Parsing Preserving Techniques in Grammar Induction

Smaranda Muresan
Department of Computer Science
Columbia University
New York, NY, 10027
smara@cs.columbia.edu

Abstract

In this paper we present the theoretical foundation of the search space for learning a class of constraint-based grammars, which preserve the parsing of representative examples. We prove that under several assumptions the search space is a complete grammar lattice, and the lattice top element is a grammar that can always be learned from a set of representative examples and a sublanguage used to reduce the grammar semantics. This complete grammar lattice guarantees convergence of solutions of any learning algorithm that obeys the given assumptions.

1 Introduction

Research in grammar induction has been relevant both to practical Natural Language Processing applications and studies of human language acquisition. Learning syntactic structures using both supervised and unsupervised methods has been particularly successful [Col99, KM04]. In recent years, however, there has been a growing need to acquire semantics as well, in order to achieve a deeper understanding of text. For practical reasons, the learning would need a small amount of annotated data, since large semantically annotated treebanks are not readily available, and are hard to build for a variety of domains. At the same time, theories of language acquisition stipulate that access to meaning is needed during language acquisition and the learner needs at least utterances paired with semantic representations to simulate the grounding of the learning process [Pin89, CN03].

Building upon these considerations, we have proposed in our previous work one approach to inducing grammars that capture both syntax and semantics [MMK04, MMK05]. We have introduced a type of constraint-based grammars, Lexicalized Well-Founded Grammars (LWFGs), which associate a syntactic-semantic representation to each string. The input to the learner is a small set of representative examples, which consist of strings paired with their syntactic-semantic representations. An ontology is used as background knowledge to provide access to meaning during learning and parsing, at the grammar rule level. We proved that these grammars can always be learned. For hypothesis (grammar rule) generation, our relational learning method uses a robust parser in the background knowledge [MMK05, pg. 23]. The key element for hypothesis generation is a technique that preserves the parsing of the current representative example for all the candidate hypotheses from which the final best rule is chosen.

In this paper, this parsing preserving technique is used for the theoretical foundation of the grammar induction search space. We explore how the search space can be organized so that all the LWFG induction algorithms converge to the same grammar. We consider additional assumptions for LWFGs, which allow us to define the search space as a complete grammar lattice. For this, we introduce the operational and denotational semantics of these grammars, as well as the set of representative examples. Finally, we define a class of derived grammars, which preserve the parsing of the representative examples, and prove that these grammars form a semantic-based complete lattice. This lattice is the search space for all the relational learning algorithms used for inducing LWFGs. We prove that for a type of LWFG conform to a sublanguage, the top element of the grammar lattice can always be learned from the representative examples and the sublanguage used for reducing the grammar semantics. The uniqueness of the complete lattice top element allows the development of several efficient algorithms for grammar induction, which use the search space in a sound way, and thus can always learn the same grammar.

In Section 2, we briefly present the Lexicalized Well-Founded Grammars introduced by [MMK04, MMK05]. In Section 3, we define the operational and denotational semantics of these grammars [MMK05]. The key concepts and properties needed to define a class of LWFGs that form a complete lattice are given in Section 4. We define the representative example set, the grammar semantics reduced to a sublanguage, and the class of derived grammars that preserve the parsing of the representative examples. In Section 5, we prove that these grammars form a semantic-based complete lattice. The grammar induction problem defined on the complete grammar lattice search space is given in Section 6, along with the proof that the lattice top element can always be learned. Conclusions are outlined in Section 7.

2 Lexicalized Well-Founded Grammars

Lexicalized Well-Founded Grammars were introduced in our previous work [MMK04, MMK05] as a type of constraint-based grammars that: 1) associate each string with a syntactic-semantic representation called *semantic molecule*; 2) have two types of constraints at the rule level: one for semantic composition and one for ontology-based semantic interpretation; and 3) introduce a partial ordering among nonterminals, which allows the ordering of grammar rules, and thus facilitate the bottom-up induction of these grammars. In this section we present only the necessary concepts needed in this paper.

Definition 1. A *syntagma*, $\sigma = (w, w')$, is a pair of a natural language string (w) and its semantic molecule (w'), and represents any unit that can be derived from a grammar.

Definition 2. A *Lexicalized Well-Founded Grammar (LWFG)* is a 6-tuple, $G = \langle \Sigma, \Sigma', N_G, R_G, P_G, S \rangle$, where:

1. Σ is a finite set of terminal symbols.
2. Σ' is a finite set of elementary semantic molecules corresponding to the set of terminal symbols. That is $w' \in \Sigma'$ iff $w \in \Sigma$, where $\sigma = (w, w')$.
3. N_G is a finite set of nonterminal symbols.
4. R_G is a partial ordering relation, $\succeq$, among the nonterminals.
5. P_G is a set of constraint rules. A constraint rule is a triple $(A, (B_1, \ldots, B_n), \Phi)$, written $A(\sigma) \rightarrow B_1(\sigma_1), \ldots, B_n(\sigma_n)\colon \Phi(\bar{\sigma})$, where $\bar{\sigma} = (\sigma, \sigma_1, ..., \sigma_n)$ s.t. $\sigma = (w, w'), \sigma_i = (w_i, {w_i}'), 1 \leq i \leq n, w = w_1 \cdots w_n, w' = w'_1 \circ \cdots \circ w'_n$, and $\circ$ is the composition operator. Sometimes, for brevity, we denote a rule by $A \rightarrow \beta\colon \Phi$, where $A \in N_G, \beta \in N_G^+$. For the rules whose left hand side are preterminals, $A(\sigma) \rightarrow$, we use the notation $A \rightarrow \sigma$.
6. $S \in N_G$ is the start nonterminal symbol, and $\forall A \in N_G, S \succeq A$.
7. All syntagmas $\sigma = (w, w')$, derived from a nonterminal A have the same category of their semantic molecules w' [1].

There are three types of rules: ordered non-recursive, ordered recursive, and non-ordered rules. A grammar rule $A \rightarrow B_1, \ldots, B_n\colon \Phi \in P_G$, is an *ordered rule*, if $\forall B_i$, we have $A \succeq B_i$. The LWFG rules have several properties: each nonterminal symbol is a left-hand side in at least one ordered non-recursive

[1] This property is used for determining the lhs nonterminal of the learned rule [MMK04, MMK05]

rule; the empty string cannot be derived from any nonterminal symbol; the rule nonterminals are augmented with syntagmas, σ; the rules are enriched with constraints, $\Phi(\bar{\sigma})$.

Definition 3. Given a LWFG, G, the *ground syntagma derivation*, '$\stackrel{*G}{\Rightarrow}$', is defined as: $\frac{A \to \sigma}{A \stackrel{*G}{\Rightarrow} \sigma}$ (if $\sigma = (w, w'), w \in \Sigma, w' \in \Sigma'$, i.e., A is a preterminal), and $\frac{B_i \stackrel{*G}{\Rightarrow} \sigma_i,\ i=1,\ldots,n, \quad A(\sigma) \to B_1(\sigma_1),\ldots,B_n(\sigma_n)\colon \Phi(\bar{\sigma})}{A \stackrel{*G}{\Rightarrow} \sigma}$.

