Learning for Deep Language Understanding

Smaranda Muresan
School of Communication and Information, Rutgers University
New Brunswick, NJ, USA
smuresan@rutgers.edu

Abstract

The paper addresses the problem of learning to parse sentences to logical representations of their underlying meaning, by inducing a syntactic-semantic grammar. The approach uses a class of grammars which has been proven to be learnable from representative examples. In this paper, we introduce tractable learning algorithms for learning this class of grammars, comparing them in terms of a-priori knowledge needed by the learner, hypothesis space and algorithm complexity. We present experimental results on learning tense, aspect, modality and negation of verbal constructions.

1 Introduction

Recently, several machine learning approaches have been proposed for mapping sentences to their formal meaning representations [Ge and Mooney, 2005; Zettlemoyer and Collins, 2005; He and Young, 2006; Wong and Mooney, 2007; Poon and Domingos, 2009; Muresan, 2008; Kwiatkowksi *et al.*, 2010]. These approaches differ in the meaning representation languages they use and the integration, or lack thereof, of the meaning representations with grammar formalisms. λ-expressions and Combinatory Categorial Grammars (CCGs) [Steedman, 1996] are used by [Zettlemoyer and Collins, 2005; Kwiatkowksi *et al.*, 2010], and ontology-based representations and Lexicalized Well-Founded Grammars (LWFGs) [Muresan, 2006; Muresan and Rambow, 2007] are used by [Muresan, 2008].

An advantage of the LWFG formalism, compared to most grammar formalisms for deep language understanding such as Head-Driven Phrase Structure Grammar (HPSG) [Pollard and Sag, 1994], Tree Adjoining Grammars (TAGs) [Joshi and Schabes, 1997], Lexical Functional Grammars (LFGs) [Bresnan, 2001], is that it is accompanied by a learnability guarantee, the search space for LWFG induction being a complete grammar lattice [Muresan, 2006; Muresan and Rambow, 2007]. Some classes of CCGs has been proven to be learnable, however, to our knowledge no tractable learning algorithms have been proposed. Moreover, LWFG is suited to learning in data-poor settings. Building large treebanks annotated with complex representations needed to learn deep grammars is a very time consuming task, which cannot be performed easily for a wide range of genres and domains.[1]

In this paper we introduce tractable algorithms for LWFG induction. The theoretical learning model proposed by Muresan and Rambow [2007] characterizes the importance of substructures in the model not simply by frequency, but rather linguistically, by defining a notion of "representative examples" that drives the acquisition process. Informally, representative examples are "building blocks" from which larger structures can be inferred via reference to a larger generalization corpus [Muresan and Rambow, 2007]. The first algorithm assumes that the learner has access to an ordered set of representative examples. In the second algorithm introduced in this paper, the learner does not have a-priori knowledge of the order of the examples, nor of which are the representative examples, learning being done from the entire generalization corpus. We compare these algorithms in terms of their complexity and hypothesis space for grammar induction. We present an experiment of learning tense, aspect, modality and negation of verbal constructions using these algorithms, and show they cover well-established benchmarks for these phenomena developed for deep linguistic formalisms. Proposing tractable learning algorithms for a deep linguistic formalism and learning methods from small amount of training data, opens the door for large scale deep language understanding.

In Section 2, we give a background on Lexicalized Well-Founded Grammars [Muresan and Rambow, 2007], including their formal definition. In Section 3, we present our two grammar learning algorithms, as well as the definition of a LWFG parser, which is used as an innate inference engine during learning. In Section 4, we compare the two algorithms in terms of complexity and hypothesis space for grammar induction. In Section 5, we describe our experiment for learning tense, aspect, modality and negation of verbal constructions. We conclude in Section 6.

2 Lexicalized Well-Founded Grammar

Lexicalized Well-Founded Grammar (LWFG) is a recently developed formalism for deep language understanding that

[1]Statistical syntactic parsers for CCGs [Hockenmaier and Steedman, 2002; Clark and Curran, 2007] have been learned from CCG-Bank [Hockenmaier and Steedman, 2007], a large treebank derived from the Penn Treebank [Marcus *et al.*, 1994].

balances expressiveness with provable learnability results [Muresan, 2006; Muresan and Rambow, 2007; Muresan, 2010]. Formally, Lexicalized Well-Founded Grammar is a type of Definite Clause Grammars (Pereira and Warren, 1980) that is decidable and learnable. In LWFG, there is a partial ordering relation among nonterminals, which allows LWFG learning from a small set of examples. Grammar nonterminals are augmented with strings and their syntactic-semantic representations, called *semantic molecules*, and grammar rules can have two types of constraints, one for semantic composition and one for semantic interpretation. Thus, LWFGs are a type of syntactic-semantic grammars.

A *semantic molecule* associated with a natural language string w, is a syntactic-semantic representation, $w' = \binom{h}{b}$, where h (head) encodes compositional information (e.g., the string syntactic category cat), while b (body) is the actual semantic representation of the string w, which in LWFG is ontology-based — the variables are either concepts or slots identifiers in an ontology. An example of a semantic molecule for the noun phrase *formal proposal* is given below:

$$(\text{formal proposal}, \begin{pmatrix} {}_h\begin{bmatrix} \text{cat} & \text{n1} \\ \text{nr} & \text{sg} \\ \text{head} & X \end{bmatrix} \\ {}_b\left\langle X_1.\text{isa = formal},\ X.Y{=}X_1,\ X.\text{isa=proposal} \right\rangle \end{pmatrix})$$

When associated with lexical items, the semantic molecules are called *elementary semantic molecules*.

