Heuristics for New Language

Peter Toshev
Sitovin Labs
peter.toshev@sitovin.com

Abstract

How do we create new language? Drawing on the observation that some languages—such as Pirahã—are almost entirely without a number system, this work explores what other features of human language could be fundamental yet remain unrealized. We propose the *super language* framework, a modular approach for iterating new forms (e.g. phonetic or morphological containers) and interpretations (e.g. semantic or pragmatic functions) on top of any base language. The resulting augmented variants are organized into a typology serving as a roadmap for experimentation. As a simple proof of concept, we present *StegaPhone*, a super language prototype that hides binary data within deliberate mispronunciations (e.g. "swim" vs. "svim"), thereby extending a phonetic layer into a parallel communication channel. To operationalize the framework, we outline a transformation engine—a rewrite-based system capable of covering the space of possible language modifications in a standardized, invertible manner. We argue that this "tip of the spear" architecture enables rapid language development and productive feedback loops across the domains in which it may be applied. Finally, we situate the significance of such a heuristic approach by considering the central role of language in communication, cognition, machine interfaces, and beyond, underscoring how new features have, and can continue, to impart transformative capabilities onto their operators.

Introduction

What is new language? And how do you make it? In this paper, we propose a framework for this line of inquiry. As a point of entry for these questions, it will be useful to foreground their context in a brief review of the lacking number system in the Pirahã language. The Pirahã people, an indigenous group in the Amazon rainforest, are reported to rely on very few and only approximate number words [1], [2]. Roughly simplified these are: *hói* ("one" or "few"), *hoí* ("two" or "some"), *baágiso* ("three or more" or "many"). A contemporary observer may find it bewildering that Pirahã speakers can manage without more precise language for numbers. So much so, they may even feel compelled to persuade the Pirahã to introduce better parametrized notions of quantity by adding linguistic objects to their language. However, in an alternate reaction, the observer could instead mirror the discovery back onto themselves and ask, what important features may my language be missing? Are there other, fundamental features whose absence will be similarly shocking to my descendants? Inspired by these questions (Figure 1), the work shared here is an effort in the heuristics of new language.

Super Language Framework

The super language framework is introduced in four progressively prescriptive and formal sections (each composed of a Description-Example pair) covering the model and the engine. They are followed by a short section on the typology and a review of the framework in select contexts.

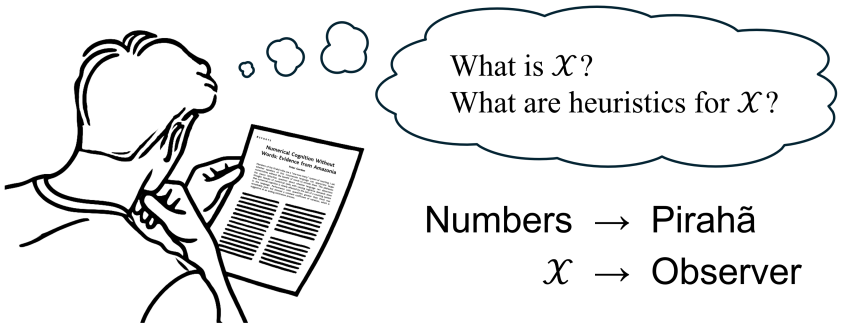


Figure 1: Illustration of a thought experiment. To parse the question "how do you make new language?", an observer analogizes (the lack of) numbers in Pirahã to possible new features in their own language.

Such a sequential unfolding is important to describe and make intuitive operation of the framework at its different levels.

Super Language Model

Description 1: Model gist

Super language is a model to promote the research, development, and adoption of new language. An instance of a super language, a *variant*, is a *base* language transformed by one or more *modifications*. Typically, a super language is designed to augment its base with new or improved features.
Casually, in expression (1) we can consider a transformation Z that maps a base language B with modifications M to its super language variant V.

$$Z : B \times M \to V \tag{1}$$

Each Z, B, M, and V drawn from the universe of possible transformations, bases, modifications, and variants, respectively, such that in shorthand we can express $\mathcal{Z} : \mathcal{B} \times \mathcal{M} \to \mathcal{V}$.

Example 1: StegaPhone prototype

StegaPhone (V) is a super language transformation (Z) from an oral English base (B) with steganographic phonetics modification (M). The choice of English is without loss of generality. In an example StegaPhone variant, a speaker can embed a hidden message in the mispronunciation of words. A speaker encodes a 0 or a 1 in any word they pronounce or mispronounce, respectively. The resulting sequence of binary values can be transformed to carry any message, for example another sequence of words.
In text, mispronunciations in StegaPhone will use phonetic misspellings. For example: "hello world" → "00" and "hello vorld" → "01" in this scheme.
In the interactive examples subsection are two demonstrations of StegaPhone to show the opportunity in this simple variant and the broader universe of super language transformations it represents.
This phonetics-based approach is a contribution in the research on linguistic steganography. It adds to the many other existing forms of text steganography explored, including steganography through translation, transliteration, VOIP or other acoustic carriers, and generative language models, as well as across both text edit- and generation-based methods [3].

Stated informally, one of the augmentations in StegaPhone is the increased dimensionality, over its oral English base, in its function as a communication channel (i.e. new bandwidth in phonetic component).

Interactive examples

There are two interactive examples at:
https://intro2024.superlang.org
Appendix A.1 provides elaboration on each.

StegaPhone playground The first is a playground interface where users can encode or decode their own StegaPhone text and modify StegaPhone parameters, such as payload alphabet or chunk size.

StegaPhonetic news The second is a demonstration of a toyish use case for this steganography technique, in which a journalist embeds fictitious stock tips by selectively mispronouncing words in a financial news broadcast.

Description 2: Model starter kit

Super language definitions

Super languages can be identified and further tiered according to a set of criteria.

Criteria A language is a *super language* if and only if it has all the following properties:

1. It has a *base* language.
2. It has one or more *modifications* over the base.
3. It has a *transformation*, defining the operation of modifications over the base.

Criteria (1), (2), and (3) together describe the trivial, tier-0 super languages.
A super language is a non-trivial, tier-1 super language if and only if it has the following additional property:

4. It contributes a new *form* or *interpretation* to the base.

It is a tier-2 super language if and only if it has the following additional property:

5. It contributes at least one *augmentation* to the base.

Base A base can be any language, or any broader communication or semiotic sign system. It can itself be a super language.

Modification A modification is a change to the base described in terms of either a form, interpretation, or both. It is a more general abstraction of a construction from construction grammar theory [4].

Form The form describes the symbolic containers of super language transformations. They will vary (Appendix B.1: Forms Tables) and include linguistic (e.g. phonetic, morphological) and extralinguistic structures (e.g. paralinguistic, kinesic). They also include containerized structures from interpretations (e.g. ontological objects).
Forms θ are drawn from the universe of possible forms Θ ($\theta \in \Theta$).

Interpretation The interpretation describes the functional roles of super language transformation. They will vary (Appendix B.2: Interpretations Tables) and range from roles described by grammar categories (e.g. tense-aspect-modality) to broader semantic and pragmatic sense-making functions. The functional roles describe the information or meaning that is contained in the forms of transformations. Interpretations ψ are drawn from the universe of possible interpretations Ψ ($\psi \in \Psi$).

Variant A super language variant is an instance of a super language for a particular choice of base, modifications, and transformation implementation.

Transformation A super language transformation, in the general case, is the specified operation of modifications over a base to create a variant. In the particular case, it is an instance of a super language transformation (e.g. a finite transformation from base string to variant string).

Augmentation Informally, an augmentation can be described as a new or improved feature of the super language or alternatively, as the purpose subserved by changes in form and interpretation introduced by the super language. They will vary (Appendix B.3: Augmentations Tables). For example, a "speed" augmentation could describe a modification which contributes an increase in the rate of information transmission per unit time.
Augmentations ϕ are drawn from the universe of possible augmentations Φ ($\phi \in \Phi$).

Super language transformation

The preceding elaboration on super language structure makes it possible to expand on expression (1) and provide a more formal outline.
First, in terms of the transformation form and interpretation *axes*.
Second, in terms of super language *representations* along those axes.
Third, in terms of an example formalization.

Axes The critical axes along which to consider the universe of possible transformations $\mathcal{Z}$ are the forms and interpretations described in the super language structure.

$$Z^{\theta} : B^{\theta} \times M^{\theta} \to V^{\theta} \tag{2}$$

$$Z^{\psi} : B^{\psi} \times M^{\psi} \to V^{\psi} \tag{3}$$

$$Z^{\theta,\psi} : B^{\theta,\psi} \times M^{\theta,\psi} \to V^{\theta,\psi} \tag{4}$$

The combination of the two in expression (4) is equivalent to (1), i.e. $Z^{\theta,\psi}$ is Z.
Example: separating a modification adding "tense" into its different possibles forms, such as a morphological structure, and its different possibles interpretations, such as new temporal encoding to represent future events.

Representations Language L, whether a base B or variant V, may be represented in a variety of ways. Therefore we describe the representation-level transformation in terms of the choice of representation domains.

$$\hat{Z} : \hat{B} \times \hat{M} \to \hat{V} \tag{5}$$

The components in $\hat{Z}$ are equivalently each drawn from the universe of possible representations, such that in shorthand we can express $\hat{\mathcal{Z}} : \hat{\mathcal{B}} \times \hat{\mathcal{M}} \to \hat{\mathcal{V}}$.
Example: for language L with a generative grammar G, super language transformation Z can have representation transformation $\hat{Z}_G$.