The ground syntagma derivation, $A \stackrel{*G}{\Rightarrow} \sigma$, is equivalent to Definite Clause Grammar provability [PW80]. The *language* of a grammar G is the set of all syntagmas generated from the start symbol S, i.e., $L(G) = \{\sigma | \sigma = (w, w'), w \in \Sigma^+, S \stackrel{*G}{\Rightarrow} \sigma\}$. The *set of all syntagmas* generated by *a grammar* G is $L_\sigma(G) = \{\sigma | \sigma = (w, w'), w \in \Sigma^+, \exists A \in N_G, A \stackrel{*G}{\Rightarrow} \sigma\}$. For a grammar G, let E be a sublanguage, such that $E \subseteq L(G)$, and let $E_\sigma \subseteq L_\sigma(G)$ be the set of subsyntagmas corresponding to the sublanguage E. We have that $L(G) \subseteq L_\sigma(G)$ and $E \subseteq E_\sigma$[2]. Extending the notation, the set of syntagmas generated by *a nonterminal* A of the grammar G is $L_\sigma(A) = \{\sigma | \sigma = (w, w'), w \in \Sigma^+, A \in N_G, A \stackrel{*G}{\Rightarrow} \sigma\}$, and the set of syntagmas generated by *a rule* $A \to \beta\colon \Phi$ of the grammar G is $L_\sigma(A \to \beta\colon \Phi) = \{\sigma | \sigma = (w, w'), w \in \Sigma^+, (A \to \beta\colon \Phi) \stackrel{*G}{\Rightarrow} \sigma\}$[3].

3 Semantics of LWFGs

Operational Semantics. Following the insight of "parsing as deduction" [SSP95], a deductive system for parsing Context-Free Grammars can serve as a method for defining their operational semantics. It has been shown that the operational semantics of a CFG corresponds to the language of the grammar [Win99]. Analogously, the operational semantics of a LWFG, G, is the set of all syntagmas generated by the grammar, $L_\sigma(G)$.

Denotational Semantics. As discussed in literature [PS84, Win99], the denotational semantics of a grammar is defined through a fixpoint of a transformational operator associated with the grammar.

Definition 4. Let $I \subseteq L_\sigma(G)$ be a subset of syntagmas generated by the grammar G. We define the *immediate syntagma derivation operator* $T_G\colon 2^{L_\sigma(G)} \to 2^{L_\sigma(G)}$, s.t.: $T_G(I) = \{\sigma \in L_\sigma(G) |$ if $(A(\sigma) \to B_1(\sigma_1), \ldots, B_n(\sigma_n)\colon \Phi(\bar{\sigma})) \in P_G \ \wedge\ B_i \stackrel{*G}{\Rightarrow} \sigma_i \ \wedge\ \sigma_i \in I$ then $A \stackrel{*G}{\Rightarrow} \sigma\}$. If we denote $T_G \uparrow 0 = \emptyset$ and $T_G \uparrow (i+1) = T_G(T_G \uparrow i)$, then we have that for $i = 1, T_G \uparrow 1 = T_G(\emptyset) = \{\sigma \in L_\sigma(G) | A \in N_G, A \to \sigma\}$. This corresponds to the syntagmas derived from preterminals, i.e., $\sigma = (w, w')$, where $w \in \Sigma, w' \in \Sigma'$. An assumption for learning LWFGs is that the rules corresponding to grammar preterminals, $A \to \sigma$, are given, i.e., $T_G(\emptyset)$ is given.

The denotational semantics of a grammar G is the least fixpoint of the immediate syntagma derivation operator. As in the case of definite logic programs, the denotational semantics is equivalent with the operational one, i.e., $L_\sigma(G) = lfp(T_G) = T_G \uparrow \omega$, where ω is the minimum limit ordinal [Tar55, vEK76].

Definition 5. Based on T_G, we can define the *ground derivation length (gdl)* for syntagmas, $gdl(\sigma)$, and the *minimum ground derivation length* for grammar rules, $mgdl(A \to \beta\colon \Phi)$:

$$\begin{aligned} gdl(\sigma) &= \min_{\sigma \in T_G \uparrow i} (i) \\ mgdl(A \to \beta\colon \Phi) &= \min_{\sigma \in L_\sigma(A \to \beta\colon \Phi)} (gdl(\sigma)) \end{aligned} \tag{1}$$

[2] In the remainder of this paper, we will use the term *sublanguage* E_σ to refer to the set of subsyntagmas corresponding to the sublanguage E.

[3] $(A \to \beta\colon \Phi) \stackrel{*G}{\Rightarrow} \sigma$ denotes the ground derivation $A \stackrel{*G}{\Rightarrow} \sigma$ obtained using the rule $A \to \beta\colon \Phi$ in the last derivation step.

4 Representative Examples Parsing Preserving Grammars

This section introduces the key concepts and properties needed to define a class of LWFGs that form a complete lattice.

Definition 6. A LWFG, G, is *unambiguous* w.r.t. a sublanguage $E_\sigma \subseteq L_\sigma(G)$ if $\forall \sigma \in E_\sigma$ there is one and only one rule $A \rightarrow \beta \colon \Phi \overset{*G}{\Rightarrow} \sigma$ [4].

Definition 7. A set of syntagmas $E_R \subseteq E_\sigma \subseteq L_\sigma(G)$ is called *representative example set* of a LWFG, G, unambiguous w.r.t a sublanguage E_σ, iff for each rule $(A \rightarrow \beta \colon \Phi) \in P_G$ there is a unique syntagma $\sigma \in E_R$ s.t. $gdl(\sigma) = mgdl(A \rightarrow \beta \colon \Phi)$.

From this definition it is straightforward that $|E_R| = |P_G|$. E_R contains the most simple syntagmas ground derived from the grammar G, and covers all the grammar rules [MMK05, pg. 15].

In order to define the search space of grammar induction as grammar lattice, we define *the rule derivation step* and *the rule generalization step* of unambiguous LWFGs, such that they are E_R *parsing preserving* and are the inverse of each other.

Definition 8. The *rule derivation step*:

$$\frac{A(\sigma_A) \rightarrow \alpha B(\sigma_B^*)\gamma \colon \Phi_A \qquad B(\sigma_B) \rightarrow \beta \colon \Phi_B}{A(\sigma_A) \rightarrow \alpha\beta\gamma \colon \Phi'_A} \tag{2}$$

is E_R *parsing preserving*, if $r_A \overset{*G}{\Rightarrow} \sigma_A \wedge r'_A \overset{*G'}{\Rightarrow} \sigma_A \wedge \sigma_A \in E_R$, where $r_A = A(\sigma_A) \rightarrow \alpha B(\sigma_B^*)\gamma \colon \Phi_A$, $r_B = B(\sigma_B) \rightarrow \beta \colon \Phi_B$, and $r'_A = A(\sigma_A) \rightarrow \alpha\beta\gamma \colon \Phi'_A$. We write $r_A \overset{r_B}{\vdash} r'_A$.

The *rule generalization step* [5]:

$$\frac{A(\sigma_A) \rightarrow \alpha\beta\gamma \colon \Phi'_A \qquad B(\sigma_B) \rightarrow \beta \colon \Phi_B}{A(\sigma_A) \rightarrow \alpha B(\sigma_B^*)\gamma \colon \Phi_A} \tag{3}$$

is E_R *parsing preserving*, if $r'_A \overset{*G'}{\Rightarrow} \sigma_A \wedge r_A \overset{*G}{\Rightarrow} \sigma_A \wedge \sigma_A \in E_R$. We write $r'_A \overset{r_B}{\dashv} r_A$.

