Derivational and Interpretational Equivalence of LIG and HG

Kalen Chang
Department of Linguistics
University of California, Los Angeles
Los Angeles, CA, USA
kalen@ucla.edu

Abstract

Linear Indexed Grammar (LIG) and Head Grammar (HG) are derivationally equivalent: not only do they generate the same string languages, each pair of equivalent grammars generates each string in the same number of ways. Moreover, any compositional interpretation for a grammar in one formalism can be transformed into a compositional interpretation of the equivalent grammar in the other formalism such that the same strings are assigned the same meanings. Although this paper focuses on these two formalisms, it serves as an example for how to compare formalisms beyond the string languages they generate.

1 Introduction

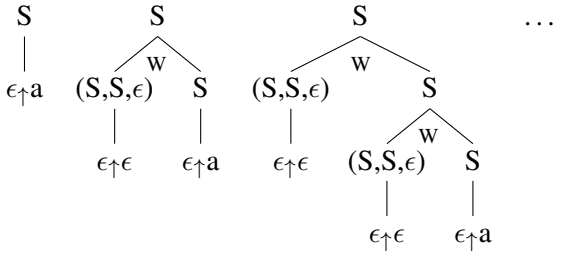
When considering different ways to compare grammar formalisms, weak generative capacity, or the string languages generated, is one common point of comparison. For example, Vijay-Shanker and Weir (1994) proved that four mildly context-sensitive grammar formalisms generate the same set of string languages. Of those four, this paper will examine two in detail: Linear Indexed Grammars (LIG) and Head Grammars (HG).

Though generating the correct set of strings is important, linguists use grammars to provide analyses of languages. They may assign internal structure to strings to explain how the subcomponents of a string relate to the whole – compositionality – or how a string may be associated with more than one meaning – ambiguity.

Thus, it is no surprise that linguists use notions beyond weak generative capacity to compare formalisms, such as comparing tree structures associated with grammars (e.g. Bresnan et al., 1982; Frank and Hunter, 2021). However, comparison across tree structures is not infallible. While some formalisms like Tree Adjoining Grammars directly derive tree structures, others, such as HG, only derive strings (or pairs thereof). The only notion of structure present in HG and string-generating formalisms are derivation structures, which record which rules were used to construct the string and how those rules interact. Though grammar formalisms vary wildly in how derivations are structured, there are still ways to compare derivations across formalisms.

For example, we might count the number of derivations assigned to a each string, so that a two-way ambiguous string in a grammar is assigned two different derivations. The translation given in Vijay-Shanker and Weir (1994) does not preserve the number of derivations per string. To see this, consider an extremely simple LIG, which has a single rule S[] → a, and a single derivation.

Following their translation, the corresponding HG has infinitely many derivations:[1]



This problem arises more generally for any pair of weakly equivalent grammars given by their translation: despite being weakly equivalent, the HG will have infinitely many derivations. In fact, no matter how many derivations generate a string in the LIG, there will be infinitely many derivations that generate that string in the HG.

I propose two ways to handle this problem. One involves changing the grammar translation so that the resulting grammar is derivationally equivalent: the LIG and the HG generate the same number of derivations per string. The other way involves

[1] The W under a node indicates wrapping two components of the left daughter around those of the right daughter.

defining a method of interpreting HG derivations given an LIG interpretation function such that the resulting interpreted HG generates the same set of string-meaning pairs, or vice versa.

In this paper, I will show both of these are possible with LIG and HG: they are both derivationally and interpretationally equivalent. After preliminary definitions in Section 2, Section 3 lays out a function to construct, given an arbitrary LIG, a derivationally equivalent HG, i.e. one which has the same number of derivations per string, in a sense comparing the multiset of strings generated. Section 4 presents the reverse translation, from HG to LIG.

In Section 5, I will prove that for any LIG equipped with a homomorphic interpretation function on derivations, there is an HG with a homomorphic interpretation function such that the two interpreted grammars generate the same set of string-meaning pairs (and vice versa). The notion of interpretational equivalence is intended to encompass any compositional property relating to meaning or relations between parts of the derivation that linguists wish to capture in a grammar as an analysis of a language.

Section 6 discusses the relation between the approach taken in this paper and a few other papers which have looked at notions of equivalence across formalisms, and Section 7 concludes the paper.

2 Definitions

2.1 Linear Indexed Grammars

Linear Indexed Grammars (Gazdar, 1988) extend Context-Free Grammars by adding stacks to nonterminals and allowing rules where exactly one daughter inherits the stack from the mother, modulo whatever changes to the stack the rule specifies. In effect, there are now an infinite number of categories represented by nonterminals with stacks, in the same way a pushdown automaton can be viewed as having an infinite number of states.

Formally, an LIG is a 5-tuple of finite sets $\mathcal{G} = (V_\text{N}, V_\text{T}, V_\text{I}, S, P)$, consisting of nonterminal symbols V_N, terminal symbols V_T, stack symbols (indices) V_I, start symbols $S \subseteq V_\text{N}$, and production rules P. Each production rule has one of two forms:

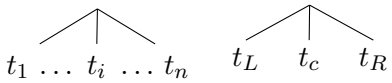
- Leaf: $A[] \to w$, where $A \in V_\text{N}, w \in V_\text{T}^*$
- Branch: $A[\zeta] \to B_1 \dots B_i[\eta] \dots B_n$, where $n \geq 1; \zeta, \eta \in V_\text{I} \cup \{\epsilon\}; \forall j.\, B_j \in V_\text{N}$

Let n denote the rank of a Branch rule, and Leaf rules have rank 0. I assume each Branch rule is either Push ($\zeta = \epsilon \neq \eta$), Pop ($\eta = \epsilon \neq \zeta$), or No Change ($\zeta = \eta = \epsilon$).[2]

Additionally, I will abbreviate such a rule as $A[\zeta] \to B_L\, B_c[\eta]\, B_R$, where $B_c = B_i$ is the designated/central daughter of A which receives the remainder of the stack, $B_L = B_1 \dots B_{i-1}$ represents the daughters to the left of the center, and similarly $B_R = B_{i+1} \dots B_n$ the daughters right of the center.

Treat LIG production rules as a ranked alphabet, where the rank of each alphabet symbol is the rank of the rule. Let the set of trees over such rules be $\mathcal{T}_\text{LIG}$ (i.e. where the number of daughters of a node matches the rank of the rule of that node). LIG derivations (or derivation trees) are elements of $\mathcal{T}_\text{LIG}$. Define $\mathcal{T}(\mathcal{G})$ as the set of trees over the rules of $\mathcal{G}$.

I will use the following abbreviation for trees, similar to rules: if $m = A[\zeta] \to B_1 \dots B_i[\eta] \dots B_n$, $t_L = t_1, \dots, t_{i-1}$, $t_c = t_i$, and $t_R = t_{i+1}, \dots, t_n$ where each $t_j \in \mathcal{T}(\mathcal{G})$, then $m = m$.



Let **cat** denote the category ($V_\text{N} \times V_\text{I}^*$) of a derivation. For trees consisting of a single node:

$$\textbf{cat}(A[] \to w) = A[]$$

For larger trees: if $\textbf{cat}^*(t_L) = B_L[]$, $\textbf{cat}(t_c) = B_c[\eta\sigma]$ ($\sigma \in V_\text{I}^*$), and $\textbf{cat}^*(t_R) = B_R[]$, then[3]

$$\textbf{cat}\left(\begin{array}{c} A[\zeta] \to B_L\, B_c[\eta]\, B_R \\ t_L \quad t_c \quad t_R \end{array}\right) = A[\zeta\sigma],$$

else it is undefined.

An LIG derivation d is well-formed with respect to an LIG $\mathcal{G}$, i.e. $\textbf{wf}_\mathcal{G}(d)$, iff $\textbf{cat}(d) = A[s]$ for $A \in V_\text{N}$, $s \in V_\text{I}^*$, and for every node r in d, $r \in P$. Let $\mathcal{D}(\mathcal{G})$ be the set of LIG derivations well-formed with respect to $\mathcal{G}$. A well-formed LIG derivation d is complete with respect to an LIG $\mathcal{G}$, i.e. $\textbf{cwf}_\mathcal{G}(d)$, iff $\textbf{cat}(d) = A[]$ for some $A \in S$. Let $\mathcal{D}_\text{LIG}$ be the set of all well-formed LIG derivations, i.e. the set of derivations which are well-formed with respect to any LIG.

[2] This assumption does not affect any equivalence results; simply replace any rule which performs more than one stack actions with a set of rules which only perform one.

[3] f^* denotes mapping f over a set or list of arguments.

A spine is a node which is not a designated daughter, along with the maximal chain of designated daughters it dominates. Define the sequence $\textbf{flagseq}(d) = [\textbf{flag}(r) \mid r \in s]$, where s is the spine of the root of d, and

$$\textbf{flag}(A[\zeta] \to A_L\, A_c[\eta]\, A_R) = \begin{cases} (_\eta & \text{if } \eta \neq \epsilon, \\ )_\zeta & \text{if } \zeta \neq \epsilon, \\ \epsilon & \text{otherwise.} \end{cases}$$

Lemma 1. For a well-formed derivation d, if $\textbf{cat}(d) = A[], A \in V_{\text{N}}$, then $\textbf{flagseq}(d)$ is a Dyck word, i.e. one generated by $S \to \epsilon \mid (_\eta\ S\ )_\eta\ S$, for each $\eta \in V_{\text{I}}$.

The yield $\textbf{y} : \mathcal{D}(\mathcal{G}) \to V_{\text{T}}^*$ is the result of reading the terminal symbols at the leaves of the tree.

$$\textbf{y}(A[] \to w) = w$$

$$\textbf{y}\left(\begin{array}{ccc} & m & \\ t_L & t_c & t_R \end{array}\right) = \textbf{y}^*(t_L)\textbf{y}(t_c)\textbf{y}^*(t_R)$$

The language of a grammar $\mathcal{L}(\mathcal{G})$ is given as the set of yields of the complete well-formed derivations of $\mathcal{G}$: $\mathcal{L}(\mathcal{G}) = \{\textbf{y}(d) \mid \textbf{cwf}_{\mathcal{G}}(d)\}$

2.2 LIG Tree contexts

In this subsection, I will define LIG tree contexts, which are closely related to LIG derivations, to serve as a useful intermediate structure in the translation from LIG to HG derivations. Intuitively, LIG tree contexts are LIG derivation trees where the rule at the bottom of the root spine has been removed. Accordingly, a context paired with a Leaf rule is isomorphic to a derivation.

LIG tree contexts $\mathcal{K}_{\text{LIG}}$ is the smallest set s.t.