In addition to a lexicon, a LWFG has a set of constraint grammar rules, where the nonterminals are augmented with pairs of strings and their semantic molecules. These pairs are called syntagmas, and denoted by $\sigma = (w, w') = (w, \binom{h}{b})$. LWFG rules have two types of constraints, one for semantic composition (Φ_c) — defines how the meaning of a natural language expression is composed from the meaning of its parts — and one for semantic interpretation (Φ_i) — validates the semantic constructions based on a given ontology. An example of a LWFG rule for a simple noun phrase is given below:

$$N1(w, \tbinom{h}{b}) \rightarrow Adj(w_1, \tbinom{h_1}{b_1}), Noun(w_2, \tbinom{h_2}{b_2}) \colon \Phi_c(h, h_1, h_2), \Phi_i(b).$$

The composition constraints Φ_c are applied to the heads of the semantic molecules and are a simplified version of "path equations" [Shieber *et al.*, 1983] (see Figure 3 for examples). These constraints are learned together with the grammar rules [Muresan, 2010]. The semantic interpretation constraints, Φ_i, represent the validation based on an ontology and are not learned. Φ_i is a predicate which can succeed or fail — when it succeeds, it instantiates the variables of the semantic representation with concepts/slots in the ontology. For example, given the phrase *formal proposal*, Φ_i succeeds and returns ⟨ X_1.isa=formal, X.manner=X_1, X.isa=proposal ⟩, while given the phrase *fair-hair proposal* it fails. The semantic interpretation constraint, Φ_i serves as a local semantic interpreter at the grammar rule level.

Muresan and Rambow [2007] formally defined LWFGs. We present below their formal definition, slightly changed by Muresan [2010], and introduce other key concepts that are needed for presenting the learning algorithms. We keep their notation for consistency.

Definition 1. *A Lexicalized Well-Founded Grammar (LWFG) is a 7-tuple, $G = \langle \Sigma, \Sigma', N_G, \succeq, P_G, P_\Sigma, S \rangle$, where:*

1. *Σ is a finite set of terminal symbols.*
2. *Σ' is a finite set of elementary semantic molecules corresponding to the terminal symbols.*
3. *N_G is a finite set of nonterminal symbols. $N_G \cap \Sigma = \emptyset$. We denote $pre(N_G) \subseteq N_G$, the set of pre-terminals (a.k.a, parts of speech)*
4. *$\succeq$ is a partial ordering relation among nonterminals.*
5. *P_G is the set of constraint grammar rules. A rule is written $A(\sigma) \rightarrow B_1(\sigma_1), \ldots, B_n(\sigma_n) \colon \Phi(\bar{\sigma})$, where $A \in (N_G - pre(N_G))$, $B_i \in N_G$, $\bar{\sigma} = (\sigma, \sigma_1, ..., \sigma_n)$ such that $\sigma = (w, w'), \sigma_i = (w_i, w_i'), 1 \leq i \leq n, w = w_1 \cdots w_n, w' = w_1' \circ \cdots \circ w_n'$, and $\circ$ is the composition operator for semantic molecules.[2] For brevity, we denote a rule by $A \rightarrow \beta \colon \Phi$, where $A \in (N_G - pre(N_G)), \beta \in N_G^+$.*
6. *P_Σ is the set of constraint grammar rules whose left-hand side are pre-terminals, $A(\sigma) \rightarrow$, $A \in pre(N_G)$. We use the notation $A \rightarrow \sigma$ for these grammar rules.*
7. *$S \in N_G$ is the start nonterminal symbol, and $\forall A \in N_G, S \succeq A$ (we use the same notation for the reflexive, transitive closure of $\succeq$).*

In LWFG due to partial ordering among nonterminals we can have ordered constraint grammar rules and non-ordered constraint grammar rules (both types can be recursive or non-recursive). A grammar rule $A(\sigma) \rightarrow B_1(\sigma_1), \ldots, B_n(\sigma_n) \colon \Phi(\bar{\sigma})$, is an *ordered rule*, if for all B_i, we have $A \succeq B_i$. In LWFGs, each nonterminal symbol is a left-hand side in at least one ordered non-recursive rule and the empty string cannot be derived from any nonterminal symbol.

2.1 Derivation in LWFG

The derivation in LWFG is called ground syntagma derivation, and it can be seen as the bottom up counterpart of the usual derivation. Given a LWFG, G, the *ground syntagma derivation* relation, $\stackrel{*G}{\Rightarrow}$, is defined as: $\frac{A \rightarrow \sigma}{A \stackrel{*G}{\Rightarrow} \sigma}$ (if $\sigma = (w, w'), w \in \Sigma, w' \in \Sigma'$, i.e., A is a preterminal), and $\frac{B_i \stackrel{*G}{\Rightarrow} \sigma_i,\ i=1,\ldots,n, \quad A(\sigma) \rightarrow B_1(\sigma_1),\ldots,B_n(\sigma_n) \colon \Phi(\bar{\sigma})}{A \stackrel{*G}{\Rightarrow} \sigma}$. In LWFGs all syntagmas $\sigma = (w, w')$ derived from a nonterminal A have the same category of their semantic molecules w' ($h_A.cat = A$). This property is used for determining the left hand side nonterminal of the learned rule.

[2]Semantic molecule bodies are just concatenated $b = [b_1, \ldots, b_n]\nu$, where ν — variable substitution— is the most general unifier of b and $b_1, \ldots, b_n$, being a particular form of the commonly used substitution [Lloyd, 2003]. The composition of the semantic molecule heads is given by the composition constraints $\Phi_c(h, h_1, \ldots, h_n)$.