The property of E_R parsing preserving means that both the initial and the modified rules ground derive the same syntagma, $\sigma_A \in E_R$. Since σ_A is a representative example, it has the minimum ground derivation length ($gdl(\sigma_A) = mgdl(r_A)$) and thus, we have that r_B is an ordered non-recursive rule (see Section 2). Moreover, both constraints Φ'_A/Φ_A are computed after each derivation/generalization step, based on the syntagmas that augment the grammar nonterminals during $(r'_A \overset{*G'}{\Rightarrow} \sigma_A)/(r_A \overset{*G}{\Rightarrow} \sigma_A)$ ground derivation. Thus, the derivation/generalization steps are the inverse of each other [6]. From both the derivation and the generalization step we have that: $L_\sigma(r_A) \supseteq L_\sigma(r'_A)$.

Definition 9. A grammar G' is *one-step derived* from a grammar G, $G \overset{r_1}{\vdash} G'$, if $\exists r, r_1 \in P_G \wedge \exists r', r_1 \in P_{G'}$, s.t. $r \overset{r_1}{\vdash} r'$, and $\forall q \neq r, q \in P_G$ iff $q \in P_{G'}$. A grammar G' is *derived* from a grammar G, $G \overset{*}{\vdash} G'$, if it is obtained from G in n-derivation steps: $G \overset{r_1}{\vdash} \cdots \overset{r_n}{\vdash} G'$, where n is finite. We extend the notation so that we have $G \overset{*}{\vdash} G$.

[4]Unambiguity is relative to syntagmas and not to language strings, which can be ambiguous. In the case of chains of unary branching rules, the ground derived equivalent syntagmas of the same string must have different categories (Definition 2, p7) and [MMK05, pg. 19].

[5]The goal of the rule derivation/generalization step is to obtain a new target grammar G'/G from G/G' by modifying a rule of G/G'. They are not to be taken as the derivation/reduction concepts in parsing.

[6]This property would not hold if in the derivation process the constraint composition were used ($\Phi'_A = \Phi_B \circ \Phi_A$) [MMK05, pg. 10].

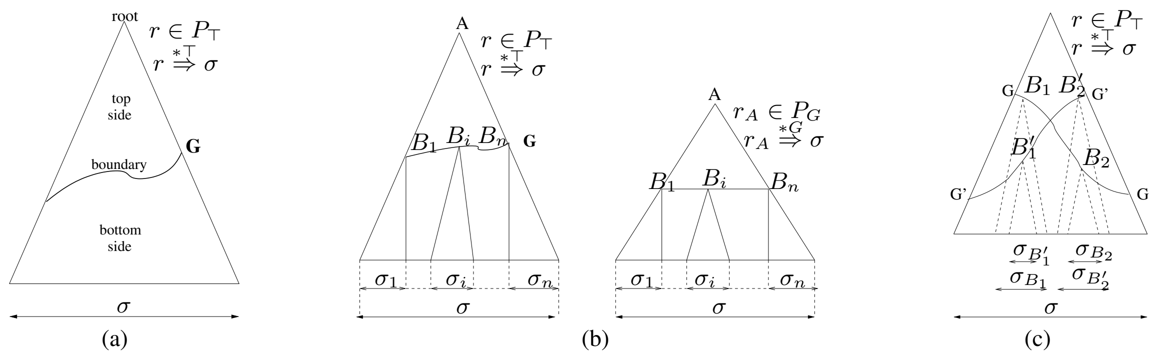


Figure 1: (a) Grammar boundary; (b) Subtree correspondence (c) Subsyntagma relations

Definition 10. A grammar G is *one-step generalized* from a grammar G', $G' \overset{r_1}{\dashv} G$, if $\exists r, r_1 \in P_G \wedge \exists r', r_1 \in P_{G'}$, s.t. $r' \overset{r_1}{\dashv} r$, and $\forall q \neq r, q \in P_G$ iff $q \in P_{G'}$. A grammar G is *generalized* from grammar G', $G' \overset{*}{\dashv} G$, if it is obtained from G' in n-generalization steps: $G' \overset{r_1}{\dashv} \cdots \overset{r_n}{\dashv} G$, where n is finite. We extend the notation so that we have $G \overset{*}{\dashv} G$.

Definition 11. Given a LWFG G and a sublanguage $E_\sigma \subseteq L_\sigma(G)$, we call $\mathbb{S}(G) = L_\sigma(G) \cap E_\sigma$ *the semantics of the grammar G reduced to the sublanguage* E_σ. Given a grammar rule $r \in P_G$, we call $\mathbb{S}(r) = L_\sigma(r) \cap E_\sigma$ *the semantics of the grammar rule* r *reduced to the sublanguage* E_σ.

Definition 12. A LWFG G is called *normalized* w.r.t. a sublanguage $E_\sigma \subseteq L_\sigma(G)$, if all grammar rules cannot be further generalized by the rule generalization step (3), such that $\mathbb{S}(r'_A) \subset \mathbb{S}(r_A)$.

Definition 13. Let $\top$ be a LWFG, normalized and unambiguous w.r.t. a sublanguage $E_\sigma \subseteq L_\sigma(\top)$, and let $E_R \subseteq E_\sigma$ be its set of representative examples. Let $\mathcal{L} = \{G | \top \overset{*}{\vdash} G\}$ be the set of grammars derivable from $\top$. We call $\top$ the *top element* of $\mathcal{L}$, and $\perp$ the *bottom element* of $\mathcal{L}$, if $\forall G \in \mathcal{L}, \top \overset{*}{\vdash} G \wedge G \overset{*}{\vdash} \perp$. The bottom element, $\perp$, is the grammar derived from $\top$, such that the right hand side of all grammar rules contains only preterminals. We have $\mathbb{S}(\top) = E_\sigma$ and $\mathbb{S}(\perp) \supseteq E_R$ (see Figure 4(b)).

Lemma 1. *For* $G, G' \in \mathcal{L}$, $G \overset{*}{\vdash} G'$ *iff* $G' \overset{*}{\dashv} G$. *Moreover,* $L_\sigma(G) \supseteq L_\sigma(G')$.

Proof. The proof is straightforward. □

Lemma 2. $\forall G \in \mathcal{L}$ *and* $\forall \sigma \in E_R$, $\exists r \in P_G$, *s.t.* $r \overset{*}{\Rightarrow} \sigma$.

Proof. The property holds for $\top$ and $\top \overset{*}{\vdash} G$ is E_R parsing preserving. □

Definition 14. If $G, G' \in \mathcal{L}$, we say that G *subsumes* G', i.e., $G \succcurlyeq G'$, iff $G \overset{*}{\vdash} G'$.

Theorem 1. *For* $G, G' \in \mathcal{L}$, *if* $G \succcurlyeq G'$ *then* $\mathbb{S}(G) \supseteq \mathbb{S}(G')$.

Proof. From Lemma 1, and Definitions 11, 14.

Definition 15. We call *boundary* of a grammar $G \in \mathcal{L}$ relative to the parse tree $r \overset{*\top}{\Rightarrow} \sigma$ [7], the right hand side of the corresponding rule $r_A \in P_G, r_A \overset{*G}{\Rightarrow} \sigma$: $bd(G) = \{B | r_A \in P_G, B \in rhs(r_A)\}$ [8](see Figure

[7]All grammars G, $\top \overset{*}{\vdash} G$, are E_R parsing preserving and all boundaries of G are in the parse trees of the ground derivations of $\top$ grammar rules.