- $\square \in \mathcal{K}_{\text{LIG}}$
- If $m \in P$ has rank $n \neq 0$; $\gamma_1, \ldots, \gamma_n \in \mathcal{K}_{\text{LIG}}$; and $\beta_1, \ldots, \beta_{i-1}, \beta_{i+1}, \ldots, \beta_n$ are Leaf rules; then $\begin{array}{ccccc} & & m & & \\ (\gamma_1, \beta_1) & \ldots & \gamma_i & \ldots & (\gamma_n, \beta_n) \end{array} \in \mathcal{K}_{\text{LIG}}$

and similarly for $\mathcal{K}(\mathcal{G})$, LIG tree contexts over a grammar $\mathcal{G}$.

It will be handy to have a function **pt** to extract the node at the bottom of the spine of a derivation (the tail of the spine), which will be a Leaf (rank 0) rule (P_L). Define $\textbf{pt} : \mathcal{T}(\mathcal{G}) \to \mathcal{K}(\mathcal{G}) \times P_L(\mathcal{G})$ as:

- For $\textbf{rank}(\beta) = 0$:
  $\textbf{pt}(\beta) = (\square, \beta)$
- For $\textbf{rank}(m) > 0$:
  $$\textbf{pt}\left(\begin{array}{ccc} & m & \\ t_L & t_c & t_R \end{array}\right) = \left(\begin{array}{ccc} & m & \\ \textbf{pt}^* t_L & \Gamma & \textbf{pt}^* t_R \end{array}, \beta\right),$$
  where $(\Gamma, \beta) = \textbf{pt}(t_c)$.

Let $\Gamma\{\Delta_1 \mapsto \Delta_2\}$ denote locating the subcontext or subtree Δ_1 in Γ and replacing it with the context or tree Δ_2. Formally, $\Gamma\{\Delta_1 \mapsto \Delta_2\} =$

$$\begin{cases} \Delta_2 & \text{if } \Gamma = \Delta_1 \\ \begin{array}{ccc} & m & \\ t_L & \gamma_c\{\Delta_1 \mapsto \Delta_2\} & t_R \end{array} & \text{if } \Gamma = \begin{array}{ccc} & m & \\ t_L & \gamma_c & t_R \end{array} \end{cases}$$

and let $\Gamma\{\Delta\} = \Gamma\{\square \mapsto \Delta\}$.

It is straightforward to show that if $\textbf{pt}(d) = (\Gamma, \beta)$, then $\Gamma\{\beta\} = d$, and so **pt** is invertible. From now on, I will treat elements of $\mathcal{T}_{\text{LIG}}$ and $\mathcal{K}_{\text{LIG}} \times P_L$ as interchangeable.

Let $\mathcal{C}(\mathcal{G})$ denote the set of well-formed LIG tree contexts over $\mathcal{G}$ — those which come from well-formed LIG derivations.

$$\mathcal{C}(\mathcal{G}) = \{\Gamma \mid d \in \mathcal{D}(\mathcal{G}), \textbf{pt}(d) = (\Gamma, \beta)\}$$

I define the notion of category for an LIG context: $\textbf{catc} : \mathcal{C}(\mathcal{G}) \times V_{\text{N}} \to V_{\text{N}} \times V_{\text{I}}^*$, which returns the category of what the context would be if it was filled by a lexical rule of the given category.

$$\textbf{catc}(\Gamma, B) = \textbf{cat}(\Gamma\{B[] \to w\})$$

Thus, $\textbf{catc}(\square, B) = B[]$.

2.3 Head Grammars

Head Grammars (Pollard, 1984; Roach, 1987) increase the expressive power of CFGs by manipulating a slightly more complicated structure: a pair of strings, separated by an arrow $\uparrow$. Derivations manipulate pairs of strings, and a new kind of rule is added: one which wraps one pair of strings around another.

Formally, an HG is a 4-tuple $G = (V_{\text{N}}, V_{\text{T}}, S, P)$, consisting of nonterminal symbols V_{N}, terminal symbols V_{T}, start symbols $S \subseteq V_{\text{N}}$, and production rules P. Each production rule has one of three forms:

- Leaves: $A \to u_\uparrow v$, where $A \in V_{\text{N}}; u, v \in V_{\text{T}}^*$
- Wrap: $A \xrightarrow{W} B_l\, B_r$, where $A, B_l, B_r \in V_{\text{N}}$
- Concat$_i$: $A \xrightarrow{C_i} B_1 \ldots B_n$, where $i \leq n$ and $A, B_1, \ldots, B_n \in V_{\text{N}}$

The rank of Leaves rules is 0, Wrap rules 2, and Concat rules n. An abbreviation similar to that for LIG rules will be applied to Concat rules: $A \xrightarrow{C_i} B_1 \dots B_n = A \xrightarrow{C_c} B_L\, B_c\, B_R$.

Treating HG production rules as a ranked alphabet, where the rank of each alphabet symbol is the rank of the rule, let the set of trees over such rules be $\mathcal{T}_{\text{HG}}$, and an HG derivation is a tree of that set. Define $\mathcal{T}(\mathcal{G})$ as the set of trees over the rules of $\mathcal{G}$.

Let **cat** denote the category (V_{N}) of a derivation.

$$\textbf{cat}(A \to u_{\uparrow} v) = A$$

If $\textbf{cat}(t_l) = B_l$ and $\textbf{cat}(t_r) = B_r$, then

$$\textbf{cat}\left(\begin{array}{c} A \xrightarrow{W} B_l\, B_r \\ \diagup \quad \diagdown \\ t_l \quad t_r \end{array}\right) = A.$$

If $\textbf{cat}^*(t_L) = B_L$, $\textbf{cat}(t_c) = B_c$, and $\textbf{cat}^*(t_R) = B_R$, then

$$\textbf{cat}\left(\begin{array}{c} A \xrightarrow{C_c} B_L\, B_c\, B_R \\ \diagup \quad | \quad \diagdown \\ t_L \quad t_c \quad t_R \end{array}\right) = A,$$

else it is undefined.

An HG derivation d is well-formed with respect to an HG $\mathcal{G}$, i.e. $\textbf{wf}_{\mathcal{G}}(d)$, iff $\textbf{cat}(d) \in V_{\text{N}}$, and for every node r in d, $r \in P$. Let $\mathcal{D}(\mathcal{G})$ be the set of HG derivations well-formed with respect to $\mathcal{G}$. A well-formed derivation d is complete with respect to an HG $\mathcal{G}$, i.e. $\textbf{cwf}_{\mathcal{G}}(d)$, iff $\textbf{cat}(d) \in S$. Let $\mathcal{D}_{\text{HG}}$ be the set of all well-formed HG derivations, i.e. derivations well-formed with respect to any HG.

The yield function $\mathbf{y} : \mathcal{D}(\text{HG}) \to V_{\text{T}}^* \times V_{\text{T}}^*$ is defined as:

$$\mathbf{y}(A \to u_{\uparrow} v) = u_{\uparrow} v$$

If $\mathbf{y}(t_l) = x_{l\uparrow} y_l$, and $\mathbf{y}(t_r) = x_{r\uparrow} y_r$:

$$\mathbf{y}\left(\begin{array}{c} A \xrightarrow{W} B_l\, B_r \\ \diagup \quad \diagdown \\ t_l \quad t_r \end{array}\right) = x_l x_{r\uparrow} y_r y_l$$

If $\mathbf{y}(t_j) = x_{j\uparrow} y_j$ for all $1 \le j \le n$:

$$\mathbf{y}\left(\begin{array}{c} A \xrightarrow{C_c} B_L\, B_c\, B_R \\ \diagup \quad | \quad \diagdown \\ t_L \quad t_c \quad t_R \end{array}\right) = x_1 y_1 \dots x_{i\uparrow} y_i \dots x_n y_n$$

The language of a grammar $\mathcal{L}(\mathcal{G})$ is defined as:

$$\mathcal{L}(\mathcal{G}) = \{uv \mid \textbf{cwf}_{\mathcal{G}}(d), \mathbf{y}(d) = u_{\uparrow} v\}$$

3 Derivational equivalence: LIG to HG

Two grammars are *derivationally equivalent* iff they generate the same number of derivations for each string. Two formalisms $\mathcal{F}_1$ and $\mathcal{F}_2$ are *derivationally equivalent* iff for each grammar $\mathcal{G}_1 \in \mathcal{F}_1$, there is some $\mathcal{G}_2 \in \mathcal{F}_2$ that is derivationally equivalent to $\mathcal{G}_1$, and vice versa. In this section, I will prove that for every LIG, there is a derivationally equivalent HG.

First, given the source LIG $\mathcal{G}$, I will define the resulting HG $\mathcal{G}' = \text{LH}(\mathcal{G})$. Next, I define a function $\textbf{lh} : \mathcal{D}_{\text{LIG}} \to \mathcal{D}_{\text{HG}}$ which transforms a well-formed LIG derivation into a well-formed HG derivation while preserving its yield and category.

To show that $\mathcal{G}$ and $\mathcal{G}'$ are derivationally equivalent, I show that the set of well-formed derivations of $\mathcal{G}$ are bijective to the derivations of $\mathcal{G}'$ via **lh**, and since **lh** preserves yields, so too are the sets of well-formed derivations for a given string. The diagram [4] below commutes, that is, $\textbf{lh}^*(\mathcal{D}(\mathcal{G})) = \mathcal{D}(\text{LH}(\mathcal{G})) = \mathcal{D}(\mathcal{G}')$.

$$\begin{array}{ccc} \text{LIG} & \xrightarrow{\text{LH}} & \text{HG} \\ \downarrow{\scriptstyle \mathcal{D}} & & \downarrow{\scriptstyle \mathcal{D}} \\ \mathcal{P}(\mathcal{D}_{\text{LIG}}) & \xrightarrow{\textbf{lh}^*} & \mathcal{P}(\mathcal{D}_{\text{HG}}) \end{array}$$

The proof proceeds by showing $\textbf{lh}^*(\mathcal{D}(\mathcal{G})) \subseteq \mathcal{D}(\text{LH}(\mathcal{G}))$, and then the reverse, relying on the fact that **lh** has an inverse $\textbf{lh}^{-1}$.

3.1 Grammar translation

Theorem 2. For any LIG $\mathcal{G}$, we can construct an HG $\text{LH}(\mathcal{G}) = \mathcal{G}'$ s.t. $\mathcal{G}$ and $\mathcal{G}'$ are derivationally equivalent, i.e. $\mathcal{D}(\mathcal{G})$ and $\mathcal{D}(\mathcal{G}')$ contain the same number of derivations per string.