2.2 Language/Sublanguage in LWFG

The *set of all syntagmas* generated by *a grammar* G is $L_\sigma(G) = \{\sigma | \sigma = (w, w'), w \in \Sigma^+, \exists A \in N_G, A \stackrel{*G}{\Rightarrow} \sigma\}$. Given a LWFG G, $E_\sigma \subseteq L_\sigma(G)$ is called a sublanguage of G. Extending the notation, given a LWFG G, the set of syntagmas generated by a *rule* $(A \rightarrow \beta \colon \Phi) \in P_G$ is $L_\sigma(A \rightarrow \beta \colon \Phi) = \{\sigma | \sigma = (w, w'), w \in \Sigma^+, (A \rightarrow \beta \colon \Phi) \stackrel{*G}{\Rightarrow} \sigma\}$, where $(A \rightarrow \beta \colon \Phi) \stackrel{*G}{\Rightarrow} \sigma$ denotes the ground derivation $A \stackrel{*G}{\Rightarrow} \sigma$ obtained using the rule $A \rightarrow \beta \colon \Phi$ in the last derivation step.

Given a LWFG G and a sublanguage E_σ (not necessarily of G), $\mathbb{S}(G) = L_\sigma(G) \cap E_\sigma$ is the set of syntagmas generated by G *reduced to the sublanguage* E_σ. Given a grammar rule $r \in P_G$, $\mathbb{S}(r) = L_\sigma(r) \cap E_\sigma$ is the set of syntagmas generated by r *reduced to the sublanguage* E_σ. These concepts will be used during learning as performance criteria.

2.3 Chains

A property of Lexicalized Well-Founded Grammars is that the category of a nonterminal is the name of the nonterminal: $\forall A \in N_G$ we have $h_A.cat = A$. As a consequence, for unary branching rules, $A \rightarrow B \colon \Phi$, where $A, B \in N_G$, the syntagmas which are ground-derived from $(A \rightarrow B \colon \Phi) \stackrel{*G}{\Rightarrow} \sigma_A$ and $B \stackrel{*G}{\Rightarrow} \sigma_B$ have the same string w and the same semantic representation b, but have different semantic molecule heads $h_A \neq h_B$. These syntagmas are called equivalent and denoted by $\sigma_A \equiv \sigma_B$.

Informally, we can define a *chain* as a set of ordered unary branching rules: $\{B_k \rightarrow B_{k-1}, B_{k-1} \rightarrow B_{k-2}, \ldots, B_2 \rightarrow B_1, B_1 \rightarrow \beta\}$ such that syntagmas generated by these rules are equivalent. The notion of chain is similar to the concept of spines in Tree Adjoining Grammars [Carreras *et al.*, 2008], but not limited to just lexical anchors. The *chain nonterminal set* for equivalent syntagmas $\sigma_i = (w, w'_i)$ is the set of nonterminals in the chain, $chs(w) = \{B_k, \ldots, B_1, B_0\}$, where $B_k \succ B_{k-1} \cdots \succ B_1$ and $B_0 = \beta$, such that $B_i \stackrel{*G}{\Rightarrow} \sigma_i$ for $0 \leq i \leq k$. Chains and chain nonterminal sets will be used to generalize grammar rules during LWFG learning.

2.4 Representative Examples

The partial ordering relation among nonterminals makes the set of nonterminals well-founded, which allows the ordering of the grammar rules, as well as the ordering of the syntagmas generated by LWFGs. This allows the definition of the representative examples of a LWFG. Muresan and Rambow [2007] informally define the representative examples E_R of a LWFG, G, as the simplest syntagmas ground-derived by the grammar G. That is, for each grammar rule there exist a syntagma which is ground-derived from it in the minimum number of steps.

3 Grammar Induction Algorithms

The theoretical learning model for LWFG induction, Grammar Approximation by Representative Sublanguage (GARS), together with a learnability theorem was introduced in [Muresan and Rambow, 2007]. LWFG induction is formulated as an Inductive Logic Programming (ILP) learning problem $\langle \vdash, LB, LE, LH \rangle$ [Kietz and Džeroski, 1994], where the provability relation, $\vdash$, is given by robust parsing and denoted $\vdash_{rp}$[3]; the language of background knowledge,LB, is the set of LWFG rules that are already learned and the elementary syntagmas corresponding to the lexicon; the language of examples, LE are syntagmas of the generalization sublanguage, which are ground atoms; and the hypothesis language, LH, is a complete LWFG lattice which preserves the parsing of representative examples, E_R.

The GARS model, as defined by Muresan and Rambow [2007], takes as input a set of representative examples, E_R, and a generalization sublanguage E_σ (finite and conformal[4]), and learns a grammar G using the above ILP-setting, such that G is unique and $E_\sigma \subseteq L_\sigma(G)$. E_R is used to construct the most specific hypotheses (grammar rules), and thus all the grammars should be able to generate these representative examples, E_R (this property is equivalent to ILP consistency property, which states that all the learned hypotheses need to be consistent with all the positive examples, and none of the negative examples [Kietz and Džeroski, 1994]). The generalization sublanguage E_σ is used during generalization, only the most general grammar being able to generate the entire sublanguage.

In this section we describes two tractable algorithms for the GARS model. The first algorithm uses an *ordered set of representative examples* (examples in E_R are ordered from the simplest to the most complex) (see Section 3.2). The reader should remember that for a LWFG G, there exists a partial ordering among the grammar nonterminals, which allows a total ordering of the representative examples of the grammar G. Thus, in this algorithm, the learner has access to the ordered set when learning the grammar. In the second algorithm, the learner does not have a-priori knowledge of the order of the examples, nor of which are the representative examples, learning being done from the entire generalization sublanguage (see Section 3.3).