[8]The notation of $bd(G), ts(G), bs(G)$ ignores the rule relative to which these concepts are defined, and in the remainder of this paper we implicitly understand that the relations hold for all grammar rules.

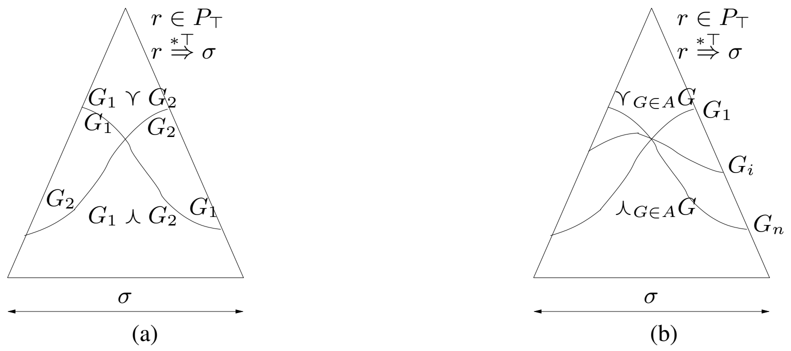


Figure 2: The lub and glb operators

1(a)). We denote by $f_N(r \stackrel{*\top}{\Rightarrow} \sigma)$ the set of nonterminals which belong to the parse tree of $r \stackrel{*\top}{\Rightarrow} \sigma$, where $f_N : P_\top \times E_R \to N_\top, r \in P_\top, \sigma \in E_R$. We call *top-side*, $ts(G)$, and respectively *bottom-side*, $bs(G)$, of grammar G relative to the parse tree $r \stackrel{*\top}{\Rightarrow} \sigma$, the sets of the nonterminals delimited by G boundary, $bd(G)$ (see Figure 1(a)):

$$ts(G) = \{B \in f_N(r \stackrel{*\top}{\Rightarrow} \sigma) | \exists B_i \in bd(G) \wedge B \succeq B_i\} \cup \{root\}$$

$$bs(G) = \{B \in f_N(r \stackrel{*\top}{\Rightarrow} \sigma) | \exists B_i \in bd(G) \wedge B \preceq B_i\}^9$$

We have that $ts(G) \cap bs(G) = bd(G)$, $ts(G) \cup bs(G) = f_N(r \stackrel{*\top}{\Rightarrow} \sigma)$ and for the top element of $\mathcal{L}$: $ts(\top) = bd(\top) \cup \{root\}$.

Lemma 3. *$\forall G \in \mathcal{L}, \forall r_A \in P_G, r_A \stackrel{*G}{\Rightarrow} \sigma$, and $\forall B_i \in rhs(r_A)$, the parse tree $B_i \stackrel{*G}{\Rightarrow} \sigma_i$ has a corresponding subtree in the parse tree $r \stackrel{*\top}{\Rightarrow} \sigma$, rooted at the same nonterminal $B_i \in bd(G)$, such that $B_i \stackrel{*\top}{\Rightarrow} \sigma_i$.*

Proof. The property holds due to the unambiguity of the $\top$ grammar and the E_R *parsing preserving property* of the rule derivation step. Moreover, the rule derivation step preserves grammar unambiguity. If $r_A \in P_G$ is $A \to B_1, \ldots, B_n$, we have that $\sigma = \sigma_1 \cdots \sigma_n$ in both parse trees $r_A \stackrel{*G}{\Rightarrow} \sigma$ and $r \stackrel{*\top}{\Rightarrow} \sigma$ (see Figure 1(b)). □

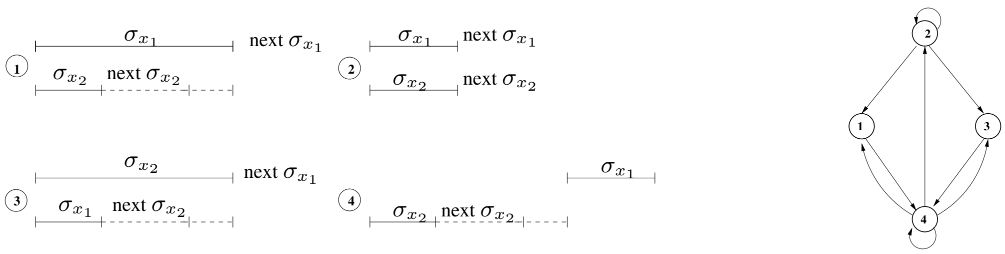
Lemma 4. *$\forall G, G' \in \mathcal{L}$, $\forall r_A \in P_G$ and $\forall r'_A \in P_{G'}$, with $r_A \stackrel{*G}{\Rightarrow} \sigma$ and $r'_A \stackrel{*G'}{\Rightarrow} \sigma$, $\sigma \in E_R$, if $B \in rhs(r_A), B' \in rhs(r'_A)$ and $B \stackrel{*G}{\Rightarrow} \sigma_B, B' \stackrel{*G'}{\Rightarrow} \sigma'_B$, then $\sigma_B \subseteq \sigma'_B \vee \sigma_B \supseteq \sigma'_B \vee \sigma_B \cap \sigma'_B = \emptyset$.*

Proof. $\top$ is a normalized and unambiguous LWFG, and $r \in P_\top$ has a unique parse tree $r \stackrel{*\top}{\Rightarrow} \sigma$. Since both G and G' are grammars derived from $\top$, the parse trees $B \stackrel{*G}{\Rightarrow} \sigma_B$ and $B' \stackrel{*G'}{\Rightarrow} \sigma'_B$ have corresponding subtrees in $r \stackrel{*\top}{\Rightarrow} \sigma$, which have the same root due to grammar unambiguity: $B \stackrel{*\top}{\Rightarrow} \sigma_B$ and $B' \stackrel{*\top}{\Rightarrow} \sigma'_B$, respectively (Lemma 3). Since no two subtrees of a tree overlap in an unambiguous grammar, the lemma property holds (Figure 1(c)). □

5 Complete Grammar Lattice

We consider the system $\mathfrak{L} = \langle \mathcal{L}, \succcurlyeq \rangle$ formed by the set $\mathcal{L}$ of the grammars derivable from $\top$, together with the binary subsumption relation $\succcurlyeq$ that establishes a partial order in $\mathcal{L}$. In order for this system to form a lattice, we must define two operators: the *least upper bound* (lub), $\curlyvee$ and the *greatest lower bound* (glb), $\curlywedge$, such that for any two elements $G_1, G_2 \in \mathcal{L}$, the elements $G_1 \curlyvee G_2, G_1 \curlywedge G_2 \in \mathcal{L}$ exist [Tar55]. The lub element of G_1, G_2 is the minimum element that has the boundary above the boundaries of G_1 and G_2. The

[9] $\succeq$ is the partial ordering relation among the LWFG nonterminals (see Definition 2, p4)



(a) (b)

Figure 3: (a) Cases of subsyntagma relations; (b) Transition diagram

glb element of G_1, G_2 is the maximum element that has the boundary below the boundaries of G_1 and G_2. Thus, lub and glb are defined such that for all grammar rules we have:

$$ts(G_1 \curlyvee G_2) = ts(G_1) \cap ts(G_2)$$
$$bs(G_1 \curlywedge G_2) = bs(G_1) \cap bs(G_2) \tag{4}$$

as can be seen in Figure 2(a). In order to have a complete lattice, the property must hold $\forall A \subseteq \mathcal{L}$:

$$ts(\curlyvee_{G \in A} G) = \bigcap_{G \in A} ts(G)$$
$$bs(\curlywedge_{G \in A} G) = \bigcap_{G \in A} bs(G) \tag{5}$$

as can be seen in Figure 2(b). These two operators are defined by Algorithms 1, and 2, which generate the rules corresponding to the grammars $G_1 \curlyvee G_2$, and $G_1 \curlywedge G_2$ based on the corresponding rules in G_1 and G_2 and the operators $\curlyvee$ and $\curlywedge$.

