predicate starts. Also
186 # narrowed the generalized VCCV rule to allow more
of the quite unlikely space of predicates with
lots
187 # of vowels before the CC pair. Probably they
should be banned (and none have ever been
proposed with
188 # more than three) but that rule is not the context
in which to arbitrarily ban half of them. Some
cleanup
189 # of the display of parses, for which updated
version of logicpreamble.py should also be
uploaded. A refinement
190 # to class "connective" checking that apparent
logical connectives are not initial segments of
predicates.
191 # This has the effect of delaying the declaration of
"connective" until after the declaration of

```
192 # "predstart".
193
194 # 12/17/2017 further refinement of the 12/16 version
      : a couple of bugs spotted.
195
196 # 12/16/2017 There should be no change in parsing
      behavior, but the predstart ruleset is shorter
197 # and more intelligible, and I realized that Complex
      doesnt need a check for the anti-slinkui test
198 # (the requirement that certain initial CVC cmapua
      be y hyphenated which replaces the slinkui test))
199 # at all: the way predstart works already ensures
      that initial CV cmapua fall off in the excluded
200 # cases, the idea being that we test the front of a
      predicate without lookahead in all cases. Also
201 # addressed the subtle point that one wasn't forced
      to pause after a predicate before following y
202 # (not likely to arise as a problem).
203
204
205 # 12/14/2017 Corrected vowel grouping to avoid
      paradoxical vowel triples which are default
206 # grouped in a way which becomes illegal if made
      explicit. SyllableA really should contain a
      final
207 # consonant: the previous form was messing up vowel
      grouping. Serious bug where end of djifoa
208 # and syllable resolution of a predicate may fail to
      agree. I think I blocked this by ensuring that
209 # final djifoa are not followed by vowels. Other
      fine tuning of the complex algorithm. Also had
210 # to repair the check for CVCCCV and CVCCVV
      predicates.
211
212
213 # 12/13/2017: added kie ( utterance ) kiu to class
      LiQuote. Did fine tuning to ensure
214 # that cmapua streams stop before <li> or <kie>,
```

that names can stop at double quotes or close
215 # parentheses, and that the capitalization rule
ignores opening parentheses as well as double
216 # quotes. One can now adorn li lu with quotes (on
the inside) in a reasonable way
217 # and adorn kie kiu with parentheses (on the inside)
in a reasonable way. One cannot
218 # *replace* these words (or any words) with
punctuation in my model of Loglan. Also,
219 # updates to comments, and # (end of utterance)
added as a marker of terminal punctuation.
220
221 # END of dated updates
222
223 # This is now done, in a first pass. That is, the
grammar is adapted and appears to work, more or
less.
224 # What is needed is comments on the lexicography and
the grammar...Phonetics has now pretty clearly
been sorted
225 # from the grammar (there are some places where the
phonetics accept grammar information with regard
to punctuation).
226
227 # Alien text is now handled somewhat differently.
Some issues to do with quoting names are not
finalized and have not been tested.
228
229 # I added -iy and -uy as VV forms allowed in general
in cmapua but not in other words; they are
always monosyllabic. What this
230 # immediately allows me to do is to give Y a name
which is not phonetically irregular! <ziy> is
supported: <yfi> is too, now.
231
232 # capitalization is roughly back to where it was in
the original, but all-caps are allowed.
233

```
234 # acronyms are liable to be horrible.
235
236 # Fixed the recursion problem in a way which will
      not be visible in ordinary parses. Streams of
      cmapua will always
237 # be broken at name or alien text markers (instead
      of using lookahead to check that we do not stand
      at the beginning
238 # of a name word or alien text word). The next
      cycle will then check for a name or alien text,
      and also check for
239 # badnamemarkers; no lookahead is happening while a
      stream of cmapua is being read except checking
      for
240 # the markers of names and alien text. This will
      change the way phonetic parses look (streams of
      cmapua will
241 # break (and sometimes resume) at name markers or
      alien text markers, but it will not change any
      grammatical
242 # parses.
243
244 #Part I Phonetics
245
246 # Mod bugs, I have implemented all of Loglan
      phonetics as described in my proposal. Borrowing
      djifoa are pretty tricky.
247
248 # I have now parsed all the words in the dictionary,
      and all single words of appropriate classes
      parse successfully.
249 # I have added alien text and quotation
      constructions which do not conform to these rules
      ; so actually
250 # all Loglan text should parse, mod some
      punctuation and capitalization issues. The
      conventions for
251 # alien text here are not the same as those in the
```

current provisional parser.

252

253 # I believe the conventions for forcing comma pauses
before vowel initial cmapua and after names

254 # except in special contexts have been enforced. In
a full grammar, one probably would want

255 # to disable pauses before vowel initial letterals (
done). This grammar also does not support the
lingering

256 # irregularities in acronyms (and won't).

257

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

259 # pauses or name markers), cmapua (qua unanalyzed
streams of cmapua units),

260 # borrowings and complexes, along with interspersed
comma pauses and marks

261 # of terminal punctuation. It does support
conventions about where commas are required

262 # and a simple capitalization rule. Streams of
cmapua break when markers initial

263 # in other forms are encountered (and may in some
cases resume when the markers

264 # are a deception).

265

266 # a likely locus for odd bugs is the group of
predstartX rules which detect apparent cmapua
which

267 # are actually preambles to predicates. These are
tricky! (and I did indeed find some lingering

268 # problems when I parsed the dictionary). Another
reason to watch this rule predstart

269 # is that it carries a lot of weight: !predstart is
used as a lightweight test

270 # that what follows is a cmapua (a point discussed
in more detail later).

271

- 272 # In reviewing this, I think that very little is
different from 1990's Loglan (the borrowing
djifoa
- 273 # are post-1989 L1, but not my creation). Some
things add precision without making anything in
1990's Loglan incorrect.
- 274 # The requirement that syllabic consonants be
doubled is new, and makes some 1990's Loglan
names incorrect.
- 275 # The requirement that names resolve into syllables
is new, and makes some 1990's Loglan names
incorrect,
- 276 # usually because they end in three consonants.
- 277 # The rule restricting final consonant pairs from
being noncontinuant/continuant is new, but
- 278 # does not affect any actual predicate ever
proposed.
- 279 # Enhancing the VccV rule to also forbid CVVV...ccV
caused one predicate to be changed
- 280 # (<haiukre> became <haiukrre>, and haiukre was a
novelty anyway, using a new name for X in X-ray)
- 281 # The exact definition of syllables and use of
syllable breaks and stress marks is new (the
close comma
- 282 # was replaced with the hyphen, so Lo,is becomes Lo
-is); but this does not make anything in 1990's
Loglan
- 283 # incorrect, it merely increases precision and
makes phonetic transcript possible.
- 284 # Forbidding doubled vowels in borrowings was new,
was already approved, and caused us to change
- 285 # <alkooli> to <alkoholi>.
- 286 # Formally allowing the CVccVV and CVcccV predicates
without y-hyphens took a proposal in 2013
because
- 287 # Appendix H was careless in describing their
abandonment of the slinkui test, but the
dictionary

288 # makes it evident that this was their intent all
along. The slinkui test had already been
289 # abandoned in the 1990s.
290 # Formally abandoning qwx was already something that
the dictionary workers in the 1990's were
working
291 # on; we completed it.
292 # Allowing glottal stop in vowel pairs and
forbidding it as an allophone of pause is a new
phonetic
293 # feature in the proposal but not reflected in the
parser, of course. Alternative pronunciations
of
294 # y and h and allowing h in final position are
invisible or do not make any 1990's Loglan
incorrect.
295 # Permitting false name markers in names was already
afoot in the 1990's and the basic outlines of
our
296 # approach were already in place. The rule
requiring explicit pauses between a name marker
not starting
297 # a name word and the beginning of the next name
word is new, but reflects something which was
already
298 # a fact about 1990's Loglan pronunciation: those
pauses had to be made in speech
299 # (and in the 1990's they had no tools to do
relevant computer tests)! The requirement
300 # that names resolve into syllables restricts which
literal occurrences of name markers are actually
301 # false name markers (the tail they induce in the
name must itself resolve into syllables).
302 # Working out the full details of borrowing djifoa
was interesting: I'm not sure that I've done
anything
303 # *new* there; explicitly noting the stress shift
in borrowing djifoa might be viewed as something

```
304 # new but it is a logical consequence of JCB's
      permission to pause after a borrowing djifoa,
      which contains
305 # explicit language about how it is to be stressed,
      and the
306 # final definition of a borrowing djifoa as simply
      a borrowing followed by -y. The shift strikes
307 # me as a really good idea anyway, because it marks
      djifoa with a pause after it as phonetically
      different
308 # in an additional way other than ending with the
      very indistinct vowel y. My rules as given here
      do not
309 # directly enforce the rule that a borrowing djifoa
      must be preceded by y but I think they
      indirectly
310 # enforce it in all or almost all cases: the
      parser tries to read a borrowing djifoa before
      reading
311 # any other kind of djifoa, so it is hard to see
      how to deploy a short djifoa in such a way that
      it would
312 # fall off the head of a borrowing without using y.
313 # These phonetics do not support certain
      irregularities in acronyms. We note that
314 # it is now allowed to insert <, mue> into an
      acronym, which would be necessary for example
315 # between a Ceo letteral and a following VCV
      letteral.
316
317 #Sounds
318
319 #all vowels
320
321 V1 <- [aeiouyAEIOUY]
322
323 #regular vowels
324
```

```
325 V2 <- [aeiouAEIOU]
326
327 #consonants
328
329 C1 <- [bcdfghjklmnpqrstvzBCDFGHJKLMNPRSTVZ]
330
331 #consonants in voiced/unvoiced pairs
332
333 Cvoiced <- [bdgjvzBDGJVZ]
334
335 Cunvoiced <- [ptkcfsPTKCFS]
336
337 # bad voice pair (or pair second term of which is h)
338 # forbidden as pairs of final consonants
339
340 Badvoice <- (Cvoiced (Cunvoiced/[Hh])/Cunvoiced (
    Cvoiced/[Hh]))
341
342 # letters
343
344 letter <- (![qwxQWX] [a-zA-Z])
345
346 # a capitalization convention which allows what our
    current one allows and also allows all-caps.
347 # if case goes down from upper case to lower case,
    it can only go back up in certain cases. This
348 # does allow capitalization of initial segments of
    words. There is a forward reference to the
    grammar
349 # in that free capitalization of embedded literals
    is permitted, and capitalization of vowels
350 # guarded with z in literals as in DaiNaizA.
351
352 lowercase <- (![qwx] [a-z])
353
354 uppercase <- (![QWX] [A-Z])
355
356 caprule <- [\"(]? &([z] V1 (!uppercase/&TAIO)/
```

```

lowercase TAI0 (!uppercase/&TAI0)/!(lowercase
uppercase.) letter (&([z] V1 (!uppercase/&TAI0)/
lowercase TAI0 (!uppercase/&TAI0)/!(lowercase
uppercase.) (letter/juncture))* !(letter/
juncture)
357
358 # syllable markers: the hyphen is always medial so
      must be followed by a letter.
359 # the stress marks can be syllable final and word
      final. A juncture is never followed
360 # by another juncture.
361
362 juncture <- (([-] &letter)/[\'*]) !juncture
363
364 stress <- [\'*] !juncture
365
366 # terminal punctuation
367
368 terminal <- ([.:?!;#])
369
370 # characters which can occur in words
371
372 character <- (letter/juncture)
373
374 # to really get all Loglan text, we should add the
      alien text constructions and the markers of alien
      text,
375 # <lie>, <lao>, <sao>, <sue> and certain quotations
      which violate the phonetic rules.
376
377 # we adopt the convention that all alien text may be
      but does not have to be enclosed in quotes.
378 # it needs to be understood that in quoted alien
      text, whitespace is understood as <, y,>; in the
      unquoted
379 # version this is shown explicitly. This handling
      of alien text is taken from the final 1990's
      treatment

```

```

380 # of Linnaeans = foreign names, and extended by us
      to replace the impossible treatment of strong
381 # quotation in 1989 Loglan.
382
383 # this is a little different from what is allowed in
      the previous provisional parser, but similar.
384 # A difference is that all the alien text markers
      are allowed to be followed by the same sorts of
      alien text.
385
386 # the forms with <hoi> and <hue> are required to
      have following quotes in written form to avoid
387 # unintended parses, which otherwise become likely
      in case of typos in non-alien text cases.
388
389 AlienText <- ([,]? [ ]+ ["] (!["].)+ ["]/ [,]? [
      ]+ (![, ]!terminal .)+ ([,]? [ ]+ [y] [,]? [ ]+
      (![, ]!terminal .)+)*
390
391 AlienWord <- &caprule ([Hh] [Oo] [Ii] juncture?
      &([,]? [ ]+ ["])/[Hh][Uu] juncture? [Ee]
      juncture? &([,]? [ ]+ ["]) / [Ll] [Ii] juncture?
      [Ee]juncture? /[Ll] [Aa] [Oo]juncture? /[Ll] [Ii]
      ] juncture? [Oo] juncture? /[Ss] [Aa] [Oo]
      juncture?/[Ss] [Uu] juncture? [Ee]juncture?)
      AlienText
392
393 # while reading streams of cmapua, the parser will
      watch for the markers of alien text.
394
395 alienmarker <- ([Hh] [Oo] [Ii] juncture? &([,]? [ ]+
      ["])/[Hh][Uu] juncture? [Ee] juncture? &([,]? [
      ]+ ["]) / [Ll] [Ii] juncture? [Ee] juncture? /[
      Ll] [Aa] [Oo] juncture? /[Ll] [Ii] juncture? [Oo]
      juncture? /[Ss] [Aa] [Oo] juncture?/[Ss] [Uu]
      juncture? [Ee] juncture?) !V1
396
397 # 5/11/18 added <lio> as an alien text marker, to

```

```

        support numerals.
398
399 # the continuant consonants and the syllabic pairs
        they can form
400
401 continuant <- [mnlrMNLr]
402
403 syllabic <- (([mM] [mM] !(juncture? [mM]))/([nN] [nN]
        ] !(juncture? [nN]))/([rR] [rR] !(juncture? [rR])
        )/([lL] [lL] !(juncture? [lL])))
404
405 # the obligatory monosyllables, and these syllables
        when broken by a usually bad syllable juncture.
406 # The i-final forms are not obligatory mono when
        followed by another i.
407
408 MustMono <- (([aeoAEO] [iI] ![iI]) /([aA] [oO]))
409
410 BrokenMono <- (([aeoAEO] juncture [iI] ![iI])/([aA]
        juncture [oO]))
411
412 # the obligatory and optional monosyllables.
        Sequences of three of the same letter
413 # are averted. Avoid formation of doubled i or u
        after ui or ui.
414
415 Mono <- (MustMono/([iI] !([uU] [uU]) V2)/([uU] !([iI]
        ] [iI]) V2))
416
417 # vowel pairs of the form found in cmapua and djifoa
        .
418 # (other than the special IY, UY covered in the
        cmapua rules)
419
420 # The mysterious prohibition controls a permitted
        phonetic exception in djifoa gluing.
421 # compua are never followed directly by vocalic
        continuants in any case.