The assumption that all the algorithms rely on is that the rules corresponding to pre-terminals (P_Σ in Definition 1) are given, i.e., they are not learned. These rules can be constructed from the lexicon. Only the set of rules corresponding to nonterminals $N_G - pre(N_G)$ (i.e., P_G in Definition 1) are learned.

Both learning algorithms use a robust parser as an innate inference engine introduced in the next section.

3.1 Robust Parser as Innate Inference Engine for Grammar Induction

The LWFG parser is a bottom-up active chart parser [Kay, 1973] that is an *effectively r-reversible* program [Neumann and van Noord, 1994], i.e., it is capable of both parsing and generation and it is guaranteed to terminate.

[3] $\vdash_{rp}$ is equivalent to ground syntagma derivation $\stackrel{*G}{\Rightarrow}$.

[4] The property of conformal introduced in [Muresan and Rambow, 2007] is similar to the notion of structural completeness of a positive sample w.r.t. an automaton [Dupont *et al.*, 1994]. This allows learning based only on positive examples.

The semantics of the LWFG parser is given by the definitions presented below. Let us consider $(w, w') \in E_\sigma \subseteq L_\sigma(G)$ a syntagma derived by a grammar G, such that $w = w_1 \cdots w_n$ is a string, $w' = \binom{h}{b}$ is its semantic molecule, and $b = b_1 \cdots b_n$ is the string semantic representation. The parser is used both during the generation of candidate hypotheses (Definition 3), and during their evaluation in order to choose the best performing one (Definition 2).

Definition 2 (Parsing for performance criteria). *We define the set of syntagmas returned by the robust parser by:* $L_\sigma(w) = \{\sigma | \sigma = (w_{ij}, w'_{ij})$, *with* $w_{ij} = w_i w_{i+1} \cdots w_{j-1}$, $w_i, \ldots, w_{j-1} \in \Sigma,\ 1 \leq i \leq j \leq n+1, s.t.\ \exists A \in N_G,\ A \stackrel{*G}{\Rightarrow} \sigma\}$

Moreover, for all syntagmas $\sigma \in L_\sigma(w)$, the parser returns the ground derivation length, $gdl_G(\sigma)$, which is the minimum number of ground-derivation steps used to derive σ from grammar G.

Definition 3 (Parsing for hypothesis generation). *When both the string* (w) *and its semantic representation* (b) *are given, we define the set of syntagmas parsed by the robust parser by* $L_\sigma(w, b) = \{\sigma | \sigma = (w_{ij}, w'_{ij}), with\ w_{ij} = w_i \cdots w_{j-1}, w'_{ij} = \binom{h_{ij}}{b_{ij}}, b_{ij} = (b_i b_{i+1} \cdots b_{j-1})\nu_{ij},\ w_i, \ldots, w_{j-1} \in \Sigma,\ w'_i = \binom{h_i}{b_i} \in \Sigma', \ldots, w'_{j-1} = \binom{h_{j-1}}{b_{j-1}} \in \Sigma'\ 1 \leq i \leq j \leq n+1,\ s.t.\ \exists A \in N_G,\ A \stackrel{*G}{\Rightarrow} \sigma\}$

It can be noticed that the parser returns all the subsyntagmas (chunks), and thus it is robust, which is essential for hypothesis generation during learning.

3.2 Grammar Learning from Ordered Representative Examples

Algorithm 1 describes the LWFG induction based on an *ordered set of representative examples*. The algorithm visits each representative example in turn, learning a new grammar rule from the current representative example $\sigma \in E_R$ using the procedure LearnRule. This rule is added to the grammar rule set P_G (P_G is added to the background knowledge K, but for clarity, in the algorithm, we give G explicitly as an argument, and not part of K). The process continues until all the representative examples are covered.

Algorithm 1: Grammar_Induction(E_R, E_σ, K)

Data: Ordered representative examples, E_R
generalization sublanguage, E_σ
Background Knowledge, K
Result: Learned grammar (rules+constraints), P_G

$P_G \leftarrow \emptyset$
foreach $\sigma \in E_R$ **do**
  $r_\sigma \leftarrow$ LearnRule(σ, G, E_σ, K)
  $P_G \leftarrow P_G \cup \{r_\sigma\}$
return P_G

We describe below the process of learning a grammar rule from the current representative example (i.e., LearnRule procedure).

Procedure LearnRule(σ, G, E_σ, K)

Let $\sigma = (w, \binom{h}{b})$
1 **Most Specific Grammar Rule Generation**
a) $w \leftarrow \min(w_1 \ldots w_n)$ s.t $b = (b_1, \ldots, b_n)\nu$
  /* $w = w_1 \ldots w_n$, $(w_j, \binom{h_j}{b_j}) \in L_\sigma(w, b)$, $1 \leq j \leq n$; n is the min num of chunks given by robust parser */
b) $chs(w_j) = \{B_j | \sigma_j \equiv (w_j, \binom{h_j}{b_j}), B_j \stackrel{*G}{\Rightarrow} \sigma_j\}, 1 \leq j \leq n$
  $chain(w_j) = \{B_j^{K_j} \rightarrow B_j^{K_j - 1}, \ldots, B_j^1 \rightarrow B_j^0\}$
c) $r \leftarrow (A \rightarrow B_1^0, \ldots, B_n^0 \colon \Phi_r)$
2 **Grammar Rule Generalization**
for $j \leftarrow 1$ **to** n **do**
  $i \leftarrow 1$
  while $r \stackrel{B_j^i \rightarrow B_j^{i-1}}{\dashv} r_g \wedge (\mathbb{S}(r) \subset \mathbb{S}(r_g))$ **do**
    $i \leftarrow i + 1$
    $r \leftarrow r_g$
return r

Procedure LearnRule has two main steps: 1) generation of the most specific grammar rule r from the current representative example σ, and 2) generalization of this rule using as performance criterion the coverage of the generalization sublanguage E_σ.