For this, the Procedure Op(G_1, G_2, i) is built based on Lemma 4. The input consists of the grammar rules $P_{G_1}(i)$ and $P_{G_2}(i)$, which ground derive the same syntagma $\sigma = E_R(i)$. The index i shows the bijective mapping between the grammar rules, P_G and the representative examples, E_R. The output consists of the corresponding rules $P_{G_1 \curlyvee G_2}(i)$, and $P_{G_1 \curlywedge G_2}(i)$ which ground derive the same representative example $\sigma = E_R(i)$ (see also Figure 4(a)).

Algorithm 1: Least_Upper_Bound(G_1, G_2)

```
for i ← 1 to |E_R| do
  ⌊ P_{G1⋎G2}(i) ← P_{G1}(i) ⋎ P_{G2}(i)  /*⋎(G1, G2, i)          */
return P_{G1⋎G2}
```

Algorithm 2: Greatest_Lower_Bound(G_1, G_2)

```
for i ← 1 to |E_R| do
  ⌊ P_{G1⋏G2}(i) ← P_{G1}(i) ⋏ P_{G2}(i)  /*⋏(G1, G2, i)          */
return P_{G1⋏G2}
```

The right hand sides r_{x_1}, r_{x_2} of the input grammar rules are traversed from left to right and the corresponding right hand sides $r_\curlyvee$, $r_\curlywedge$ of the output grammar rules are computed. For each right nonterminal x_1, x_2 of the input rules, the syntagmas $\sigma_{x_1}, \sigma_{x_2}$, which derive from them, are computed. The nonterminal whose ground derived syntagma includes the other's syntagma, is appended to $r_\curlyvee$, while the other nonterminal is appended to $r_\curlywedge$ (see case 1, and 3 in Procedure Op). For the equality case (case 2), the nonterminal is appended to both rules $r_{\curlyvee}$ and $r_{\curlywedge}$. Based on Lemma 4, we have four cases in Procedure `Op`, illustrated in Figure 3(a), where the syntagmas in two consecutive steps are shown. The 4th case necessarily follows after case 1 and 3, where the nonterminal x_2 is appended to $r_{\curlywedge}$. We noticed that in case 3, r_{x_1} and r_{x_2} are swapped. The transition diagram among the 4 cases is shown in Figure 3(b). A full cycle of Procedure `Op` is exemplified in Figure 4(a), where the ground derived subsyntagmas are also shown.

At the end of the Procedure `Op` both $\Phi_{\curlyvee}$ and $\Phi_{\curlywedge}$ are computed, based on the corresponding rules previously computed. [10] This is in accordance with the principle that the rule derivation/generalization steps are the inverse of each other, since lub is a generalization, while glb is a derivation.

Procedure `Op` (G_1, G_2, i)

/* $P_{G_1}(i) \curlyvee P_{G_2}(i)$ *or* $P_{G_1}(i) \curlywedge P_{G_2}(i)$ */
$l \leftarrow lhs(P_{G_1}(i))$ /* $= lhs(P_{G_2}(i))$ */
$\sigma \leftarrow E_R(i)$
$r_{x_1} \leftarrow rhs(P_{G_1}(i)))$ $r_{x_2} \leftarrow rhs(P_{G_2}(i)))$
$r_{\curlyvee} \leftarrow r_{\curlywedge} \leftarrow \emptyset$
$x_1 \leftarrow next(r_{x_1})$ $x_2 \leftarrow next(r_{x_2})$
while $x_1 \neq \emptyset \vee x_2 \neq \emptyset$ **do**
  $x_1 \stackrel{*G_1}{\Rightarrow} \sigma_{x_1} \subseteq \sigma$
  $x_2 \stackrel{*G_2}{\Rightarrow} \sigma_{x_2} \subseteq \sigma$
  if $\sigma_{x_1} \subseteq \sigma_{x_2} \vee \sigma_{x_1} \supseteq \sigma_{x_2}$ **then**
1   **if** $\sigma_{x_1} \supset \sigma_{x_2}$ **then**
    $r_{\curlyvee} \leftarrow r_{\curlyvee} @\, x_1(\sigma_{x_1})$ $r_{\curlywedge} \leftarrow r_{\curlywedge} @\, x_2(\sigma_{x_2})$ /*@ *is the concatenation operator* */
2   **if** $\sigma_{x_1} = \sigma_{x_2}$ **then**
    /*$x_1 = x_2$ [11] */
    $r_{\curlyvee} \leftarrow r_{\curlyvee} @\, x_1(\sigma_{x_1})$ $r_{\curlywedge} \leftarrow r_{\curlywedge} @\, x_2(\sigma_{x_2})$
3   **if** $\sigma_{x_1} \subset \sigma_{x_2}$ **then**
    $r_{\curlyvee} \leftarrow r_{\curlyvee} @\, x_2(\sigma_{x_2})$ $r_{\curlywedge} \leftarrow r_{\curlywedge} @\, x_1(\sigma_{x_1})$ $r_{x_1} \longleftrightarrow r_{x_2}$
   $x_1 \leftarrow next(r_{x_1})$
   $x_2 \leftarrow next(r_{x_2})$
4 **else**
   $r_{\curlywedge} \leftarrow r_{\curlywedge} @\, x_2(\sigma_{x_2})$ $x_2 \leftarrow next(r_{x_2})$
$\Phi_{\curlyvee} \leftarrow$ Generate_Constraints($l \rightarrow r_{\curlyvee}$)
$\Phi_{\curlywedge} \leftarrow$ Generate_Constraints($l \rightarrow r_{\curlywedge}$)
if $Op = \curlyvee$ **then** return $l \rightarrow r_{\curlyvee} : \Phi_{\curlyvee}$ **else** return $l \rightarrow r_{\curlywedge} : \Phi_{\curlywedge}$
function next(r)
  $x \leftarrow first(r)$
  $r \leftarrow rest(r)$
  return x

Lemma 5. *The system $\mathfrak{L} = \langle \mathcal{L}, \succeq \rangle$ together with the lub and glb operators computed by Algorithms 1 and 2, guarantees that for any two grammars $G_1, G_2 \in \mathcal{L}$ the following property holds:$G_1 \curlyvee G_2 \succeq G_1, G_2 \succeq G_1 \curlywedge G_2$ ($G_1 \curlyvee G_2 \stackrel{*}{\vdash} G_1, G_2 \stackrel{*}{\vdash} G_1 \curlywedge G_2$).*

Proof. From Procedure `Op` (Figure 3 and Figure 4(a)), it results that the boundaries $bd(G_1 \curlyvee G_2)$ and $bd(G_1 \curlywedge G_2)$ are computed with respect to (4) such that the theorem property is guaranteed for each grammar rule (see Figure 2(a)). □

[10] The constraints can be computed based on the syntagmas which augment the grammar nonterminals (Definition 2, p5), [MMK05, pg. 10].