Formalization example An example formalization considers the form of super language transformation (Z^θ) in terms of a generative grammar representation ($\hat{Z}_G^\theta$) that can be described by:

$$\hat{Z}_G^\theta : G_B \times (\Delta N, \Delta\Sigma, \Delta P, \delta S) \rightarrow G_V \tag{6}$$

In this domain, a generative grammar $G = (N, \Sigma, P, S)$.
Appendix C.3a provides an elaboration on the transformation described in equation (6).

Super language classification

Evaluations for criteria (4) and (5) will vary, even for the same super language variant.
Determining whether a super language introduces a new form, interpretation or augmentation requires a system of reference, which must be included as a part of the determination. Such a determination is possible as a *classification* executed by a *classifier*. A sketch to this determination approach is in Super Language Typology.
Classifications according to form, interpretation, and augmentation are key because they can provide quick answers to the questions "what has changed", "how is the change used", and "what purpose does the change serve" when describing a super language. The Transformation Variation tables (Appendix B) provide categories, with some interspersed examples, along these three critical axes of variation. They are a start to the parametrization of the space of possible super language transformations that can serve as a potent roadmap for variant experimentation.
Beyond these categories, there are other valuable classification varieties, such as the categorization of downstream super language uses into *applications*. Informally, applications can be described as the systems in which variants can manifest their augmentations over a base.

Important properties

Designed vs. descriptive Super language can be designed (e.g. StegaPhone), when they are a prescriptive creation on top of a base. They can be descriptive, when an already existing language is cast as the super language transformation of another. For example, a verbal language can be cast as a descriptive super language of oral language which was modified to include writing.

One for all modifications A modification can be one or more modifications. By default, a combination is assumed to be nested, where each modification is applied independently and serially.

Tip of the spear transformation Critically, a base does not need to be described beyond the narrow scope relevant to the super language transformation. The transformations applying modifications over a base do not need to produce the underlying structures of the base (e.g. its generative grammar). They must only define how a subset of base structures changes. In concert with the other model properties, the focus on the modifications over a base enables a tip of the spear approach to language design to unencumber both experimentation and adoption of new features. Traditional language creation and language learning often suffer from "cold start" problems, which the modular incrementality in the super language approach is crafted to address.

Example 2: StegaPhone in the model

The example StegaPhone variant can be more precisely defined (Table 1) as a super language in the framework model. We provide an example formalization in terms of the variant's form in the generative grammar representation domain.

StegaPhone transformation components

Table 1: Example super language model components of a StegaPhone variant.

Component	**Value**
Super language	StegaPhone
Base	EN-US (oral)
Modification	Steganographic Phonetics
Mod Form	Pronunciations of words
Mod Interpretation	{Correct pronunciations = 0, Mispronunciations = 1}
Variant	StegaPhone-201
Transformation	StegaPhone over EN-US (oral) encodes word-wise a <Payload string> where each symbol is encoded as the binary value of its index in the <Payload alphabet>, set to {←, ↑, Ⓐ, Ⓑ}.
Example Transformation	Base string: "hello world" Variant string: "hello vorld" Payload string: "↑" ("01" binary index)

StegaPhone formalization example

Consider the StegaPhone transformation Z_{sp}, defining the map from its oral English base B_{en} and steganographic phonetics modification M_{sp} to the variant V_{sp}:

$$Z_{sp} : B_{en} \times M_{sp} \to V_{sp} \tag{7}$$

For a generative grammar G_{en} of the oral English base B_{en}, the associated StegaPhone transformation ($\hat{Z}^{\theta}_{sp}$) in this domain can be described by:

$$\hat{Z}^{\theta}_{sp} : G_{en} \times (\Delta N, \Delta\Sigma, \Delta P, \delta S) \to G_{sp} \tag{8}$$

While G_{en} is unwieldy, the resulting $G_{sp} = (N_{sp}, \Sigma_{sp}, P_{sp}, S_{sp})$ can be made very simple using the super language framework.

StegaPhone generative grammar The components of G_{sp} will be:

- Non-terminals $N_{sp} = \{\text{Sentence}, \text{WordSequence}, \text{Word}, S\}$ where WordSequence is Sentence with one or more Word removed.
- Terminals $\Sigma_{sp} = \Sigma \cup \Sigma'$ where the sets Σ and Σ' are those of the correctly, and incorrectly, pronounced English words, respectively.
- Production Rules $P_{sp} =$
  1. $S \to$ Sentence
  2. Sentence $\to$ WordSequence
  3. WordSequence $\to$ Word
  4. WordSequence $\to$ Word WordSequence
  5. Word $\to w$, $\forall w \in \Sigma$ (correctly pronounced words)
  6. Word $\to w'$, $\forall w' \in \Sigma'$ (mispronounced words)

- Start symbol S

In the interpretation $\mathcal{I}$ of the language V_{sp}, the output of interpretation function I will vary across Σ and Σ', where $I(w) = 0$ and $I(w') = 1$. The values 0 and 1 are binary elements in {0,1} that constitute the payload.
Appendix C.3b provides derivation and additional details.

StegaPhone classification

On a first pass, it can be more suitable to consider variant classifications informally, guided by categories outlined in Appendix B, before doing so with strict and explicit methods.

Form Pronunciations are a phonetic element of speech. Phonetics are an example of the broader set of forms in the linguistic category of containers (Appendix B.1).

Interpretation The word-wise encoding of binary bits in a payload creates an additional stream of communication that can be identified as a "parallelization". This parallelization is an example in the broader "Information Encoding & Channels" category of functional roles (Table 8 in Appendix B.2).

Augmentation Intuitively, use of StegaPhone to modify the sending and receiving of messages should increase the security or the bandwidth of the communication channel in which they are transmitted through. In the framework, we can consider the potential of the "parallelization" identified in StegaPhone to see if it may qualify as an augmentation (Appendix B.3).
For example, we can consider the "surprisal" augmentation target. A surprisal augmentation is an increase in information per unit word (e.g. a higher load of meaning per word following a lexicon expansion). Such an increase is a consequence of the StegaPhone transformation and therefore makes "surprisal" an augmentation of the variant.

Application The typical application for steganography, including in this phonetic form, is information security. A less obvious application for the StegaPhone super language would be to investigate its potential for intelligence development. For example, carrying out a study of StegaPhone for its effects on activities subserved by the prefrontal cortex (PFC), such as working memory or attention control. Such a study could investigate if speakers of StegaPhone see improved performance in listening span tasks, which has been associated with improved learning and comprehension [5]. It could also explore new variations on listening span tasks, requiring individuals to encode and decode payloads in a series of utterances. It is through such trials that StegaPhone, or other super language variants, can be identified as good candidates for broader adoption.

Exploration & exploitation in the framework

Iteration with new ideas on revised objectives is natural in the super language framework. For example, the implementation of StegaPhone described has limitations in the context of human adoption.
The cognitive load from payload encoding and decoding, where a StegaPhone speaker must track words of an utterance as a binary data stream, is very high. It can be greatly reduced by limiting the size of the payload alphabet or by introducing significant chunking, but such changes may be unlikely in making unassisted StegaPhone speech practical.
This limitation could be addressed by considering more operable uses to the StegaPhone modification, or more precisely, by keeping its phonetic form (mispronunciations) but changing its interpretation

(word-wise encoding of bits). For instance, rather than having the payload be a separate message with symbols encoded as their indices, the interpretation could be a small set of actions that change semantic components of the cover. For a trivial example, a mispronunciation could be the logical negation of clause: "I will go" → "I will go" so that "I vill go" → "I will not go", which is a more human-operable variation and likely a better start towards an application in intelligence development.
The rapid iterations to calibrated but flexible end goals in transformation design is key to developing valuable super language prototypes.

Super Language Engine

Description 3: Engine primer

Transformation engine motivation

The super language model, comprised of criteria, components, and classifications, provides valuable scaffolding and a roadmap for new language. However, it does not lay out a system for the production and operation of super language. Such a system should establish a set of methods to parse, generate, and operate super language across the variety of forms and interpretations (including their derivatives and abstractions) that their transformations could take on. The system must cover the universe of possible transformations but it must also standardize the making of super language so that it is agnostic to the specification details of any particular variant. The methods should be simple for more reproducible and scalable development of super languages. In the *transformation engine* described below, we present such a system to operationalize the super language framework.

Transformation engine brief

Resetting the informal notation, a transformation from language (L) to super language (Z) can be executed by the transformation engine outlined in Equations (9), (10) and Figure 2.

$$Z[M]\colon B \times U \to V \times W \tag{9}$$

$$Z^{-1}[M]\colon V \times W \to B \times U \tag{10}$$

The transformation engine proceduralizes super language design first by standardizing the definition of a variant in a *transformation specification* that defines the transformation mapping. This specification is the recipe necessary to implement and execute transformations from base to variant, and vice versa. The transformation engine maps base strings to variant strings per the modifications defined in the transformation specification. Furthermore, it maps (Z) combinations of base strings ($b \in B$) and modification commands ($u \in U$) to variant strings ($v \in V$) and modification artifacts ($w \in W$). It acts on symbolic expressions of both the structures and the operations in super language.
In general, the bases and variants are the structures of super language transformations and the commands and artifacts are operations over those structures. Accordingly, bases and variants will primarily correspond to the forms of the transformation whereas the commands and artifacts will primarily correspond to the interpretations. Of course, all components are symbolically expressed, so this distinction highlights the respective tasks of the four components in the engine. The significant variation in transformations from bases/commands to variants/artifacts will often be underpinned by the motivating augmentations or other relevant super language classifications.