Most specific rule generation. In order to generate the most specific grammar rule, the robust parser first produces the minimum number of chunks that cover $\sigma = (w, \binom{h}{b})$ — starting from the string w and its semantic representation b (see Definition 3). Let us assume that our current representative example is $\sigma = (formal\ proposal, \binom{h}{b})$, where the semantic molecule $\binom{h}{b}$ was given in Section 2. And let us assume that the background knowledge contains the pre-terminal rules P_Σ for the adjective *formal* and the noun *proposal* (i.e., $Adj \rightarrow (formal, \binom{h_1}{b_1})$; $Noun \rightarrow (proposal, \binom{h_2}{b_2}))$, as well as previously learned grammar rules and constraints P_G (e.g., $N1 \rightarrow Noun \colon \Phi_1$).[5] Using this background knowledge, the robust parser returns the chunks that cover the example, namely the adjective *formal*, and the noun *proposal*, respectively. Next, in step 1b, for each chunk w_j, the robust parser determines the set of nonterminals from which $\sigma_j = (w_j, w'_j)$ is ground-derived, i.e., $chs(w_j)$, as well as the corresponding chains. In our example, we have that syntagma corresponding to the adjective *formal* can be derived only from the pre-terminal Adj, while the noun *proposal* can be derived from the nonterminal $N1$ and the pre-terminal $Noun$. Thus, the only chain that we have is for the noun *proposal*, and it only contains one unary branching rule: $chain(proposal) = \{N1 \rightarrow Noun\}$. In step 1c, the most specific rule r is generated such that its left-hand side nonterminal is determined from the category annotated in the representative example σ, $h.cat = A$ and the

[5]For readability we only provide the context-free backbone of the grammar rules, and the semantic composition constraints. Notations and examples are similar to the ones used by Muresan and Rambow [2007] to exemplify the search space, for reference.

arguments of each nonterminal B_j^0 from its right-hand side are generalized. In our example, the most specific rule is $N1 \rightarrow Adj \;\; Noun\colon \Phi_2$, where the left-hand side nonterminal is given by the category of the representative example, in this case `n1` (see Section 2). The compositional constraints Φ_2 are learned as well (Muresan [2010] gave an algorithm for constraint learning, which we used in this paper).

Grammar Rule Generalization. In the second step, the rule r is generalized by unary branching rules to a new rule r_g, as long as $\mathbb{S}(r) \subset \mathbb{S}(r_g)$. We denote by $r \overset{B_j^i \to B_j^{i-1}}{\dashv} r_g$ the generalization of r by unary branching rule $B_j^i \rightarrow B_j^{i-1}$ in $chain(w_j)$. In other words, the most specific rule is generalized based on unary branching rules, obtaining thus a set of candidate grammar rules. The performance criterion in choosing the best grammar rule among these candidate hypotheses is the number of examples in the generalization sublanguage E_σ that can be parsed using the candidate grammar rule r_{gi} in the last ground derivation step, together with the previous learned rules, i.e., $|\mathbb{S}(r_{gi})|$. In our example, we obtain two candidate grammar rules: $r_{g1} = (N1 \rightarrow Adj \;\; Noun\colon \Phi_2)$ and $r_{g2} = (N1 \rightarrow Adj \;\; N1\colon \Phi_3)$. Given the generalization sublanguage E_σ={ *formal proposal, loud clear noise, the loud noise*} the learner will generalize to the recursive rule r_{g2}, since this rule covers more examples in E_σ.

3.3 General Grammar Learning

Algorithm 1 presented in the previous section requires the learner to have knowledge of the representative examples, and the "true" order of those examples (i.e., from simpler to more complex). In this section we introduce a new general algorithm for LWFG induction, which does not require such a-priori knowledge. This algorithm has big practical implications, since when modeling complex language phenomena it might not be realistic to provide the right order of examples, or even to know which are the representative examples, without relying on deep linguistic knowledge. Algorithm 2 is an iterative grammar induction algorithm, which learns based on a set of examples E^* that can be either an unordered representative example set E_R^u (when we have knowledge of the representative examples), or the entire generalization corpus E_σ (when we do not have such knowledge).

Initially, for each example $\sigma \in E^*$, a rule r_σ is learned using the LearnRule procedure (stage 1). Each rule covers at least the example from which it was learned (ILP consistency property), and we have that $|P_G| = |E^*|$, property which will remain true at each iteration step of the algorithm (stage 2).

