Besri Lamda Grammar
with Texts and Vocabulary

Nicolas Antonio Cloutier

February 12, 2024

λ

—R'e Teri-Pratxe

Table I: Consonants

	Labial	**Alveolar**	**Postalveolar**	**Palatal**	**Velar**	**Glottal**
Plosive	p	t d			k g	ʔ
Nasal	m	n		ɲ		
Trill		r				
Fricative	f	s	ʃ			h
Approximant	w	l			(w)	

Table II: Romanization of consonants

	Labial	**Alveolar**	**Postalveolar**	**Palatal**	**Velar**	**Glottal**
Plosive	p	t d			c/qu g/gu	’
Nasal	m	n	ñ			
Trill		r				
Fricative	f	z/c	x			j
Approximant	hu	l				

Figure I: Vowels

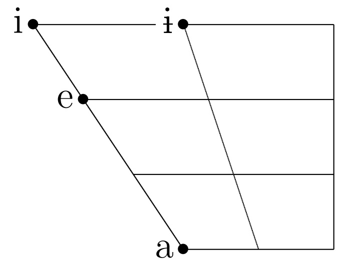
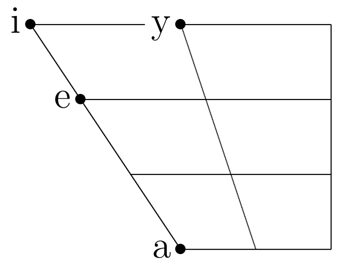


Figure II: Romanization of vowels



In unstressed syllables, /a/ is prounced as [ɐ], and /i/ as [ɪ]. Spanish rules are followed when multiple romanizations given. For example, /si/ is written as ⟨ci⟩, but /sa/ is written as ⟨za⟩, /gi/ is written as ⟨gui⟩ but /ga/ is written as ⟨ga⟩. All syllables in Landa are (C)(C)V. Adjacent vowels are treated as nuclei of separate syllables. Stress can be varied, and is marked by the acute diacritic, unless stress is on the penultimate syllable of a multisyllabic word. If a monosyllabic word receives stress, its vowel is marked with an acute.

2 Grammar

2.1 *sre* ⋯ *xi* ⋯ *sre* ⋯ — `let`

A `let` statement can be used to define a variable, and state its equality to some known concept. In Besri Lamda, all variables begin with vowels, but the specific form is up to the speaker. Because the semantic value of variables is determined through the process of speech, the phonological structure of this variable is completely unimportant so long as it is defined properly. This means that variables have no internal strucutre in terms of morphology, and are essentially atoms that store semantic meaning.

The first part of a `let` statement is *sre*, which marks that a `let` statement is beginning. After this, the name of the variable should be given. This is then followed by *xi*, which declares the semantic class of the variable, which is, just as the variable itself, immutable. After this, another *sre* is given, followed by the definition of the word. Take the following example:

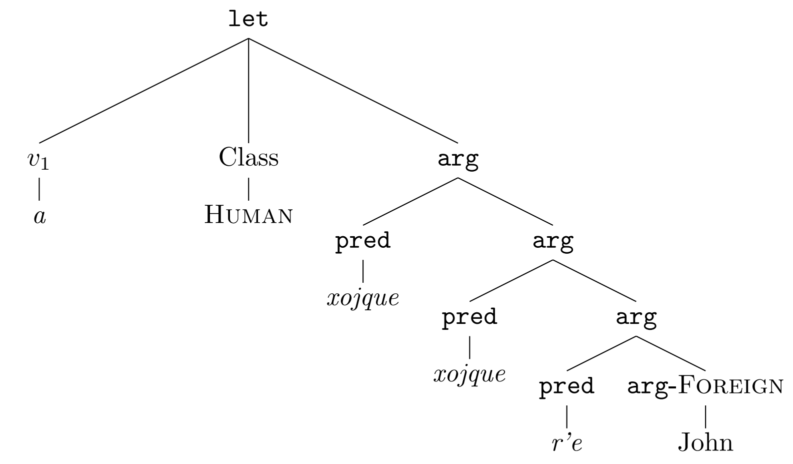
(1)	sre	a	xi	jú	sre	xojque	xojque	r'e	John
	`let`.BEGIN	v_1	CLASS	HUMAN	`let`.END	parent	parent	from.name	John

Let *a* refer to John's grandparent.