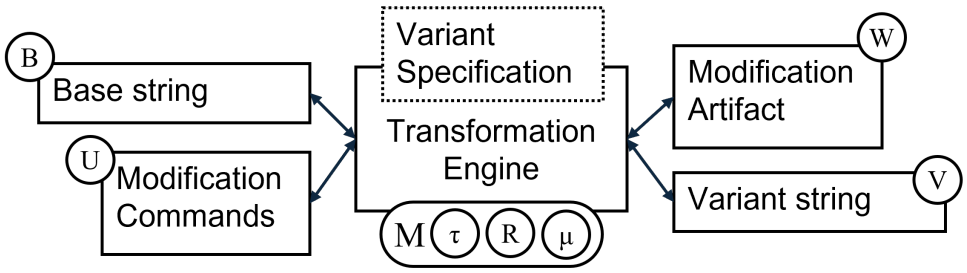


Figure 2: Transformation engine components. The forward transformation is left-to-right and the inverse is right-to-left.

Important properties

Symbols of structures & operations There are a few important choices in the transformation engine to make it fit for the creation and use of super language. Key to standardizing the design of super languages is the fundamental unit in which they manifest: the symbol. Symbols, and their sequences in strings, can represent both the structures of super language and operations over those structures. This is critical because it will not be sufficient for a system to parse and generate the forms of super language strings. It must also direct and enable control of the interpretations of super language strings.

Packets of forms & interpretations Key to the control of super language transformations, whether in terms of their structures or operations, is the packetization of forms and interpretations. The transformation engine enables fine-grained control of super language in its discretized base-commands, variant-artifact pairs which makes for flexible and extensible super language design. This modular architecture of the transformations enables rapid evolution in their specifications, varying the expression and control of their forms and interpretations (including their abstractions, derivatives, other variations in their representations).

Example 3: StegaPhone engine sketch

When considering how the StegaPhone variant may be implemented as an engine transformation, there are a variety of possibilities for the commands and artifact components of the transformation. For the same base string b and variant string v, we can consider a variety of examples in commands U and artifacts W.

Fixed base & variant Example $b, v \in B, V$.
$b =$ "We will swim today."
$v =$ "We vill svim today."

Example commands Example modification commands $u \in U$.
$u_1 = \{$Payload string = "0110"$\}$
$u_2 = \{$Payload alphabet = $\{\leftarrow, \uparrow, \text{Ⓐ}, \text{Ⓑ}\}$; Payload string = "↑Ⓐ"$\}$
Here the payload string of u_2, which also has binary index sequence "0110", creates the same variant as u_1 but its meaning is different because of the change in alphabet. These couple varieties of U show that modification commands can enable different interpretations to the same form output.

Example artifacts Example modification artifacts w in W.
w_1 = <Payload string>
w_2 = <Payload alphabet>
w_3 = none
Here the artifact output chosen by the sender will create different requirements for the receiver of the super language transformation. In w_1, the payload is transmitted directly, whereas in w_2, the receiver must decode the payload themselves using the alphabet provided. In w_3, significant burden is placed on the receiver. Without additional context, they must intuit the sender's expectation for the steganographic package they have received.

This simple sketch emphasizes further the modularity and flexibility of the transformation engine. It can be used as a reference template to understand and consider variations in bases/variants, commands/artifacts, and the forms/interpretations that can describe them.

Description 4: Engine formal outlines

Transformation engine definition

The transformations in Equations (11) and (12) to and from a super language variant (V) and its associated modification artifact (W) are those prescribed by the application of its one or more modifications (M), including its tokenization (τ), rewrite rules (R), and modification function (μ) with possible inverse (μ^{-1}), over its base (B) and modification commands (U). For clarity, the explanations here describe a transformation with modifications subsumed in one modification (M).

$$Z[M(\tau, R, \mu)] \colon B \times U \to V \times (W_{int} \times W_{key}) \tag{11}$$

$$Z^{-1}[M(\tau, R, \mu^{-1})] \colon V \times (W_{int} \times W_{key}) \to B \times U \tag{12}$$

A variant is defined by the set of possible super language transformations (Z) over its base. Components of this transformation are described further in Appendix C.1: Transformation Specification Components. Note we overload Z across the super language definitions (Descriptions 1 and 2) and the engine definitions (Descriptions 3 and 4). When written with brackets to explicitly disambiguate, $Z[\cdot]$ denotes an engine instantiated by a specification.

Engine algorithm gist We provide a gist of the transformation engine algorithm and a more thorough step-by-step overview is in Appendix C.2: Transformation Engine Algorithm.
For a choice of variant with specification (Z), the engine tokenizes (τ) the base ($b \in B$) input, finding targets for the rewrite rule (R) substitutions. It sequentially iterates through the tokens and rewrite rules, and on each match calls the modification function (μ) which carries the modification commands input ($u \in U$). It either applies or does not apply the substitution and updates the modification artifact ($w \in W$). On termination, it outputs the variant ($v \in V$) and modification artifact, comprised of the interpretation (w_{int}) and inverse key (w_{key}).
The system preserves the necessary information (at worst the complete sequence of applied rewrite rules) to perform the inverse transformation, recovering the source base string and modification commands for a given variant string and modification artifact.

Engine ingredients and recipe Table 2 outlines the "recipe" for the transformation engine's making of super language given the provided "ingredients" in the components of the variant.

Table 2: Engine transformation specification components.

Component	Variables	Description
Base string	$b \in B$	Valid strings in the base (B).
Variant string	$v \in V$	Valid strings in the variant (V).
Tokenization	τ	Parsing of string input into tokens and non-token substrings.
Rewrite Rules	$r \in R$	Ordered set of pattern-replacement pairs (α, β) for token substitutions $(\alpha \to \beta)$ over a string.
Mod Function	μ	Implements modification $M(\tau, R, \mu)$ of transformation (Z). Inputs: base (b), commands (u) Outputs: variant (v), artifact (w)
Mod Commands	$u \in U$	User inputs to modification function (μ) directing transformation of base to a target variant and/or target mod artifact.
Mod Artifact	$w \in W$	Output of the terminating mod function call (μ_N), comprised of interpretation and inverse key subcomponents.
Interpretation	$w_{int} \in W_{int}$	Artifact subcomponent for functional role served by transformation.
Inverse Key	$w_{key} \in W_{key}$	Artifact subcomponent with key material for inverse mod function (μ^{-1}).
Inverse Mod Function	μ^{-1}	Implements inverse modification $M^{-1}(\tau, R, \mu^{-1})$ of inverse (Z^{-1}). Inputs: variant (v), artifact (w) Outputs: base (b), commands (u)

Important properties

String rewriting recipes Key to the generation and control of super language symbols and their sequences is the ordered set of rewrite rules. Such an approach standardizes super languages into recipes for the universe of possible transformation. It is also a natural paring with the super language tip of the spear property because when considering transformations, the intuition will often be "how should string x become string y?"

Completeness As a rewrite system, any transformation can generate any recursively enumerable language (similarly can create any language generated unrestricted grammar). Such a transformation is equivalent to a Turing machine [6]. Of course, the resulting variant is not required to be Turing complete.

Recursive rewriting Often it is convenient to recurse transformations and their inverses. In recursion, the tokenizing step simply parses the base string into substrings, each of which can become the initial base string of a recursed transformation, which may itself be further nested. This can be useful when modification substitutions (i.e. patterns → replacements) are functions on the token targets (e.g. regular expression operations).

Vanilla super language vs. transformation engine In Appendix C.3, an example, formal description of super language (C.3a) and StegaPhone (C.3b) is provided which do not include the transformation engine. The example description uses a traditional, generative grammar approach to language. While such an approach can leverage some of the important properties in the model, it has several limitations that are addressed by the transformation engine. For example, it does make use of the tip of the spear property by creating an intermediary super language to avoid having to define the generative grammar for the original base, oral English. However, even with the addition of the production rules tasked with converting correct pronunciations and mispronunciations to 0's and 1's, the

resulting super language is not an operable system as much as it is just a formal parsing method. It is missing the packetized form and interpretations included in the transformation engine.

Variety in specification components There is a great diversity in the possible bases/variants and the modifications/transformations that relate them, such as variations across their form, interpretation, augmentation, or other classifications. The commands and artifacts introduced in the engine can also differ across these variations. They are often more complex expressions compared to bases and variants because they carry more of the interpretative, rather than structural, load in a transformation. As types, they can range from traditional strings, to functions, to more abstract computing objects and other representations.

Variety in transformation implementations It is not strictly necessary to implement the engine's transformations in terms of an ordered set of rewrite rules. For the consistency of the framework, the system should behave as a rewriting system (roughly deterministic up to non-deterministic factors in the modification function). However, for some variants (e.g. complex, context-dependent tokenizations and substitutions) it may be much more convenient to first implement the transformation using different methods. For instance, there may be cases that are better suited to statistical/ML-based methods, such as an autoregressive language model fine-tuned to perform substring- and token-wise transformations. It is then possible to add verification guardrails for these alternate implementations to preserve the intended rewrite system logic. Moreover, the alternate implementations can themselves be used to build out the explicit rewrite system implementation of the transformation.

Self-correction Reductively, super languages can have errors. Errors such as a disagreement between the intention of a variant (captured via its informal description) and a string output by an implementation of the variant. This can happen when developing complex transformations or when exploring more black-box implementation alternatives, such as in neural network models.
The self-correcting mechanism of super language is not unlike that in the accretive mixing of linguistic alterations in the evolution of natural languages. In the iteration of natural language, there is communal experimentation that changes forms and interpretations. Adoption of the modifications is a social process in which novel properties are shared and refined. There is a collective error correction and sometimes so-called errors may actually be explorations of future forms and interpretations.
Super language creation and use should share this self-correcting process because iteration in the tip of the spear approach to variant transformation can mirror the incremental patchworking in natural language evolution. Removing an excessive focus on error-free formalizations at the outset is important to avoid derailing efforts towards new language prototypes.