```

```

422
423 VV <- (!(MustMono V2 juncture? V2 juncture? [Rr] [Rr
      ]) (!BrokenMono V2 juncture? V2)
424
425 # the next vocalic unit to be chosen from a stream
      of vowels
426 # in a predicate or name. This is different than in
      our Sources
427 # and formally described in the proposal.
428
429 NextVowels <- (MustMono/(V2 &MustMono)/Mono/!( [Ii]
      juncture [Ii] V1) !( [Uu] juncture [Uu] V1) V2)
430
431 # 5/11/18 forbidding consonantal vowels to follow
      the same vowel.
432
433 # the doubled vowels that trigger the rule that one
      of them must be stressed
434
435 DoubleVowel <- (([aA] juncture? [aA])/([eE] juncture
      ? [eE])/([oO] juncture? [oO])/([iI] juncture [iI]
      )/([uU] juncture [uU])/[iI] [Ii] &[iI]/[Uu] [uU]
      &[uU])
436
437 # the mandatory "vowel" component of a syllable
438
439 Vocalic <- (NextVowels/syllabic/[Yy])
440
441 # the permissible initial pairs of consonants, and
      the same pairs possibly
442 # broken by syllable junctures.
443
444 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]))

```

445

```

446 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]))

```

447

```

448 # the permissible initial consonant groups in a
      syllable. Adjacent consonants should be initial
      pairs.

```

```

449 # The group should not overlap a syllabic pair.
      Such a group is of course followed by a vocalic
      unit.

```

450

```

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

```

452

```

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

```

454


```

455 InitialConsonants <- ((!syllabic C1 &Vocalic)/(!C1
    syllabic) Initial &Vocalic)/(&Initial C1 !(C1
    syllabic) Initial !syllabic &Vocalic))
456
457 # the forbidden medial pairs and triples. These are
    forbidden regardless of placement
458 # of syllable breaks.
459
460 # each of these is actually a single consonant
    followed by an initial, and the idea was to
    identify CVC-CCV junctions which
461 # would be hard to pronounce. But the placement of
    the syllable break is not relevant to the
    exclusion of the sequence.
462 # Notice that the continuant syllabic pairs are
    excluded: this prevents final consonants from
    being included in such pairs.
463
464 NoMedial2 <- ([[Bb] juncture? [Bb]])/([Cc] juncture?
    [Cc])/([Dd] juncture? [Dd])/([Ff] juncture? [Ff])
    /([Gg] juncture? [Gg])/([Hh] juncture? C1)/([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]))
465
466 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]))

```

467

468 # The syllable.

469

470 # there are no formal rules about syllables as such
in our Sources, which is odd since471 # the definition of predicates depends on the
placement of stresses on syllables.

472

473 # The first rule enforces the special point needed
in complexes that474 # a CVC syllable is preferred to a CV syllable where
possible; we economically apply475 # the same rule for default placement of syllable
breaks everywhere, which is, with476 # that exception, that the break comes as soon as
possible.

477

478 # the SyllableB approach is taken if the following
syllable would otherwise start with a syllabic
pair.

479

480 # the reason for this approach is that if one
syllabizes a well formed complex in this way...481 # the syllable breaks magically fall on the djifoa
boundaries. This does mean that the482 # default break in <cabro> is <cab-ro>, which feels
funny but is harmless. Explicitly breaking

483 # it <ca-bro> will also parse correctly.

```

484
485 SyllableA <- (C1 V2 FinalConsonant (!Syllable
      FinalConsonant)?)
486
487 SyllableB <- (InitialConsonants? Vocalic (!Syllable
      FinalConsonant)? (!Syllable FinalConsonant)?)
488
489 Syllable <- ((SyllableA/SyllableB) juncture?)
490
491 # The final consonant in a syllable. There may be
      one or two final consonants. A pair of final
492 # consonants may not be a non-continuant followed by
      a continuant. A final consonant may not
493 # start a forbidden medial pair or triple.
494
495 # The rule that a final consonant pair may not be a
      non-continuant followed by a continuant
496 # is natural and obvious but not in our Sources.
      Such a pair of consonants would seem to
497 # naturally form another syllable.
498
499
500 # a pair of final consonants cannot be differently
      voiced
501
502 FinalConsonant <- !syllabic !(&Badvoice C1 !Syllable
      ) (!(!continuant C1 !Syllable continuant) !
      NoMedial2 !NoMedial3 C1 !(juncture? (V2/syllabic)
      ))
503
504 #!( (!MaybeInitial)C1 juncture? !syllabic C1 juncture
      ? !syllabic C1) !(&MaybeInitial C1 juncture C1 !(
      juncture? C1))
505
506 # Here are various flavors of syllable we may need.
507
508 # this is a portmanteau definition of a bad syllable
      (the sort not allowed in a borrowing).

```

```

509
510 SyllableD <- &(InitialConsonants? ([Yy]/DoubleVowel/
      BrokenMono/&Mono V2 DoubleVowel/!MustMono &Mono
      V2 BrokenMono)) Syllable
511
512 # this (below) is the kind of syllable which can
      exist in a borrowed predicate:
513 # it cannot start with a continuant pair, it cannot
      have a y as vocalic unit,
514 # and its vocalic unit (whether it has one or two
      regular vowels)
515 # cannot be involved in a double vowel or an
      explicitly broken
516 # mandatory monosyllable.
517
518 BorrowingSyllable <- !syllabic (!SyllableD) Syllable
519
520 # this is the final syllable of a predicate. It
      cannot be followed
521 # without pause by a regular vowel.
522
523 VowelFinal <- InitialConsonants? Vocalic juncture? !
      V2
524
525 # syllables with syllabic consonant vocalic units
526 # this class is only used in borrowings, and we *
      could* reasonably
527 # require it to be followed by a vowel. But I won't
      for now.
528 # for gluing this restriction would work, but we
      might literally borrow predicates
529 # with syllabic continuant pronunciations.
530
531 SyllableC <- (&(InitialConsonants? syllabic)
      Syllable)
532
533 # syllables with y
534

```

```

535 SyllableY <- (&(InitialConsonants? [Yy]) Syllable)
536
537 # an explicitly stressed syllable.
538
539 StressedSyllable <- ((SyllableA/SyllableB) [\']*])
540
541 # a final syllable in a word, ending in a consonant.
542
543 NameEndSyllable <- (InitialConsonants? (syllabic/
      Vocalic &FinalConsonant) FinalConsonant?
      FinalConsonant? stress? !letter)
544
545 # the pause classes actually hang on the letter
      before the pause.
546
547 # whitespace which might or might not be a pause.
548
549 maybepause <- (V1 [\']*]? [ ]+ C1)
550
551 # explicit pauses: these are whitespace before a
      vowel or after a consonant, or comma marked
      pauses.
552
553 pause <- ((C1 [\']*]? [ ]+ &letter)/(letter [\']*]? [
      ]+ &V1)/(letter [\']*]? [,] [ ]+ &letter))
554
555 # these are final syllables in words followed by
      whitespace which might not be a pause.
556 # the definition actually doesnt mention the
      maybepause class.
557
558 MaybePauseSyllable <- InitialConsonants? Vocalic
      ['*]? &([ ]+ &C1)
559
560 # The full analysis of names.
561
562 # a name word (without initial marking) is
      resolvable into syllables and ends with a

```

```

consonant.
563
564 PreName <- ((Syllable &Syllable)* NameEndSyllable)
565
566 # this is a busted name word with whitespace in it
    -- but not whitespace at which one has to pause.
567
568 BadPreName <- (MaybePauseSyllable [ ]+/Syllable &
    Syllable)* NameEndSyllable
569
570 # This is a name marker followed by a consonant
    initial name word without pause.
571
572 # I deployed a minimal set of name marker words; I
    can add the others whenever.
573 # I have decided (see below) to retain the social
    lubrication words as vocative markers
574 # *without* making them name markers, so one must
    pause <Loi, Djan>. By not allowing
575 # freemods right after vocative markers in the
    vocative rule, I make <Loi hoi Djan> work as well
    ,
576 # without pause.
577
578 # 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
    ?) [ ]* &C1 &caprule PreName)
579
580 MarkedName <- &caprule ((([Ll] !pause [Aa] juncture
    ?)/ ([Hh] [Oo] !pause [Ii] juncture?) / ([Hh] [
    Uu] juncture? !pause [Ee] juncture?) /([Ll] [Ii]
    juncture? !pause [Uu] juncture?)/[Gg][Aa] !pause
    [Oo] juncture?/[Mm][Uu] juncture? !pause [Ee]
    juncture?) [ ]* &C1 &caprule PreName)

```

```

581
582
583 # This is an unmarked name word with a false name
      marker in it.
584
585 FalseMarked <- (&PreName (!MarkedName character)*
      MarkedName)
586
587 # This is the full definition of name words.  These
      are either marked consonant initial names without
      pause defined above,
588 # names without false name markers beginning with
      explicit pauses (either comma marked or vowel-
      initial)
589 # and name markers followed, with or without pause,
      by name words.  In the latter case there must be
      at least
590 # whitespace before a vowel initial name.
591
592 # a series of names without false name markers and
      names marked with ci, separated by spaces, may be
      appended.
593
594 # there is a look ahead at the grammar: a NameWord
      can be followed without explicit pause (there is
      whitespace and
595 # a pause in speech!) by another
596 # kind of utterance only in a serial name when what
      follows is of the form <ci> predunit, to be
      included
597 # in the name.
598
599 NameWord <- (&caprule MarkedName/([,] [ ]+ !
      FalseMarked &caprule PreName)/(&V1 !FalseMarked &
      caprule PreName)/&caprule ((([Ll] [Aa] juncture?)
      /([Hh] [Oo] [Ii] juncture?)/([Cc] &pause [Ii]
      juncture?)/([Ll] [Ii] juncture? [Uu] juncture?)/[
      Mm] [Uu] juncture? [Ee] juncture?/[Gg] [Aa] [Oo]

```

```

    juncture?) !V1 [,]? [ ]* &caprule PreName))([,]?[
    ]+ !FalseMarked &caprule PreName/[ ,]?[ ]+ &([Cc]
    &pause [Ii]) NameWord)* &([ ]* [Cc] [Ii]
    predunit/&([,] [ ]+ /terminal/[\" ])/!.) ./!.)
600
601 # this is the minimal set of name marker words we
    are using. We may add more.
602
603 # I am contemplating adding the words of social
    lubrication as name markers, but in a more
    restricted
604 # way that in the last provisional parser, in which
    I made them full-fledged vocative markers. [
    Actually,
605 # I preserved their status as vocative markers
    without restoring their status as name markers,
    in the latest version].
606
607 # adding <mue> as a name marker
608
609 namemarker <- ([Ll] [Aa] juncture?/[Hh][Oo][Ii]
    juncture?/[Hh] [Uu] juncture? [Ee] juncture?)/[
    Cc] &pause [Ii] juncture?/[Ll][Ii] juncture? [Uu]
    juncture?/[Gg][Aa][Oo] juncture?/[Mm] [Uu]
    juncture? [Ee] juncture?) !V1
610
611 # this is the bad name marker phenomenon that needs
    to be excluded. This captures the idea
612 # that what follows the name could be pronounced
    without pause as a name word according to the
613 # orthography, but the fact that whitespace is
    present shows that this is not the intention.
614
615 # it is worth noting that name markers at heads of
    name words pass this test
616 # (because I omitted the test that what follows is
    not a PreName in the interests
617 # of minimizing lookahead);

```



```
618 # but this test is only applied to strings that have
      already been determined not to
619 # be of class NameWord.
620
621 badnamemarker <- namemarker !V1 [, ]? [ ]*
      BadPreName
622
623 # we test for the bad name marker condition at the
      beginning of each stream of cmapua,
624 # and streams of cmapua stop before name markers (
      and may resume at a name marker
625 # if neither a NameWord nor the bad marker condition
      is found).
626
627 # We have at any rate completely solved the phonetic
      problem of names and their markers.
628
629 # predicate start tests: the idea is the same as
      class "connective" above, to recognize
630 # the start of a predicate without recursive appeals
      to the whole nasty definition of predicate.
631 # The reason to do it is to recognize when CV^n
      followed by CC cannot be a cmapua unit.
632
633 # New implementation 4/28/2019. This allows only (C
      )V(V)(V) before the pair of vowels, for much less
634 # potential lookahead.
635
636 Vthree <- (V2 juncture?) (V2 juncture?) (V2 juncture
      ?)
637
638 Vfour <- (V2 juncture?) (V2 juncture?) (V2 juncture
      ?) (V2 juncture?)
639
640 # predicate starting with two or three consonants:
      rules out CC(C)V(V) forms. Junctures in
641 # the initial consonant group ignored.
642
```

```

643 predstartA1 <- (&MaybeInitial C1 juncture?
    MaybeInitial/MaybeInitial) &V2 !(V2 stress !Mono
    V2) !(V2 juncture? V2 !character) !(V2 juncture?
    !character)
644
645 # an apparent cmapua unit followed by a consonant
    group which cannot start a predicate -- CV(V)
    case
646
647 predstartA2 <- C1 V2 juncture? (V2 juncture?)? !
    predstartA1 C1 juncture? C1
648
649 # a stressed CV^n before a consonant group (CV(V)
    case)
650
651 predstartA3 <- C1 !Vthree (!StressedSyllable V2
    juncture?)? &StressedSyllable V2 V2? juncture? C1
    juncture? C1
652
653 # other (C)V^n followed by nonpredicate
654
655 predstartA4 <- C1? V2 juncture? (V2 juncture?)? (V2
    juncture?)? !predstartA1 !(MaybeInitial V2) C1
    juncture? C1
656
657 # other stressed (C)V^n followed by consonant group
658
659 predstartA5 <- C1? !Vfour (!StressedSyllable V2
    juncture?)? (!StressedSyllable V2 juncture?)? &
    StressedSyllable V2 V2? juncture? !(MaybeInitial
    V2) C1 juncture? C1
660
661 # forms with y; implemented CVVhy alternative for
    CVV cmapua
662
663 predstartA6 <- C1 (V2 juncture?) (V2 juncture? [Hh
    ]?/C1 juncture? (C1 juncture?)?) [Yy]
664

```

```
665 predstart <- predstartA1/predstartA2/predstartA3/  
    predstartA4/predstartA5/predstartA6  
666  
667 # it is worth noting that in the sequel we have  
    systematically replaced tests &Cmapua  
668 # with !predstart. The former involves lots of  
    lookahead and was causing recursion crashes  
669 # in Python. The phonetics and the grammar are both  
    structured so that any string  
670 # starting with a name marker is tested for NameWord  
    -hood before it is tested for  
671 # cmapua-hood; the only thing it is tested for later  
    is predicate-hood, and predstart  
672 # is a rough and ready test that something might be  
    a predicate (and at any rate  
673 # cannot be a cmapua).  
674  
675 # this class requires pauses before it, after all  
    the phonetic word classes.  
676 # what is being recognized is the beginning of a  
    logical connective.  
677  
678 # To avoid horrible recursion problems, giving this  
    a concrete phonetic definition  
679 # without much lookahead. This can go right up in  
    the phonetics section if it works  
680 # (and here it is!).  
681  
682 # single vowel cmapua syllables early for  
    connectives  
683  
684 a <- ([Aa] !badstress juncture? !V1)  
685  
686 e <- ([Ee] !badstress juncture? !V1)  
687  
688 i <- ([Ii] !badstress juncture? !V1)  
689  
690 o <- ([Oo] !badstress juncture? !V1)
```

```

691
692 u <- ([Uu] !badstress juncture? !V1)
693
694
695 Hearly <- (!predstart [Hh])
696
697 Nearly <- (!predstart [Nn])
698
699
700 # these appear here for historical reasons and could
       be moved later
701
702
703 connective <- [ ]* !predstart ([Nn] [Oo] juncture?)*
       (a/e/o/u/Hearly a/Nearly UU) juncture? !V2 (!(
       predstart [Ff] [Ii]) (!(predstart [Mm] [Aa])) (!(
       predstart [Zz] [Ii]))
704
705
706 # cmapua units starting with consonants. This is
       the exact description from NB3. The fancy tail
       in each of the
707 # three cases is enforcing the rule about pausing
       before a following predicate if stressed.
708
709 # consonant initial cmapua units may not be followed
       by vowels without pause.
710
711 # I am adding <iy> and <uy> (always monosyllable,
       yuh and wuh) as vowel pairs permitted in VV and
       CVV cmapua units.
712 # it is worth noting that the "yuh" and "wuh"
       pronunciations of these diphthongs
713 # are surprising to the English-reading eye.
714 # The use for this envisaged is that the name <ziy>
       of Y becomes easy to introduce. Adding word
       space
715 # is always nice, and these words seem pronounceable