[11] We need to include the statement $if\ x_2 \succ x_1\ then\ x_1 \longleftrightarrow x_2$ iff chains of unary branching rules are considered.

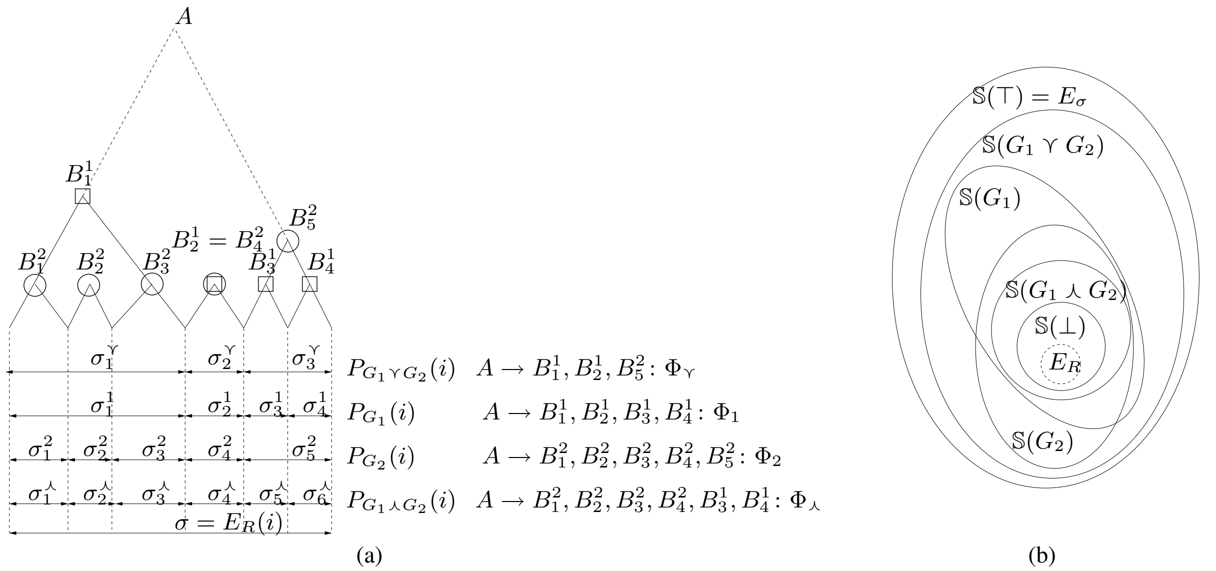


Figure 4: (a) Example of computing lub and glb; (b) Grammar semantics reduced to the sublanguage E_σ

Theorem 2. *The system $\mathfrak{L} = \langle \mathcal{L}, \succcurlyeq \rangle$ together with the lub and glb operators computed by Algorithms 1 and 2, forms a complete lattice.*

Proof. Besides the property given in Lemma 5, lub and glb operators are computed w.r.t. (5) (see Figure 2(b)), such that we have $ts(\curlyvee_{G\in\mathcal{L}} G) = \bigcap_{G\in\mathcal{L}} ts(G) = ts(\top)$, $bs(\curlywedge_{G\in\mathcal{L}} G) = \bigcap_{G\in\mathcal{L}} bs(G) = bs(\bot)$, which gives the uniqueness of $\top$ and $\bot$ elements. $\square$

Similar to the subsumption relation, $\succcurlyeq$, the lub, $\curlyvee$ and glb, $\curlywedge$ operators are semantic-based.

Theorem 3. *In the complete lattice $\mathfrak{L} = \langle \mathcal{L}, \succcurlyeq \rangle$, $\forall G_1, G_2 \in \mathcal{L}$ we have:*

$$\begin{aligned} &\mathbb{S}(G_1 \curlyvee G_2) \supseteq \mathbb{S}(G_1) \cup \mathbb{S}(G_2). \\ &\mathbb{S}(G_1 \curlywedge G_2) \subseteq \mathbb{S}(G_1) \cap \mathbb{S}(G_2) \end{aligned} \tag{6}$$

Proof. The proof is straightforward from Theorem 1 and Lemma 5. $\square$

Thus, the complete grammar lattice is semantic-based (see Figure 4(b)). It is straightforward to prove that the complete grammar lattice $\mathfrak{L} = \langle \mathcal{L}, \succcurlyeq \rangle$ has all the known properties (i.e., idempotency, commutativity, associativity, absorption, $\top$ and $\bot$ laws, distributivity). In proving these properties, it is crucial that Procedure `Op` always computes the constraints $\Phi_\curlyvee$ and $\Phi_\curlywedge$ at the end, and thus they are independent of the order in which the operators are applied.

6 Grammar Induction

According to the assumption from Section 3, the rules corresponding to the grammar preterminals (POS) are given (i.e., $T_G(\emptyset)$). Thus, for a given representative example set, E_R, we can construct the grammar $\bot$ using a robust, active chart parser [Kay73] ($P_\bot = Bottom(E_R)$) (Definition 2, p7, and [MMK05, pg. 14, 23]). In order to build the $\top$ element, we need to apply the grammar generalization procedure starting from the $\bot$ element. This requires knowledge of the sublanguage E_σ, because the partial order of the grammar lattice is semantic-based ($\mathbb{S}(G) = L_\sigma(G) \cap E_\sigma$).

The grammar generalization is determinate if the rule generalization step is determinate.

Definition 16. A grammar rule $r'_A \in P_G$ is *determinate generalizable* by the rule generalization step (3) if $\exists\beta \in rhs(r'_A)$ and $\exists!\ r_B = B \rightarrow \beta\colon \Phi_B$ (i.e., one and only one rule r_B), s.t. $r'_A \overset{r_B}{\dashv} r_A$ with $\mathbb{S}(r'_A) \subset \mathbb{S}(r_A)$. We use the notation $r'_A \overset{1\subset}{\dashv} r_A$ for the determinate generalization step with semantic increase.

Definition 17. A LWFG G is *conform* w.r.t. a sublanguage $E_\sigma \subseteq L_\sigma(G)$ iff G is normalized and unambiguous w.r.t. E_σ and the rule derivation step (2) is determinate ($\exists! r_B$) and guarantees the decrease in rule semanitcs ($\mathbb{S}(r_A) \supset \mathbb{S}(r'_A)$) for all grammars derived from G. We use the notation $r_A \overset{1\supset}{\vdash} r'_A$ for the determinate derivation step with semantic decrease (rule r_B is unique and thus not specified).

As a consequence, the only rule generalization steps (3) allowed in the grammar induction process, are those which guarantee the same semantic relation $\mathbb{S}(r'_A) \subset \mathbb{S}(r_A)$ [12], which assures that all the generalized grammars belong to the grammar lattice. We use the notation $r'_A \overset{r_B\subset}{\dashv} r_A$ for the generalization step with semantic increase (it can be nondeterminate, and thus r_B must be specified).

Definition 18. In a LWFG $\top$ conform w.r.t. a sublanguage E_σ, we call *chain*, a set of ordered rules $chain_\top = \{B_k \rightarrow B_{k-1}\colon \Phi_{k\top}, \dots, B_2 \rightarrow B_1\colon \Phi_{2\top}, B_1 \rightarrow \beta\colon \Phi_{1\top}\}$, such that $B_k \succ \cdots \succ B_2 \succ B_1$. All the chain rules, but the last, are unary branching rules. The last rule is the minimal chain rule.