This theorem will be proved over the next few subsections. First, I present the construction of $\text{LH}(\mathcal{G})$. Given an LIG $\mathcal{G} = (V_{\text{N}}, V_{\text{T}}, V_{\text{I}}, S, P)$, construct HG $\mathcal{G}' = \text{LH}(\mathcal{G}) = (V'_{\text{N}}, V_{\text{T}}, S, P')$ such that

- $V'_{\text{N}} = V_{\text{N}} \cup \underline{V_{\text{N}}} \cup \{\frac{A\eta}{B} \mid A \in V_{\text{N}}, B \in V_{\text{N}} \cup \underline{V_{\text{N}}}, \eta \in V_{\text{I}} \cup \{\epsilon\}\}$, where $\underline{V_{\text{N}}} = \{\underline{A} \mid A \in V_{\text{N}}\}$
- For each rule in P: $A[\zeta] \to A_L\, A_c[\eta]\, A_R$; add to P': $\frac{A\zeta}{\underset{\sim}{B}} \xrightarrow{C_c} A_L\, \frac{A_c\eta}{B}\, A_R$, for all $B \in V_{\text{N}}$, where $\underset{\sim}{B}$ represents two separate instances of the rule, one with B and one with $\underline{B}$.

[4] $\mathcal{P}$ denotes the powerset.

- For each rule in P: $A[] \to w$; add to P': $\underset{\sim}{A} \to \epsilon_{\uparrow} w$.
- Add these HG rules to P', for all $A, B, D \in V_{\text{N}}$ and $\eta \in V_{\text{I}}$:
  - $\frac{A\eta}{B} \xrightarrow{W} \frac{A\epsilon}{\underline{D}}\ \frac{D\eta}{\underline{B}}$ COMP(OSE)
  - $A \xrightarrow{W} \frac{A\epsilon}{\underline{B}}\ \underline{B}$ FILL
  - $\frac{A\epsilon}{A} \to \epsilon_{\uparrow}\epsilon$ EMPTY

The construction of the HG $\mathcal{G}'$ is similar to that given in Vijay-Shanker and Weir (1994), with several modifications to ensure derivational equivalence. The additional nonterminals $\underset{\sim}{V_{\text{N}}}$ indicate a subtree whose root is a rule originating in the LIG (either a Leaf or Branch rule). This will prevent spurious applications of the HG structural rules (COMP, FILL, and EMPTY).

Additionally, there are additional nonterminals written as fractions: $\frac{A\eta}{B}$, presented in Vijay-Shanker and Weir (1994) as (A, B, η). This HG nonterminal corresponds to an LIG tree context that would have category $A[\eta\sigma]$, if the gap was filled in with a tree with category $B[\sigma]$. Combining these fraction nonterminals with Wrap rules allows the HG to emulate the stack actions of the LIG.

3.2 The lh and lch translation functions

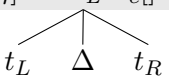
Let $\mathcal{G}$ be an arbitrary LIG, and $\mathcal{G}' = \text{LH}(\mathcal{G})$. Then $\mathbf{lh} : \mathcal{D}(\mathcal{G}) \to \mathcal{D}(\mathcal{G}')$ converts well-formed LIG derivations to well-formed HG derivations. Since $\mathbf{pt}$ can produce pairs of LIG contexts and Leaf rules from trees, I define $\mathbf{lh}$ on some derivations with the tail of the spine already separated for convenience. After defining $\mathbf{lh}$, I prove that the HG derivation produced matches the category of the source LIG derivation.

- $\mathbf{lh}(B[] \to w) = B \to \epsilon_{\uparrow} w$
- $\mathbf{lh}(\Gamma\{B[] \to w\}) = \dfrac{A \xrightarrow{W} \frac{A\epsilon}{\underline{B}}\ \underline{B}}{\mathbf{lch}(\Gamma, \underline{B}) \qquad \underline{B} \to \epsilon_{\uparrow} w}$,

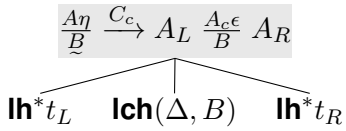
  where $A \to \ldots$ is at the root of Γ.

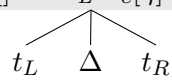
A separate function to translate LIG tree contexts combined with an LIG category to HG derivations is given as $\mathbf{lch} : \mathcal{C}(\mathcal{G}) \times \underset{\sim}{V_{\text{N}}} \to \mathcal{D}(\mathcal{G}')$.

- $\mathbf{lch}(\Box, B) = \frac{B\epsilon}{B} \to \epsilon_{\uparrow}\epsilon$
- If $\Gamma = A[\eta] \to A_L\ A_c[]\ A_R$ and $\eta \in V_{\text{I}} \cup \{\epsilon\}$,

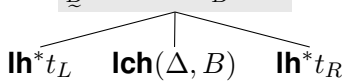
  then $\mathbf{lch}(\Gamma, \underset{\sim}{B}) =$

  
- If $\Gamma = A[] \to A_L\ A_c[\eta]\ A_R$ and $\eta \in V_{\text{I}}$, then

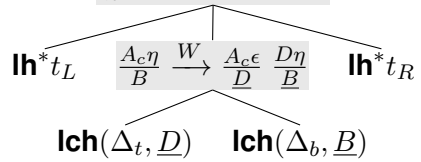
  there are two subcases. Decompose $\mathbf{flagseq}(\Gamma)$ into $({}_{\eta}u)_{\eta}v$, where u, v are Dyck words. Let Δ_b be the context at the node corresponding to $)_{\eta}$ in the decomposition, and let $\Delta_t = \Delta\{\Delta_b \mapsto \Box\}$.

  (a) If $\Delta_t = \Box$ (i.e. the root of Δ is a Pop η rule), then $\mathbf{lch}(\Gamma, \underset{\sim}{B}) =$

  (b) Otherwise,

  $\mathbf{lch}(\Gamma, \underset{\sim}{B}) =$

  where $D[\eta] \to \ldots$ is the top node of Δ_b.

Note that the $\mathbf{flagseq}$ of Δ_t is exactly u, a Dyck word, and that of Δ_b is exactly $)_{\eta}v$, with v a Dyck word. Since the root of Δ_b has $)_{\eta}$ as its flag, the root node of Δ_b is the Pop η rule that corresponds to the Push η rule at the root of Γ.

Theorem 3. $\mathbf{lh}$ preserves categories $\mathbf{cat}(\mathbf{lh}(d))[] = \mathbf{cat}(d)$, and thus the image of $\mathbf{lh}$ contains only well-formed HG derivations.

Proof. The proof will proceed by induction on these two statements:

(i) for $d \in \mathcal{D}(\text{LIG})$, if $\mathbf{cat}(d) = A[]$, then $\mathbf{cat}(\mathbf{lh}(d)) = A$

(ii) for $\Gamma \in \mathcal{C}(\text{LIG})$, if $\mathbf{catc}(\Gamma, B) = A[\eta]$ $(\eta \in V_I \cup \{\epsilon\})$ and $\mathbf{lch}(\Gamma, B)$ is defined, then $\mathbf{cat}(\mathbf{lch}(\Gamma, \underset{\sim}{B})) = \frac{A\eta}{\underset{\sim}{B}}$

Base cases:

(i) $\mathbf{cat}(B[] \to w) = B[]$
$\mathbf{cat}(\mathbf{lh}(B[] \to w)) = \mathbf{cat}(B \to \epsilon_{\uparrow} w) = B$

(ii) $\mathbf{catc}(\Box, B) = B[]$
$\mathbf{cat}(\mathbf{lch}(\Box, B)) = \mathbf{cat}(\frac{B\epsilon}{B} \to \epsilon_{\uparrow}\epsilon) = \frac{B\epsilon}{B}$

Inductive cases:

(i) Let $d = \Gamma\{B[] \to w\}$, and $\mathbf{cat}(d) = A[] = \mathbf{catc}(\Gamma, B)$.

Let $d' = \mathbf{lh}(d) = \dfrac{A \xrightarrow{W} \frac{A\epsilon}{\underline{B}}\ \underline{B}}{\mathbf{lch}(\Gamma, \underline{B}) \quad \underline{B} \to \epsilon_{\uparrow} w}$.

By the IH (ii), since $\mathbf{catc}(\Gamma, B) = A[]$, $\mathbf{cat}(\mathbf{lch}(\Gamma, \underline{B})) = \frac{A\epsilon}{\underline{B}}$, so $\mathbf{cat}(d') = A$.

(ii) Let $\Gamma = \dfrac{A[\zeta] \to A_L\ A_c[\eta]\ A_R}{t_L \quad \Delta \quad t_R}$ with $\mathbf{catc}(\Gamma, B) = A[\zeta]$, $\mathbf{catc}(\Delta, B) = A_c[\eta]$.

Since $\mathbf{cat}^*(t_L) = A_L[]$, by the IH(i), $\mathbf{cat}^*(\mathbf{lh}^*(t_L)) = A_L$, and similarly for t_R. Let $d' = \mathbf{lch}(\Gamma, \underline{B})$. It follows that (ii) is true at Γ by considering two possibilities for Γ:

(a) If $\eta = \epsilon$: $d' = \dfrac{\frac{A\zeta}{\underline{B}} \xrightarrow{C} A_L\ \frac{A_c\epsilon}{B}\ A_R}{\mathbf{lh}^* t_L \quad \mathbf{lch}(\Delta, B) \quad \mathbf{lh}^* t_R}$.

Since $\mathbf{catc}(\Delta, B) = A_c[]$, by the IH $\mathbf{cat}(\mathbf{lch}(\Delta, B)) = \frac{A_c\epsilon}{B}$, so $\mathbf{cat}(d') = \frac{A\zeta}{\underline{B}}$.

(b) If $\eta \neq \epsilon$:

$d' = \dfrac{\frac{A\epsilon}{\underline{B}} \xrightarrow{C} A_L\ \frac{A_c\eta}{B}\ A_R}{\mathbf{lh}^* t_L \quad \frac{A_c\eta}{B} \xrightarrow{W} \frac{A_c\epsilon}{\underline{D}}\ \frac{D\eta}{\underline{B}} \quad \mathbf{lh}^* t_R \qquad \mathbf{lch}(\Delta_t, \underline{D}) \quad \mathbf{lch}(\Delta_b, \underline{B})}$,

where Δ_t and Δ_b are as defined above in **lch**.

As noted above, the root of Δ_b is a Pop rule $D[\eta] \to \ldots$, and the designated daughter of that node is a context whose **flagseq** is a Dyck word. Thus, by the IH (i), $\mathbf{catc}(\Delta_b, B) = D[\eta]$ and by the IH (ii) $\mathbf{cat}(\mathbf{lch}(\Delta_b, \underline{B})) = \frac{D\eta}{\underline{B}}$.

Also, as noted above, $\mathbf{flagseq}(\Delta_t)$ is a Dyck word. By the IH (ii), $\mathbf{catc}(\Delta, B) = A_c[]$ and we know $\Delta_t = \Delta\{\Delta_b \mapsto \square\}$, so $\mathbf{catc}(\Delta_t, D) = A_c[]$. By the IH (ii), $\mathbf{cat}(\mathbf{lch}(\Delta_t, \underline{D})) = \frac{A_c\epsilon}{\underline{D}}$

Thus, $\mathbf{cat}(d') = \frac{A\epsilon}{B}$. □

Corollary 4. If d is a complete well-formed derivation of LIG $\mathcal{G}$, then $\mathbf{lh}(d)$ is a complete well-formed derivation of HG $\mathcal{G}' = \text{LH}(\mathcal{G})$. This shows $\mathbf{lh}^*(\mathcal{D}(\mathcal{G})) \subseteq \mathcal{D}(\text{LH}(\mathcal{G}))$.