```

```

. I also made <yfi> possible: Y now has
phonetically
716 # regular names.
717
718 CmapuaUnit <- (C1 Mono juncture? V2 !(['*] [ ]* &C1
predstart) juncture? !V1/C1 (VV/[Ii][Yy]/[Uu][Yy
]) !(['*] [ ]* &C1 predstart) juncture? !V1/C1 V2
!(['*] [ ]* &C1 predstart) juncture? !V1)
719
720 # 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.
721 # 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
722 # word due to inexplicit whitespace (doesn't satisfy
the bad name marker condition).
723
724 # we force explicit comma pauses before logical
connectives, but not before vowel initial cmapua
in general;
725 # other conditions force at least whitespace, which
does stand for a pause, before such words.
726
727 # detect starts of quotes or parentheses with <li>
or <kie>
728
729 likie <- ([Ll] [Ii] juncture? !V1/[Ki] [Ii] juncture
? [Ee] juncture? !V1)
730
731 # a special provision is made for NO UI forms as
single words. <yfi> is supported.
732
733 Cmapua <- &caprule !badnamemarker (!predstart (VV/[
Ii][Yy]/[Uu][Yy]) !(['*] [ ]* &C1 predstart)
juncture? NOI/!predstart [Nn] [Oo] juncture? !
predstart (VV/[Ii][Yy]/[Uu][Yy]) !(['*] [ ]* &C1

```

```

predstart) juncture?/((!predstart (VV/[Ii][Yy]/[
Uu][Yy]) !(['*] [ ]* &C1 predstart) juncture?)+ /
((!predstart V1 !(['*] [ ]* &C1 predstart)
juncture?)/ !predstart CmapuaUnit) (!namemarker !
alienmarker !likie !predstart CmapuaUnit)*)/!
predstart V2 !(['*] [ ]* &C1 predstart) juncture
?) !V1 !(C1+ juncture) !([ ]* connective)
734
735 # I have apparently now completely solved the
      problem of parsing cmapua as well as name words.
736
737 # Now for predicates.
738
739 # the elementary djifoa (not borrowings)
740
741 # various special flavors of these djifoa will be
      needed.
742 # These are the general definitions.
743
744 # The NOY and Bad forms are for use for testing
      candidate borrowings for resolution
745 # with bad syllable break placements. Borrowings do
      not contain Y...
746
747 # CVV djifoa with phonetic hyphens.
748
749 # added checks to all cmapua classes: the vowel
      final ones, when not phonetically hyphenated,
      cannot
750 # be followed by a regular vowel. This is crucial
      for getting the syllable analysis and the djifoa
751 # analysis to end at the same point.
752
753 # allowing h to be inserted before y in CVVy djifoa
      for a CVVhy form.
754
755 # allowing -r glue to be expressed as -rr
756

```

```

757 CVV <- C1 VV (juncture? [Hh]? [Yy] [-]? &(Complex) /
    juncture? [Rr] [Rr]? juncture? &C1/[Nn] juncture?
    &[Rr]/juncture? !V2)
758
759 CVVNoHyphen <- C1 VV juncture? !V2
760
761 CVVHiddenStress <- C1 &DoubleVowel V1 [-]? V1 ([-]?
    [Hh]? [Yy] [-]? &Complex /[Rr] [-]? &C1/[Nn] [-]?
    &[Rr]/[-]? !V2)
762
763 CVVFinalStress <- C1 VV (['*] [Hh]? [Yy] [-]? &
    Complex /[Rr] ['*] &C1/['*] [Rr] [Rr] juncture? &
    C1/[Nn] ['*] &[Rr]/['*] !V2)
764
765 CVVNOY <- C1 VV (juncture? [Rr] [Rr]? juncture? &C1
    /[Nn] juncture? &[Rr]/juncture? !V2)
766
767 CVVNOYFinalStress <- C1 VV ([Rr] ['*] &C1/['*] [Rr]
    [Rr] juncture? &C1/[Nn] ['*] &[Rr]/['*] !V2)
768
769 CVVNOYMedialStress <- C1 !BrokenMono V2 ['*] V2 [-]?
    !V2
770
771 # CCV djifoa with phonetic hyphens.
772
773 CCV <- Initial V2 (juncture? [Yy] [-]? &letter/
    juncture? !V2)
774
775 CCVStressed <- Initial V2 (['*] [Yy] [-]? &letter
    /['*] !V2)
776
777 CCVNOY <- Initial V2 juncture? !V2
778
779 CCVBad <- MaybeInitial V2 juncture? !V2
780
781 CCVBadStressed <- MaybeInitial V2 ['*] !V2
782
783

```

```

784 # CVC djifoa with phonetic hyphens.  These cannot be
      final and are always followed by a consonant (
      well, the
785 # -y form may be followed by a vowel...
786 # an eccentric syllable break is supported if the
      CVC is y-hyphenated:
787 # <me-ky-kiu> and <mek-y-kiu> are both legal.  The
      default is the latter.
788
789 CVC <- (C1 V2 !NoMedial2 !NoMedial3 C1 (juncture? [
      Yy] [-]? &letter/juncture? &C1)/C1 V2 juncture C1
      [Yy] [-]? &letter)
790
791 CVCStressed <- (C1 V2 !NoMedial2 !NoMedial3 C1 (['*]
      [Yy] [-]? &letter/['*] &letter)/C1 V2 ['*] C1 [
      Yy] [-]? &letter)
792
793 CVCNOY <- C1 V2 !NoMedial2 !NoMedial3 C1 juncture? &
      C1
794
795 CVCBad <- C1 V2 !NoMedial2 !NoMedial3 juncture? C1 &
      C1
796
797 CVCNOYStressed <- C1 V2 !NoMedial2 !NoMedial3 C1
      ['*] &C1
798
799 CVCBadStressed <- C1 V2 !NoMedial2 !NoMedial3 ['*]
      C1 &C1
800
801 # the five letter forms (always final in complexes)
802
803 CCVCV <- Initial V2 juncture? C1 V2 [-]? !V2
804
805 CCVCVStressed <- Initial V2 ['*] C1 V2 [-]? !V2
806
807 CCVCVBad <- MaybeInitial V2 juncture? C1 V2 [-]? !V2
808
809 CCVCVBadStressed <- MaybeInitial V2 ['*] C1 V2 [-]?

```



```

!V2
810
811 CVCCV <- (C1 V2 juncture? Initial V2 [-]? !V2/C1 V2
      !NoMedial2 C1 juncture? C1 V2 [-]? !V2)
812
813 CVCCVStressed <- (C1 V2 ['*] Initial V2 [-]? !V2/C1
      V2 !NoMedial2 C1 ['*] C1 V2 [-]? !V2)
814
815 # the medial five letter djifoa
816
817 CCVCY <- Initial V2 juncture? C1 [Yy] [-]?
818
819 CVCCY <- (C1 V2 juncture? Initial [Yy] [-]?/C1 V2 !
      NoMedial2 C1 juncture? C1 [Yy] [-]?)
820
821 CCVCYStressed <- Initial V2 ['*] C1 [Yy] [-]?
822
823 CVCCYStressed <- (C1 V2 ['*] Initial [Yy] [-]?/C1 V2
      !NoMedial2 C1 ['*] C1 [Yy] [-]?)
824
825 # to reason about resolution of borrowings into both
      syllables and djifoa (we want to exclude the
      latter
826 # but we need to define it adequately) we need to
      recognize where to stop. A predicate word ends
      either
827 # at a non-character (not a letter or syllable mark:
      whitespace, comma or terminal punctuation) or it
828 # has an explicit or deducible penultimate stress.
      Borrowings do not contain doubled vowels, so they
829 # have to have explicit stress in the latter case.
830
831 # analysis: the stressed tail consists of a
      stressed syllable followed by an unstressed
      syllable.
832 # identifying an unstressed final syllable is
      complicated by recognizing which CVV combinations
      can

```

```

833 # be one syllable. This will either be an
      explicitly stressed syllable followed by a single
      syllable
834 # or a syllable suitable to be stressed followed by
      an explicitly final syllable. CVV djifoa can
835 # contain both syllables in a tail and of course the
      five letter djifoa have to be tails. A never
      stressed
836 # SyllableC (with a continuant) may intervene.
837
838 # tail of a borrowing with an explicit stress
839
840 BorrowingTail1 <- !SyllableC &StressedSyllable
      BorrowingSyllable (!StressedSyllable &SyllableC
      BorrowingSyllable)? !StressedSyllable &
      BorrowingSyllable VowelFinal
841
842 # tail of a borrowing or borrowing djifoa with no
      explicit stress
843
844 BorrowingTail2 <- !SyllableC BorrowingSyllable (!
      StressedSyllable &SyllableC BorrowingSyllable)? !
      StressedSyllable &BorrowingSyllable VowelFinal
      (&[Yy]/!character)
845
846 # tail of a stressed borrowing djifoa, different
      because stress is shifted to the end
847
848 BorrowingTail3 <- !SyllableC !StressedSyllable
      BorrowingSyllable (!StressedSyllable &SyllableC
      BorrowingSyllable)? &BorrowingSyllable
      InitialConsonants? Vocalic ['*] &[Yy]
849
850 BorrowingTail <- BorrowingTail1 / BorrowingTail2
851
852 # short forms that are ruled out: CCVV and CCCVV
      forms.
853

```

```

854 CCVV <- (InitialConsonants V2 juncture? V2 juncture?
      !character / InitialConsonants V2 ['*] !Mono V2
      juncture?)
855
856 # VCCV and some related forms are ruled out (rule
      predstartF above is about this)
857
858 # a continuant syllable cannot be initial in a
      borrowing and there cannot be successive
      continuant
859 # syllables. There really ought to be no more than
      one!
860
861 # borrowing, before checking that it doesnt resolve
      into djifoa
862
863 PreBorrowing <- &predstart!CCVV!Cmapua!SyllableC(!
      BorrowingTail!(StressedSyllable)!(SyllableC
      SyllableC)BorrowingSyllable)* BorrowingTail
864
865 # ditto for an explicitly stressed borrowing
866
867 StressedPreBorrowing <- &predstart!CCVV!Cmapua!
      SyllableC(!BorrowingTail!(StressedSyllable)!(
      SyllableC SyllableC)BorrowingSyllable)*
      BorrowingTail1
868
869 # borrowing djifoa without explicit stress (before
      resolution check)
870
871 PreBorrowing2 <- &predstart!CCVV!Cmapua!SyllableC(!
      BorrowingTail!(StressedSyllable)!(SyllableC
      SyllableC)BorrowingSyllable)* BorrowingTail2
872
873 # stressed borrowing djifoa (before resolution check
      ).
874
875 PreBorrowing3 <- &predstart!CCVV!Cmapua!SyllableC(!

```

```

      BorrowingTail3!(StressedSyllable)!(SyllableC
      SyllableC)BorrowingSyllable)* BorrowingTail3
876
877 # Now comes the problem of trying to say that a
      preborrowing cannot resolve into cmapua. The
      difficulty is with
878 # recognizing the tail, so making sure that the two
      resolutions stop in the same place.
879
880 # we know because it is a borrowing that there is at
      most one explicit stress, and it has to fall
881 # in one of the cmapua! This should make it doable.
882
883 # borrowing djifoa are terminated with y, so the
      final djifoa needs to take this into account
884
885 # the idea behind both djifoa analyses is the same.
      If we end with a final djifoa followed by
886 # a non-character, we improve our chances of ending
      the syllable analysis at the same point. We
      control
887 # this by identifying djifoa with stresses in them:
      a medially stressed djifoa must be the last one
888 # (and the syllable analysis will find its stressed
      syllable and end at its final syllable, the fact
889 # that djifoa cannot be followed by vowels ensuring
      that the syllable analysis cannot overrun its end
      .
890 # When the djifoa is finally stressed, the complex
      analysis ends with a further djifoa guaranteed to
      have
891 # just one syllable, and the syllable analysis again
      will stop in the same place. The medial five
      letter forms
892 # and borrowing djifoa of course are finally
      stressed mod an additional unstressed syllable
      which is skipped
893 # by the syllable analysis, because it allows one to

```

```

        ignore an actually penultimate syllable with y
        or
894 # a syllabic consonant. In the case where we never
        find a stress and end up at a final djifoa, the
        syllable
895 # analysis will carry right through to the same
        final point.
896
897 # in the attempted resolution of borrowings, our
        life is easier because we do not have
898 # borrowing djifoa or medial five letter forms to
        consider, or any forms with y-hyphens.
899
900 RFinalDjifoa <- (CCVCVBad/CVCCV/CVVNoHyphen/CCVBad/
        CVCBad) (&[Yy]/!character)
901
902 RMediallyStressed <- (CCVCVBadStressed/CVCCVStressed
        /CVVNOYMedialStress)
903
904 RFinallyStressed <- (CVVNOYFinalStress/
        CCVBadStressed/CVCBadStressed/CVCNOYStressed)
905
906 BorrowingComplexTail <- (RMediallyStressed/
        RFinallyStressed (&(C1 Mono) CVVNoHyphen/CCVBad)/
        RFinalDjifoa)
907
908 ResolvedBorrowing <- (!BorrowingComplexTail(CVVNOY/
        CCVBad/CVCBad))* BorrowingComplexTail
909
910 # borrowed predicates
911
912 Borrowing <- !ResolvedBorrowing &caprule
        PreBorrowing !([ ]* (connective))
913
914 # explicitly stressed borrowed predicates
915
916 StressedBorrowing <- !ResolvedBorrowing &caprule
        StressedPreBorrowing !([ ]* &V1 Cmapua)

```

```

917
918 #This is the shape of non-final borrowing djifoa.
      Notice that a final stress is allowed.
919 #The curious provision for explicitly stressing a
      borrowing djifoa and pausing is supported.
920
921 # borrowing djifoa without explicit stress (stressed
      ones are not of this class!)
922 # Note that one can pause after these (explicitly,
      with a comma, in which case the stress must be
      explicit too)
923
924 BorrowingDjifoa <- !ResolvedBorrowing &caprule
      PreBorrowing2 (['*'] [y] [,] [ ]+/junction? [y]
      [-]?)
925
926 # stressed borrowing djifoa finally implemented!
927
928 StressedBorrowingDjifoa <- !ResolvedBorrowing &
      caprule PreBorrowing3 [y] [-]? ([,] [ ]+)?
929
930 # We resolve complexes twice, once into syllables
      and once into djifoa. We again have to ensure
      that
931 # we end up in the same place! The syllable
      resolution is very similar to that of borrowings;
932 # the unstressed middle syllable of the tail can be
      a SyllableY, and can also be a
933 # SyllableC if the final djifoa is a borrowing.
934
935 # A stressed borrowing djifoa with the property that
      the tail is still a phonetic complex is
936 # a unit for this analysis.
937
938 # note here that I specifically rule out a complex
      being followed without pause by y. I do not rule
939 # this out for the vowel final djifoa because they
      can be followed by y at the end of a borrowing

```

```

940 # djifoa.
941
942 PhoneticComplexTail1 <- !SyllableC !SyllableY &
    StressedSyllable Syllable (!StressedSyllable &(
    SyllableC/SyllableY) Syllable)? !StressedSyllable
    !SyllableY VowelFinal !V1
943
944 PhoneticComplexTail2 <- !SyllableC !SyllableY
    Syllable (!StressedSyllable &(SyllableC/SyllableY
    ) Syllable)? !StressedSyllable !SyllableY
    VowelFinal !character
945
946 PhoneticComplexTail <- PhoneticComplexTail1 /
    PhoneticComplexTail2
947
948 # note the explicit predstart test here.
949
950 PhoneticComplex <- &predstart!CCVV!Cmapua!SyllableC(
    StressedBorrowingDjifoa &PhoneticComplex/!
    PhoneticComplexTail!(StressedSyllable)!(SyllableC
    SyllableC) Syllable)* PhoneticComplexTail
951
952 # the analysis of final djifoa and stressed djifoa
    differs only in details from
953 # what is above for resolution of borrowings. The
    issues about CVV djifoa with doubled
954 # vowels are rather exciting.
955
956 # a stressed borrowing djifoa with the tail still a
    phonetic complex is a black box unit for
957 # this construction.
958
959 # My approach imposes the restriction on JCB's "
    pause after a borrowing djifoa" idea that what
    follows
960 # the pause must itself contain a penultimate stress
    : <igllu'ymao> is a predicate but <igllu'y, mao>
    is not.