Example 4: StegaPhone in the engine

StegaPhone example engine specifications

Example implementation of StegaPhone in the transformation engine in Table 3

StegaPhone example engine execution

Example execution of StegaPhone in the transformation engine in Figure 3.

Table 3: StegaPhone transformation specification for engine.

Component	Variables	Example
Base string	$b \in B$	English (EN-US)
Variant string	$v \in V$	StegaPhone (StegaPhone-201)
Tokenization	τ	words (e.g. "swim")
Rewrite Rules	$r \in R$	word → mispronounced word (e.g. "swim" → "svim").
Mod Function	μ	Encodes {0, 1} as binary bits in {word, mispronounced word} of Base "cover" string for each index of Mod Commands "payload" characters.
Mod Commands	$u \in U$	1. Set <Payload alphabet>. 2. Set <Chunk size>. 3. Set <Payload string>.
Mod Artifact	$w \in W$	$w = (w_{int}, w_{key})$
Interpretation		w_{int} = <Payload string>
Inverse Key		w_{key} = {<Payload alphabet>, <Chunk size>}
Inverse Mod Function	μ^{-1}	Decodes and returns the <Payload string> token-wise from Variant string given {<Payload alphabet>, <Chunk size>} of Inverse Key.

Super Language Typology

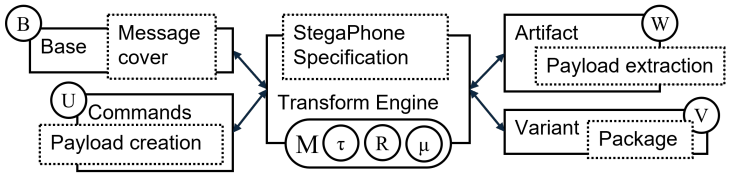
The typology of super language can be considered in terms of the variety in transformation from base and variant. We can initiate a typology first from the Description 2 outline of super language model components and their key axes of variation including forms (θ), interpretations (ψ), and augmentations (ϕ). Typology along those axes is elaborated on in Appendix B: Transformation Variation and can be further expanded in scope. Future work will develop this expansion in terms of super language representation domains or applications and in terms of the engine modification commands and artifacts (Description 4 and Appendix C.1). In order to explicitly attribute types or classes, the typology can be made more precise via *transformation classification*.

Transformation classification brief In super language, a *classifier* implements the *classification* of an *argument* for a particular *target*. The domain of the classifier is designed around its argument (e.g. a variant) and the codomain is designed around its target (e.g. speed). The arguments include are modifications, variants, or super language transformation instances. These are the objects classified into *classes* or their *categories*. The targets included are drawn from forms, interpretations, augmentations, applications, among other categories. Classifiers of super languages can be formalized as a function (often a set function or a measure) mapping its argument to a target.
In Equation (13), a classifier ($\hat{C}$) implements the classification (C) of an argument (C_{arg}) with respect to a target (C_{tar}). Occasionally the classifier will require additional input (C_{in}) to execute the classification.

$$C \colon \hat{C}(C_{arg}(C_{in})) \to C_{tar} \tag{13}$$

The classifiers, and in particular the targets of the classification it defines, will vary. In some cases, the classifier may be a simple test for the existence of a particular form (e.g. does the modification introduce a new phoneme) or interpretation (e.g. does it introduce parallelization). In other cases, especially for augmentation or application targets, the classification may be in more continuous units (e.g. change in "surprisal" augmentation from base to variant).

StegaPhone Example Transformation (Z) and Inverse (Z^{-1})				
Base string (b)	We	will	swim	today.
Tokenization (τ)	token 1	token 2	token 3	token 4
Modification Commands (u)	u[1] Payload alphabet		Σ = {←, ↑, Ⓐ, Ⓑ}	
	u[2] Chunk size		1 bit per 1 word	
	u[3] Payload string		↑Ⓐ	
Payload enc/dec‡	0	1	1	0
Variant string (v)	We	vill	svim	today.
Modification Artifact (w)	Interpretation (w_{int})	<Payload string>		
	Inverse Key (w_{key})	<Payload alphabet> <Chunk size>		



Modification Command Set (U[3])						
Payload strings (all u[3])		Variant strings (all v in V(b), given u[1], u[2])				
Indices	Chars	tok1	tok2	tok3	tok4	v's
[00,00]	←←	We	will	swim	today.	01
[00,01]	←↑	We	will	swim	tozay.	02
[00,10]	←Ⓐ	We	will	svim	today.	03
[00,11]	←Ⓑ	We	will	svim	tozay.	04
[01,00]	↑←	We	vill	swim	today.	05
[01,01]	↑↑	We	vill	swim	tozay.	06
[01,10]	↑Ⓐ	We	vill	svim	today.	07†
[01,11]	↑Ⓑ	We	vill	svim	tozay.	08
[10,00]	Ⓐ←	Ve	will	swim	today.	09
[10,01]	Ⓐ↑	Ve	will	swim	tozay.	10
[10,10]	ⒶⒶ	Ve	will	svim	today.	11
[10,11]	ⒶⒷ	Ve	will	svim	tozay.	12
[11,00]	Ⓑ←	Ve	vill	swim	today.	13
[11,01]	Ⓑ↑	Ve	vill	swim	tozay.	14
[11,10]	ⒷⒶ	Ve	vill	svim	today.	15
[11,11]	ⒷⒷ	Ve	vill	svim	tozay.	16

Figure 3: Example execution of StegaPhone in the transformation engine. Top-to-bottom direction is the transformation (Z) and bottom-to-top is the inverse (Z^{-1}). Variant (v) at † is the output or input string in the transformation or its inverse, respectively. Encoding or decoding at ‡ is executed by the modification function (μ), or its inverse (μ^{-1}).

Transformation classification network Together the classifiers, the arguments, and the targets form a *transformation classification network* (Figure 4). Such a network is valuable as a tractable way to chart and explore of the universe of possible transformations, especially since different classifiers may return different outputs for the same arguments.

Super Language Review

Contexts to the framework

We consider a selection of important contexts in which the super language framework can be reviewed, and for which advantages or opportunities in super language can be considered.

Creating new language

A key lens through which to review super languages are other efforts toward new language. Two typical, but unwieldy, categories of these efforts are natural language and constructed language [7].
Super language can be described as either natural or constructed. It can be partially natural depending on the choice of base, or entirely natural (e.g. the descriptive verbal super language). In fact, the super language framework is based on key qualities of natural language evolution (e.g. iterative alterations).
Like constructed language (or *conlangs*), super languages can include deliberate design. There are some efforts in the conscious engineering of language, like those of controlled natural language (CNL) [8], that also focus specifically on changes to a base language.
There are also some computational linguistic tools (e.g. Prolog [9] and FCG Editor [10]) that can support language design and experimentation.

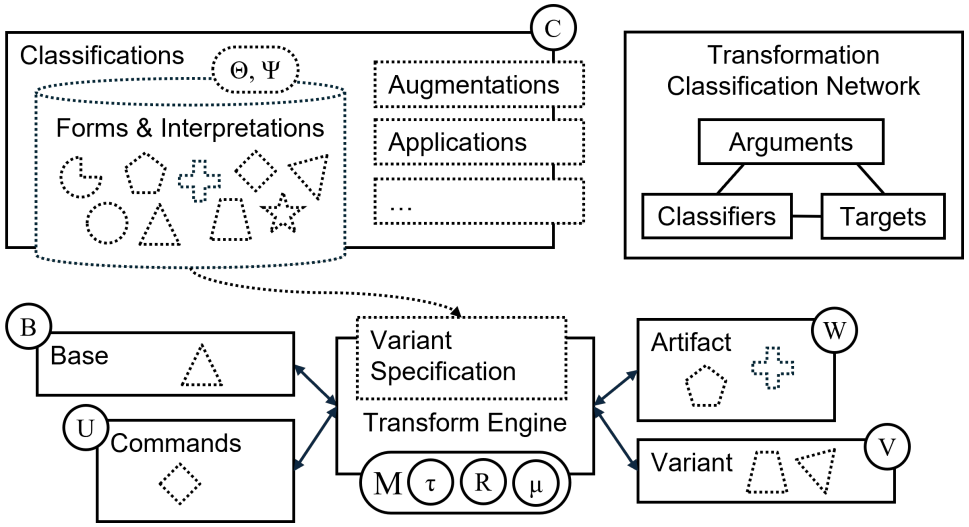


Figure 4: Modular design and cataloging of super language via engine transformation specifications and classifications.

With an elaboration beyond the scope of this review, these alternatives have different limitations. They do not have a modular and extensible transformation architecture for the rapid, tip of the spear augmentation to language. The extremely limited adoption of conlangs conveys the need for new paradigms in language modification.

Specifying language

Language type

Another key lens through which to review super languages is other attempts at the explicit and implicit typologizing of language. There are formal efforts in typological linguistics, such as in the context of construction grammar theories [11]. There are more applied efforts such as in the parametrization of language features to create and solve Linguistics Olympiad problem sets [12], [13].
Typological studies have further developed with the emergence of machine representations of language (including language-mediated content) and their intersection with various human ontologies. An important development in this area is that of "evaluations", testing linguistic reasoning in language models [14], [15].
These different efforts can supply components in super language (e.g. possible forms/interpretations) but they have limitations. They are often backward looking, focusing on the historical cataloging and recycling of existing features of language rather than serving as a roadmap towards new language. They can be impractical, focusing on hyper-detailed specifications rather than an intuitive menu of new features.