Proof. This follows fairly straightforwardly from the previous theorem. Given any complete well-formed derivation $d \in \mathcal{D}(\mathcal{G})$, we can easily inspect that each rule used in $\mathbf{lh}(d)$ is a rule of $P_{\mathcal{G}'}$. Furthermore, $\mathcal{G}$ and $\mathcal{G}'$ share the same start symbols S, so $\mathbf{lh}(d)$ must be complete and well-formed with respect to $\mathcal{G}'$. □

The proof that **lh** preserves yields is very similar to that of Theorem 3 and will not be provided here. Moreover, it follows as a corollary of Theorem 10.

Example 1. Here is an example of an LIG $\mathcal{G}_1$ which generates $\{ww \mid w \in \{a, b\}^*\}$. Figure 1 shows an LIG derivation $d_1 \in \mathcal{D}(\mathcal{G}_1)$ for *abbabb*, and the HG derivation $\mathbf{lh}(d_1)$, which is in $\mathcal{D}(\text{LH}(\mathcal{G}_1))$.
$\mathcal{G}_1$ = ({S,T,A,B},{a,b},{1,2},{S},P_1) where P_1 = {S[] → A S[1], S[] → B S[2], S[] → T[], T[1] → T[] A, T[2] → T[] B, T[] → ϵ, A[] → a, B[] → b}

3.3 Derivational equivalence

In this subsection, I complete the proof of Theorem 2 by proving that $\mathcal{D}(\text{LH}(\mathcal{G})) \subseteq \mathbf{lh}^*(\mathcal{D}(\mathcal{G}))$. To do that, I will show that **lh** and **lch** are invertible, with their inverses identified below. $\mathbf{lh}^{-1} : \mathcal{D}(\mathcal{G}') \to \mathcal{C}(\mathcal{G}) \times P_{\text{L}}(\mathcal{G})$ is defined over HG derivations with a simple category A, and $\mathbf{lch}^{-1} : \mathcal{D}(\mathcal{G}') \to \mathcal{C}(\mathcal{G}) \times \underset{\sim}{V_{\text{N}}}$ over those with a fractional category $\frac{A\eta}{\underline{B}}$.

(i) $\mathbf{lh}^{-1}(B \to \epsilon_{\uparrow} w) = (\square, B[] \to w)$

(ii) if $d' = \dfrac{A \xrightarrow{W} \frac{A\epsilon}{\underline{B}}\ \underline{B}}{\Gamma \quad \underline{B} \to \epsilon_{\uparrow} w}$, then $\mathbf{lh}^{-1}(d') = (\mathbf{lch}^{-1}(\Gamma), B[] \to w)$

(iii) $\mathbf{lch}^{-1}(\frac{B\epsilon}{B} \to \epsilon_{\uparrow}\epsilon) = (\square, B)$

(iv) if $d' = \dfrac{\frac{A\eta}{\underline{B}} \xrightarrow{C_c} A_L\ \frac{A_c\epsilon}{B}\ A_R}{t_L \quad t_c \quad t_R}$, then

$\mathbf{lch}^{-1}(d') = \left(\dfrac{A[\eta] \to A_L\ A_c[]\ A_R}{\mathbf{lh}^{-1*} t_L \quad \mathbf{lch}^{-1} t_c \quad \mathbf{lh}^{-1*} t_R}, \underline{B} \right)$

(v) if $d' = \dfrac{\frac{A\epsilon}{\underline{B}} \xrightarrow{C} A_L\ \frac{A_c\eta}{B}\ A_R}{t_L \quad \frac{A_c\eta}{B} \xrightarrow{W} \frac{A_c\epsilon}{\underline{D}}\ \frac{D\eta}{\underline{B}} \quad t_R \qquad t_t \quad t_b}$, then

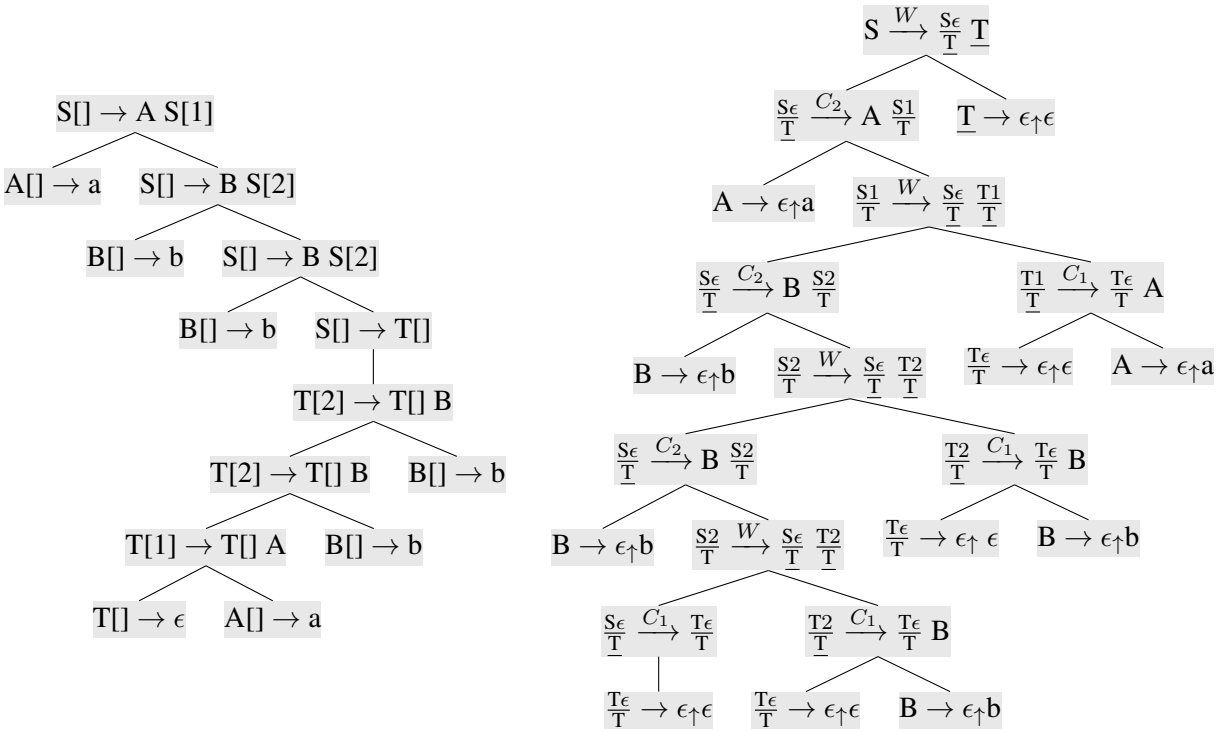


Figure 1: Derivation of *abbabb* in LIG $\mathcal{G}_1$ (left) and HG $\text{LH}(\mathcal{G}_1)$ (right)

$$\mathbf{lch}^{-1}(d') = \left(\begin{array}{c} A[] \to A_L\, A_c[\eta]\, A_R \\ \mathbf{lh}^{-1*}t_L \quad (\mathbf{lch}_0^{-1}t_t)\{\mathbf{lch}_0^{-1}t_b\} \quad \mathbf{lh}^{-1*}t_R \end{array}, \underset{\sim}{B} \right),$$

where $\mathbf{lch}_0^{-1}$ denotes the first element returned by $\mathbf{lch}^{-1}$.

It is not hard to verify that these functions composed with **lh** and **lch**, respectively, return the identity. I show this is the case for (v):

$$\mathbf{lch}(\mathbf{lch}^{-1}(d')) = \begin{array}{c} \frac{A\epsilon}{\underset{\sim}{B}} \xrightarrow{C} A_L\, \frac{A_c\eta}{B}\, A_R \\ \mathbf{lh}^*(\mathbf{lh}^{-1*}t_L) \quad \frac{A_c\eta}{B} \xrightarrow{W} \frac{A_c\epsilon}{\underline{D}}\ \frac{D\eta}{\underline{B}} \quad \mathbf{lh}^*(\mathbf{lh}^{-1*}t_R) \\ \mathbf{lch}(\mathbf{lch}^{-1}t_t) \quad \mathbf{lch}(\mathbf{lch}^{-1}t_b) \end{array}$$

By induction, $\mathbf{lh}^*(\mathbf{lh}^{-1*}t_{L/R}) = t_{L/R}$ and $\mathbf{lch}(\mathbf{lch}^{-1}t_{t/b}) = t_{t/b}$, and we know that $(\mathbf{lch}^{-1}t_t)\{\mathbf{lch}^{-1}t_b\}$ will be split by **lch** into $\mathbf{lch}^{-1}t_t$ and $\mathbf{lch}^{-1}t_b$. This is because $\mathbf{cat}(t_t) = \frac{A_c\epsilon}{\underline{D}}$, so $\mathbf{cat}(\mathbf{lch}^{-1}t_t, \underline{D}) = A_c[]$; thus $\mathbf{lch}^{-1}t_t$ is a context whose **flagseq** will be a Dyck word.

If $\Gamma = A[\zeta] \to A_L\, A_c[\eta]\, A_R$, then

$$\begin{array}{ccc} t_L & \Delta & t_R \end{array}$$

$$\mathbf{lch}^{-1}(\mathbf{lch}(\Gamma, \underset{\sim}{B})) = \left(\begin{array}{c} A[] \to A_L\, A_c[\eta]\, A_R \\ \mathbf{lh}^{-1}(\mathbf{lh}^*t_L) \quad \Delta_x \quad \mathbf{lh}^{-1}(\mathbf{lh}^*t_R) \end{array}, \underset{\sim}{B} \right), \text{ where}$$

$\Delta_x = (\mathbf{lch}_0^{-1}(\mathbf{lch}(\Delta_t, \underline{D})))\{\mathbf{lch}_0^{-1}(\mathbf{lch}(\Delta_b, \underline{B}))\}$. By induction, we know $\Delta_x = \Delta_t\{\Delta_b\} = \Delta$, and also that $\mathbf{lh}^{-1*}(\mathbf{lh}^*t_{L/R}) = t_{L/R}$; therefore, $\mathbf{lch}^{-1}(\mathbf{lch}(\Gamma, \underset{\sim}{B})) = (\Gamma, \underset{\sim}{B})$.