```

```
961 # while <iglluy', gudmao> is a predicate.
962
963 # the analysis of the djifoa resolution process is
    the same as above, with additional remarks
964 # about doubled vowel syllables: notice that where
    the complex tail involved a doubled vowel
    syllable
965 # without explicit stress, we insist on that djifoa
    or the single-syllable next djifoa ending in
966 # a non-character: in the absence of explicit
    stress, we always rely on whitespace or
    punctuation
967 # to indicate the end of the predicate.
968
969 # all sorts of subtleties about borrowings and
    borrowing djifoa are finessed by always looking
    for
970 # them first. There are no restrictions re fronts
    of borrowings or borrowing djifoa looking like
    regular
971 # djifoa; the fact that borrowing djifoa end in y
    and borrowings do not contain y makes it always
972 # possible to tell when one is looking at the head
    of a borrowing djifoa. Regular djifoa just
    before a borrowing
973 # djifoa need to be y-hyphenated so as not to be
    absorbed into the front of the borrowing (I don't
    believe
974 # that I actually need to impose a formal rule to
    this effect, though I am not absolutely certain;
    it would
975 # be difficult to formulate [and does appear in the
    previous version, where it is a truly
    unintelligible piece
976 # of PEG code]).
977
978 FinalDjifoa <- (Borrowing/CCVCV/CVCCV/CVVNoHyphen/
    CCVNOY) !character
```



```

979
980 MediallyStressed <- (StressedBorrowing/CCVCVStressed
    /CVCCVStressed/CVVNOYMedialStress)
981
982 FinallyStressed <-(StressedBorrowingDjifoa/
    CCVCYStressed/CVCCYStressed/CVVFfinalStress/
    CCVStressed/CVCStressed)
983
984 ComplexTail <- (CVVHiddenStress (&(C1 Mono)
    CVVNoHyphen/CCVNOY) !character/FinallyStressed
    (&(C1 Mono) CVVNoHyphen/CCVNOY)/MediallyStressed/
    FinalDjifoa)
985
986 PreComplex <- (!CVVHiddenStress (!ComplexTail)(
    StressedBorrowingDjifoa &PhoneticComplex/
    BorrowingDjifoa/CVCCY/CCVCY/CVV/CCV/CVC))*
    ComplexTail
987
988 # originally I had complicated tests here for the
    conditions under which an initial
989 # CVC cmapua has to be y-hyphenated: I was being
    wrong headed, the predstart rules
990 # already enforce this (in the bad cases, the
    initial CV- falls off). The user will
991 # simply find that they cannot put the word together
    otherwise. The previous version
992 # did need this test because it actually used full
    lookahead to check for the start of a predicate.
993
994 Complex <- &caprule &PreComplex PhoneticComplex !([
    ]* (connective))
995
996 # format for the LI quote and KIE parenthesis
997
998 LiQuote <- (&caprule [Ll][Ii]juncture? comma2? [\" ]
    phoneticutterance [\" ] comma2? &caprule [Ll][Uu]
    juncture? !([ ]* connective)/(&caprule [Kk][Ii]
    juncture?[Ee]juncture? comma2? [(
```

```

    phoneticutterance []] comma2? &caprule [Kk][Ii]
    juncture?[Uu]juncture? !([ ]* connective)))
999
1000 # the condition on Word that a Cmapua is not
    followed by another Cmapua
1001 # with mere whitespace between was used by <liu>
    quotation, but is now redundant,
1002 # because I have required that <liu> quotations be
    closed with explicit pauses in all cases.
1003
1004 Word <- (NameWord / Cmapua !([ ]*Cmapua)/ Complex/
    CCVNOY)
1005
1006 # it is an odd point that all borrowings parse as
    complexes -- so when I parsed all the words the
    first time they all
1007 # parsed as complexes. A borrowing is a complex
    consisting of a single final borrowing djifoa!
1008 # I did redesign this so that borrowings are parsed
    as borrowings. (This is the class
1009 # I used to parse the dictionary).
1010
1011 # Yes, CVC djifoa do get parsed as names in the
    dictionary, so the CVC case here is redundant. I
    actually
1012 # think that only the CCV djifoa actually get parsed
    as such.
1013
1014 SingleWord <- (Borrowing !./Complex !./ Word !./
    PreName !. /CCVNOY) !.
1015
1016 # name word appearing initially without leading
    spaces is important, because one type of NameWord
    includes a leading comma.
1017
1018 phoneticutterance1 <- (NameWord /[ ]* LiQuote/[ ]*
    NameWord/[ ]* AlienWord/[ ]*Cmapua/[ ]* '--'/[ ]*
    '...'/[ ]* Borrowing![y]/[ ]* Complex/[ ]* (

```

```
CCVNOY)))+
1019
1020 phoneticutterance <- (phoneticutterance1/[ , ][ ]+ /
    terminal)+
1021
1022 # consonants and vowel groups in cmapua
1023
1024 # as noted above, !predstart stands in for the
    computationally disastrous &Cmapua
1025
1026 badstress <- ['*] [ ]* &C1 predstart
1027
1028 B <- (!predstart [Bb])
1029
1030 C <- (!predstart [Cc])
1031
1032 D <- (!predstart [Dd])
1033
1034 F <- (!predstart [Ff])
1035
1036 G <- (!predstart [Gg])
1037
1038 H <- (!predstart [Hh])
1039
1040 J <- (!predstart [Jj])
1041
1042 K <- (!predstart [Kk])
1043
1044 L <- (!predstart [Ll])
1045
1046 M <- (!predstart [Mm])
1047
1048 N <- (!predstart [Nn])
1049
1050 P <- (!predstart [Pp])
1051
1052 R <- (!predstart [Rr])
1053
```

```

1054 S <- (!predstart [Ss])
1055
1056 T <- (!predstart [Tt])
1057
1058 V <- (!predstart [Vv])
1059
1060 Z <- (!predstart [Zz])
1061
1062 # the monosyllabic classes may be followed by one
      vowel
1063 # if they start a Cvv-V cmapua unit; the others may
      never
1064 # be followed by vowels. Classes ending in -b are
1065 # used in Cvv-V cmapua units.
1066
1067 # the single vowel classes were moved before the
      class
1068 # connective in the phonetics section.
1069
1070
1071 V3 <- juncture? V2 !badstress
1072
1073 AA <- ([Aa] juncture? [Aa] !badstress juncture? !V1)
1074
1075 AE <- ([Aa] juncture? [Ee] !badstress juncture? !V1
      )
1076
1077 AI <- ([Aa] [Ii] !badstress juncture? !(V1))
1078
1079 AO <- ([Aa] [Oo] !badstress juncture? !(V1))
1080
1081 AIb <- ([Aa] [Ii] !badstress juncture? &(amp;V2
      juncture? !V1))
1082
1083 AOb <- ([Aa] [Oo] !badstress juncture? &(amp;V2
      juncture? !V1))
1084
1085 AU <- ([Aa] juncture? [Uu] !badstress juncture? !V1

```

```

    )
1086
1087 EA <- ([Ee] juncture? [Aa] !badstress juncture? !V1
    )
1088
1089 EE <- ([Ee] juncture? [Ee] !badstress juncture? !V1
    )
1090
1091 EI <- ([Ee] [Ii] !badstress juncture? !(V1))
1092
1093 EIb <- ([Ee] [Ii] !badstress juncture? &(amp;V2
    juncture? !V1))
1094
1095 EO <- ([Ee] juncture? [Oo] !badstress juncture? !V1
    )
1096
1097 EU <- ([Ee] juncture? [Uu] !badstress juncture? !V1
    )
1098
1099 IA <- ([Ii] juncture? [Aa] !badstress juncture? !(
    V1))
1100
1101 IE <- ([Ii] juncture? [Ee] !badstress juncture? !(
    V1))
1102
1103 II <- ([Ii] juncture? [Ii] !badstress juncture? !(
    V1))
1104
1105 IO <- ([Ii] juncture? [Oo] !badstress juncture? !(
    V1))
1106
1107 IU <- ([Ii] juncture? [Uu] !badstress juncture? !(
    V1))
1108
1109 IAb <- ([Ii] juncture? [Aa] !badstress juncture?
    &(amp;V2 juncture? !V1))
1110
1111 IEb <- ([Ii] juncture? [Ee] !badstress juncture? &(

```

```

    V2 juncture? !V1))
1112
1113 IIb <- ([Ii] juncture? [Ii] !badstress juncture? &(
    V2 juncture? !V1))
1114
1115 IOb <- ([Ii] juncture? [Oo] !badstress juncture? &(
    V2 juncture? !V1))
1116
1117 IUb <- ([Ii] juncture? [Uu] !badstress juncture?
    &(V2 juncture? !V1))
1118
1119 OA <- ([Oo] juncture? [Aa] !badstress juncture? !V1
    )
1120
1121 OE <- ([Oo] juncture? [Ee] !badstress juncture? !V1
    )
1122
1123 OI <- ([Oo] [Ii] !badstress juncture? !(V1))
1124
1125 OIb <- ([Oo] [Ii] !badstress juncture? &(V2
    juncture? !V1))
1126
1127 OO <- ([Oo] juncture? [Oo] !badstress juncture? !V1
    )
1128
1129 OU <- ([Oo] juncture? [Uu] !badstress juncture? !
    V1)
1130
1131 UA <- ([Uu] juncture? [Aa] !badstress juncture? !(
    V1))
1132
1133 UE <- ([Uu] juncture? [Ee] !badstress juncture? !(
    V1))
1134
1135 UI <- ([Uu] juncture? [Ii] !badstress juncture? !(
    V1))
1136
1137 UO <- ([Uu] juncture? [Oo] !badstress juncture? !(

```

```

    V1))
1138
1139 UU <- ([Uu] juncture? [Uu] !badstress juncture? !(
    V1))
1140
1141 UAb <- ([Uu] juncture? [Aa] !badstress juncture?
    &(V2 juncture? !V1))
1142
1143 UEb <- ([Uu] juncture? [Ee] !badstress juncture? &(
    V2 juncture? !V1))
1144
1145 UIb <- ([Uu] juncture? [Ii] !badstress juncture?
    &(V2 juncture? !V1))
1146
1147 UOb <- ([Uu] juncture? [Oo] !badstress juncture? &(
    V2 juncture? !V1))
1148
1149 UUb <- ([Uu] juncture? [Uu] !badstress juncture? &(
    V2 juncture? !V1))
1150
1151 # adding the new IY and UY, which might see use some
    time.
1152 # they are mandatory monosyllables but do not take a
    possible additional
1153 # following vowel as the regular ones do. So far
    only used in <ziy>.
1154
1155 IY <- [Ii] [Yy] !badstress juncture? !V1
1156
1157 UY <- [Uu] [Yy] !badstress juncture? !V1
1158
1159 # this is a pause not required by the phonetics.
    This is the only
1160 # sort of pause which could in principle carry
    semantic freight (the
1161 # pause/GU equivalence beloved of our Founder) but
    we have abandoned
1162 # this. There is one place, after initial <no> in

```

```

    an utterance, where
1163 # a pause can have effect on the parse (but not on
    the meaning, I believe,
1164 # unless a word break is involved).
1165
1166 # this class should NEVER be used in a context which
    might follow
1167 # a name word. In previous versions, pauses after
    name words were included
1168 # in the name word; this is not the case here, so a
    PAUSE
1169 # after a name word would not be recognized as a
    mandatory pause.
1170
1171 # in any event, as long as we stay away from pause/
    GU equivalence, this
1172 # is not a serious issue!
1173
1174 # this class does do some work in the handling of
    issues surrounding the legacy
1175 # shape of APA connectives, concerning which the
    less said, the better.
1176
1177 PAUSE <- [,] [ ]+ !(V1/connective) &caprule
1178
1179 # more punctuation
1180
1181 comma <- [,] [ ]+ &caprule
1182
1183 comma2 <- [,]? [ ]+ &caprule
1184
1185 # Part II Lexicography
1186
1187 # In this section I develop the grammar of words in
    Loglan. I'll work by editing the original
    provisional PEG grammar.
1188
1189 # I place the start of this section exactly here,
```



```

        just before two final items of
1190 # punctuation, because these items of punctuation
        look forward not only to lexicography
1191 # but to the full grammar!
1192
1193 # the end of utterance symbol <#> should be added in
        the phonetics
1194 # section as a species of terminal marker. Done. We
        do *not* actually
1195 # endorse use of this marker, but we can notionally
        support it and it is in
1196 # our sources.
1197
1198 end <- (([ ]* '#' [ ]+ utterance)/([ ]+ !.)/!.)
1199
1200 # this rule allows terminal punctuation to be
        followed by an inverse vocative,
1201 # a frequent occurrence in Leith's novel, and
        something which makes sense.
1202
1203 period <- (([!.:;?] (&end/([ ]+ &caprule))) (invvoc
        period?))?
1204
1205 # Letters with y will be special cases
1206 # idea: allow IY and UY (always monosyllables) as
        vowel combinations in cmapua only.
1207 # done: Y has a name now. <yfi> is also added.
1208
1209 # the classes in this section after this point are
        the cmapua word classes of Loglan (if they begin
        with [ ]* or a word class).
1210 # I suppose the alien text classes are not really
        word classes, but they are lexicographic items,
        as it were.
1211 # Paradoxically, the PA and NI classes admit
        internal explicit pauses. So of course do
        predicate words!
1212

```

```

1213 # Loglan does admit true multisyllable cmapua:
      there are words made of cmapua units which have
      joints between
1214 # units at which one cannot pause without breaking
      the word.  Lojban, I am told, does not.
1215
1216 # this version has the general feature that the
      quotation and alien text constructions are not
      hacked:
1217 # they are supported by the phonetic rules (as dire
      exceptions, of course) and the grammatical
      constructions
1218 # conform with the phonetic layer.  Alien text and
      utterances quoted with <li>...<lu> can be
      enclosed in double quotes.
1219 # LI only supports full utterances, for the moment.
      All alien text constructors take the same class
      as argument:
1220 # the vocative and inverse vocative *require* quotes
      to avoid misreading ungrammatical expressions
      with typos
1221 # as correct (inverse) vocatives.
1222
1223 # the names <yfi>, <ziy> for Y are supported.  The
      Ceo names are left as they are.  I decided that a
      second short series
1224 # of letteral pronouns is actually a reasonable use
      of short words, and the Ceio words are there for
      other uses.
1225
1226 TAI0 <- (V1 juncture? M a/V1 juncture? F i/V1
      juncture? Z i/!predstart C1 AI/!predstart C1 EI/!
      predstart C1 AIb u/!predstart C1 EIb (u)/!
      predstart C1 E0/ Z [Ii] V1 !badstress juncture? !
      V1 (M a?))
1227
1228 # a negative suffix used in various contexts.
      Always a suffix:  its use as a prefix in tenses