Language change

Similar to typological research describing language in terms of their forms and functions, other relevant efforts seek to describe language in terms of their change over time. There are a variety of approaches including historical, comparative, evolutionary, diachronic and other change-based investigations of language.

For example, human language has been a key enabler for culture, intergenerational information transfer, and the socialization of our species at scale [16]. Dissecting these impacts in various units of change can support the development of super language, helping characterize the salience that it seeks out in the search of new transformations.

Applying language

The applying of language here refers to its impact as both the subject and object of its use in different domains.

Neural bases

The neural basis, and inversely the neural consequences, of language are important vectors to consider for the development of super language.
There is a two-way interaction between the instantiation of language and the neural coding schemes that produce it. This is true in horizontal sense, in which language, as instrument of synaptic plasticity, supports a variety of cognitive functions. For instance, learning can be strongly mediated by reading. It is true in a more vertical sense, in which language is acting more directly on the structures it is subserved by. For instance, research of the PFC has been shown to be key for the capacity for language for both (i) an individual, through measurements during their development into early adulthood [17], and (ii) the species as a whole, through the hyperscaling of the PFC in humans compared to the relative stagnation in the PFC of its closest relatives [18].
Moreover, evolutionary evidence has suggested sudden phase transitions in a coevolutionary trajectory of language and the neural systems that underlie it [19]. Study of the acceleration triggers in this history can inform iteration in super language design.

Universe of applications

Beyond the duality in language and its neural bases, there are many other intersecting domains in which to consider language as both subject and object. They can be viewed hierarchically, as upstream (e.g. cognitive) or downstream (e.g. social influence) from where language is viewed as being expressed (e.g. speech). They can be viewed more functionally, as inner dialog or tools for thought of the individual, as communicative functions between groups, as distributed ecological/environmental states, as neural patterns in machine learning models, etc.
Super language design can be informed in terms of its subject-object interplay across the different domains of application.

Language of technologies

In the universe of applications, one neighborhood which has been significantly explored is the practice of language in technology. This area has seen the explosion of perhaps nominally constructed languages: Boolean algebra for digital circuits, Morse code for telegraphy, programming languages for computers, domain specific languages (e.g. for mathematics, such as Lean or Mathematica), etc. More recently, neural network technologies have greatly improved machine capabilities in human language. Such a machine, based on large language models, can now use the vast collection of natural language (primarily in human expressions stored on the internet) to create representations of meaning and build out a great, artificial intelligence.
Whether these machines exhibit intelligent, or only intelligent-adjacent, features, they are by design significantly mediated by expressions in natural language. They are also by design operated by humans

via natural language. Future work will expand on the value of new language in and beyond its role as interface between human and technology. But in this context it stands to reason that it will be fruitful to experiment and iterate on what has served as both the generative source and the interface for this machine intelligence. Super language can provide framework for such iterative exploration.

Discussion & Future Work

We have introduced a new framework, super language, and presented an example variant prototype, StegaPhone, to evolve our methods for language creation and adoption. The framework, including the underlying model of language, transformation engine approach, and typological map, have been outlined to encourage attempts at new language. The modular architecture proposed, with rapid iteration across forms and interpretations, leverages the different strengths in natural language evolution and deliberative constructed language design to establish a system of a language augmentation. In subsequent work, we will elaborate on the framework, its context, and share additional prototypes. This proof-of-concept, in terms of the framework and example prototype, is inspired by the pluripotent mechanisms of language and aspires towards new heuristics for their future.

References

[1] Peter Gordon. Numerical Cognition Without Words: Evidence from Amazonia. *Science*, 306(5695):496–499, October 2004.

[2] Michael C. Frank, Daniel L. Everett, Evelina Fedorenko, and Edward Gibson. Number as a cognitive technology: Evidence from Pirahã language and cognition. *Cognition*, 108(3):819–824, September 2008.

[3] Zachary Ziegler, Yuntian Deng, and Alexander Rush. Neural Linguistic Steganography. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Hong Kong, China, 2019. Association for Computational Linguistics.

[4] Adele E Goldberg. Constructions: A new theoretical approach to language. *Trends in Cognitive Sciences*, 7(5):219–224, May 2003.

[5] Margarete Imhof. Listening Span Tests. In Debra L. Worthington and Graham D. Bodie, editors, *The Sourcebook of Listening Research*, pages 394–401. Wiley, 1 edition, September 2017.

[6] Emil L. Post. Formal Reductions of the General Combinatorial Decision Problem. *Journal of Symbolic Logic*, 8(1):50–52, 1943.

[7] Christine Schreyer. Constructed Languages. *Annual Review of Anthropology*, 50(1):327–344, October 2021.

[8] Tobias Kuhn. A Survey and Classification of Controlled Natural Languages. *Computational Linguistics*, 40(1):121–170, March 2014.

[9] Jan Wielemaker, Tom Schrijvers, Markus Triska, and Torbjörn Lager. SWI-Prolog. *Theory and Practice of Logic Programming*, 12(1-2):67–96, 2012.

[10] Remi Van Trijp, Katrien Beuls, and Paul Van Eecke. The FCG Editor: An innovative environment for engineering computational construction grammars. *PLOS ONE*, 17(6):e0269708, June 2022.

[11] William Croft. *Ten Lectures on Construction Grammar and Typology*. BRILL, September 2020.

[12] Bozhidar Bozhanov and Ivan Derzhanski. Rosetta Stone Linguistic Problems. In Ivan Derzhanski and Dragomir Radev, editors, *Proceedings of the Fourth Workshop on Teaching NLP and CL*, pages 1–8, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.

[13] Vlad A. Neacșu. *Linguistics Olympiad. Training Guide*. Number 13 in Textbooks in Language Sciences. Language Science Press, Berlin, 2024.

[14] Nathan Chi, Teodor Malchev, Riley Kong, Ryan Chi, Lucas Huang, Ethan Chi, R. McCoy, and Dragomir Radev. ModeLing: A Novel Dataset for Testing Linguistic Reasoning in Language Models. In Michael Hahn, Alexey Sorokin, Ritesh Kumar, Andreas Shcherbakov, Yulia Otmakhova, Jinrui Yang, Oleg Serikov, Priya Rani, Edoardo M. Ponti, Saliha Muradoğlu, Rena Gao, Ryan Cotterell, and Ekaterina Vylomova, editors, *Proceedings of the 6th Workshop on Research in Computational Linguistic Typology and Multilingual NLP*, pages 113–119, St. Julian's, Malta, March 2024. Association for Computational Linguistics.

[15] Andrew M. Bean, Simi Hellsten, Harry Mayne, Jabez Magomere, Ethan A. Chi, Ryan Chi, Scott A. Hale, and Hannah Rose Kirk. LINGOLY: A Benchmark of Olympiad-Level Linguistic Reasoning Puzzles in Low-Resource and Extinct Languages, October 2024.

[16] Mark Pagel. Q&A: What is human language, when did it evolve and why should we care? *BMC Biology*, 15(1):64, December 2017.

[17] Kate Teffer and Katerina Semendeferi. Chapter 9 - Human prefrontal cortex: Evolution, development, and pathology. In Michel A. Hofman and Dean Falk, editors, *Progress in Brain Research*, volume 195 of *Evolution of the Primate Brain*, pages 191–218. Elsevier, January 2012.

[18] J.B. Smaers, J. Steele, C.R. Case, A. Cowper, K. Amunts, and K. Zilles. Primate Prefrontal Cortex Evolution: Human Brains Are the Extreme of a Lateralized Ape Trend. *Brain Behavior and Evolution*, 77(2):67–78, April 2011.

[19] Rafael Vieira Bretas, Yumiko Yamazaki, and Atsushi Iriki. Phase transitions of brain evolution that produced human language and beyond. *Neuroscience Research*, 161:1–7, December 2020.

Appendix

Appendix A: StegaPhone Demonstration

Additional information on the StegaPhone interactive examples described in Example 1.

Appendix A.1: StegaPhone interactive examples details

There are two interactive examples at:
https://intro2024.superlang.org

StegaPhone playground

In the StegaPhonetic playground, the vast possibilities in the choice of payload alphabet hints at the different uses to this super language prototype.
A natural first choice for an alphabet is an ordered list of letters {a, b, c, ...}. This is roughly how the characters are organized in the American Standard Code for Information Interchange (ASCII) table. Using the ASCII table alphabet, the word "hi" would be represented as binary string "0110100001101001", requiring at least 16 words (and a specific 7 of them mispronounced) to embed the payload.
But the payload does not itself need to be a string of English words like the cover that carries it. The payload itself could be much simpler and yet operate in a wholly different system of communication. For example, consider a "controller alphabet" $\{\text{Input}_1, \text{Input}_2, ..., \text{Input}_n\}$ where each character instead represents a command to an input.
Without loss of generality we can consider an example controller alphabet like {←, ↑, →, Start-Stop, Yes, No, Error}. The arrows specify a directional "movement" command whereas the remaining are "interaction" commands: Start/Stop to start and stop communication; Yes and No commands for a generalizable binary user input typically used for "approve" and "reject", respectively; Error for some choice of error signaling or correction.

StegaPhonetic news

StegaPhonetic news is a toy demonstration of a fictional, malicious use case for this steganography technique. It includes a set of fictional, financial news segments for the cover and a set of fictional stock tips for the payload.
Stock tips are in the form <up/down arrow><company stock ticker symbol>.
For example, ↑VZ for the tip "buy shares of Verizon, Inc." or ↓BLK for "sell shares of BlackRock, Inc."
Stock tips are randomly selected for embedding into the cover.
This simple example is a reminder of the dual-use nature of steganography and of language in general. Ethical considerations are important during the development of steganography techniques and of super language prototypes in general.