At each iteration step (stage 2), the rule corresponding to the current example σ is deleted from the existing set of grammar rules ($P_{G^-} \leftarrow P_G - \{r_\sigma\}$). If by deletion of r_σ we have that $|\mathbb{S}(G^-)| = |\mathbb{S}(G)|$ (step 3) — i.e., there exist a more general rule $r_{\sigma'}$ that covers the current example σ, as well as the example it was learned from σ' — we keep in P_G only this more general rule, and we delete from E^* the example with the maximum ground derivation length (σ_{max}). In other words, we keep the most general rule in P_G and the simplest example in E^*. Otherwise, the algorithm (re)learns the rule

Algorithm 2: General_Grammar_Induction(E^*, E_σ, K)

Data: E^* can be either E_R^u or E_σ
generalization sublanguage E_σ
Background Knowledge, K
Result: Learned grammar (rules+constraints), P_G, Representative examples,E_R

$P_G \leftarrow \emptyset$
1 **foreach** $\sigma \in E^*$ **do**
  $r_\sigma \leftarrow$ LearnRule(σ, G, E_σ, K)
  $P_G \leftarrow P_G \cup \{r_\sigma\}$
2 **repeat**
  $O_G \leftarrow P_G$
  foreach $\sigma \in E^*$ **do**
    $P_{G^-} \leftarrow P_G - \{r_\sigma\}$ // $\sigma \in \mathbb{S}(r_\sigma)$
3   **if** $\mathbb{S}(G) = \mathbb{S}(G^-)$ **then**
      Let $\sigma' \neq \sigma$ s.t. $\sigma \in \mathbb{S}(r_{\sigma'})$ // $\sigma' \in \mathbb{S}(r_{\sigma'})$
      if $gdl_{G^-}(\sigma) \geq gdl_{G^-}(\sigma')$ **then**
        $\sigma_{max} \leftarrow \sigma$
      else
        $\sigma_{max} \leftarrow \sigma'$
      $E^* \leftarrow E^* - \{\sigma_{max}\}$
4   **else**
      $r_\sigma \leftarrow$ LearnRule(σ, G^-, E_σ, K)
      $P_G \leftarrow P_{G^-} \cup \{r_\sigma\}$
until $O_G = P_G$
$E_R \leftarrow E^*$
return P_G, E_R

from the current example σ based on all the other rules, using the procedure LearnRule, and then it adds the new rule to the set of grammar rules P_G (step 4). The algorithm iterates until the grammar learned during the last step is the same as the grammar learned in the previous step ($P_G = O_G$). At the end, E^* will contain just the representative examples.

Algorithm 2 is an instance of "theory revision", because after a grammar ("theory") is learned during the first stage, the grammar ("theory") is revised (by deleting and adding rules). In the next section we discuss the complexity of these two algorithms and the property of the hypothesis space.

4 Hypothesis Space and Algorithm Complexity

Muresan and Rambow [2007] have proven that the search space for LWFG induction is a complete grammar lattice, giving a learnability theorem which states that: if E_R is the set of representative examples associated with a LWFG G conformal with respect to a sublanguage $E_\sigma \supseteq E_R$, then G can always be learned from E_R and E_σ as the top element ($\top = G$) of the complete grammar lattice. In Figure 1 the complete grammar lattice is shown in blue, and we have $|P_G| = |E_R|$. This property of the search space holds for algorithms where the learner has knowledge of the representative examples, being it ordered or unordered.

However, in the case of Algorithm 2 when we learn directly from E_σ, the search space only converges to a complete grammar lattice (Figure 1). Initially when we have that $|P_G| = |E^*| > |E_R|$, the search space is not a complete

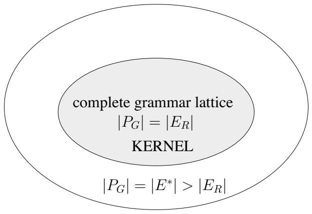

Figure 1: Hypothesis space

grammar lattice since we have grammar rules that do not preserve the parsing of the representative examples. For example, let us assume that we start with E_σ={loud clear noise, loud noise, noise} and we only have as background knowledge the pre-terminal rules. The grammar rule first learned by Algorithm 2 for the example *loud clear noise* would be $N1 \rightarrow Adj\ Adj\ Noun: \Phi$. This rule will not parse the representative example *loud noise*, and will be outside the complete grammar lattice, as given in [Muresan and Rambow, 2007]. However these rules will be eliminated (stage 2), and in the end only the rules which preserve the parsing of the representative examples are kept. Thus, the search space will converge to a complete grammar lattice ($|P_G| = |E_R|$). The algorithm is guaranteed to converge to the lattice top element ($\top$), since the rule generalization in LearnRule is the inverse of the rule specialization operator used by Muresan and Rambow [2007] to define the complete grammar lattice.

All algorithms introduced in this paper are polynomial. This result provides an advance in practical grammar learning for deep language understanding. While learnability results have been proven for some classes of Combinatorial Categorial Grammars [Steedman, 1996], to our knowledge no tractable learning algorithm has been proposed.

First, the procedure LearnRule is linear on the length of the learned rules and has the complexity O($|\beta| * max(|chs(w_j)|) * |E_\sigma| * |\sigma|^3$), where β is the right hand side of grammar rules. We assume a constant bound on the length of the grammar rules. Given this, the complexity of Algorithm 1 is $|E_R|$ multiplied with the complexity of the procedure LearnRule, i.e., O($|E_R| * |\beta| * max(|chs(w_j)|) * |E_\sigma| * |\sigma|^3$).

Algorithm 2 terminates after $|E_R|$ iterations in the worst case. The complexity of each iteration step is $|E^*|$ multiplied with the complexity of the procedure LearnRule, thus the overall complexity of Algorithm 2 is O($|E_R| * |E^*| * |\beta| * max(|chs(w_j)|) * |E_\sigma| * |\sigma|^3$).