```

```

    was a mistake in NB3 and I
1229 # think still supported in LIP.  Ambiguities
    demonstrably followed from this usage (an example
    of how the demonstration
1230 # of non-ambiguity of 1989 Loglan was compromised by
    the opaque lexicography).
1231
1232 NOI <- (N OI)
1233
1234 # the logical connectives.  [A0] is the class of
    core logical connectives.  [A] is the fully
    decorated logical connective with
1235 # possible nu- (always in nuno- or nuu) and no-
    prefixes, possible -noi suffix, and possible (
    problematic) PA suffix, closed
1236 # with -fi (our new proposal) or an explicit pause.
1237
1238 A0 <- &Cmapua (a/e/o/u/H a/N UU)
1239
1240 A <- [ ]* !predstart !TAIO (N [o])? AO NOI? !([ ]+
    PANOPAUSES PAUSE) !(PANOPAUSES !PAUSE [ ,]) (
    PANOPAUSES ((F i)/&PAUSE))?)
1241
1242 # 4/18 in connected sentpreds, fi must be used to
    close, not a pause.
1243
1244 # A2 <- [ ]* !predstart !TAIO (N [o])? AO NOI? !([
    ]+ PANOPAUSES PAUSE) !(PANOPAUSES !PAUSE [ ,]) (
    PANOPAUSES (F i))?)
1245
1246
1247
1248 # A not closed with -fi or a pause
1249
1250 ANOFI <- [ ]* (!predstart !TAIO ( (N [o])? AO NOI?
    PANOPAUSES?))
1251
1252 A1 <- A

```

```

1253
1254 # versions of A with different binding strength
1255
1256 ACI <- (ANOFI C i)
1257
1258 AGE <- (ANOFI G e)
1259
1260 # a tightly binding series of logical connectives
      used to link predicates
1261 # this also includes the fusion connective <ze> when
      used between predicates.
1262
1263 CAO <- (( (N o)? ((C a)/(C e)/(C o)/(C u)/(Z e)/(C i
      H a)/N u C u)) NOI?)
1264
1265 CA1 <- (CAO !([ ]+ PANOPAUSES PAUSE) !(PANOPAUSES !
      PAUSE [ ,]) (PANOPAUSES ((F i)/&PAUSE)))?)
1266
1267 CA1NOFI <- (CAO PANOPAUSES?)
1268
1269 CA <- ([ ]* CA1)
1270
1271 # the fusion connective when used in arguments
1272
1273 ZE2 <- ([ ]* (Z e))
1274
1275 # sentence connectives. [I] is the class of
      utterance initiators (no logical definition).
1276 # the subsequent classes are inhabited by sentence
      logical connectives with various binding
1277 # strengths.
1278
1279 I <- ([ ]* !predstart !TAIO i !([ ]+ PANOPAUSES
      PAUSE) !(PANOPAUSES !PAUSE [ ,]) (PANOPAUSES ((F
      i)/&PAUSE)))?)
1280
1281 ICA <- ([ ]* i ((H a)/CA1))
1282

```

```

1283 ICI <- ([ ]* i CA1NOFI? C i)
1284
1285 IGE <- ([ ]* i CA1NOFI? G e)
1286
1287 # forethought logical connectives
1288
1289 KAO <- ((K a)/(K e)/(K o)/(K u)/(K i H a)/(N u K u))
1290
1291 # causal and comparative modifiers
1292
1293 KOU <- ((K OU)/(M OI)/(R AU)/(S OA)/(M OU)/(C IU))
1294
1295 # negative and converse forms
1296
1297 KOU1 <- (((N u N o)/(N u)/(N o)) KOU)
1298
1299 # the full type of forethought connectives, adding
      the causal and comparative connectives
1300
1301 KA <- ([ ]* ((KAO)/((KOU1/KOU) K i)) NOI?)
1302
1303 # the last component of the KA...KI... structure of
      forethought connections
1304
1305 KI <- ([ ]* (K i) NOI?)
1306
1307 # causal and comparative modifiers which are *not*
      forethought connectives
1308
1309 KOU2 <- (KOU1 !KI)
1310
1311 # a test used to at least partially enforce the
      penultimate stress rule on quantifier predicates
1312
1313 BadNISstress <- ((C1 V2 V2? stress (M a)? (M OA)? NI
      RA)/(C1 V2 stress V2 (M a)? (M OA)? NI RA))
1314
1315 # root quantity words, including the numerals

```

```

1316
1317 NIO <- (!BadNISTress ((K UA)/(G IE)/(G IU)/(H IE)/(H
      IU)/(K UE)/(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)))
1318
1319 # the class of SA roots, which modify quantifiers
1320
1321 SA <- (!BadNISTress ((S a)/(S i)/(S u)/(IE (comma2?
      !IE SA)?)) NOI?)
1322
1323 # the family of quantifiers which double as suffixes
      for the quantifier predicates
1324 # 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>.
1325 # No action here, just a remark.
1326
1327 RA <- (!BadNISTress ((R a)/(R i)/(R o)/R e/R u))
1328
1329 # re and ru added to class RA 5/11/18
1330
1331 # quantifier units consisting of a NI or RA root
      with <ma> 00 or <moa> 000 appended; to <moa> one
      can further
1332 # append a digit to iterate <moa>: <fomoate> is
      four billion, for example. <rimoa>, a few
      thousand.
1333
1334 # a NI1 or RA1 may be followed by a pause before
      another NI word other than a numerical predicate;
1335 # one is allowed to breathe in the middle of long
      numerals. I question whether the pause
1336 # provision makes sense in RA1.
1337
1338 NI1 <- ((NIO (!BadNISTress M a)? (!BadNISTress M OA
      NIO*?)) (comma2 !(NI RA) &NI)?)

```

```

1339
1340 RA1 <- ((RA (!BadNISTress M a)? (!BadNISTress M OA
      NIO*)?) (comma2 !(NI RA) &NI)?)
1341
1342 # a composite NI word, optional SA prefix before a
      sequence of NI words or a RA word,
1343 # or a single SA word [which will modify a default
      quantifier not expressed],
1344 # possibly negated, connected with CA0 roots to
      other such constructs.
1345
1346 NI2 <- (( (SA? (NI1+/RA1))/SA) NOI? (CA0 ((SA? (NI1
      +/RA1))/SA) NOI?)*
1347
1348 # a full NI word with an acronymic dimension (
      starting with <mue>, ending with a pause) or <cu>
      appended. I need to look up <cu>
1349 # and figure out its semantics. An arbitrary name
      word may now be used as a dimension, as well.
1350
1351 NI <- ([ ]* NI2 (&(M UE) Acronym (comma/&end/&period
      ) !(C u)/comma2? M UE comma2? PreName !(C u))? (C
      u)?)
1352
1353 # mex is now identical with NI, but it's in use in
      later rules.
1354
1355 mex <- ([ ]* NI)
1356
1357 # a word used for various tightly binding
      constructions: a sort of verbal hyphen.
1358 # also a name marker, which means phonetic care is
      needed (pause after constructions with <ci>).
1359
1360 CI <- ([ ]* (C i))
1361
1362 # Acronyms, which are names (not predicates as in
      1989 Loglan) or dimensions (in NI above).

```

```

1363 # units in acronym are TAI0 letterals, zV short
      forms for vowels, the dummy unit <mue>, and NI1
1364 # quantity units. NI1 quantity units may not be
      initial. <mue> units may be preceded by pauses.
1365 # An acronym has at least two units.
1366
1367 # it is worth noting that acronyms, once viewed as
      names, could be entirely suppressed as a feature
      of the
1368 # grammar by really making them names (terminate
      them with -n). I suppose a similar approach
      would work
1369 # for dimensions, allowing any name word to serve as
      a dimension. <mue> would be a name marker for
      use
1370 # with dimensions in this case. <temuedain>, three
      dollars. Now supported.
1371
1372 Acronym <- ([ ]* &caprule ((M UE)/TAI0/(Z V2 !V2))
              ((comma &Acronym M UE)/NI1/TAI0/(Z V2 (!V2/(Z &V2
              ))))+)
1373
1374 # the full class of letterals, including the <gao>
      construction whose details I should look at.
1375
1376 TAI <- ([ ]* (TAI0/((G AO) !V2 [ ]* (PreName/
              Predicate/CmapuaUnit))))
1377
1378 # atomic non-letteral pronouns.
1379
1380 #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.
1381
1382 DAO <- ((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 00)/(D A0))

1383

1384 # letterals (not including <gao> constructions and atomic pronouns optionally suffixed with a digit.

One should pause after the

1385 # suffixed forms, because <ci> is a name marker.

1386

1387 DA1 <- ((TAIO/DA0) (C i ![] NIO)?)

1388

1389 # general pronoun words.

1390

1391 DA <- ([]* DA1)

1392

1393 # 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.

1394

1395 PA0 <- (NI2? (N u !KOU)? ((G IA)/(G UA)/(P 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)/(KOU !(N OI) !KI)) (N OI)? ZI?)

1396

1397 # the form used for actual prepositions and suffixes to A words, with minimal pauses allowed.

1398 # these are built by concatenating KOU2 and PA0 units, then linking these with CA0 roots (which can take

1399 # no- prefixes and -noi suffixes, and next to which one *can* pause), optionally suffixed with a class ZI suffix.

1400

```

1401 PANOPAUSES <- ((KOU2/PA0)+ ((comma2? CA0 comma2?) (
      KOU2/PA0)+)*
1402
1403 # prepositional words
1404
1405 PA3 <- ([ ]* PANOPAUSES)
1406
1407 # class PA can appear as tense markers or as
      relative modifiers without arguments; here pauses
1408 # are allowed not only next to CA0 units but between
      KOU2/PA units. Like NI words, PA
1409 # words are a class of arbitrary length
      constructions, and we think breaths within them
1410 # (especially complex ones) are natural.
1411
1412 PA <- ((KOU2/PA0)+ (((comma2? CA0 comma2?)/(comma2 !
      mod1a)) (KOU2/PA0)+)* !modifier
1413
1414 PA2 <- ([ ]* PA)
1415
1416 GA <- ([ ]* (G a))
1417
1418 # the class of tense markers which can appear before
      predicates.
1419
1420 PA1 <- ((PA2/GA))
1421
1422 # suffixes which indicate extent or remoteness/
      proximity of the action of prepositions.
1423
1424 ZI <- ((Z i)/(Z a)/(Z u))
1425
1426 # the primitive description building "articles".
      These include <la> which requires special
1427 # care in its use because it is a name marker.
1428
1429 LE <- ([ ]* ((L EA)/(L EU)/(L OE)/(L EE)/(L AA)/(L e
      )/(L o)/(L a)))

```

```
1430
1431 # articles which can be used with abstract
      descriptions: these include some quantity words.
1432 # this means that some abstract descriptions are
      semantically indefinites: I wonder if this
1433 # could be improved by having a separate abstract
      indefinite construction.
1434
1435 LEFORPO <- ([ ]* ((L e)/(L o)/NI2))
1436
1437 # the numerical/quantity article.
1438
1439 LIO <- ([ ]* (L IO))
1440
1441 # structure words for the ordered and unordered list
      constructions.
1442
1443 LAU <- ([ ]* (L AU))
1444
1445 LOU <- ([ ]* (L OU))
1446
1447 LUA <- ([ ]* (L UA))
1448
1449 LUO <- ([ ]* (L UO))
1450
1451 ZEIA <- ([ ]* Z EIb a)
1452
1453 ZEIO <- ([ ]* Z EIb o)
1454
1455 # initial and final words for quoting Loglan
      utterances.
1456
1457 LI1 <- (L i)
1458
1459 LU1 <- (L u)
1460
1461 # quoting Loglan utterances, with or without
      explicit double quotes (if they appear, they must
```

```

1462 # appear on both sides). The previous version
      allowed quotation of names; likely this should
1463 # be restored.
1464
1465 LI <- ([ ]* LI1 comma2? utterance0 comma2? LU1/[ ]*
      LI1 comma2? [" ] utterance0 [" ] comma2? LU1)
1466
1467 # the foreign name construction. This is an alien
      text construction
1468
1469 LA0 <- ([ ]* &([Ll] [Aa] [Oo]junction?) AlienWord)
1470
1471 # the strong quotation construction. This is an
      alien text construction.
1472
1473 LIE <- ([ ]* &([Ll] [Ii] juncture? [Ee]junction?)
      AlienWord)
1474
1475 LIO1 <- ([ ]* &([Ll] [Ii] juncture? [Oo]junction?)
      AlienWord)
1476
1477
1478
1479 # I am not sure this class is used at all.
1480
1481 LW <- Cmapua
1482
1483 # articles for quotation of words
1484
1485 LIU0 <- ((L IU)/(N IU))
1486
1487 # this now imposes the condition that an explicit
      comma pause (or terminal punctuation, or end)
      must appear at the end of the
1488 # Word or PreName quoted with <liu>. This seems
      like a good idea, anyway.
1489
1490 # this class appeals to the phonetics. Words and

```

```

    PreNames can be quoted.  The ability to quote
    names
1491 # here may remove the need to quote them with <li
    >...<lu>.  Of course, some Words are in fact
    phrases rather
1492 # than single words:  we will see whether the
    privileges afforded are used.  The final clause
    allows
1493 # use of letterals as actual names of letters.
1494
1495 # added <niu>:  didn't make it a name marker.
1496
1497 LIU1 <- ([ ]* ([Ll]/[Nn])[iI] juncture? [Uu]
    juncture? !V1 comma2? (PreName/Word) &(comma/
    terminal/end) /[ ]*(L II TAI ))
1498
1499 # the construction of foreign and onomatopoeic
    predicates.  These are alien text constructions.
1500
1501 SUE <- ([ ]* &([Ss] [Uu] juncture? [Ee] juncture?/[
    Ss] [Aa] [Oo] juncture?) AlienWord)
1502
1503 # left marker in a predicate metaphor construction
1504
1505 CUI <- ([ ]* (C UI) )
1506
1507 # other uses of GA
1508
1509 GA2 <- ([ ]* (G a) )
1510
1511 # ge/geu act as "parentheses" to make an atomic
    predicate from a complex metaphorically
1512 # and logically connected predicates; <ge> has
    other left marking uses.
1513
1514 GE <- ([ ]* (G e) )
1515
1516 GEU <- ([ ]* ((C UE)/(G EU)) )

```

```
1517
1518 # final marker of a list of head terms
1519
1520 GI <- ([ ]* ((G i)/(G OI)) )
1521
1522 # used to move a normally prefixed metaphorical
      modifier after what it modifies.
1523
1524 GO <- ([ ]* (G o) )
1525
1526 # marker for second and subsequent arguments before
      the predicate; NEW
1527
1528 GIO <- ([ ]* (G IO) )
1529
1530 # the generic right marker of many constructions.
1531
1532 GU <- ([ ]* (G u) )
1533
1534 # various flavors of right markers.
1535
1536 # It should be noted that at one point I executed a
      program of simplifying these to
1537 # reduce the likelihood that multiple <gu>'s would
      ever be needed to close an utterance.
1538 # first of all, I made the closures leaner, moving
      them out of the classes closed
1539 # to their clients so that they generally can be
      used only when needed.
1540 # Notably, the grammar of <guu> is quite different.
      Second,
1541 # I introduced some new flavors of right marker.
      All can be realized with <gu>,
1542 # but if one knows the right flavor one can close
      the right structure with a single
1543 # right closure.
1544
1545 # right markers of subordinate clauses (argument
```