It is important to note that for every derivation in $\mathcal{D}(\mathcal{G}')$, either $\mathbf{lh}^{-1}$ or $\mathbf{lch}^{-1}$ is defined. To see why, consider all the kinds of rules that could be at the root of an arbitrary derivation d' in $\mathcal{D}(\mathcal{G}')$. If the root of d' is a Leaves or Fill rule, $\mathbf{lh}^{-1}(d)$ is defined in cases (i) and (ii). If the root of d' is an Empty rule or a Concat rule which did not come from an LIG Push rule, $\mathbf{lch}^{-1}(d)$ is defined in cases (iii) and (iv). The only time a Compose rule appears in an HG derivation is when it is dominated by a Concat-from-Push rule, which is defined in (v).

Thus, every $d' \in \mathcal{D}(\text{LH}(\mathcal{G}))$ can be mapped to a derivation in $\mathcal{D}(\mathcal{G})$. Since $\mathbf{lh}^{-1*}(\mathcal{D}(\text{LH}(\mathcal{G}))) \subseteq \mathcal{D}(\mathcal{G})$, this proves that $\mathcal{D}(\text{LH}(\mathcal{G})) \subseteq \mathbf{lh}^*(\mathcal{D}(\mathcal{G}))$. And since $\mathcal{D}(\mathcal{G})$ and $\mathcal{D}(\text{LH}(\mathcal{G}))$ are bijective and the bijection **lh** preserves yields, each derivation set contains the same number of derivations per string generated. This completes the proof of Theorem 2, that LH constructs derivationally equivalent HGs from LIGs.

4 Derivational equivalence: HG to LIG

In this section, I will prove the reverse of the previous section: that for every HG, there is a derivationally equivalent LIG. The proof will proceed with the same structure.

First, given a source HG $\mathcal{G}$, I will define the resulting LIG $\text{HL}(\mathcal{G}) = \mathcal{G}'$. Next, I define a function $\mathbf{hl}_\alpha : \mathcal{D}_{\text{HG}} \to \mathcal{D}_{\text{LIG}}$ which transforms a well-formed HG derivation into a well-formed LIG derivation while preserving its yield and category.

To show that $\mathcal{G}$ and $\mathcal{G}'$ are derivationally equivalent, I show that the set of well-formed derivations of $\mathcal{G}$ stand in correspondence to the derivations of $\mathcal{G}'$ via **hl**. I will show that the diagram below commutes, that is, $\mathbf{hl}^*_\alpha(\mathcal{D}(\mathcal{G})) = \mathcal{D}(\text{HL}(\mathcal{G})) = \mathcal{D}(\mathcal{G}')$.

$$\begin{array}{ccc} \text{HG} & \xrightarrow{\text{HL}} & \text{LIG} \\ \downarrow \mathcal{D} & & \downarrow \mathcal{D} \\ \mathcal{P}(\mathcal{D}_{\text{HG}}) & \xrightarrow{\mathbf{hl}^*_\alpha} & \mathcal{P}(\mathcal{D}_{\text{LIG}}) \end{array}$$

The proof proceeds by showing $\mathbf{hl}^*_\alpha(\mathcal{D}(\mathcal{G})) \subseteq \mathcal{D}(\text{HL}(\mathcal{G}))$, and then the reverse. I show that $\mathbf{hl}_\alpha$ has an inverse $\mathbf{hl}^{-1}_\alpha$ and is thus bijective, so there is a one-to-one correspondence between HG and LIG derivations.

4.1 Grammar translation

Theorem 5. For any HG $\mathcal{G}$, there is an LIG $\mathcal{G}' = \text{HL}(\mathcal{G})$ s.t. $\mathcal{G}$ and $\mathcal{G}'$ are derivationally equivalent.

Again, this theorem will be proven through this section. I first present the construction of the equivalent LIG. Given an HG $\mathcal{G} = (V_{\text{N}}, V_{\text{T}}, S, P)$, construct LIG $\mathcal{G}' = \text{HL}(\mathcal{G}) = (V'_{\text{N}}, V_{\text{T}}, V_{\text{N}}, S, P')$ such that:

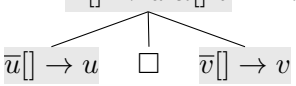
- $V'_{\text{N}} = V_{\text{N}} \sqcup \{\alpha\} \sqcup \overline{V_{\text{T}}}$, where $\overline{V_{\text{T}}} = \{\overline{u} \mid u \in V_{\text{T}}\}$
- For each rule in P of the form $A \xrightarrow{C_c} A_L\, A_c\, A_R$, add to P' the LIG rule $A[] \to A_L\, A_c[]\, A_R$.
- For each rule in P: $A \xrightarrow{W} B\, D$, add to P': $A[] \to B[D]$ and $\alpha[D] \to D[]$.
- For each rule in P: $A \to u_{\uparrow} v$, add to P': $A[] \to \overline{u}\ \alpha[]\ \overline{v}$, $\overline{u}[] \to u$, and $\overline{v}[] \to v$.
- Add to P': $\alpha[] \to \epsilon$. ALPHA

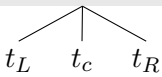
The constructed LIG is similar to the original HG, but there are several new nonterminals: $\overline{V_{\text{T}}}$, which are solely to introduce the corresponding terminals; and α, which serves as a gap in the LIG tree structure, allowing the LIG to emulate the bipartite nature of HG strings.

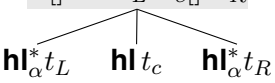
The indices of the LIG are just the nonterminals of the HG. Intuitively, if an HG rule wraps B around D, the corresponding LIG rule will push D on the stack, to construct a D subtree inside the B subtree at the bottom of its spine. It is in this way that the constructed LIG can use the stack to emulate the HG Wrap rules.

4.2 The hl translation function

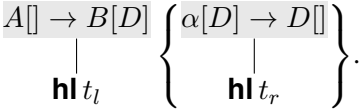
hl : $\mathcal{D}_{\text{HG}} \to \mathcal{C}_{\text{LIG}}$ converts HG trees into LIG contexts. $\mathbf{hl}_\alpha$ fills the context resulting from **hl** with the rule $\alpha[] \to \epsilon$ to produce a well-formed LIG derivation which is equivalent to the HG derivation.

- $\mathbf{hl}(A \to u_{\uparrow} v) =$ 

.
- If $d =$ 

then $\mathbf{hl}(d) =$ 

.
- If $d =$ 

$\mathbf{hl}(d) =$ 

.
- $\mathbf{hl}_\alpha(d) = \mathbf{hl}(d)\{\alpha[] \to \epsilon\}$

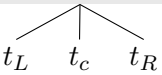
The **hl** translation function preserves categories and well-formedness. First, I introduce a lemma about context composition.

Lemma 6. Composition lemma for categories Let $\Gamma, \Delta \in \mathcal{C}(\text{LIG})$. Let $\sigma_1, \sigma_2 \in V_I^*$. If $\mathbf{catc}(\Gamma, B) = A[\sigma_1]$ and $\mathbf{catc}(\Delta, D) = B[\sigma_2]$, then $\mathbf{catc}(\Gamma\{\Delta\}, D) = A[\sigma_1\sigma_2]$.

Proof. Induction over the structure of Γ. □

Theorem 7. **hl** preserves categories
If $d \in \mathcal{D}_{\text{HG}}$, then $\mathbf{hl}_\alpha(d)$ is a well-formed LIG derivation. In addition, **hl** preserves categories to the extent that $\mathbf{catc}(\mathbf{hl}(d), \alpha) = \mathbf{cat}(d)[]$.

Proof. Induction will proceed on the statement $\textbf{catc}(\textbf{hl}(d), \alpha) = \textbf{cat}(d)[]$.
Base case:
$d = A \to u_{\uparrow} v$. $\textbf{cat}(d) = A$.
$\textbf{catc}(\textbf{hl}(d), \alpha) = A[]$.
Inductive cases:
(i) If $d = A \xrightarrow{C_c} A_L\, A_c\, A_R$ and $\textbf{cat}(d) = A$,



then $\textbf{cat}(t_{L/c/R}) = A_{L/c/R}$. By induction, $\textbf{catc}(\textbf{hl}(t_{L/c/R}), \alpha) = \textbf{cat}(t_{L/c/R})[] = A_{L/c/R}[]$. So, $\textbf{catc}(\textbf{hl}(d), \alpha) = A[]$.
(ii) If $d = A \xrightarrow{W} B\, D$, and $\textbf{cat}(d) = A$, then

$\textbf{cat}(t_l) = B$ and $\textbf{cat}(t_r) = D$. By induction, $\textbf{catc}(\textbf{hl}(t_l), \alpha) = \textbf{cat}(t_l)[] = B[]$, and $\textbf{catc}(\textbf{hl}(t_r), \alpha) = \textbf{cat}(t_r)[] = D[]$.
Let $\Gamma = \textbf{hl}(t_l)$ and $\Delta = \alpha[D] \to D[]$, so

$\textbf{catc}(\Delta, \alpha) = \alpha[D]$ and $\textbf{hl}(d) = A[] \to B[D]$.

Applying Lemma 6 where $\textbf{catc}(\Gamma, \alpha) = B[]$ and $\textbf{cat}(\Delta, \alpha[]) = \alpha[D]$ yields $\textbf{catc}(\Gamma\{\Delta\}, \alpha) = B[D]$. Thus, $\textbf{catc}(\textbf{hl}(d), \alpha[]) = A[]$. □

Corollary 8. If d is a complete well-formed derivation of HG $\mathcal{G}$, then $\textbf{hl}_\alpha(d)$ is a complete well-formed derivation of LIG $\mathcal{G}' = \text{HL}(\mathcal{G})$. This shows $\textbf{hl}^*_\alpha(\mathcal{D}(\mathcal{G})) \subseteq \mathcal{D}(\text{HL}(\mathcal{G}))$.

Proof. This follows fairly straightforwardly from the previous theorem. Given any complete well-formed derivation $d \in \mathcal{D}(\mathcal{G})$, we can easily inspect that each rule used in $\textbf{hl}_\alpha(d)$ is a rule of $P_{\mathcal{G}'}$. Furthermore, $\mathcal{G}$ and $\mathcal{G}'$ share the same start symbols S, so $\textbf{hl}_\alpha(d)$ must be complete and well-formed with respect to $\mathcal{G}'$. □

Again, the proof of yield preservation is not difficult, and would follow the same structure as Theorem 7. It also follows as a corollary of Theorem 12.