Annotation Effort: Algorithm 1 and Algorithm 2 when $E^* = E_R^u$ require a small amount of annotation since only the representative examples E_R need to be fully annotated — the sublanguage E_σ used for generalization can be just weakly annotated (i.e., bracketed) or even unannotated. In turn, Algorithm 2 when $E^* = E_\sigma$, requires a larger annotation effort since the entire E_σ set needs to be fully annotated (i.e., utterances and their semantic molecules). To alleviate this effort, we have developed an annotation tool that integrates the parser and the lexicon in order to provide the user with the utterance and the semantic molecules of the chunks, so the user does not need to write the semantic representation by hand.

5 Experiments

An advantage of providing polynomial algorithms for LWFG learning is that instead of writing grammars for deep linguistic processing by hand, we can learn such grammars from examples. Capturing tense, aspect, modality and negation of verbal constructions has been one of the classic benchmarks for deep linguistic processing grammar formalisms, since modeling these phenomena is crucial for temporal reasoning and interpretation of facts and beliefs.

In our experiment, to learn sentence structures with subject and verbal constructions including auxiliary verbs — which capture tense, aspect, modality and negation —- we used 20 representative examples E_R (phrases annotated with their semantic molecules, similarly to the example in Section 2), and 106 examples in the generalization sublanguage E_σ. All examples were derived from Quirk et al.'s English grammar [Quirk *et al.*, 1972].

All algorithms converged to the same grammar, which contains 20 rules, including the compositional constraints, also learned. Convergence of the algorithms to the same target grammar is guaranteed by the theoretical results presented in Section 4.

In Figure 3, we show a sample from our learned grammar corresponding to auxiliary constructions dealing with modals, negation, subject-verb agreement, tense and aspect, as well as the corresponding learned constraints for two of the rules. We have 4 nonterminals for auxiliaries. $AV0$ models simple form of auxiliaries "be" and "have" as well as modal auxiliaries and the periphrastic auxiliary "do", together with subject agreement and inversion; $AV0$ is also used for modeling constructions with relative pronouns used either in questions or relative clause constructions; $AV1$ introduces negation; $AV2$ introduces the modals and future tense; $AV3$ introduces the perfect aspect; while $AV4$ introduces the progressive form of the auxiliary "to be", which will be used in conjunction with the passive constructions (e.g., "*she may have been being* examined by ..."). In this grammar we have the following chain of nonterminals $\{AV4, AV3, AV2, AV1, AV0\}$. Even if for this grammar of auxiliaries the rules of the grammar seem simple, and someone might wonder if they could have been written by hand, the complexity of the task is emphasized in the learned constraints. Moreover, the grammar becomes more complex as we introduce complex noun phrases that could generalize to Sbj, as well as other verbal constructions. In Figure 2, we show the representative examples used to learn the 8th and 14th grammar rules and constraints, respectively. The examples in the generalization sublanguage used for the 8th rule are {*someone is not, who is not, he is not, he can not, he has not*}.

To test our learned grammar we have used the Test Suites for Natural Language Processing (TSNLP) [Lehmann *et al.*, 1996], since it includes specific benchmarks for tense, aspect and modality (C_Tense_Aspect_Modality) and negation (C_Negation), and it has been used for testing grammars for deep linguistic processing. Using our learned grammar, our parser correctly recognized the 157 positive examples for tense, aspect and modality, and rejected all 38 ungrammatical examples. For negation, out of 289 positive examples

Sample Representative Examples
8. (someone is not, [cat:av1,stype:s,vtype:aux,vft:fin,int:no,dets:y,aux:be,neg:y,tense:pr,pers:(_,3),nr:sg,pf:no,pg:no,headS:X,head:Y]$\bowtie$ $\langle$X.isa=someone,Y.tense=pr,Y.neg=y $\rangle$)
14. (someone is being),[cat:av4,stype:s,vtype:aux,vft:fin,int:no,dets:y,aux:be,neg:no,tense:pr,pers:(_,3),nr:sg,pf:no,pg:y,headS:X,head:Y] $\bowtie$ $\langle$ X.isa=someone, Y.tense=pr,Y.pg=y $\rangle$)

Figure 2: Sample representative examples. The examples are represented as (w, h $\bowtie$ b) instead of (w,$\binom{h}{b}$)