```

        modifiers).
1546 # <gui> closes a different class than in the trial
        .85 grammar, with
1547 # similar but on the whole better results.
1548
1549 GUIZA <- ([ ]* (G UI) (Z a) )
1550
1551 GUIZI <- ([ ]* (G UI) (Z i) )
1552
1553 GUIZU <- ([ ]* (G UI) (Z u) )
1554
1555 GUI <- (!GUIZA !GUIZI !GUIZU ([ ]* (G UI) ))
1556
1557 # right markers of abstract predicates and
        descriptions.
1558 # probably the forms with z are to be preferred (and
        the other
1559 # two are not needed) but I preserve all five
        classes for now.
1560
1561 GUO <- ([ ]* (G UO) )
1562
1563 GUOA <- ([ ]* (G UOb a/G UO Z a) )
1564
1565 GUOE <- ([ ]* (G UOb e) )
1566
1567 GUOI <- ([ ]* (G UOb i/G UO Z i) )
1568
1569 GUOO <- ([ ]* (G UOb o) )
1570
1571 GUOU <- ([ ]* (G UOb u/G UO Z u) )
1572
1573 # right marker used to close term (argument/
        predicate modifier) lists.
1574 # it is important to note that in our grammar GUU is
        not a component of
1575 # the class termset, nor is it a null termset: it
        appears in other classes

```

```
1576 # which include termsets as an option to close them.
      The effects are similar
1577 # to those in the trial.85 grammar, but there is
      less of a danger that
1578 # extra unexpected closures will be needed.
1579
1580 GUU <- ([ ]* (G UU) )
1581
1582 # a new closure for arguments in various contexts
1583
1584 GUUA <- ([ ]* (G UUb a) )
1585
1586 # a new closure for sentences. In particular, it
1587 # may have real use in closing up the scope of a
      list of
1588 # fronted terms before a series of logically
      connected sentences.
1589
1590 GIUO <- ([ ]* (G IUb o) )
1591
1592 # right marker used to close arguments tightly
      linked with JE/JUE.
1593
1594 GUE <- ([ ]* (G UE) )
1595
1596 # a new closure for descpreds
1597
1598 GUEA <- ([ ]* (G UEb a) )
1599
1600
1601 # used to build tightly linked term lists.
1602
1603 JE <- ([ ]* (J e) )
1604
1605 JUE <- ([ ]* (J UE) )
1606
1607 # used to build subordinate clauses (argument
      modifiers).
```



```

1608
1609 JIZA <- ([ ]* ((J IE)/(J AE)/(P e)/(J i)/(J a)/(N u
      J i)) (Z a) )
1610
1611 JIOZA <- ([ ]* ((J IO)/(J AO)) (Z a) )
1612
1613 JIZI <- ([ ]* ((J IE)/(J AE)/(P e)/(J i)/(J a)/(N u
      J i)) (Z i) )
1614
1615 JIOZI <- ([ ]* ((J IO)/(J AO)) (Z i) )
1616
1617 JIZU <- ([ ]* ((J IE)/(J AE)/(P e)/(J i)/(J a)/(N u
      J i)) (Z u) )
1618
1619 JIOZU <- ([ ]* ((J IO)/(J AO)) (Z u) )
1620
1621 JI <- (!JIZA !JIZI !JIZU ([ ]* ((J IE)/(J AE)/(P e)
      /(J i)/(J a)/(N u J i)) ))
1622
1623 JIO <- (!JIOZA !JIOZI !JIOZU ([ ]* ((J IO)/(J AO)) )
      )
1624
1625 # case tags, both numerical position tags and the
      optional semantic case tags.
1626
1627 DIO <- ([ ]* ((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)?
1628
1629 # markers of indirect reference. Originally these
      had the same grammar as case tags,
1630 # but they are now different.
1631
1632 LAE <- ([ ]* ((L AE)/(L UE)) )
1633
1634 # <me> turns arguments into predicates, <meu> closes
      this construction.
1635

```

```

1636 ME <- ([ ]* ((M EA)/(M e)) )
1637
1638 MEU <- ([ ]* M EU )
1639
1640 # reflexive and conversion operators:  first the
      root forms, then those with
1641 # optional numerical suffixes.
1642
1643 NUO <- ((N UO)/(F UO)/(J UO)/(N u)/(F u)/(J u))
1644
1645 NU <- [ ]* (((N u/N UO) !([ ]+ (NIO/RA)) (NIO/RA)?)/
      NUO)+ freemod?
1646
1647 # abstract predicate constructors (from sentences)
1648
1649 # I do *not* think
1650 # that <poia> will really be confused with <po ia>,
      particularly
1651 # since we do require an explicit pause before <ia>
      in the latter case,
1652 # but I record this concern:  the forms with z might
      be preferable.
1653
1654 P01 <- ([ ]* ((P o)/(P u)/(Z o)))
1655
1656 P01A <- ([ ]* ((P 0Ib a)/(P UIb a)/(Z 0Ib a)/(P o Z
      a)/(P u Z a)/(Z o Z a)))
1657
1658 P01E <- ([ ]* ((P 0Ib e)/(P UIb e)/(Z 0Ib e)))
1659
1660 P01I <- ([ ]* ((P 0Ib i)/(P UIb i)/(Z 0Ib i)/(P o Z
      i)/(P u Z i)/(Z o Z i)))
1661
1662 P01O <- ([ ]* ((P 0Ib o)/(P UIb o)/(Z 0Ib o)))
1663
1664 P01U <- ([ ]* ((P 0Ib u)/(P UIb u)/(Z 0Ib u)/(P o Z
      u)/(P u Z u)/(Z o Z u)))
1665

```

```

1666 # abstract predicate constructor from simple
      predicates
1667
1668 POSHORT1 <- ([ ]* ((P OI)/(P UI)/(Z OI)))
1669
1670 # word forms associated with the above abstract
      predicate root forms
1671
1672 PO <- ([ ]* P01 )
1673
1674 POA <- ([ ]* P01A )
1675
1676 POE <- ([ ]* P01E )
1677
1678 POI <- ([ ]* P01E )
1679
1680 POO <- ([ ]* P010 )
1681
1682 POU <- ([ ]* P01U )
1683
1684 POSHORT <- ([ ]* POSHORT1 )
1685
1686 # register markers
1687
1688 DIE <- ([ ]* ((D IE)/(F IE)/(K AE)/(N UE)/(R IE)) )
1689
1690 # vocative forms: I still have the words of social
      lubrication as
1691 # vocative markers.
1692
1693 HOI <- ([ ]* ((H OI)/(L OI)/(L OA)/(S IA)/(S IE)/(S
      IU)) )
1694
1695 # the verbal scare quote. The quantifier suffix
      indicates how many preceding words are affected;
1696 # this is an odd mechanism.
1697
1698 JO <- ([ ]* (NIO/RA/SA)? (J o) )

```

```

1699
1700 # markers for forming parenthetical utterances as
      free modifiers.
1701
1702 KIE <- ([ ]* (K IE) )
1703
1704 KIU <- ([ ]* (K IU) )
1705
1706 KIE2 <- [ ]* K IE comma2? [(]
1707
1708 KIU2 <- [ ]* [)] comma2? K IU
1709
1710 # marker for forming smilies.
1711
1712 SOI <- ([ ]* (S OI) )
1713
1714 # a grab bag of attitudinal words, including but not
      restricted to the VV forms.
1715
1716 UIO <- (!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 IEb i)))
1717
1718 # negative forms of the attitudinals. The ones with
      <no> before the two vowel forms are a phonetic
      exception. The others
1719 # should also be (though they present no
      pronunciation problem) so that they are resolved
      as single words.
1720
1721 NOUI <- (([ ]* UIO NOI)/([ ]* N [o] juncture? comma?
      [ ]* UIO ))
1722
1723 # all attitudinals (adding the discursives nefi,
```

```

    tofi... etc)
1724 # there is a technical problem with mixing UIO roots
      of VV and CVV shapes.
1725
1726 UI1 <- ([ ]* (UIO+/(NI F i)) )
1727
1728 # the inverse vocative marker
1729
1730 HUE <- ([ ]* (H UE))
1731
1732 # occurrences of <no> as a word rather than an affix
      .
1733
1734 NO1 <- ([ ]* !KOU1 !NOUI (N o) !(comma2? Z AO comma2
      ? Predicate) !([ ]* KOU) !([ ]* (JIO/JI/JIZA/
      JIOZA/JIZI/JIOZI/JIZU/JIOZU)) )
1735
1736 # a technical closure for the alternative parser
      approach: the "large subject marker"
1737
1738 GAA <- (NO1 freemod?)* ([ ]* (G AA))
1739
1740
1741 # Names, acronyms and PreNames from above.
1742
1743 AcronymicName <- Acronym &(comma/period/end)
1744
1745 DJAN <- (PreName/AcronymicName)
1746
1747 # predicate words which are phonetically cmapua
1748
1749 # "identity predicates". Converses are provided as
      a new proposal.
1750
1751 BI <- ([ ]* (N u)? ((B IA)/(B IE)/(C IE)/(C IO)/(B
      IA)/(B [i])) )
1752
1753 # interrogative and pronoun predicates

```

```

1754
1755 LWPRED A <- ((H e)/(D UA)/(D UI)/(B UA)/(B UI))
1756
1757 # here I should reinstall the <zao> proposal.
1758
1759 # the predicate words defined above in the phonetics
      section
1760
1761 Predicate <- (CmapuaUnit comma2? Z A0 comma2?)*
      Complex (comma2? Z A0 comma2? Predicate)?
1762
1763 # predicate words, other than the "identity
      predicates" of class [BI]
1764 # these include the numerical predicates (NI RA),
      also cmapua phonetically.
1765
1766 # we are installing John Cowan's <zao> proposal here
      , experimentally, 4/15/2019
1767
1768 PRED A <- ([ ]* &caprule (Predicate/LWPRED A/(![ ] NI
      RA)) )
1769
1770 # Part 3: The Grammar Proper
1771
1772 # right markers turned into classes.
1773
1774 guoa <- (PAUSE? (GUOA/GU) freemod?)
1775
1776 guoe <- (PAUSE? (GUOE/GU) freemod?)
1777
1778 guoi <- (PAUSE? (GUOI/GU) freemod?)
1779
1780 guoo <- (PAUSE? (GUOO/GU) freemod?)
1781
1782 guou <- (PAUSE? (GUOU/GU) freemod?)
1783
1784 guo <- (!guoa !guoe !guoi !guoo !guou (PAUSE? (GUO/
      GU) freemod?))

```

```
1785
1786 guiza <- (PAUSE? (GUIZA/GU) freemod?)
1787
1788 guizi <- (PAUSE? (GUIZI/GU) freemod?)
1789
1790 guizu <- (PAUSE? (GUIZU/GU) freemod?)
1791
1792 gui <- (PAUSE? (GUI/GU) freemod?)
1793
1794 gue <- (PAUSE? (GUE/GU) freemod?)
1795
1796 guea <- (PAUSE? (GUEA/GU) freemod?)
1797
1798 guu <- (PAUSE? (GUU/GU) freemod?)
1799
1800 guua <- (PAUSE? (GUUA/GU) freemod?)
1801
1802 giuo <- (PAUSE? (GIUO/GU) freemod?)
1803
1804 meu <- (PAUSE? (MEU/GU) freemod?)
1805
1806 geu <- GEU
1807
1808 # Here note the absence of pause/GU equivalence.
1809
1810 gap <- (PAUSE? GU freemod?)
1811
1812 # this is the vocative construction. It can appear
    early because all of its components are marked.
1813
1814 # the intention is to indicate who is being
    addressed. This can be handled via a name, a
    descriptive argument, a predicate or an
1815 # alien text name (the last must be quoted). The
    complexities of these grammatical constructions
    can be deferred until they are
1816 # introduced.
1817
```

```

1818 # HOIO <- [ ]* [Hh] [Oo] [Ii] juncture?
1819
1820 # restore words of social lubrication as vocative
      markers but not as name markers: <loi, Djan>
1821
1822 # I do not allow a freemod to intervene between a
      vocative marker and the associated
1823 # utterance, to avoid unintended grabbing of
      subjects by the words of social lubrication when
      they are used
1824 # as vocative markers. This lets <Loi, Djan> and <
      Loi hoi Djan> be equivalent. The comma needed in
      the
1825 # first because the social lubrication words are in
      this version not name markers.
1826
1827 HOIO <- ([ ]* ((([Hh] OI)/([Ll] OI)/([Ll] OA)/([Ss]
      IA)/([Ss] IE)/([Ss] IU)))) juncture? !V1
1828
1829 voc <- (HOIO comma2? name /(HOI comma2? descpred
      guea? namesuffix?)/(HOI comma2? argument1 guua?)
      /[ ]* &([Hh] [Oo] [Ii] juncture?) AlienWord)
1830
1831 # this is the inverse vocative. It can appear early
      because all of its components are marked.
1832
1833 # 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
1834 # handled for the vocative; there is the further
      option of a sentence (the [statement] class) and
      there is a strong closure option
1835 # for the case where an argument is used (to avoid
      it inadvertantly expanding to a sentence).
1836
1837 HUEO <- [ ]* &caprule [Hh] [Uu] juncture? [Ee]
      juncture? !V1
1838

```



```
1839 invvoc <- (HUE0 comma2? name/HUE freemod? descpred
      guea? namesuffix?/(HUE freemod? statement giuo?)
      /(HUE freemod? argument1 guu?)/[ ]* &([Hh] [Uu]
      juncture? [Ee] juncture?) AlienWord)
1840
1841
1842 # this is the class of free modifiers. Most of its
      components are head marked (those that aren't
      appear just above),
1843 # and it is useful for it to appear early because
      these things appear everywhere in subsequent
      constructions. A free modifier,
1844 # of whatever sort, is a freely insertable gadget
      which modifies the immediately preceding
      construction, or the entire utterance
1845 # if it is initial.
1846
1847 # NOUI is a negated attitudinal word. UI1 is an
      attitudinal word: these express an emotional
      attitude toward the
1848 # assertion (noting that EI marks questions (yes or
      no answer expected) and SEU marks utterances as
      answers).
1849
1850 # SOI creates smilies in a general sense: <soi
      crano> indicates that the listener should imagine
      the speaker smiling;
1851 # similarly for other predicates.
1852
1853 # DIE and NO DIE are register markers, communicating
      the social attitude of the speaker toward the
      one addressed: <die> for
1854 # example is "dear"
1855
1856 # KIE...KIU constructs a full parenthetical
      utterance as a comment, which can be enclosed in
      actual parentheses inside
1857 # the marker words.
```

```

1858
1859 # J0 is a scare quote device.
1860
1861 # the comma is a freemod with no semantic content:
      this is a device for discarding phonetically
      required pauses
1862 # and the speaker's optional pauses alike. The
      pause before a non-pause marked prename is part
      of the NameWord and so
1863 # is excluded. Ellipses and dashes are fancy pauses
      supported as freemods.
1864
1865 freemod <- ((NOUI/(SOI freemod? descpred guea?)/DIE
      /(NO1 DIE)/(KIE comma? utterance0 comma? KIU)/(
      KIE2 comma? utterance0 comma? KIU2)/invvoc/voc/(
      comma (!(FalseMarked PreName))/JO/UI1/([ ]* '...'
      ([ ]* &letter)?)/([ ]* '--' ([ ]* &letter)?))
      freemod?)
1866
1867 # the classes juelink to linkargs describe very
      tightly bound arguments which can be firmly
      attached to predicates in
1868 # the context of metaphorical modifications and the
      use of predicates in descriptive arguments.
1869
1870 # note that we allow predicate modifiers (
      prepositional phrases) to be bound with <je/jue>
      which is not
1871 # allowed in 1989 Loglan, but which we believe is
      supported in Lojban.
1872
1873 juelink <- (JUE freemod? (term/(PA2 freemod? gap?)))
1874
1875 links1 <- (juelink (freemod? juelink)* gue?)
1876
1877 links <- ((links1/(KA freemod? links freemod? KI
      freemod? links1)) (freemod? A1 freemod? links1)*)
1878

```