Implementation information

Mispronunciations The mispronunciation of any word is represented by a particular misspelling of that word. The mispronunciation of that word in an utterance is a phonetic reading of its misspelling. This is also the case when providing the text for artificial speech synthesis to a text-to-speech model. There are many words for which the current implementation does not generate a mispronunciation by

way of phonetic misspelling. In those cases, there is a catchall condition that adds the paralanguage "uh" following any word that must be mispronounced.

Payload alphabet The payload alphabet is any ordered set of characters. A character from the payload is embedded in the cover using the binary value of its index in the alphabet.

Chunk size This reduces the frequency of mispronounced words by increasing the total number of words required to encode any given payload. A chunk is the set of words associated with one bit in the payload. At least one of those words must be mispronounced for the bit to take on a value of 1. This adjustment makes it easier to produce and understand an utterance in StegaPhone. It can also make any message hidden in StegaPhone speech more resistant to steganalysis. The default chunk size is 1 so that one word in the cover corresponds to one bit in the payload.

Appendix B: Transformation Variation

Transformation variation across forms and interpretations, as well as other classification targets in augmentations, applications, etc.

Appendix B.1: Forms Tables

Forms: Simple structures

Forms of simple structure provided in Table 4.

Table 4: Simple container structures for transformation forms.

Category	Description/example
Symbol	The element "→" in an alphabet defined by the set {←,↑,→,↓,Ⓐ,Ⓑ}.
Substring	The sequence [→,→,↓,Ⓑ] or "→→↓Ⓑ" in string "→→↑↓Ⓐ↓Ⓑ".
Character	Character "h" in "hello world".
Word	Word "hello" in "hello world".
Clause	"I will swim" in "The forecast is unclear. If it is sunny, I will swim."
Sentence	"The forecast is unclear." in "The forecast is unclear. If it is sunny, I will swim."

Forms: Linguistic structures

Forms of linguistic structure provided in Table 5.

Table 5: Linguistic container structures for transformation forms.

Category	Description/example
Phonological/ Phonetic	Phonemes, pronunciations, allophones, articulatory gestures and other elements of speech production/perception. (StegaPhone example has this form structure)
Prosodic	Supra segmental elements of speech such as pitch, intonation, stress, rhythm.
Morphological	Morphemes. Both content (e.g. lake) and function (-ed suffix in English grammatical marker for past tense, as in "they jumped into the lake").
Syntactic	Syntactic functions that define the generation and parsing of morpheme sequences, including the relationship between constituents and clauses
Pragmatic	Different forms of context which affect meaning in language.
Lexical	Vocabulary, subsets of lexical corpuses.
Grammatical	Broader grammatical category to encompass the variety of grammatical structures used to define a language. For example, production rules in a formal grammar to recover part-of-speech tokens.
Semantic	Meaning/interpretation of symbols or sequences of symbols.
...	...

Forms: Extralinguistic structures

Forms of extralinguistic structure provided in Table 6.

Table 6: Extralinguistic container structures for transformation forms.

Category	Description/example
Paralinguistic	Nonlexical elements of speech (e.g. filler sounds like "uh") ...
Proxemic	Containers for space and distance ...
Chronemic	Orientations and organizations of time ...
Modal	Medium meta-category (e.g. body language). Kinesic, Haptic/Tactile, Gustatory, Olfactory, Oculesic, ...
Technological	Technology containers such as in cryptographic keys, blockchain operations ...
Sociocultural	E.g. belief systems, artistic artifacts ...
Semiotic	E.g. evolutionary, ecological, environmental containers, other ecosystem contexts ...
...	...

Forms: Interpretation structures

Forms can themselves be containerized into functional structures. They can be made from any of the interpretations (including ontological objects) outlined in Appendix B.2: Interpretations Tables. An example demonstration of this role reversal follows below.

T–V distinction modification Consider a modification $M[TV]$:
Mod Interpretation: control of social distance by a speaker when addressing an individual.
Mod Form: different second-person pronouns (e.g. French *tu* vs. *vous*).

T–V distinction squared modification Consider another modification $M[TV^2]$:
Mod Interpretation: topic shifting in discourse (e.g. changing of topic like "moving on").
Mod Form: use of the $M[TV]$ modification.
Note the original $M[TV]$ does not necessarily need to be in a specific container, like pronouns.
The form here has become the control of social distance in the T–V distinction.

Interpretation container example Consider this sequence of example transformations:
English: I would like to help you (informal). Moving on, the next item is ...
English + $M[TV]$: I would like to help thou. Moving on, the next item is ...
English + $M[TV]$ + $M[TV^2]$: I would like to help thou. The next item is ...
The last illustrates use of an interpretation structures as a form container.

Appendix B.2: Interpretations Tables

Interpretations: Categories

Interpretation categories provided in Table 7.

Table 7: Categories of functional roles for interpretations in super language transformation.

Category	Description text
Argument-Structure / Event-Structure	Establishes how events are conceptualized, specifying who does what to whom (Agent, Patient, Recipient, etc.) and how participants relate to each other in a clause. By systematically encoding roles, language can represent complex actions with multiple participants and outcomes.
Tense-Aspect-Modality (TAM)	Encodes the temporal and modal dimensions of events: when something happens (tense), whether it is ongoing or completed (aspect), and the speaker's attitude toward its likelihood or evidence (modality). This allows language to interweave time and epistemic stance into event descriptions, broadening discussion from not just what happens, but when and with what certainty.
Voice & Perspective	Alters how events are framed by shifting emphasis among participants (active, passive, reflexive, middle, etc.). This allows language to reshape perspective, highlighting or downplaying different participants and influencing how responsibility, agency, or viewpoint is perceived.
Information-Structure	Manages how new vs. given info is packaged, which elements are in focus or backgrounded, and how topics are introduced, maintained or shifted. By structuring the flow of information, language can guide attention, emphasize contrasts, and maintain coherence in discourse.
Illocutionary Force / Speech-Act	Encodes the speaker's communicative intent—whether making a statement, asking a question, giving a command, or expressing emotion. This transforms language into an interactive tool for performing actions (telling, requesting, commanding) rather than just describing.
Discourse-Structuring	Organizes communication at a level beyond single clauses, marking transitions, topic shifts, and parentheticals. This ensures extended discourse or conversation remains coherent and navigable with participants guiding the flow of conversation across multiple sentences or turns and shaping how ideas connect at a higher level than single clauses.
Pragmatic / Social-Indexical	Conveys social relationships, stance, politeness, identity markers, and contextual nuances of interaction. Enhances language beyond raw propositional content as a social tool, signaling formality, hierarchy, in-group vs. out-group status, or personal attitudes.
Idiomatic / Figurative	Conveys non-literal or partially compositional expressions (e.g. idioms, metaphors, set phrases). Enhancing language as a cultural, memetic tool with culturally rich, creative, and often evocative dimensions that expand expressive power beyond direct description, enabling speakers to convey nuanced or imaginative meaning.

Continued on next page

Table 7 – *Continued from previous page*

Category	Description text
Conceptual / Ontological	Introduces genuinely new concepts, entities, or ontological objects into the language, or significantly reorganizes conceptual boundaries. By adding or redefining what can be expressed (e.g. new scientific phenomena, cultural practices, or abstract concepts), language extends its semantic horizon and speakers can refer to and think about something previously unnamed or only vaguely conceptualized.
Information Encoding & Channels	Adds or modifies the methods by which information is symbolically represented—new alphabets, parallelization, additional semantic layers. It does not necessarily introduce new concepts, but transforms how existing concepts are delivered, such as increasing expressive capacity, security, or throughput in communication.
...	...

Interpretations: Examples

Interpretation examples provided in Table 8.

Table 8: Example functional roles for interpretations in super language transformation.

Category	Example	Example description	Example text
Argument-Structure / Event-Structure	Ditransitive	Conveys an Agent, an Indirect Object/Recipient, and a Direct Object/Theme in one event.	He baked her a cake.
	Caused-Motion	Emphasizes an Agent causing a Theme to move along a path.	She kicked the ball into the goal.
	Resultative	Presents an Agent causing an object to reach a new state.	He hammered the metal flat.
Tense-Aspect-Modality (TAM)	Past Tense	Places the action in a completed timeframe.	I walked home.
	Progressive	Emphasizes that the action is ongoing.	I am walking home.
	Modality	Conveys possibility, necessity, or uncertainty.	She might arrive soon.
Voice & Perspective	Passive	Centers the patient/theme while demoting the agent.	The cake was eaten (by John).
	Middle	Frames the subject as undergoing the action without a specified agent.	This car drives smoothly.
	Reflexive	Emphasizes that the agent and patient are the same entity.	They criticized themselves.
Information-Structure	Cleft	Spotlighting or isolating a key participant or piece of information.	It was Sarah who fixed the sink.