The rules of a chain must ground derive equivalent representative syntagmas $\sigma_{B_k} \equiv \cdots \equiv \sigma_{B_1}$ (see Figure 5(a)), i.e., syntagmas that have the same string and the same semantic representation, but different categories (see Definition 2, p7 and [MMK05, pg. 19]).

For the $\perp$ grammar of a lattice that has $\top$ as its top element, the aforementioned chain becomes $chain_\perp = \{B_k \rightarrow \beta_\perp\colon \Phi_{k\perp}, \dots, B_2 \rightarrow \beta_\perp\colon \Phi_{2\perp}, B_1 \rightarrow \beta_\perp\colon \Phi_{1\perp}\}$, where $\beta_\perp$ contains only preterminals and the rule order is unknown. By the parsing preserving property of the rule derivation step, the same equivalent representative syntagmas can be ground derived from the $chain_\perp$ rules (see Figure 5(b)).

We denote by $chain = \{r_k, \dots, r_2, r_1\}$, one or more chains in any lattice grammar, where the rule order is unknown. The minimal chain rules, $r_m = min(chain)$, can always be determined if $r_m \in chain$ s.t. $\forall r \in chain - \{r_m\} \wedge r_m \overset{r}{\dashv} r_{mg}$ we have that $\mathbb{S}(r_m) = \mathbb{S}(r_{mg})$. By the consequence of the conform property, the generalization step $r_m \overset{r=}{\dashv} r_{mg}$ is not allowed, since it does not produce any increase in rule semantics. That is, a minimal chain rule cannot be generalized by any other chain rule, with an increase in its semantics.

Given $chain_\perp$ and the aforementioned property of the minimal chain rules, we can recover $chain_\top$ by Procedure `chains_recovery`.

Lemma 6. *Given a LWFG $\top$ conform w.r.t. a sublanguage E_σ, for any grammar G derived from $\top$, all rules are determinate generalizable if all chains of the grammar $\top$ (i.e., all $chain_\top$) are known (e.g., recovered by Procedure* `chains_recovery`*).*

Proof. The only case of rule generalization step nondeterminism, with semantic increase, is introduced by the derivation of the unary branching rules of ordered $chain_\top$, which yields the unordered $chain_\perp$, where $B_i \rightarrow \beta_\perp \overset{B_j \rightarrow \beta_\perp \subset}{\dashv} B_i \rightarrow B_j$, holds for all $B_j \prec B_i$. Thus, keeping (or recovering) the ordered $chain_\top$ in any grammar G derived from $\top$, all the other grammar rules are determinate generalizable. □

Algorithm 4 builds the lattice $\top$ element, $\top \leftarrow Top(E_R, E_\sigma)$. In step 1, the Procedure `chains_recovery` detects all $chain = chain_\perp$, which contain rules with identical right-hand side. In step 2, all $chain_\perp$ rules are transformed in $chain_\top$ form, by generalizing them through the minimal chain rule (see also Figure 5(c)). The generalization step $r \overset{r_m\subset}{\dashv} r_g$ guarantees the semantic increase $\mathbb{S}(r_g) \supset \mathbb{S}(r)$ for all the rules r which

[12] This property allows the grammar induction based only on positive examples.

are generalized through r_m, thus being the inverse of the rule derivation step in the grammar lattice. The rules r are either chain rules, or rules having the same left hand side as the chain rules. The returned set $P_\perp$ contains all $chain_\top$ unary branching rules of the $\top$ grammar. Therefore, in Algorithm 4 the set $P_\top$ initially contains determinate generalizable rules, and the "while loop" can determinately generalize all the grammar rules. An example showing the full trace of Procedure `chains_recovery` is given in Appendix A.

Algorithm 4: Top(E_R, E_σ)

$P_\perp \leftarrow Bottom(E_R)$
$P_\top \leftarrow$ `chains_recovery` $(P_\perp, E_R, E_\sigma)$ /* $P_\top$ is determinate generalizable */
while $\exists r \in P_\top \; s.t. \; r \overset{1\subset}{\dashv} r_g$ **do**
  $r \leftarrow r_g$
return $P_\top$

Procedure `chains_recovery`$(P_\perp, E_R, E_\sigma)$

while $E_R \neq \emptyset$ **do**
1 $\sigma \leftarrow first(E_R)$
  $chain \leftarrow \{r \in P_\perp | r \overset{*\perp}{\Rightarrow} \sigma_r \wedge \sigma_r \equiv \sigma\}$ /* $chain = chain_\perp$ */
  $lhs\_chain \leftarrow \{lhs(r) | r \in chain\}$
  $E_R \leftarrow E_R - \{\sigma_r \in E_R | r \in chain \wedge r \overset{*\perp}{\Rightarrow} \sigma_r\}$
2 **while** $|chain| > 1$ **do**
    /* chain recovery */
    $r_m \leftarrow min(chain)$ /* r_m cannot be generalized with semantic increase */
    $chain \leftarrow chain - \{r_m\}$
    $lhs\_chain \leftarrow lhs\_chain - \{lhs(r_m)\}$
    foreach $r \in P_\perp \wedge lhs(r) \in lhs\_chain \; s.t. \; r \overset{r_m\subset}{\dashv} r_g$ **do**
      $r \leftarrow r_g$
return $P_\perp$ /* The returned $P_\perp$ contains all $chain_\top$ */

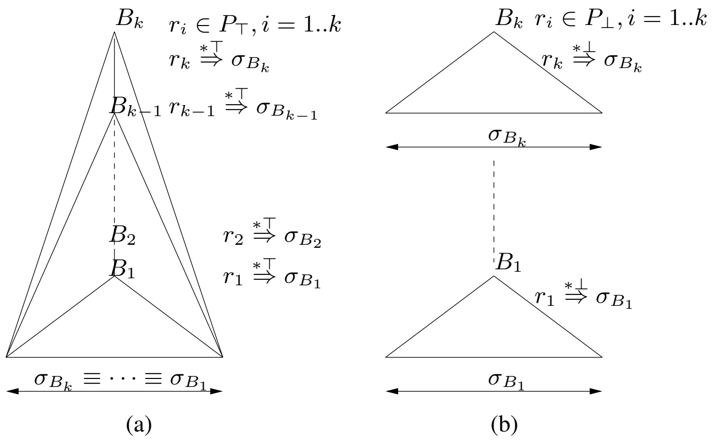
Theorem 4. *If E_R is the set of representative examples associated with a LWFG G conform w.r.t. a sublanguage $E_\sigma \supseteq E_R$, then the procedure Top(E_R, E_σ) computes the lattice $\top$ element such that $\top = G$.*

Proof. Since G is normalized (Definition 12), none of its rule can be generalized with increase in semantics. Starting with the $\perp$ element, after applying Procedure `chains_recovery`, all rules that can be generalized, with increase in semantics, through the rule generalization step (3), are determinate generalizable (Lemma 6, Definition 16, 17) . Since the rule generalization step and the rule derivation step are the inverse of each other, the process of grammar generalization from $\perp$ to $\top$ is the inverse of the derivation process from G to $\perp$, which is finite. This means that regardless of the grammar sequence $\perp, G_1, \ldots, G_n, \top$, there is a derivation process that yields the inverse sequence $G, G_n, \ldots, G_1, \perp$. Since $\mathbb{S}(G) = E_\sigma$ and $\mathbb{S}(G_i)$ are increasing (because G is conform to E_σ), the generalization process ends at the semantic limit $\mathbb{S}(\top) = E_\sigma$, and thus $\top = G$. $\square$

If the hypothesis of Theorem 4 holds, then any grammar induction algorithm that uses the complete lattice search space can converge to the lattice top element, using different search strategies.