```

1879 jelink <- (JE freemod? (term/(PA2 freemod? gap?)))
1880
1881 linkargs1 <- (jelink freemod? (links/gue?))
1882
1883 linkargs <- ((linkargs1/(KA freemod? linkargs
      freemod? KI freemod? linkargs1)) (freemod? A1
      freemod? linkargs1)*)
1884
1885 # class abstractpred supports the construction of
      event, property, and quantity predicates from
      sentences. These are
1886 # closable with <guo> if introduced with <po,pu,zo>
      and closable with suffixed variants of <guo> if
      introduced with suffixed
1887 # variants of <po,pu,zo> (a NEW idea but it is clear
      that closure of these predicates (and of the
      more commonly
1888 # used associated descriptions) is an important
      issue).
1889
1890 abstractpred <- ((POA freemod? uttAx guoa?)/(POA
      freemod? sentence guoa?)/(POE freemod? uttAx guoe
      ?)/(POE freemod? sentence guoe?)/(POI freemod?
      uttAx guoi?)/(POI freemod? sentence guoi?)/(POO
      freemod? uttAx guoo?)/(POO freemod? sentence guoo
      ?)/(POU freemod? uttAx guou?)/(POU freemod?
      sentence guou?)/(PO freemod? uttAx guo?)/(PO
      freemod? sentence guo?))
1891
1892 # predunit1 describes the truly atomic forms of
      predicate.
1893
1894 # PREDA is the class of predicate words (the
      phonetic predicate words along with the special
      phonetic cmapua which are predicates, listed
1895 # above under the PREDA rule. NU PREDA handles
      permutations and identifications of arguments of
      PREDAs.

```

```

1896
1897 # SUE contains the alien text constructions with <
      sao> and <sue>, semantically quite different but
      syntactically handled
1898 # in the same way.
1899
1900 # <ge>...<geu/cue> (the closing optional) can
      parenthesize a fairly complex predicate phrase
      and turn it into an atomic form.  These
1901 # forms can have conversion or reflexive operators (
      NU) applied.  I should look into why the class
      handled in the conversion case
1902 # is different.  An important use of this is in
      metaphor constructions, but it has other
      potential uses.
1903
1904 # abstractpred is the class of abstraction
      predicates just introduced above.  These are
      treated as atomic in this grammar:  it should
1905 # be noted that their privileges in the trial.85
      grammar are (absurdly) limited.
1906
1907 # <me>...<meu> (the closing optional, but important
      to have available) forms predicates from
      arguments, the predicate being true of the
1908 # objects to which the argument refers.  <Ti me le
      mrenu> :  this is one of the men we are talking
      about.
1909
1910 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?)
1911
1912 # <no> binds very tightly to predunit1:  a possibly
      multiply negated predunit1 (or an unadorned
      predunit1) is a predunit2.

```

```
1913
1914 predunit2 <- ((N01 freemod?)* predunit1)
1915
1916 # an instance of N02 is one not absorbed by a
      predunit. Example: <Da no kukra prano> X is a
      slow (not-fast) runner vs
1917 # <Da no ga kukra prano> (X is not a fast runner,
      and in fact may not run at all).
1918
1919 N02 <- (!predunit2 N01)
1920
1921 # a predunit3 is a predunit2 with tightly attached
      arguments.
1922
1923 predunit3 <- ((predunit2 freemod? linkargs)/
      predunit2)
1924
1925 # a predunit is a predunit3 or a predunit3 converted
      by the short-scope abstraction operators
1926 # <poi/pui/zoi> to an abstraction predicate. This
      is the kind of predicate which can appear as
1927 # a component in a serial name.
1928
1929 predunit <- ((POSHORT freemod?)? predunit3)
1930
1931 # a further "atomic" (because tightly packaged) form
      is a forethought connected pair
1932 # of predicates (this being the full predicate class
      defined at the end of the process)
1933 # possibly closed with <guu>, possibly multiply
      negated as well.
1934
1935 # the closure with guu eliminated the historic rule
      against kekked heads of metaphors.
1936
1937 kekpredunit <- ((N01 freemod?)* KA freemod?
      predicate freemod? KI freemod? predicate guu?)
1938
```

```

1939 # there follows the construction of metaphorically
      modified predicates ,
1940 # along with tightly logically linked predicates.
1941
1942 # CI and simple juxtaposition of predicates both
      represent modification of the second
1943 # predicate by the first. We impose no semantic
      conditions on this modification,
1944 # except in the case of modification by predicates
      logically linked with CA,
1945 # which do distribute logically in the expected way
      both as modifiers and as modified.
1946 # We do not regard <preda1 pred2> as necessarily
      implying pred2: we do regard
1947 # it as having the same place structure as pred2.
      It is very often but not always
1948 # a qualification or kind of pred2; in any case it
      is a relation analogous to pred2.
1949
1950 # modification with CI binds most tightly.
1951
1952 # we eliminated the distinction between the series
      of sentence and description
1953 # predicate preliminary classes: there seems to be
      no need for it even in the
1954 # trial.85 grammar.
1955
1956 despredA <- ((predunit/kekpredunit) (freemod? CI
      freemod? (predunit/kekpredunit))*
1957
1958 # this is logical connection of predicates with the
      tightly binding CA
1959 # series of logical connectives. CUI can be used to
      expand the scope of
1960 # a CA connective over a metaphor on the left. <ge
      >...<geu> is used to expand
1961 # scope on the right (and could also be used on the
      left, it should be noted).

```

```

1962 # descpredC is an internal of despredB assisting the
      function of CUI.
1963 # the !PREDA in front of CUI is probably not needed.
1964
1965 despredB <- ((!PREDA CUI freemod? despredC freemod?
      CA freemod? despredB)/despredA)
1966
1967 despredC <- (despredB (freemod? despredB)*)
1968
1969 # tight logical linkage of despredB's
1970
1971 despredD <- (despredB (freemod? CA freemod? despredB
      )*)
1972
1973 # chain of modifications of despredD's (grouping to
      the left)
1974
1975 despredE <- (despredD (freemod? despredD)*)
1976
1977 # the GO construction allows inverse modification:
      <preda1 GO preda2> is <preda2 preda1> as it were.
1978 # there are profound effects on grouping.
1979
1980 descpred <- ((despredE freemod? GO freemod? descpred
      )/despredE)
1981
1982 # this version which appears in sentence predicates
      as opposed to descriptions differs
1983 # in allowing loosely linked arguments (termsets)
      instead of those linked with <je/jue> for the
      predicate
1984 # moved to the end by GO.
1985
1986 # 4/17/2019 shared argument experiment
1987
1988 # sentpred <- (( (KA freemod? sentpred freemod? KI
      freemod? sentpred !guu)/ (despredE freemod? GO
      freemod? barepred)/despredE) ) (A2 freemod? ((

```

```

        despredE freemod? G0 freemod? barepred)/despredE)
    )*
1989
1990 sentpred <- ((despredE freemod? G0 freemod? barepred
    )/despredE)
1991
1992 # sentpred <- ((despredE freemod? G0 freemod?
    barepred)/despredE) (A1 ((despredE freemod? G0
    freemod? barepred)/despredE))*
1993
1994 # the construction of predicate modifiers (
    prepositional phrases usable as terms along with
    arguments).
1995
1996 mod1a <- (PA3 freemod? argument1 guua?)
1997
1998 # note special treatment of predicate modifiers
    without actual arguments.
1999 # the !barepred serves to distinguish these
    predicate modifiers from actual
2000 # "tenses" (predicate markers).
2001
2002 mod1 <- ((PA3 freemod? argument1 guua?)/(PA2 freemod
    ? !barepred gap?))
2003
2004 # forethought connection of modifiers. There is
    some subtlety in
2005 # how this is handled.
2006
2007 kekmod <- ((N01 freemod?)* (KA freemod? modifier
    freemod? KI freemod? mod))
2008
2009 mod <- (mod1/((N01 freemod?)* mod1)/kekmod)
2010
2011 # afterthought connection of modifiers
2012
2013 modifier <- (mod (A1 freemod? mod))*
2014

```



```

2015 # the serial name is a horrid heterogenous
      construction! It can involve
2016 # components of all three of the major phonetic
      classes essentially!
2017
2018 # However, I believe I have the definition right,
      with all the components
2019 # correctly guarded :-)
2020
2021 name <- (PreName/AcronymicName) (comma2? !
          FalseMarked PreName/comma2? &([Cc] [Ii]) NameWord
          /comma2? CI predunit !(comma2? (!FalseMarked
          PreName))/comma2? CI AcronymicName)* freemod?
2022
2023 LA0 <- [ ]* [Ll] [Aa] juncture?
2024
2025 LANAME <- (LA0 comma2? name)
2026
2027 # general constructions of arguments with "articles
      ".
2028
2029 # the rules here have the "possessive" construction
      as in <lemi hasfa; le la Djan, hasfa> embedded.
      These are not the same
2030 # construction in 1989 Loglan, though speakers might
      think they are. Here they are indeed the same.
      The "possessor" cannot
2031 # be "indefinite" (cannot start with a quantifier
      word); the possessor can be followed by a tense,
      as in
2032 # <le la Djan, na hasfa>, "John's present house", by
      analogy with <lemina hasfa>, which is accepted
      by LIP (because
2033 # LIP accepts <lemina> as a word).
2034
2035 # there are other subtleties to be reviewed.
2036
2037 #descriptn <- (!LANAME ((LAU wordset1)/(LOU wordset2

```

```

)/(LE freemod? ((!mex arg1a freemod?)? (PA2
freemod?)?)? mex freemod? descpred)/(LE freemod?
 ((!mex arg1a freemod?)? (PA2 freemod?)?)? mex
freemod? arg1a)/(GE freemod? mex freemod?
descpred)/(LE freemod? ((!mex arg1a freemod?)? (
PA2 freemod?)?)? descpred)))
2038
2039 descriptn <- (!LANAME ((LAU wordset1)/(LOU wordset2)
/(LE freemod? ((!mex arg1a freemod?)? (PA2
freemod?)?)? (mex freemod? arg1a/mex freemod?
descpred/descpred))/(GE freemod? mex freemod?
descpred)))
2040
2041
2042 # abstract descriptions. Note that abstract
descriptions are closed with <guo> entirely
independently of abstract predicates:
2043 # <le po preda guo> does not have a grammatical
component <po preda guo>. This avoids the double
closure often apparently necessary
2044 # in Lojban.
2045
2046 abstractn <- ((LEFORPO freemod? POA freemod? uttAx
guoa?)/(LEFORPO freemod? POA freemod? sentence
guoa?)/(LEFORPO freemod? POE freemod? uttAx guoe
?)/(LEFORPO freemod? POE freemod? sentence guoe?)
/(LEFORPO freemod? POI freemod? uttAx guoi?)/(
LEFORPO freemod? POI freemod? sentence guoi?)/(
LEFORPO freemod? P00 freemod? uttAx guoo?)/(
LEFORPO freemod? P00 freemod? sentence guoo?)/(
LEFORPO freemod? POU freemod? uttAx guou?)/(
LEFORPO freemod? POU freemod? sentence guou?)/(
LEFORPO freemod? PO freemod? uttAx guo?)/(LEFORPO
freemod? PO freemod? sentence guo?))
2047
2048 # a wider class of basic argument constructions.
Notice that LANAME is always read by preference
to descriptn.

```

```

2049
2050 namesuffix <- (&(comma2 !FalseMarked PreName/[ ]* [
      Cc][Ii] juncture? comma2? (PreName/AcronymicName)
      ) ([ ]* [Cc][Ii] juncture? comma2?/comma2)? name)
2051
2052 arg1 <- (abstractn/(LIO freemod? descpred guea?)/(
      LIO freemod? argument1 guua?)/(LIO freemod? mex
      gap?)/LIO1/LAO/LANAME/(descriptn guua? namesuffix
      ?)/LIU1/LIE/LI)
2053
2054 # this adds pronouns (incl. the fancy <gao>
      letterals) and the option of left marking an
      argument with <ge>
2055
2056 arg1a <- ((DA/TAI/arg1/(GE freemod? arg1a)) freemod
      ?)
2057
2058 # argument modifiers (subordinate clauses)
2059
2060 argmod1 <- ((([ ]* (N o) [ ]*)? ((JI freemod?
      predicate)/(JIO freemod? sentence)/(JIO freemod?
      uttAx)/(JI freemod? modifier)/(JI freemod?
      argument1)))/((([ ]* (N o) [ ]*)? (((JIZA freemod?
      predicate) guiza?)/((JIOZA freemod? sentence)
      guiza?)/((JIOZA freemod? uttAx) guiza?)/((JIZA
      freemod? modifier) guiza?)/(JIZA freemod?
      argument1 guiza?)))/((([ ]* (N o) [ ]*)? ((JIZI
      freemod? predicate guizi?)/(JIOZI freemod?
      sentence guizi?)/(JIOZI freemod? uttAx guizi?)/(
      JIZI freemod? modifier guizi?)/(JIZI freemod?
      argument1 guizi?)))/((([ ]* (N o) [ ]*)? ((JIZU
      freemod? predicate guizu?)/(JIOZU freemod?
      sentence guizu?)/(JIOZU freemod? uttAx guizu?)/(
      JIZU freemod? modifier guizu?)/(JIZU freemod?
      argument1 guizu?))))))
2061
2062 # we improved the trial.85 grammar by closing not
      argmod1 but argmod with <gui>. But the labelled

```

```

        argument modifier constructors
2063 # when building an argmod1 have the argmod1
        construction closed with the corresponding
        labelled right marker, of course. Thus
2064 # gui and guiza actually have different grammar.
2065
2066 # trial.85 did not provide forethought connected
        argument modifiers, and we also see no need for
        them,
2067 # though they could readily be added.
2068
2069 argmod <- (argmod1 (A1 freemod? argmod1)* gui?)
2070
2071 # affix argument modifiers to a definite argument
2072
2073 arg2 <- (arg1a freemod? argmod*)
2074
2075 # build a possibly indefinite argument from an
        argument:  to le mrenu
2076
2077 arg3 <- (arg2/(mex freemod? arg2))
2078
2079 # build an indefinite argument from a predicate
2080
2081 indef1 <- (mex freemod? descpred)
2082
2083 # affix an argument modifier to an indefinite
        argument
2084
2085 indef2 <- (indef1 guua? argmod*)
2086
2087 indefinite <- indef2
2088
2089 # link arguments with the fusion connective <ze>
2090
2091 arg4 <- ((arg3/indefinite) (ZE2 freemod? (arg3/
        indefinite))*))
2092

```

```
2093 # forethought connection of arguments. Note use of
      argx
2094
2095 arg5 <- (arg4/(KA freemod? argument1 freemod? KI
           freemod? argx))
2096
2097 # arguments with possible negations followed by
           possible indirect reference constructions.
2098
2099 argx <- ((N01 freemod?)* (LAE freemod?)* arg5)
2100
2101 # afterthought connection with the tightly binding
           ACI connectives
2102
2103 arg7 <- (argx freemod? (ACI freemod? argx)?)
2104
2105 # afterthought connection with the usual A
           connectives. Can't start with GE
2106 # to avoid an ambiguity (to which 1989 Loglan is
           vulnerable) involving AGE connectives.
2107
2108 arg8 <- (!GE (arg7 freemod? (A1 freemod? arg7)*))
2109
2110 # afterthought connection (now right grouping,
           instead of the left grouping above)
2111 # using the AGE connectives. GUU can be used to
           affix an argument modifier at this top level.
2112
2113 argument1 <- (((arg8 freemod? AGE freemod? argument1
                )/arg8) (GUU freemod? argmod)*)
2114
2115 # possibly negated and case tagged arguments. We (
           unlike 1989 Loglan) are careful
2116 # to use argument only where case tags are
           appropriate.
2117
2118 argument <- ((N01 freemod?)* (D10 freemod?)*
                argument1)
```