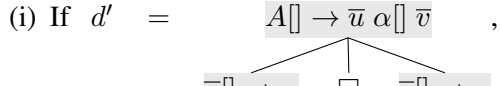
Example 2. Figure 2 shows an example of an HG $\mathcal{G}_2$ which generates $\{ww \mid w \in \{a, b\}^*\}$, and the equivalent LIG $\mathcal{G}'_2$ resulting from HL. The figure also shows a $\mathcal{G}_2$ derivation for *abbabb*, and the $\mathcal{G}'_2$ derivation resulting from **hl**.
$\mathcal{G}_2 =$ ({S,T,U,A,B}, {a,b}, {S},P_2), where $P_2 =$ {S $\xrightarrow{C_2}$ A T, T $\xrightarrow{W}$ S A, S $\xrightarrow{C_2}$ B U, U $\xrightarrow{W}$ S B, A $\to \epsilon_{\uparrow}$a, B $\to \epsilon_{\uparrow}$b}

4.3 Inverting hl

In this subsection, I complete the proof of Theorem 5 by proving that $\mathcal{D}(\text{HL}(\mathcal{G})) \subseteq \textbf{hl}^*_\alpha(\mathcal{D}(\mathcal{G}))$. To do that, I will show that $\textbf{hl}_\alpha$ has an inverse, $\textbf{hl}^{-1}_{-\alpha}$, and use it to show that $\textbf{hl}^{-1*}_{-\alpha}(\mathcal{D}(\text{HL}(\mathcal{G}))) \subseteq \mathcal{D}(\mathcal{G})$.

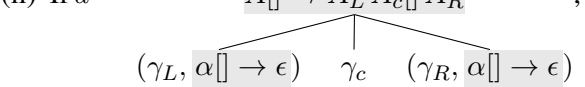
Since the last step of $\textbf{hl}_\alpha$ is to add $\alpha[] \to \epsilon$, the first step of $\textbf{hl}^{-1}_{-\alpha}$ is to remove it: $\textbf{hl}^{-1}_{-\alpha}(d') = \textbf{hl}^{-1}(d'\{\alpha[] \to \epsilon \mapsto \Box\})$. Then, $\textbf{hl}^{-1}$ is the inverse of **hl**. $\textbf{hl}^{-1}$ can be spelled out as:

(i) If $d' =$

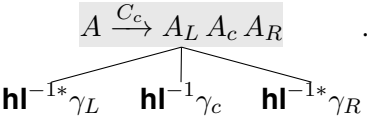


, then $\textbf{hl}^{-1}(d') = A \to u_{\uparrow} v$.

(ii) If $d' =$

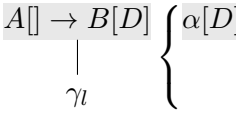


, then $\textbf{hl}^{-1}(d') =$

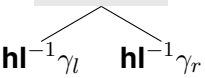


.

(iii) If $d' =$



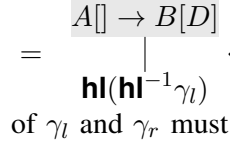
, then $\textbf{hl}^{-1}(d') =$



. That is, d' is a LIG context whose root node is a Push D rule; this means somewhere along the spine must be the corresponding Pop D rule because $\textbf{flagseq}(d')$ is a Dyck word. The corresponding Pop rule is found in the same way as in the **lh** translation function.

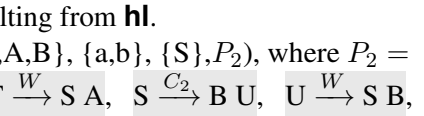
It is not difficult to verify that $\textbf{hl} \circ \textbf{hl}^{-1} = \textbf{id}$ and $\textbf{hl}^{-1} \circ \textbf{hl} = \textbf{id}$; I show case (iii) below:

$\textbf{hl}(\textbf{hl}^{-1}(d')) =$



.
The **flagseq** of γ_l and γ_r must be Dyck words as well, and γ_l and γ_r have an α-shaped gap, meaning $\textbf{hl}^{-1}$ is defined over them, and by induction, $\textbf{hl}(\textbf{hl}^{-1}\gamma_{l/r}) = \gamma_{l/r}$

For $\textbf{hl}^{-1} \circ \textbf{hl}$: If $d = A \xrightarrow{W} B\, D$, then $\textbf{hl}(d) =$



. The intuition here is that since t_l is a well-formed HG derivation, the

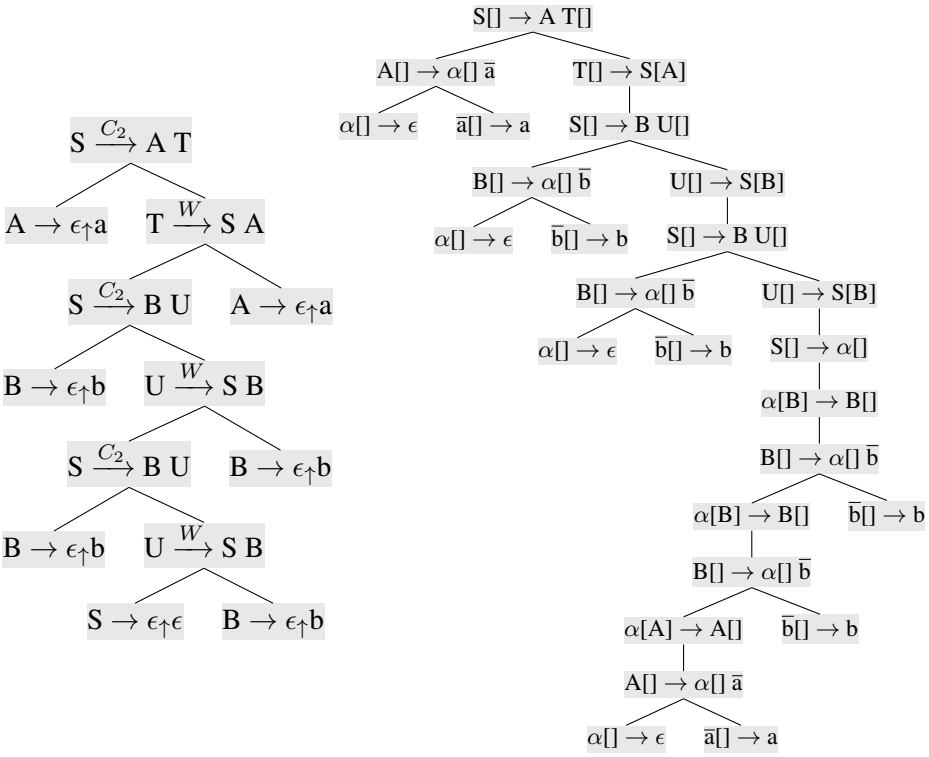


Figure 2: Derivation of *abbabb* in HG $\mathcal{G}_2$ (left) and LIG $\text{HL}(\mathcal{G}_2)$ (right)

flagseq of $\mathbf{hl}(t_l)$ will be a Dyck word, so $\mathbf{hl}^{-1}$ will split $\mathbf{hl}(d)$ at the Pop D corresponding to the Push D at the root of $\mathbf{hl}(d)$. Thus, $\mathbf{hl}^{-1}(\mathbf{hl}\, d) = d$.

$$\begin{array}{c} A \xrightarrow{W} B\,D \\ \mathbf{hl}^{-1}\,\mathbf{hl}(t_l) \quad \mathbf{hl}^{-1}(\mathbf{hl}\, t_r) \end{array}$$

Combining the fact that $\mathbf{hl}$ and $\mathbf{hl}^{-1}$ are inverses with Theorem 7 results in the category preservation of $\mathbf{hl}^{-1}_{-\alpha}$: $\mathbf{cat}(d') = \mathbf{cat}(\mathbf{hl}^{-1}_{-\alpha} d')[]$. Given any (complete) well-formed derivation $d' \in \mathcal{D}(\text{HL}(\mathcal{G}))$, $\mathbf{hl}^{-1*}_{-\alpha}(d')$ will be a (complete) well-formed derivation in $\mathcal{D}(\mathcal{G})$. Therefore, $\mathcal{D}(\mathcal{G})$ and $\mathcal{D}(\text{HL}(\mathcal{G}))$ are bijective. Additionally, since $\mathbf{hl}_{\alpha}$ preserves yields, the number of derivations per string is the same in $\mathcal{G}$ and $\text{HL}(\mathcal{G})$. This completes the proof of Theorem 5, that HL constructs derivationally equivalent LIGs from HGs.

5 Interpretational equivalence

Let us now consider the ways grammars can match semantics to syntactic structures, following the approach taken by Miller (1999). An interpreted grammar consists of a grammar paired with a rule interpretation function: $(\mathcal{G}, \mu)$, where μ is a function from rules of $\mathcal{G}$ to functions over the interpretation domain M. The interpretation domain M can be a set of any kind of elements: lambda terms, strings, and even trees. Two interpreted grammars are *interpretationally equivalent* iff they generate the same set of string-meaning pairs using their respective interpretation functions. And two formalisms $\mathcal{F}_1$ and $\mathcal{F}_2$ are *interpretationally equivalent* iff for each interpreted grammar $\mathcal{G}_1 \in \mathcal{F}_1$ with interpretation function μ_1, there is some $(\mathcal{G}_2, \mu_2)$ (with $\mathcal{G}_2 \in \mathcal{F}_2$) that is interpretationally equivalent to $(\mathcal{G}_1, \mu_1)$, and vice versa.

Formally, for a grammar $\mathcal{G}$, let μ be a function which assigns interpretations to rules of $\mathcal{G}$: it maps each rank 0 rule to an element of M, and each rank n rule to an n-ary function $M^n \to M$. Then, let $[\![\cdot]\!]^{\mu}$ be the homomorphic extension of the rule interpretation function μ, defined over $\mathcal{D}(\mathcal{G})$: for β rank 0, $[\![\beta]\!]^{\mu} = \mu(\beta)$, and for m rank n,

$$\left[\!\!\left[\begin{array}{c} m \\ t_1 \quad \ldots \quad t_n \end{array}\right]\!\!\right]^{\mu} = \mu(m)[\![t_1]\!]^{\mu} \ldots [\![t_n]\!]^{\mu}.$$

Additionally, define the interpretation function $[\![\cdot]\!]^{\mu}$ over tree contexts in such a way that if $d = \Gamma\{\beta\}$, then $[\![d]\!]^{\mu} = [\![\Gamma]\!]^{\mu}(\mu(\beta))$.

- $[\![\Box]\!]^{\mu} = \lambda x.\, x$
- If $\Gamma = \begin{array}{c} m \\ t_L \quad \gamma_c \quad t_R \end{array}$, then

$$[\![\Gamma]\!]^{\mu} = \lambda x.\, \mu(m)[\![t_L]\!]^{\mu}([\![\gamma_c]\!]^{\mu}(x))[\![t_R]\!]^{\mu}.$$

Lemma 9. Composition lemma for interpretations
The interpretation of the composition of two tree contexts Γ, Δ is function composition:
$[\![\Gamma\{\Delta\}]\!]^\mu = \lambda x.\, [\![\Gamma]\!]^\mu([\![\Delta]\!]^\mu x)$

Proof. Induction over the structure of Γ. □

The proofs to show interpretational equivalence between LIG and HG will proceed as follows: for any LIG $\mathcal{G}$ equipped with an arbitrary rule interpretation function μ, I will show how to construct a rule interpretation function μ' for the HG $\text{LH}(\mathcal{G})$, such that for all $d \in \mathcal{D}(\mathcal{G})$, $[\![d]\!]^\mu = [\![\mathbf{lh}(d)]\!]^{\mu'}$. In essence, in addition to translating grammars to grammars with LH, and derivations to derivations with **lh**, we can also translate interpretation functions to interpretation functions. I will also show the same for the reverse translation: how to construct an function to interpret derivations resulting from **hl**, in grammars resulting from HL.