Continued on next page

Table 8 – *Continued from previous page*

Category	Example	Example description	Example text
	Topic-Marking	Clearly labels the topic or theme of the sentence.	Japanese: *Gakkou wa atarashii desu.* (English translation: The school is new.)
Illocutionary Force / Speech-Act	Imperative	Converts an utterance into a direct command.	Close the door!
	Yes/No Question	Requests confirmation or denial from the listener.	Are you leaving now?
	Exclamative	Expresses strong emotion or heightened emphasis.	What a gorgeous day!
Discourse-Structuring	Discourse Marker	Signals a shift or elaboration in discourse.	Well, let us consider another point...
	Topic Shift	Announces a change in the subject matter or focus of discussion.	Moving on to our next agenda item...
	Parenthetical	Inserts side remarks or comments without derailing the main clause.	He is—if you can believe it—already at the finish line.
Pragmatic / Social-Indexical	T/V Distinctions	Marks social distance or familiarity via separate second-person pronouns.	French *tu* vs. *vous*
	Stance Markers	Inserts the speaker's attitude, confidence, or emotion into the utterance.	Frankly, I think we should cancel the event.
Idiomatic / Figurative	Metaphor	Maps one conceptual domain onto another to create layered understanding.	Time is money.
	Idioms	Fixed, culturally entrenched phrases with meanings not always derivable from components.	kick the bucket, spill the beans
	Extended Figurative	Uses elaborate imagery to convey emotion or attitude.	He was skating on thin ice with that comment.
Conceptual / Ontological	Coining Neologisms	Creates a word for a previously unknown/unnamed phenomenon.	quark, quasar, meme
	Conceptual (Re)Framing	Alters how a domain is understood (e.g. cyclical time) without coining new words.	"Time is a circle" (requiring change in temporal framing)
	Category Merging	Combines or splits existing categories (e.g. color terms) to reflect new conceptual boundaries.	Merging "blue" and "green" into one color category

Continued on next page

Table 8 – *Continued from previous page*

Category	**Example**	**Example description**	**Example text**
Information Encoding & Channels	Parallelization (StegaPhone example)	New channel for sequence (often binary bits) in the token containers of the associated form.	StegaPhone “swim” vs. “svim” for binary 0 vs. 1
	Semantic Representation	Changes of semiotic signs and symbols carrying meaning.	Make letters, not words, the smallest unit of meaning that can stand on its own.
	Color-Coded Grammar	Uses color to indicate grammatical categories (e.g. tense or case), clarifying or speeding comprehension. Details of color coding scheme in associated form.	Red text for verbs, blue text for nouns, etc.
	Morphological Fusion	Merges multiple grammatical markers (e.g. tense + aspect + evidential) into a single affix for high information density.	One suffix for past + perfect + reported: -pecrep
...	...	...	...

Appendix B.3: Augmentations Tables

Augmentations: Categories & Examples

Augmentation categories and examples provided in Table 9.

Table 9: Sketch of categories and examples for augmentations in super language transformation.

Category	**Category description**	**Example**	**Example description**
Speed	$+\Delta$ forms/interps per unit time (i.e. increase in information rate).	...	...
Quantity	$+\Delta$ number of forms/interps (ignores distance from existing forms/interps).	...	...
Capacity	$+\Delta$ number of forms/interps (considers distance from existing forms/interps). Often measuring expressive potential in sender/receiver messages.	...	...
Surprisal	$+\Delta$ information load per unit (e.g. symbol/token) so more meaning compressed per unit.	Words-to-Alphabet	Words $\rightarrow$ unique symbol in new orthography (e.g. apple $\rightarrow$ symbol1, pear $\rightarrow$ symbol2, ...)
Density	...	...	...
Dimensionality	$+\Delta$ number of channels modes of expression.	Multimodal Layering	Modal encodings with new semantic cues.
	$+\Delta$ degrees of freedom per unit expression or communication.	StegaPhone	Embed binary (1)0 in (mis)pronunciation of words.
Accuracy	...	...	...
Precision	...	...	...
Privacy	...	...	...
Security	$+\Delta$ resistance against adversarial interception of messages. $-\Delta$ message decipherability per unit communication.	Crypto Grammar	Cryptographic primitives (e.g. hash function) in morphological or syntactic containers.
		Dialog Checksum	Linguistic adaptation to longitudinal parity check.
Parsimony	...	...	...
Robustness	...	...	...
Intelligence Loading	$+\Delta$ in [Capacity] augmentation per unit modification (e.g. *g-factor* correlation with tests of users learning new forms/interps in variant).	...	...
Learning	...	...	...
...	...	...	...

Appendix C: Transformation Engine Information

Additional information for the transformation engine described in Description 3 and Description 4. For clarity, modifications are absorbed in one modification (M).

Appendix C.1: Transformation Specification Components

Base (B) Base B or string instances $b \in B$ are comprised of symbols from the base alphabet Σ_B.

Variant (V) Variant V or string instances $v \in V$ are comprised of symbols from the variant alphabet Σ_V. In a transformation, variants are comprised of interwoven sequences of substrings from the base and token replacements from applied substitutions. Variants manifest the form in a transformation and thereby serve as a target of control via modification commands.

Modification (M) Modification $M(\tau, R, \mu)$ includes the tokenization (τ) over a base (B) and variant (V), rewrite system including ordered set of rewrite rules (R), the modification function (μ). Application of one or more modifications over a base defines a variant. A modification may be one or more modifications. The inverse of the modification (M^{-1}) must include the definition for the inverse modification function (μ^{-1}).

Tokenization (τ) The tokenization is a function over strings of the base, or variant, that will return the targets for the rewrite rules (R) patterns, or replacements. There are a variety of tokenizations available including: words, sentences, (word, part-of-speech), and many more (Appendix B: Transformation Variation).

Tokens (T) The resulting base/variant tokens are the patterns/replacements of the rewrite rules. Token alphabet (Σ_B) will supplement the base and variant alphabets to execute the parsing and tokenizing.

Rewrite Rules (R) The rewrite rules are ordered set of pairs of patterns (α) and replacements (β) for substitutions ($\alpha \rightarrow \beta$) over a string. Substitutions are possible at tokens matching patterns/replacements. Each rewrite rule r_j in R is given by: $r_j = (\alpha_j, \beta_j, \mu)$ where α_j is the "pattern", β_j is the "replacement", and μ is the modification function. The rewrite rules (R) form a Rewrite System (RS).

Rewrite System (RS) The rewrite system is the ordered set of rewrite rules (R) where modification function (μ) is the degenerate default. It can produce all possible values a variant string can take on following super language transformation (and similarly all possible values a base string can take on following inverse transformation). Following tokenization, RS can generate all possible $v \in V$ for $b \in B$, and vice versa. This is an important property. As a result, it is possible to enumerate the set of all possible forms a transformation can create independently of the set of all possible interpretations.

Modification Function (μ) The rewrite system makes it possible (i) to manifest the forms in super language transformation. Modification function (μ) extends this implementation so that is also possible to (ii) control the forms, as well as to (iii) manifest and (iv) control the interpretations of a transformation. The modification function will implement the modification or modifications of a transformation.

Inverse Modification Function (μ^{-1}) The inverse modification function is the inverse of the modification function. It may need to be further specified when there are additional requirements on the inversion (e.g. must be bijective, injective, or surjective).

Modification Commands (U) Modification commands (U) are inputs to the modification function (μ) and will vary. They specify operation over the base string to generate a target variant string, a target artifact, or both. They enable the control of both form and interpretation in the transformation. Without loss of generality, let the commands be a symbolic sequence drawn from another alphabet, the commands alphabet (Σ_U).

Modification Artifact (W) Modification artifact (W) is output to the modification function (μ) and will vary. They manifest the interpretation in a transformation and thereby serve as a target of control via modification commands. Without loss of generality, let the artifacts be a symbolic sequence drawn from another alphabet, the artifact alphabet (Σ_W). The artifact is composed of two subcomponents, the interpretation (W_{int}) and the inverse key (W_{key}).

Modification Artifact Interpretation (W_{int}) The interpretation is the functional role of the transformation. It can vary greatly with very minor adjustments to this role (see Example 3 for an illustration).

Modification Artifact Inverse Key (W_{key}) The inverse key is the minimum data necessary to execute the inverse transformation. At most, the key will include the sequences of all modifications applied in a transformation. Often, for a given variant, there are more computationally efficient implementations of the inverse transformations. For example in StegaPhone, the base string is the correct pronunciation all words in the variant. The commands, in this case the payload, are retrieved from word-wise decoding of the binary data stream. Neither of these require a decompiling of the rewrite sequences.

Appendix C.2: Transformation Engine Algorithm

Table 10 outlines the steps of the transformation engine, enabling generation of super language transformations and their inverse via transformation specification recipes. Example 4 provides a transformation specification outline for the StegaPhone example variant.

Table 10: Transformation engine steps mapping base and commands to variant and artifact, and vice versa.

Steps	**Description**
Step 0	Receive transformation specification (Z, Z^{-1}) and begin engine execution. $\hookrightarrow$ Start Sender mode and go to Step 1. $\hookrightarrow$ Start Receiver mode and go to Step 6.
Step 1	Receive base string ($b \in B$) input.
Step 2	Execute tokenization (τ) over base string to create tokens (T).
Step 2a	(optional) Execute the rewrite system (RS) over base string.
Step 3	Receive modification commands ($u \in U$), if any. May require function F_U to map variety of inputs to direct output in V and W.
Step 4	Execute the transformation.
Step 4a	Call initialization mod function (μ_0) to begin rewriting, iterating (ℓ) through each substring (s_n) and token match (t_k). Mod function call at each iteration is $\mu(x_\ell \vert \ell) = \mu(x_\ell) = \mu_\ell$ where ℓ is either a token match k or non-token substring n. Mod function has iterative input $x_\ell = \mu'(x_{\ell-1})$ and output $\mu(x_\ell) = (\delta_\ell, w_\ell, \mu'_\ell)$. The $\delta_\ell, w_\ell, \mu'_\ell$ are the values of rewrite rule substitution, modification artifact ($w_\ell = (w_{int}, w_{key})_\ell$), and modification function state, respectively, at iteration ℓ.
Step 4b	If ℓ is n, call mod function μ_n with δ_n output always 0.
Step 4c	If ℓ is k, call mod function μ_k (i.e. μ_{ji}). $r_k(\alpha_j, \beta_j, \mu_{ji})$ is iteration having rewrite rule r_k with token i matching pattern α_j To determine whether to apply substitution ($\alpha \to \beta$), evaluate δ_k in μ_k. $\delta_k = \begin{cases} 1, \text{ apply substitution} \\ 0, \text{ skip} \\ \text{-1, apply \& skip (degenerate transformation)} \end{cases}$
Step 4d	Following last iteration, execute terminating call (μ_N).
Step 5	Return variant (v), artifact (w), Z having determined which $v, w \in V, W$ for source $b, u \in B, U$. $\hookrightarrow$ End Sender mode and exit.
Step 6	Execute inverse transformation procedure (Z^{-1}) Unless otherwise implemented (very likely), artifact inverse key (w_{key}) has recorded transformation steps as a rewrite sequence (in each μ'_ℓ). This key can be used to decompile the transformation in reverse to retrieve source base string and commands for given variant string and artifact.
Step 6a	Return base (b), commands (u), Z^{-1} having determined which $b, u \in B, U$ for source $v, w \in V, W$. $\hookrightarrow$ End Receiver mode and exit.