```
2119
2120
2121
2122
2123
2124
2125 # an argument which is actually case tagged.
2126
2127 argxx <- (&((N01 freemod?)* DIO) argument)
2128
2129 # arguments and predicate modifiers actually
      associated with predicates.
2130
2131 term <- (argument/modifier)
2132
2133 # a term list consisting entirely of modifiers.
2134
2135 modifiers <- (modifier (freemod? modifier)*)
2136
2137 # a term list consisting entirely of modifiers and
      tagged arguments.
2138
2139 modifiersx <- ((modifier/argxx) (freemod? (modifier/
      argxx))*)
2140
2141 # the subject class is a list of terms (arguments
      and predicate modifiers) in which all but
      possibly one
2142 # of the arguments are tagged, and there is at least
      one argument, tagged or otherwise.
2143
2144 subject <- ((modifiers freemod?)? ((argxx subject)/(
      argument (modifiersx freemod?)?)))
2145
2146 # this case is identified as an aid to experimental
      termination of argument lists
2147
2148 statement1 <- (subject freemod? (GIO freemod? terms1
```

```
    )? predicate)
2149
2150 # these classes are exactly argument, but are used
    to signal
2151 # which argument position after the predicate an
    argument occupies.
2152 # I think the grammar is set up so that these will
    actually
2153 # never be case tagged, though the grammar does not
    expressly forbid it.
2154
2155 # I am trying a simple version of the "alternative
    parser" approach:
2156 # a term list will refuse to digest an argument
    which starts a new
2157 # SVO sentence (statement1).
2158
2159 argumentA <- !statement1 argument
2160
2161 # argumentA <- argument
2162
2163 argumentB <- !statement1 argument
2164
2165 # argumentB <- argument
2166
2167 argumentC <- !statement1 argument
2168
2169 # argumentC <- argument
2170
2171 argumentD <- !statement1 argument
2172
2173 # argumentD <- argument
2174
2175 # for argument lists not guarded against absorbing a
    following subject
2176
2177 argumentA1 <- argument
2178
```

```
2179 argumentB1 <- argument
2180
2181 argumentC1 <- argument
2182
2183 argumentD1 <- argument
2184
2185 # a general term list. It cannot contain more than
      four untagged arguments (they will be labelled
2186 # with the lettered subclasses given above).
2187
2188 terms <- ((modifiersx? argumentA (freemod?
      modifiersx)? argumentB? (freemod? modifiersx)?
      argumentC? (freemod? modifiersx)? argumentD?)/
      modifiersx)
2189
2190 # terms list not guarded against absorbing a
      following subject
2191
2192 terms1 <- ((modifiersx? argumentA1 (freemod?
      modifiersx)? argumentB1? (freemod? modifiersx)?
      argumentC1? (freemod? modifiersx)? argumentD1?)/
      modifiersx)
2193
2194 # innards of ordered and unordered list
      constructions. These are something I totally
      rebuilt, as they were in a totally
2195 # unsatisfactory state in trial.85. Note the use of
      comma words to separate items in lists.
2196
2197 word <- (arg1a/indef2)
2198
2199 words1 <- (word (ZEIA word)*)
2200
2201 words2 <- (word (ZEIO word)*)
2202
2203 wordset1 <- (words1? LUA)
2204
2205 wordset2 <- (words2? LUO)
```



```
2206
2207 # the full term set type to be affixed to predicates
      .
2208
2209 # forethought connection of term lists
2210
2211 termset1 <- (terms/(KA freemod? termset2 freemod?
      guu? KI freemod? termset1))
2212
2213 # afterthought connection of term lists. There are
      cunning things going on here getting <guu>
2214 # to work correctly. Note that <guu> is NOT a null
      term list as it was in trial.85.
2215
2216 termset2 <- (termset1 (guu &A1)? (A1 freemod?
      termset1 (guu &A1)?)*)
2217
2218 # there is an interesting option here of a list of
      terms followed by <go> followed by a predicate
2219 # intended to metaphorically modify the predicate to
      which the terms are affixed. Is there a reason
2220 # why we cannot have a more complex construction in
      place of terms?
2221
2222 termset <- ((terms freemod? GO freemod? barepred)/
      termset2)
2223
2224 # this is the untensed predicate with arguments
      attached. Here is the principal locus
2225 # of closure with <guu>, but it is deceptive to say
      that <guu> merely closes barepred,
2226 # as we have seen above, for example in [termset2].
2227
2228 # modified for 4/17/2019 shared argument experiment
2229
2230 barepred <- (sentpred freemod? ((termset guu?)/(guu
      (&termset))))?)
2231
```

```

2232 # barepred <- (sentpred freemod? ((termset guu?)/(
      guu (&termset/&A1))))?)
2233
2234 # tensed predicates
2235
2236 markpred <- (PA1 freemod? barepred)
2237
2238 # there follows an area in which my grammar looks
      different from trial.85.  Distinct parallel forms
      for
2239 # marked and unmarked predicates are demonstrably
      not needed even in trial.85.  The behavior of the
      ACI
2240 # connectives is plain weird in trial.85; here we
      treat ACI connectives in the same way as A
      connectives, but
2241 # binding more tightly.
2242
2243 # units for the ACI construction following --
      possibly multiply negated bare or marked
      predicates.
2244
2245 # adding shared termsets to logically connected
      predicates are handled differently here than in
      trial.85,
2246 # which uses a very elegant but dreadfully left-
      grouping rule which a PEG cannot handle.  Any
      realistic situation
2247 # should be manageable.
2248
2249 backpred1 <- ((N02 freemod?)* (barepred/markpred))
2250
2251 # ACI connected predicates.  Shared termsets are
      added.  Notice how we first group backpred1's
      then recursively
2252 # group backpreds.
2253
2254 backpred <- (((backpred1 (ACI freemod? backpred1))+

```

```

freemod? ((termset guu?)/(guu &termset))?) ((ACI
freemod? backpred)+ freemod? ((termset guu?)/(guu
&termset))?)?)/backpred1)
2255
2256 # A connected predicates; same comments as just
      above. Cannot start with GE to fix ambiguity
      with AGE connectives.
2257
2258 predicate2 <- (!GE (((backpred (A1 !GE freemod?
      backpred)+ freemod? ((termset guu?)/(guu &termset
      ))?) ((A1 freemod? predicate2)+ freemod? ((
      termset guu?)/(guu &termset))?)?)/backpred))
2259
2260 # predicate2's linked with right grouping AGE
      connectives (A and ACI are left grouping).
2261
2262 predicate1 <- ((predicate2 AGE freemod? predicate1)/
      predicate2)
2263
2264 # identity predicates from above, possibly negated
2265
2266 identpred <- ((N01 freemod?)* (BI freemod? argument1
      guu?))
2267
2268 # predicates in general. Note that identity
      predicates cannot be logically connected
2269 # except by using forethought connection (see above)
      .
2270
2271 predicate <- (predicate1/identpred)
2272
2273
2274 # The gasent is a basic form of the Loglan sentence
      in which the predicate leads.
2275 # The basic structure is <PA word (usually a tense)
      or <ga>) followed optionally by terms followed
      optionally by
2276 # <ga> followed by terms. The list of terms after <

```

```

    ga> (if present) will either contain
2277 # at least one argument and no more than one
    untagged argument
2278 # (a subject) [gasent1] or all the arguments of the
    predicate [gasent2]. We deprecate other
    arrangements possible in
2279 # 1989 Loglan because they would cause unexpected
    reorientation of the arguments already given
    before <ga> as second
2280 # and further arguments were read after <ga>. [
    barepred] is an untensed predicate possibly with
    arguments; [sentpred]
2281 # is "simply a verb", i.e., a predicate without
    arguments.
2282
2283 # there is a semantic change from 1989 Loglan
    reflected in a grammar change here:
2284 # in [gasent1] the final (ga subject) is optional.
    When it does not appear, the resulting
2285 # sentence is an observative (a sentence with
    subject omitted), not an imperative.
2286 # Imperatives for us are unmarked.
2287
2288 # In the alternative version, the use of the large
    subject marker GAA can prevent inadvertant
    absorption of a preceding trailing argument into
    a statement
2289
2290 # 4/22 allowing general predicates in gasent.
    Otherwise the spaces of observatives and
    imperatives become quite confused.
2291
2292 #gasent1 <- ((NO1 freemod?)* (GAA? freemod? PA1
    freemod? barepred (GA2 freemod? subject?))
2293
2294 gasent1 <- ((NO1 freemod?)* (GAA? freemod? &markpred
    predicate (GA2 freemod? subject?))
2295

```

```
2296 gasent2 <- ((N01 freemod?)* (GAA? PA1 freemod?  
    sentpred modifiers? (GA2 freemod? subject freemod  
    ? GIO? freemod? terms?)))  
2297  
2298 gasent <- (gasent2/gasent1)  
2299  
2300 # this is the simple Loglan sentence in various  
    basic orders. The form "gasent" is discussed  
    just above.  
2301 # Predicate modifiers  
2302 # can be prefixed to the gasent. The final form  
    given here is the basic SVO sentence. The "  
    subject" class is a list of terms  
2303 #(arguments and predicate modifiers) containing at  
    most one un-case-tagged argument. The most  
    general SVO form is subject, followed optionally  
2304 #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  
2305 #in which preceding constructions with errors in  
    them may supply extra unintended arguments. It  
    should be noted in NB3 that JCB envisioned  
2306 #a single argument before the predicate, followed by  
    the predicate, which may itself contain further  
    arguments. A gasent may optionally be negated  
2307 #(even multiple times).  
2308  
2309 # re <gio> and some other changes, in his comments  
    on the NB3 grammar JCB often notes restrictions  
    on appearances of term lists which he  
2310 # intends but which he thought were hard to  
    implement in the machine grammar. The appearance  
    of just one argument before the "verb"  
2311 # 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  
2312 # consider it a virtue to have unmarked SOV after
```

```

    observing unintended parses appearing in the
    Visit text). Another example of this
2313 # (which would not have been hard for JCB to
    implement, in fact) is our restriction of the
    form "terms gasent" to "modifiers gasent".
2314 # His comments make it clear that he does not want
    arguments among those terms.
2315
2316 statement <- (gasent/(modifiers freemod? gasent)/(
    subject freemod? GAA? freemod? (GIO freemod?
    terms1)? predicate))
2317
2318 # this is a forethought connected basic sentence.
    It is odd (and actual odd results can be
    exhibited) that the final segment in both
2319 # of these rules is of the very general class uttA1,
    which includes some quite fragmentary utterances
    usually intended as answers.
2320
2321 # 12/20/2017 I rewrote the rule in a more compact
    form. This rule looks ahead to the class [
    sentence] which we now develop;
2322 # for the moment notice that [sentence] will include
    [statement].
2323
2324 # 4/14 tentatively allowing initial modifiers here
    and leaving this out of uttA0 which replaces
    uttA1 below.
2325 # The intention is to eliminate weird sentence
    fragments.
2326
2327 keksent <- modifiers? freemod? (NO1 freemod?)* (KA
    freemod? headterms? freemod? sentence freemod? KI
    freemod? uttA0)
2328
2329 # sentence negation. We allow this to be set off
    from the main sentence with a mere pause, because
    generally

```

```
2330 # it does not differ in meaning from the result of
      negating the first argument or predicate modifier
      .
2331
2332 neghead <- ((NO1 freemod? gap)/(NO2 PAUSE))
2333
2334 # this class includes [statement], predicate
      modifiers preceding a predicate (which may
      contain arguments), a statement,
2335 # a predicate, and a keksent. Of these, the first
      and third are imperatives.
2336
2337 # in the alternative version, the large subject
      marker GAA can prevent inadvertant absorption of
      preceding trailing arguments into a statement
2338
2339 # 4/23/2019 added actual rule for imperative
      sentences. This should not
2340 # affect the parse in any essential way.
2341
2342 imperative <- ((modifiers freemod?)? GAA? !gasent
      predicate)
2343
2344 sen1 <- (neghead freemod?)* (imperative/statement/
      keksent)
2345
2346 # sen1 <- ((neghead freemod?)* ((modifiers freemod?
      GAA? !gasent predicate)/statement/GAA? predicate/
      keksent))
2347
2348 # the class [sentence] consists of sen1's
      afterthought connected with A connectives
2349
2350 sentence <- (sen1 (ICA freemod? sen1)*)
2351
2352 # [headterms] is a list of terms (arguments and
      predicate modifiers) ending in <gi>. Preceding a
      [sen1] with these
```

```

2353 # causes all predicates in the [sen1] to share these
      arguments. We propose either that the headterms
      arguments be directly
2354 # 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
2355 # of the headterms list, and the list is inserted at
      the appropriate position in each component
      sentence. Neither of these is
2356 # the condition described in Loglan I, which
      presupposes that we always know what the last
      argument of each predicate used is.
2357
2358 headterms <- (terms GI)+
2359
2360 # this is the previous class prefixed with a list of
      fronted terms.
2361 # we think the <giuo> closure might prove useful.
2362
2363 uttAx <- (headterms freemod? sentence giuo?)
2364
2365 # weird answer fragments
2366
2367 uttA <- ((A1/mex) freemod?)
2368
2369 # a broad class of utterances, including various
      things one would usually only say as answers.
      Notice
2370 # that this utterance class can take terminal
      punctuation.
2371
2372 uttA0 <- sen1/uttAx
2373
2374 uttA1 <- ((sen1/uttAx/links/linkargs/argmod/(
      modifiers freemod? keksent)/terms/uttA/N01)
      freemod? period?)
2375
2376 # possibly negated utterances of the previous class.

```



```
2377
2378 uttC <- ((neghead freemod? uttC)/uttA1)
2379
2380 # utterances linked with more tightly binding ICI
      sentence connectives. Single sentences are of
      this class
2381 # if not linked with ICI or ICA.
2382
2383 uttD <- ((sentence period? !ICI !ICA)/(uttC (ICI
      freemod? uttD)*))
2384
2385 # utterances of the previous class linked with ICA.
      I went to some trouble to ensure that a
      freestanding
2386 # [sentence] is actually parsed as a sentence, not a
      composite uttD, which was a deficiency, if not
      an ambiguity of
2387 # LIP and of the trial.85 grammar.
2388
2389 uttE <- (uttD (ICA freemod? uttD)*)
2390
2391 # utterances of the previous class linked with I
      sentence connectives.
2392
2393 uttF <- (uttE (I freemod? uttE)*)
2394
2395 # the utterance class for use in the context of
      parenthetical freemods or quotations, in which it
      does not go to end of text.
2396
2397 utterance0 <- (!GE ((!PAUSE freemod period?
      utterance0)/(!PAUSE freemod period?)/(uttF IGE
      utterance0)/uttF/(I freemod? uttF?)/(I freemod?
      period?)/(ICA freemod? uttF)) (&I utterance0)?)
2398
2399 # Notice that there are two passes here: the parser
      first checks that the entire utterance
2400 # is phonetically valid, then returns and checks for
```

```

        grammatical validity.
2401
2402 # the full utterance class. This goes to end of
        text, and incorporates the phonetics check. This
        incorporates the only situations
2403 # in which a freemod is initial. The IGE
        connectives bind even more loosely than the I
        connectives and right-group instead of
2404 # left grouping.
2405
2406 utterance <- &(phoneticutterance !.) (!GE ((!PAUSE
        freemod period? utterance)/(!PAUSE freemod period
        ? (&I utterance)? end)/(uttF IGE utterance)/(I
        freemod? period? (&I utterance)? end)/(uttF (&I
        utterance)? end)/(I freemod? uttF (&I utterance)?
        end)/(ICA freemod? uttF (&I utterance)? end)))

```