5.1 LIG to HG

Theorem 10. **lh** preserves interpretations
Given an LIG $\mathcal{G}$ with interpretation function μ, we can construct an HG interpretation function μ' for $\mathcal{G}' = \text{LH}(\mathcal{G})$ such that for every derivation $d \in \mathcal{D}(\mathcal{G})$, $[\![d]\!]^\mu = [\![\mathbf{lh}(d)]\!]^{\mu'}$.

- $\mu'(B \to \epsilon_\uparrow w) = \mu(B[] \to w)$
- If $m = A[\zeta] \to A_L\, A_c[\eta]\, A_R$
  and $m' = \frac{A\zeta}{\underset{\sim}{B}} \xrightarrow{C_i} A_L\, \frac{A_c\eta}{B}\, A_R$,
  then $\mu'(m') = \lambda t_L t_c t_R x.\, \mu m t_L(t_c x) t_R$
- $\mu'(\frac{A\eta}{B} \xrightarrow{W} \frac{A\epsilon}{\underline{D}}\ \frac{D\eta}{\underline{B}}) = \lambda xyz.\, x(yz)$ COMP
- $\mu'(A \xrightarrow{W} \frac{A\epsilon}{\underline{B}}\ \underline{B}) = \lambda xy.\, xy$ FILL
- $\mu'(\frac{A\epsilon}{A} \to \epsilon_\uparrow\epsilon) = \lambda x.\, x$ EMPTY

Proof. We want to show that both **lh** and **lch** preserve interpretations. We prove this by induction on both of these statements:

- $\forall d \in \mathcal{D}(\mathcal{G}).[\![d]\!]^\mu = [\![\mathbf{lh}(d)]\!]^{\mu'}$
- $\forall \Gamma \in \mathcal{C}(\mathcal{G}), B \in \underset{\sim}{V_N}.[\![\Gamma]\!]^\mu = [\![\mathbf{lch}(\Gamma, B)]\!]^{\mu'}$ whenever $\mathbf{lch}(\Gamma, B)$ is defined

Base cases:

(i) If $d = B[] \to u$, then $[\![d]\!]^\mu = \mu(B[] \to u)$.
$[\![\mathbf{lh}(d)]\!]^{\mu'} = [\![B \to \epsilon_\uparrow u]\!]^{\mu'} = \mu(B[] \to u)$.

(ii) If $\Gamma = \square$, then $[\![\Gamma]\!]^\mu = \lambda x.\, x$.
$[\![\mathbf{lch}(\Gamma, B)]\!]^{\mu'} = [\![\frac{B\epsilon}{B} \to \epsilon_\uparrow\epsilon]\!]^{\mu'} = \lambda x.\, x$

Induction:

(i) If $d = \Gamma\{\beta\}$, then $[\![d]\!]^\mu = [\![\Gamma]\!]^\mu(\mu(\beta))$. By the inductive hypothesis, for any B, $[\![\Gamma]\!]^\mu = [\![\mathbf{lch}(\Gamma, B)]\!]^{\mu'}$, where defined.

$$
\begin{aligned}
&[\![\mathbf{lh}(d)]\!]^{\mu'} \\
&= \mu'(\text{FILL})([\![\mathbf{lch}(\Gamma, B)]\!]^{\mu'})([\![\mathbf{lh}(\beta)]\!]^{\mu'}) \\
&= [\![\mathbf{lch}(\Gamma, B)]\!]^{\mu'}([\![\mathbf{lh}(\beta)]\!]^{\mu'}) \\
&= [\![\Gamma]\!]^\mu(\mu(\beta))
\end{aligned}
$$

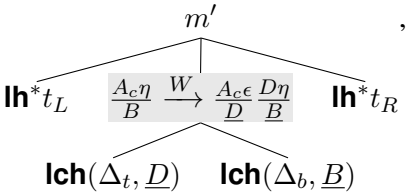
(ii) If $\Gamma =$  and

$m = A[\zeta] \to A_L A_c[\eta] A_R$, then
$[\![\Gamma]\!]^\mu = \lambda x.\, \mu(m)[\![t_L]\!]^\mu([\![\Delta]\!]^\mu(x))[\![t_R]\!]^\mu$.

By induction, $[\![t_{L/R}]\!]^\mu = [\![\mathbf{lh}(t_{L/R})]\!]^{\mu'}$ and $[\![\Delta]\!]^\mu = [\![\mathbf{lch}(\Delta, B)]\!]^{\mu'}$ for any B s.t $\mathbf{lch}(\Delta, B)$ is defined. There are two kinds of Γ, depending on the root node:

(a) If m is not a Push rule ($\eta = \epsilon$), then

$$
\begin{aligned}
&[\![\mathbf{lch}(\Gamma, B)]\!]^{\mu'} \\
&= \mu'(\mathbf{lh}(m))[\![\mathbf{lh}(t_L)]\!]^{\mu'}[\![\mathbf{lch}(\Delta, B)]\!]^{\mu'}[\![\mathbf{lh}(t_R)]\!]^{\mu'} \\
&= (\lambda t_L t_c t_R x.\, \mu(m) t_L(t_c x) t_R)[\![t_L]\!]^\mu[\![\Delta]\!]^\mu[\![t_R]\!]^\mu \\
&= \lambda x.\, \mu(m)[\![t_L]\!]^\mu([\![\Delta]\!]^\mu x)[\![t_R]\!]^\mu
\end{aligned}
$$

(b) If m is a Push rule ($\eta \neq \epsilon$), then let $m' = \frac{A\epsilon}{\underset{\sim}{B}} \xrightarrow{C_i} A_L\, \frac{A_c\eta}{B}\, A_R$. Thus, $\mathbf{lch}(\Gamma, \underset{\sim}{B}) =$  , where

$\Delta_t\{\Delta_b\} = \Delta$.

$[\![\Delta]\!]^\mu = [\![\Delta_t\{\Delta_b\}]\!]^\mu = \lambda x.\, [\![\Delta_t]\!]^\mu([\![\Delta_b]\!]^\mu x)$ by Lemma 9.

By the IH, $[\![\Delta_t]\!]^\mu = [\![\mathbf{lch}(\Delta_t, \underline{D})]\!]^{\mu'}$ and $[\![\Delta_b]\!]^\mu = [\![\mathbf{lch}(\Delta_b, \underline{B})]\!]^{\mu'}$.

$$
\begin{aligned}
&[\![\mathbf{lch}(\Gamma, \underline{B})]\!]^{\mu'} \\
&= \mu'(m')[\![\mathbf{lh}(t_L)]\!]^{\mu'}(\mu'(\textsc{comp}) \\
&\quad [\![\mathbf{lch}(\Delta_t, \underline{D})]\!]^{\mu'}[\![\mathbf{lch}(\Delta_b, \underline{B})]\!]^{\mu'})[\![\mathbf{lh}(t_R)]\!]^{\mu'} \\
&= \mu'(m')[\![t_L]\!]^{\mu}(\mu'(\textsc{comp})[\![\Delta_t]\!]^{\mu}[\![\Delta_b]\!]^{\mu})[\![t_R]\!]^{\mu} \\
&= \mu'(m')[\![t_L]\!]^{\mu}[\lambda z.\, [\![\Delta_t]\!]^{\mu}([\![\Delta_b]\!]^{\mu}(z))][\![t_R]\!]^{\mu} \\
&= \mu'(m')[\![t_L]\!]^{\mu}[\![\Delta]\!]^{\mu}[\![t_R]\!]^{\mu} \\
&= \lambda x.\, \mu(m)[\![t_L]\!]^{\mu}([\![\Delta]\!]^{\mu}x)[\![t_R]\!]^{\mu}
\end{aligned}
$$

So **lh** and **lch** both preserve the results of any arbitrary interpretation function μ via μ'. □

Corollary 11. Weak equivalence and preservation of yields by **lh** follows from interpretational equivalence and preservation of interpretations. Let $\mu = \mathbf{y}$, the natural yield function for LIGs, and $\mu' = \mathbf{y}$, the natural yield function for HGs, with a pair of strings being equivalent to a function from strings to strings: $u_{\uparrow}v = \lambda x.\, uxv$.

5.2 HG to LIG

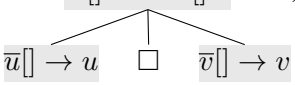
This subsection shows the reverse: how to preserve interpretations in the translation from HG to LIG. LIG interpretations in this section will involve lists, so here are a few preliminary definitions regarding lists: $[x]^{\downarrow} = x$ and $x{:}[y, \ldots] = [x, y, \ldots]$.

Theorem 12. **hl** preserves interpretations
Given an HG $\mathcal{G}$ with μ, we can construct an interpretation function μ' for $\textsc{hl}(\mathcal{G})$, such that $\forall d \in \mathcal{D}(\mathcal{G}).\, [[d]\!]^{\mu}] = [\![\mathbf{hl}(d)]\!]^{\mu'}[\,]$, i.e. $[\![d]\!]^{\mu} = ([\![\mathbf{hl}(d)]\!]^{\mu'}[\,])^{\downarrow}$. That is, the interpretation of $\mathbf{hl}(d)$ applied to the empty list is equal to the interpretation of d contained in a list. To define μ':

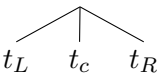
- $\mu'(\overline{u} \to u) = [\,]$
- $\mu'(A[\,] \to \overline{u}\ \alpha[\,]\ \overline{v}) = \lambda usv.(\mu(A \to u_{\uparrow}v)){:}s$
- If $m = A \xrightarrow{C_c} A_L\, A_c\, A_R$
  and $m' = A[\,] \to A_L\, A_c[\,]\, A_R$,
  then $\mu'(m') = \lambda x(y{:}s)z.\, (\mu(m)(x^{\downarrow})y(z^{\downarrow})){:}s$.
- If $m = A \xrightarrow{W} B\, D$,
  then $\mu'(A[\,] \to B[D]) = \lambda(x{:}y{:}s).\, (\mu mxy){:}s$
  and $\mu'(\alpha[D] \to D[\,]) = \lambda x.\, x$.
- $\mu'(\alpha[\,] \to \epsilon) = [\,]$ ALPHA

Proof. Note that since the image of **hl** is $\mathcal{C}(\text{LIG})$, the interpretation of contexts applies.
Base case:
$d = A \to u_{\uparrow}v$, so $[\![d]\!]^{\mu} = \mu(A \to u_{\uparrow}v)$.