Here, a variable *a* (glossed as variable 1, or v_1) is instantiated and declared to refer to an object of the *jú* (HUMAN) semantic class. Further, it is defined to refer to the parent of the parent of the person named John. This is achieved first through looking at the final name, and then applying the predicate *r'e* to it, which is of the type [FOREIGN → HUMAN], that is, it takes a proper noun and returns a human. Specifically, this predicate returns a human that is of a particular name given the proper noun passed as its sole argument. Next, the predicate *xojque*, when applied to a human argument, has the type [HUMAN → HUMAN], and returns a parent of a human. Thus, applying this function to its own result would give the grandparent of the initial argument, that being *r'e John* (or the person referred to by the name "John"), and would finally return the value of John's grandparent, which is immutably stored in the variable *a*, which, if this were a larger text, would continue to be glossed as v_1. Mathematically, this statement may be written as follows:

$$v_1 \in \{\ x \in \mathcal{H} : x = p \circ p \circ n(\text{John})\ \}$$

Where v_1 is the variable in this instance referred to by *a*, $\mathcal{H}$ is the HUMAN semantic class, p is the predicate to find the parent of a human, and n is the predicate to find a human given their name. This can be represented in a syntax tree as follows:

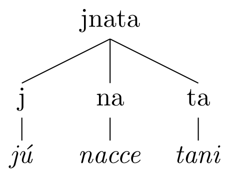


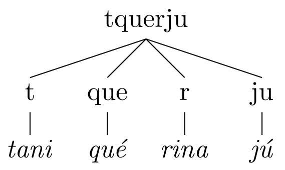
2.2 *la* ⋯ *se* ⋯ *ny* — function declaration

Functions in Besri Lamda are mathematically pure and can be declared using the structure *la* ⋯ *se* ⋯ *ny*. After the *la*, an arbitrary name of a function is given.[1] After this, the particle *se* signals a semantic class declaration. For functions, the first consonant of the semantic class name is used for each successive argument, ending in the return type, with the first vowel from each second class inserted as phonotactically necessary. Stress information is removed. For example, *jnata* would mark a function as taking two arguments, of semantic classes human and action, respectively, and return an animate non-human value. Another example could be *tquerju*, which would be a function

[1]By convention, these variable names begin with vowels and by grammatical necessity do not begin with *e*.

taking arguments of types animate non-human, concept, and inanimate, and returning a type human. These can be visualized below:





After this, the particle *ny* declares the body of the function itself, which is an evaluation of predefined functions and arguments in the language composed to create complexity. Arguments are referred to by common variable names. The first argument is always *e*. All vowels in argument names are *e*, and they all begin with this phoneme. In between these vowels, consonants are worked through two at a time, with *e* being put where it is phonotactically necessary. The progression of consonants follows a standard ordering, as follows: /p/ ⇒ /t/ ⇒ /d/ ⇒ /k/ ⇒ /g/ ⇒ /ʔ/ ⇒ /m/ ⇒ /n/ ⇒ /ɲ/ ⇒ /r/ ⇒ /f/ ⇒ /s/ ⇒ /ʃ/ ⇒ /h/ ⇒ /w/ ⇒ /l/. This is the same order that is given by following the IPA chart for Besri Lamda right-to-left, top-to-bottom. For example, in every function with five arguments, the five arguments are, in order: *e*, *epe*, *ete*, *ede*, and *eque*. Once the end is reached, another consonant is added, producing e.g. *epte* or *egmene*, depending on how many arguments the function takes.

2.3 *gte* — monadic `bind`

2.4 *nci* ··· *nci* — multiple monadic `bind`

2.5 *xe* ··· *xe* — `where`

2.6 *ne* — the `Statement` monad

2.7 *nja* — function composition

2.8 Function calling

2.9 Multi-argument functions

3 Semantics and Lexicon

There are five semantic classes in Besri Lamda: the Human class, only for humans, the Action class, for actions that can be carried out, the Animate non-human class, for animals, the Concept class, for abstract concepts, and the Inanimate class, for non-abstract, non-animate physical objects. Arguments can be broken into these classes, with cognate arguments in different semantic classes having different but often related meanings. Similarly, a single predicate can have several different but related meanings when taking differnent numbers of inputs and from different classes. These are defined in section 3.2, along with their class signatures. There is also a sixth semantic class: the Foreign class, for loan words and proper nouns.

3.1 Arguments

3.1.1 The human class: jú

je	"The generic argument; a human."
tweme	"woman."
zñawa	"man."

3.1.2 The action class: nacce

bary	"To speak."
cazte	"To compute."
jen	"The generic argument; to do."
mxeca	"To track the time."

3.1.3 The animate non-human class: tani

je	"The generic argument; an animal."
tweme	"female."
zñawa	"male."

3.1.4 The concept class: qué

bary	"Human speech."
cazte	"Mathematics and computation."
je	"The generic argument; a concept."
mxeca	"Time."

3.1.5 The inanimate class: rina

cazte	"Computer."
je	"The generic argument; a thing."
mxeca	"Clock."

3.2 Predicates

bary [Human $\rightarrow$ Concept] : Someone's speech.
besri [Foreign $\rightarrow$ Concept] : A language given its name.
cazte [Human $\rightarrow$ Action] : To understand someone.
r'e [Foreign $\rightarrow$ Human] : A person given their name.
xojque [Action $\rightarrow$ Union{Human, Animate}] : The performer of an action.
xojque [Animal $\rightarrow$ Animal] : The parent of an animal.
xojque [Concept $\rightarrow$ Concept] : The origin of a concept.
xojque [Human $\rightarrow$ Human] : The parent of a person.
xojque [Inanimate $\rightarrow$ Union{Human, Animate}] : The creator of an object.

4 Short texts