7 Conclusions

In this paper we have presented the theoretical foundation of the search space for the grammar induction problem. We defined a set of grammars derived from a normalized LWFG, G, which preserve the parsing of



$$
\begin{array}{cccc}
B_k \rightarrow \beta_\perp & B_k \rightarrow B_1 & \dots & \mathbf{B_k \rightarrow B_{k-1}} \\
B_{k-1} \rightarrow \beta_\perp & B_{k-1} \rightarrow B_1 & \dots & \\
\vdots & \vdots & & \\
B_2 \rightarrow \beta_\perp & \mathbf{B_2 \rightarrow B_1} & & \\
\mathbf{B_1} \rightarrow \beta_\perp & & &
\end{array}
$$

(c)

Figure 5: (a,b) Parsing trees for chain rules (in $chain_\top$ and $chain_\perp$, respectively); (c) The iterations of step 2 in Procedure `chains_recovery` ($chain_\top$ contains the diagonal rules)

the representative examples set, E_R and whose semantics is reduced to a sublanguage E_σ. We proved that this set of grammars forms a semantic-based complete lattice, which has as its top element the grammar G. The set of representative examples E_R and the sublanguage E_σ are key elements in defining the complete grammar lattice. We proved that given E_R and E_σ, the lattice top element can always be learned, as a LWFG conform w.r.t. E_σ.

The search space formed by the complete grammar lattice can guarantee the convergence of all the grammar learning algorithm solutions, if they obey the conform grammar properties defined in this paper. This result is important for developing efficient relational learning algorithms for LWFG induction.

References

[CN03] Peter Culicover and Andrzej Nowak. *Dynamical Grammar: Minimalism, Acquisition, and Change*. Oxford University Press, 2003.

[Col99] Michael Collins. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, 1999.

[Kay73] Martin Kay. The MIND System. In Randall Rustin, editor, *Natural Language Processing*, pages 155–188. Algorithmics Press, New York, 1973.

[KM04] Dan Klein and Christopher D. Manning. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedings of the ACL'04*, 2004.

[MMK04] Smaranda Muresan, Tudor Muresan, and Judith Klavans. Inducing constraint-based grammars from a small semantic treebank. In *Proceedings of AAAI Spring Symposium on Language Learning: An Interdisciplinary Perspective*, Stanford University, 2004.

[MMK05] Smaranda Muresan, Tudor Muresan, and Judith Klavans. Lexicalized Well-Founded Grammars: Learnability and merging. Technical Report CUCS-027-05, Columbia University, New York, NY, 2005.

[Pin89] Steven Pinker. *Learnability and Cognition: The Acquisition of Argument Structure*. MIT Press, 1989.

[PS84] Fernando C. Pereira and Stuart M. Shieber. The semantics of grammar formalisms seen as computer languages. In *Proceeding of the ACL'84*, pages 123–129, 1984.

[PW80] Fernando C. Pereira and D.H.D Warren. Definite Clause Grammars for language analysis. *Artificial Intelligence*, 13:231–278, 1980.

[SSP95] Stuart Shieber, Yves Schabes, and Fernando Pereira. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24(1-2):3–36, 1995.

[Tar55] A. Tarski. Lattice-theoretic fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[vEK76] M. van Emden and R. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 4:733–742, 1976.

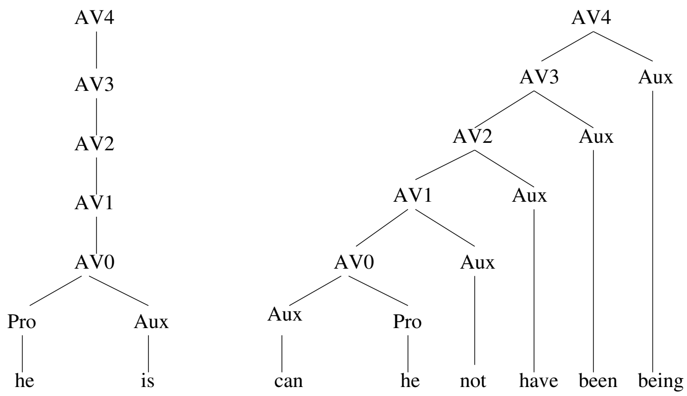
[Win99] S. Wintner. Compositional semantics for linguistic formalisms. In *Proceedings of the ACL'99*, 1999.

A Example

Table 1 shows the trace of Procedure `chains_recovery` for auxiliary verb constructions. It can be noticed that at the end, all the rules are determinate generalizable. For simplicity, only the strings corresponding to E_R are shown (i.e., their semantic molecules are not given; examples of E_R with semantic molecules are presented in [MMK05, pg. 38]). The constraints Φ, which are not shown, are computed at each steps for the generalized rules. The sublanguage E_σ used for the generalization process is not given. As a note, the Pro nonterminal will be furthered generalized to Sbj. Figure 6 shows examples of parse trees corresponding to the ground derivation in the grammar returned by Procedure `chains_recovery`.

E_R	$P_\perp$	*Generalized Rules at:*			
		Iteration 1	*Iteration 2*	*Iteration 3*	*Itereration 4*
he is	* $\mathbf{AV0} \rightarrow \mathbf{Pro}, \mathbf{Aux}$				
is he	$AV0 \rightarrow Aux, Pro$				
he is	* $AV1 \rightarrow Pro, Aux$	* $\mathbf{AV1} \rightarrow \mathbf{AV0}$			
he is not	$AV1 \rightarrow Pro, Aux, Aux$	$AV1 \rightarrow AV0, Aux$			
he is	* $AV2 \rightarrow Pro, Aux$	* $AV2 \rightarrow AV0$	* $\mathbf{AV2} \rightarrow \mathbf{AV1}$		
he can be	$AV2 \rightarrow Pro, Aux, Aux$	$AV2 \rightarrow AV0, Aux$	$AV2 \rightarrow AV1, Aux$		
he is	* $AV3 \rightarrow Pro, Aux$	* $AV3 \rightarrow AV0$	* $AV3 \rightarrow AV1$	* $\mathbf{AV3} \rightarrow \mathbf{AV2}$	
he has been	$AV3 \rightarrow Pro, Aux, Aux$	$AV3 \rightarrow AV0, Aux$	$AV3 \rightarrow AV1, Aux$	$AV3 \rightarrow AV2, Aux$	
he is	* $AV4 \rightarrow Pro, Aux$	* $AV4 \rightarrow AV0$	* $AV4 \rightarrow AV1$	* $AV4 \rightarrow AV2$	* $AV4 \rightarrow AV3$
he is being	$AV4 \rightarrow Pro, Aux, Aux$	$AV4 \rightarrow AV0, Aux$	$AV4 \rightarrow AV1, Aux$	$AV4 \rightarrow AV2, Aux$	$AV4 \rightarrow AV3, Aux$

Table 1: A trace of Procedure `chains_recovery` for auxiliary verb constructions with 4 iterations. Chain rules are marked with *, and the minimal chain rules are in bold.



(a) (b)

Figure 6: (a) Chain rule deriving a syntagma from E_R; (b) Non-chain rule deriving a syntagma from E_σ