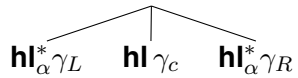
$\mathbf{hl}(d) = A[\,] \to \overline{u}\ \alpha[\,]\ \overline{v}$, so



$[\![\mathbf{hl}(d)]\!]^{\mu'}[\,] = \mu'(A[\,] \to \overline{u}\ \alpha[\,]\ \overline{v})[\,] = (\lambda s.\, \mu(A \to u_{\uparrow}v){:}s)[\,] = [\mu(A \to u_{\uparrow}v)] = [[d]\!]^{\mu}]$.
Inductive cases:
(i) If $m = A \xrightarrow{C_c} A_L\, A_c\, A_R$ and $d = m$, then $[\![d]\!]^{\mu} = \mu(m)[\![t_L]\!]^{\mu}[\![t_c]\!]^{\mu}[\![t_R]\!]^{\mu}$.



Let $m' = A[\,] \to A_L\, A_c[\,]\, A_R$, so
$\mathbf{hl}(d) = m'$.



And by the IH, $[[\![t_{L/c/R}]\!]^{\mu}] = [\![\mathbf{hl}\,\gamma_{L/c/R}]\!]^{\mu'}[\,]$.

$$
\begin{aligned}
&[\![li(d)]\!]^{\mu'}([\,]) \\
&= (\lambda x.\, \mu'(m')[\![(\mathbf{hl}^*\gamma_L, \textsc{alpha})]\!]^{\mu'} \\
&\qquad ([\![\mathbf{hl}\,\gamma_c]\!]^{\mu'}x)[\![(\mathbf{hl}^*\gamma_R, \textsc{alpha})]\!]^{\mu'})[\,] \\
&= \mu'(m')([\![\mathbf{hl}^*\gamma_L]\!]^{\mu'}[\,])([\![\mathbf{hl}\,\gamma_c]\!]^{\mu'}[\,])([\![\mathbf{hl}^*\gamma_R]\!]^{\mu'}[\,]) \\
&= \mu'(m')[[\![t_L]\!]^{\mu}][[\![t_c]\!]^{\mu}][[\![t_R]\!]^{\mu}] \\
&= (\mu(m)([[\![t_L]\!]^{\mu}]^{\downarrow})([\![t_c]\!]^{\mu})([[\![t_R]\!]^{\mu}]^{\downarrow})){:}[\,] \\
&= [\mu(m)[\![t_L]\!]^{\mu}[\![t_c]\!]^{\mu}[\![t_R]\!]^{\mu}]
\end{aligned}
$$

(ii) If $m = A \xrightarrow{W} B\, D$ and $d = m$, then

$[\![d]\!]^{\mu} = \mu(m)[\![t_l]\!]^{\mu}[\![t_r]\!]^{\mu}$.
Let $\Gamma = A[\,] \to B[D]$ with $\mathbf{hl}\,t_l$ and $\Delta = \alpha[D] \to D[\,]$ with $\mathbf{hl}\,t_r$,
so $\mathbf{hl}(d) = \Gamma\{\Delta\}$. By induction, $[[\![t_{l/r}]\!]^{\mu}] = [\![\mathbf{hl}\,t_{l/r}]\!]^{\mu'}[\,]$. This entails $[\![\mathbf{hl}\,t_l]\!]^{\mu'} = \lambda x.\, [\![t_l]\!]^{\mu}{:}x$.
$[\![\Delta]\!]^{\mu'} = \lambda y.\, (\lambda x.\, x)([\![\mathbf{hl}\,t_r]\!]^{\mu'}y) = [\![\mathbf{hl}\,t_r]\!]^{\mu'}$.
$[\![\Gamma]\!]^{\mu'} = \lambda z.\, (\lambda(x{:}y{:}s).\, (\mu mxy){:}s)([\![\mathbf{hl}\,t_l]\!]^{\mu'}z)$.

$$
\begin{aligned}
&[\![\mathbf{hl}(d)]\!]^{\mu'}[\,] = [\![\Gamma\{\Delta\}]\!]^{\mu'}[\,] \\
&= [\![\Gamma]\!]^{\mu'}([\![\Delta]\!]^{\mu'}[\,]) \text{ (by Lemma 9)} \\
&= (\lambda(x{:}y{:}s).\, (\mu mxy){:}s)([\![\mathbf{hl}\,t_l]\!]^{\mu'}([\![\mathbf{hl}\,t_r]\!]^{\mu'}[\,])) \\
&= (\lambda(x{:}y{:}s).\, (\mu mxy){:}s)((\lambda x.\, [\![t_l]\!]^{\mu}{:}x)[[\![t_r]\!]^{\mu}]) \\
&= [\mu(m)[\![t_l]\!]^{\mu}[\![t_r]\!]^{\mu}]
\end{aligned}
$$

Finally, since $\mathbf{hl}_{\alpha}(d) = \mathbf{hl}(d)\{\alpha[\,] \to \epsilon\}$, $[\![\mathbf{hl}_{\alpha}(d)]\!]^{\mu'} = [[\![d]\!]^{\mu}]$. □

Again, weak equivalence and preservation of yields by $\mathbf{hl}_{\alpha}$ follows as an instance of μ. In addition, if we let μ be the identity function on $\mathcal{D}(\mathcal{G})$, μ' will emulate $\mathbf{hl}^{-1}_{-\alpha}$.

6 Discussion

The translation from LIG to HG presented in Vijay-Shanker and Weir (1994) formed the basis for this paper; because they only considered generated string languages, their translations did not preserve the number of derivations per string. Thus, to study equivalences deeper than of sets of strings, Section 3.1 presented a modified version of their grammar translation which is indeed derivation-count preserving.

However, it is worth noting that when paired with the semantic translation defined in Theorem 10, the original translation algorithm in Vijay-Shanker and Weir (1994) is also string-meaning preserving, despite not producing derivationally equivalent grammars. The HGs produced from their translation contain infinite families of spurious ambiguities, but the semantic translation in this paper is defined in such a way to overlook these instances of spurious ambiguity, treating them as semantically vacuous. This shows that derivational equivalence is not a prerequisite to interpretational equivalence. In general, derivational equivalence may become a less useful notion when working with infinitely ambiguous grammars, but interpretational equivalence does not.

Other researchers have proposed ways to compare generative capacity at a level beyond strings, such as Koller and Kuhlmann (2009) and Schiffer and Maletti (2021). Most notably, Kanazawa (2014) shows that the set of derivation trees of arboreal indexed grammars (with categories stripped of indices at each node) is equal to the set of trees derived by simple context-free tree grammars. In the base case, this equivalence roughly reduces down to the deindexed derivation trees of LIG being equal to the trees derived by Tree-Adjoining Grammars (TAG), the latter of which has essentially been proven strongly equivalent to HG in Fujiyoshi and Kasai (2000).

The goal of establishing notions of equivalences beyond string languages is to consider what kinds of properties each formalism is capable of ascribing to each string. If these properties, such as semantics, are computed compositionally from the derivations, then in comparing interpretations, we are implicitly comparing derivations which have that interpretation. In theory, it is not enough to establish that the derived trees of a tree grammar are in correspondence with the deindexed derivation trees of an index grammar, since there could be multiple derivations per derived tree in the former case, or multiple derivations which are identical modulo indices in the latter case. However, it is easy to see that given the semantic translation developed here, Kanazawa's result can be strengthened to interpretational equivalence.

7 Conclusion

In this paper, I explored two kinds of equivalence beyond weak equivalence and proved they both apply to LIG and HG. Two grammars are derivationally equivalent if they generate the same number of derivations per string, and two grammars are interpretationally equivalent if they generate the same set of string-interpretation pairs.

These proofs were facilitated by the functions $\mathbf{lh}$ and $\mathbf{hl}_\alpha$, which translated derivations from a grammar in one formalism to an equivalent grammar in the other formalism. It is important to note that these functions did not depend on the grammars themselves in any way. This means that they could be defined over $\mathcal{D}_{\text{LIG}}$ and $\mathcal{D}_{\text{HG}}$, showing that all LIG and HG derivations stand in a bijection.

This paper can serve as a model for how to rigorously compare recursive mechanisms across formalisms. The notions of derivational and interpretational equivalences shed light on how the generative structures of different grammar formalisms relate. In this case, the equivalence between LIG and HG reveals how working with a linear stack-based category system is equivalent on some fundamental level to the ability to wrap pairs of strings.

Acknowledgments

I would like to thank Dylan Bumford, Tim Hunter, Laurel Perkins, Robert Frank, and audiences at UCLA SynSem and MoL for invaluable feedback and discussion. This paper has also benefited from the comments of three anonymous reviewers.

References

Joan Bresnan, Ronald M. Kaplan, Stanley Peters, and Annie Zaenen. 1982. Cross-Serial Dependencies in Dutch. Linguistic Inquiry, 13(4):613–635. Publisher: MIT Press.

Robert Frank and Tim Hunter. 2021. Variation in mild context-sensitivity: Derivational state and structural monotonicity. Evolutionary Linguistic Theory, 3(2):181–214.

A. Fujiyoshi and T. Kasai. 2000. Spinal-Formed Context-Free Tree Grammars. Theory of Computing Systems, 33(1):59–83.

Gerald Gazdar. 1988. Applicability of indexed grammars to natural languages. In U. Reyle and C. Rohrer, editors, Natural Language Parsing and Linguistic Theories, pages 69–94. Springer Netherlands, Dordrecht.

Makoto Kanazawa. 2014. A Generalization of Linear Indexed Grammars Equivalent to Simple Context-Free Tree Grammars. In Formal Grammar, volume 8612, pages 86–103, Berlin, Heidelberg. Springer Berlin Heidelberg.

Alexander Koller and Marco Kuhlmann. 2009. Dependency trees and the strong generative capacity of CCG. In Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics, pages 460–468, Athens, Greece. Association for Computational Linguistics.

Philip H. Miller. 1999. Strong Generative Capacity: The Semantics of Linguistic Formalism. Number no. 103 in CSLI Lecture Notes. CSLI Publications, Stanford, CA.

Carl Jesse Pollard. 1984. Generalized phrase structure grammars, head grammars, and natural language. Ph.D. thesis, Stanford University, Stanford, CA.

Kelly Roach. 1987. Formal Properties of Head Grammars. In Alexis Manaster-Ramer, editor, Mathematics of Language: Proceedings of a conference held at the University of Michigan, Ann Arbor, October 1984, pages 293–348. John Benjamins Publishing Company.

Lena Katharina Schiffer and Andreas Maletti. 2021. Strong Equivalence of TAG and CCG. Transactions of the Association for Computational Linguistics, 9:707–720.

K. Vijay-Shanker and D. J. Weir. 1994. The equivalence of four extensions of context-free grammars. Mathematical Systems Theory, 27(6):511–546.