Appendix C.3: Engineless Super Language

Appendix C.3a: Super language in generative grammar representation

The super language framework and prototypes can be expressed in the abstractions of generative grammar theories.

Let base language L be any language over an alphabet Σ. Let G be a grammar of this base L. Grammar $G = (N, \Sigma, P, S)$ where N: Finite set of non-terminal symbols, Σ: Finite set of terminal symbols (alphabet), P: Finite set of production rules, $S \in N$: Start symbol.

Super language is a function that maps languages to languages, $Z : \mathcal{L} \to \mathcal{L}$ where $\mathcal{L}$ is the set of all languages. A super language Z is a transformation of language L to a new language $Z(L)$. For any particular transformation Z from L to $Z(L)$ there may be more than way to define it.

The modifications on the language can be defined in terms of the operator $\hat{Z}$, $\hat{Z} : \mathcal{G} \to \mathcal{G}$ where $\mathcal{G}$ is the set of all grammars. More precisely, the Z is Z^θ and the $\hat{Z}$ is $\hat{Z}^\theta$ since this example in the generative grammar domain concerns the transformation *form*, not the *interpretation*. For clarity, here Z^θ and $\hat{Z}^\theta$ are written as Z and $\hat{Z}$. Applied to a grammar G, it produces a modified grammar, $G_{\hat{Z}} = \hat{Z}(G)$ which becomes the grammar of the super language $Z(L)$.
In summary:

$$Z(L) = L(G_{\hat{Z}}) = L(\hat{Z}(G))$$

$\hat{Z}$ can be defined as a tuple of functions acting on the components of G:

$$\hat{Z} = (f_N, f_\Sigma, f_P, f_S)$$

where:
$f_N : N \to N_{\hat{Z}}$, modifying the set of non-terminals.
$f_\Sigma : \Sigma \to \Sigma_{\hat{Z}}$, modifying the set of terminals.
$f_P : P \to P_{\hat{Z}}$, modifying the set of production rules.
$f_S : S \to S_{\hat{Z}}$, modifying the start symbol.

In this case, there is a simplified definition of $\hat{Z}$ using set operations.
The components of the tuple are the set operation applied to the corresponding component of the grammar tuple.

$$\hat{Z} = (\Delta N, \Delta\Sigma, \Delta P, \delta S)$$

Excluding the starting symbol, the components of the grammar operator can be described in terms of the set additions and deletions.

$$\hat{Z} = ((\Delta N^+, \Delta N^-), (\Delta\Sigma^+, \Delta\Sigma^-), (\Delta P^+, \Delta P^-), \delta S)$$

Non-terminals modification $\Delta N = (\Delta N^+, \Delta N^-)$, where $\Delta N^+ \subseteq N_{\text{new}}$, non-terminals to be added; $\Delta N^- \subseteq N$, non-terminals to be removed.

Terminals modification $\Delta\Sigma = (\Delta\Sigma^+, \Delta\Sigma^-)$, where $\Delta\Sigma^+ \subseteq \Sigma_{\text{new}}$, terminals to be added; $\Delta\Sigma^- \subseteq \Sigma$, terminals to be removed.

Production Rules modification $\Delta P = (\Delta P^+, \Delta P^-)$, where $\Delta P^+ \subseteq P_{\text{new}}$, production rules to be added; $\Delta P^- \subseteq P$, production rules to be removed.

Start Symbol modification $\delta S = \begin{cases} S' & \text{if the start symbol is modified, where } S' \in N \cup \Delta N^+ \\ S & \text{if the start symbol is not modified} \end{cases}$

The modified grammar $G_{\hat{Z}}$ is generated from simple set operations.

Non-terminals: $N_{\hat{Z}} = (N \cup \Delta N^+) \setminus \Delta N^-$
Terminals: $\Sigma_{\hat{Z}} = (\Sigma \cup \Delta\Sigma^+) \setminus \Delta\Sigma^-$
Production Rules: $P_{\hat{Z}} = (P \cup \Delta P^+) \setminus \Delta P^-$
Start Symbol: $S_{\hat{Z}} = \delta S$

A super language Z_0 can be the base of a new super language Z, such as in a nested super language, $Z(Z_0(L)) = L((G_{\hat{Z}_0})_{\hat{Z}})$. A super language can be characterized by one or more sets of operator sequences $\{\hat{Z}_1, \hat{Z}_2, \ldots, \hat{Z}_n\}$.

Appendix C.3b: StegaPhone in generative grammar representation

Suppose base L (or L_{en}) is the English oral language, in particular the set of all correctly pronounced English sentences. Defining formally the generative grammar G (or G_{en}) that generates all of oral English is challenging. By extension, doing so for all of StegaPhone (G_{sp} for Z_{sp}) would be more challenging still.

To address these challenges, we can instead create an intermediary super language Z_0 that will overgenerate valid strings in L ($Z_0 \supset L_{en}$) to create a simplified G_0.

$$\hat{Z}_0 : G_{en} \to G_{\hat{Z}_0}$$
$$G_0 = G_{\hat{Z}_0} = (N_0, \Sigma_0, P_0, S_0)$$

The components of the intermediary super language grammar G_0 are below.
Non-terminals: $N_0 = \{\text{Sentence}, \text{WordSequence}, \text{Word}, S\}$
Terminals: $\Sigma_0 = \{\, w \mid w \text{ is an English word}\}$, i.e. all correctly pronounced English words in Σ_{en} of base
Production rules: $P_0 = P_{parser} \cup \{S \to \text{Sentence}\}$ where P_{parser} has the following rules:
1. Sentence $\to$ WordSequence
2. WordSequence $\to$ WordSequence Word
3. WordSequence $\to$ Word
4. Word $\to$ w, $\forall w \in \Sigma_0$

Start symbol: S
This approach will avoid the need to explicitly describe all details of base grammar G_{en}. In general, this simplification is critical for less encumbered exploration of new language prototypes.

We start with the operator expression:

$$\hat{Z} = ((\Delta N^+, \Delta N^-), (\Delta\Sigma^+, \Delta\Sigma^-), (\Delta P^+, \Delta P^-), \delta S)$$

We define a simple operator $\hat{Z}_0$ to apply to G:

$$\hat{Z}_0 = ((\{\text{WordSequence}, \text{Word}\}, N_{en} \setminus \{S, \text{Sentence}\}), (\emptyset, \emptyset), (P_{parser}, P_{en} \setminus \{S \to \text{Sentence}\}), \emptyset)$$

Applying $\hat{Z}_0$ to G_{en} creates intermediary super language Z_0. The resulting grammar G_0 parses valid oral English sentence of L_{en} into its component words. Z_0 is now more amenable for the StegaPhone super language transformation Z_{sp}.

$$\hat{Z}_{sp} = ((\emptyset, \emptyset), (\Sigma', \emptyset), (\{\text{Word} \to w',\ \forall w' \in \Delta\Sigma^+\}, \emptyset), \emptyset)$$

where the terminal additions $\Delta\Sigma^+$ is Σ', the set of all mispronounced English words. For each $w \in \Sigma_{\hat{Z}_0}$, add an incorrectly pronounced version w', $\Delta\Sigma^+ = \Sigma'$ where $\Sigma' : \{w' \mid w \in \Sigma_{\hat{Z}_0}\}$.
We can chain the super language transformations together, $\hat{Z} = \hat{Z}_{sp}(\hat{Z}_0)$ and in terms of its generative grammar, $G_{\hat{Z}_{sp}} = \hat{Z}_{sp}(G_{\hat{Z}_0}) = \hat{Z}_{sp}(\hat{Z}_0(N_{en}, \Sigma_{en}, P_{en}, S_{en}))$.
Applying this chain produce the components of G_{sp} stated in Example 2.
The super language includes all sentences where each word can be either correctly pronounced ($w \in \Sigma$) or incorrectly pronounced ($w' \in \Delta\Sigma'$).

For the interpretation, the binary scheme has correct pronunciation ($w \in \Sigma$) encode 0 and incorrect pronunciation ($w' \in \Delta\Sigma'$) encode 1.

Example transformation Choose sentence $s \in L_{en}$ to create $Z(s)$.
s_{en} = "hello world"
$Z_0(s_{en})$ = "hello world"

$$Z_{sp}(Z_0(s)) = \begin{cases} \text{"hello world", binary payload "00"} \\ \text{"hello vorld", binary payload "01"} \\ \text{"ello world", binary payload "10"} \\ \text{"ello vorld", binary payload "11"} \end{cases}$$

Super language $Z_{sp}(Z_0(L_{en}))$ extends oral English with a new grammar for stegaphonetic communication.