Sample Learned Grammar P_G
$Sbj(\binom{h}{b}) \rightarrow Pro(\binom{h_1}{b_1}):\ \Phi_{c1}(h,h_1),\Phi_i(b)$
$Sbj(\binom{h}{b}) \rightarrow NNP(\binom{h_1}{b_1}):\ \Phi_{c2}(h,h_1),\Phi_i(b)$
$Sbj(\binom{h}{b}) \rightarrow WP(\binom{h_1}{b_1}):\ \Phi_{c3}(h,h_1),\Phi_i(b)$
$AV0(\binom{h}{b}) \rightarrow Sbj(\binom{h_1}{b_1}),\ Aux(\binom{h_2}{b_2}):\ \Phi_{c4}(h,h_1,h_2),\Phi_i(b)$
$AV0(\binom{h}{b}) \rightarrow Aux(\binom{h_1}{b_1}),\ Sbj(\binom{h_2}{b_2}):\ \Phi_{c5}(h,h_1,h_2),\Phi_i(b)$
$AV0(\binom{h}{b}) \rightarrow WP(\binom{h_1}{b_1}), Aux(\binom{h_2}{b_2}):\ \Phi_{c6}(h,h_1,h_2),\Phi_i(b)$
$AV1(\binom{h}{b}) \rightarrow AV0(\binom{h_1}{b_1}):\ \Phi_{c7}(h,h_1),\Phi_i(b)$
$AV1(\binom{h}{b}) \rightarrow AV0(\binom{h_1}{b_1}),\ Aux(\binom{h_2}{b_2}):\ \Phi_{c8}(h,h_1,h_2),\Phi_i(b)$
$AV2(\binom{h}{b}) \rightarrow AV1(\binom{h_1}{b_1}):\ \Phi_{c9}(h,h_1),\Phi_i(b)$
$AV2(\binom{h}{b}) \rightarrow AV1(\binom{h_1}{b_1}),\ Aux(\binom{h_2}{b_2}):\ \Phi_{c10}(h,h_1,h_2),\Phi_i(b)$
$AV3(\binom{h}{b}) \rightarrow AV2(\binom{h_1}{b_1}):\ \Phi_{c11}(h,h_1),\Phi_i(b))$
$AV3(\binom{h}{b}) \rightarrow AV3(\binom{h_1}{b_1}),\ Aux(\binom{h_2}{b_2}):\ \Phi_{c12}(h,h_1,h_2),\Phi_i(b)$
$AV4(\binom{h}{b}) \rightarrow AV3(\binom{h_1}{b_1}):\ \Phi_{c13}(h,h_1),\Phi_i(b)$
$AV4(\binom{h}{b}) \rightarrow AV3(\binom{h_1}{b_1}),\ Aux(\binom{h_2}{b_2}):\ \Phi_{c14}(h,h_1,h_2),\Phi_i(b)$

Sample learned compositional constraints Φ_c	
$\Phi_{c8}(h,h_1,h_2)$ =	{h.cat=av1,h.stype=h_1.stype, h.vtype=h_1.vtype,h.vtype=h_2.vtype, h.vft=fin,h_1.vft=fin,h.int=h_1.int, h.dets=h_1.dets,h.aux=h_1.aux,h.neg=h_2.neg, h.tense=h_1.tense,h.pers=h_1.pers,h.nr=h_1.nr, h.pf=h_1.pf,h.pg=h_1.pg,h.headS=h_1.headS, h.head=h_1.head,h.head=h_2.head,h_1.cat=av0, h_1.neg=no, h_2.cat=aux, h_2.aux=not}
$\Phi_{c14}(h,h_1,h_2)$ =	{h.cat=av4,h.stype=h_1.stype, h.vtype=h_1.vtype,h.vtype=h_2.vtype, h.vft=fin,h_1.vft=fin,h.int=h_1.int, h.dets=h_1.dets,h.aux=be,h_1.aux=be, h_2.aux=be,h.neg=h_1.neg, h.tense=h_1.tense,h.pers=h_1.pers,h.nr=h_1.nr, h.pf=h_1.pf,h.pg=h_2.pg,h.headS=h_1.headS, h.head=h_1.head,h.head=h_2.head,h_1.cat=av3, h_1.pg=no,h_2.cat=aux,h_2.vft=nfin}

Figure 3: Sample learned grammar rules and constraints for auxiliary verbs. pre(N_G)={Pro,NNP,WP, Aux}

G	She'll have been being seen .
G	He must have been succeeding .
G	She would not be being seen .
G	He will not have succeeded .
G	He might not have been seen .
UG	She may be being seeing .
UG	He did succeeding.
UG	He not would be succeeding .
UG	He not might have been being seen .

Table 1: Example of grammatical(G) and ungrammatical(UG) utterances, accepted and rejected by the learned grammar, repectively

the parser covered 288[6] and rejected all the negative examples (129). Example of correct utterances and ungrammatical utterances are given in Table 1.

Our parser returns the semantic representation of the utterance. For example, for the utterance *She might not have been being seen* the parser returns $\langle 1.isa = she, 2.mod = might, 2.neg = y, 2.tense = pr, 2.pg = y, 2.pf = y, 2.isa = see, 2.ag = 1\rangle$ (where *pg* and *pf* relates to verb's aspect: progressive and perfective, respectively; and *ag* refers to the semantic role *agent* of the verb *see*, which is *she*).

[6]The example not covered was *He not might have been succeeding*, which we believe should not be part of the grammatical examples, since it has negation before the modal *might*.

6 Conclusions and Future Work

We have described two polynomial algorithms for Lexicalized Well-Founded Grammar learning, which are guaranteed to learn the same unique target grammar. We have discussed these algorithms in terms of their search space, complexity and a priori knowledge that the learner needs to have (ordered vs. unordered representative examples, knowledge of the representative examples vs. absence of such knowledge). We have described an experiment of learning tense, aspect, modality and negation of verbal constructions using these algorithms, and show they covered well-established benchmarks for these phenomena developed for deep linguistic formalisms. Proposing tractable learning algorithms for a deep linguistic grammar formalism opens the door for large scale deep language understanding. Moreover, being able to learn from a small amount of data will enable rapid adaptation to different domains or text syles.

We are currently extending the grammar with probabilities in two ways: 1) adding probabilities at the rule level similar to other probabilistic grammars, and 2) modeling a weighted ontology (thus Φ_i behaves as a soft constraint, rather than a hard one).

Acknowledgements

The author acknowledges the support of the NSF (SGER grant IIS-0838801). Any opinions, findings, or conclusions are those of the author, and do not necessarily reflect the views of the funding organization.

References

[Bresnan, 2001] Joan Bresnan. *Lexical-Functional Syntax*. Oxford: Blackwell, 2001.

[Carreras *et al.*, 2008] Xavier Carreras, Michael Collins, and Terry Koo. Tag, dynamic programming and the perceptron for efficient, feature-rich parsing. In *Proceedings of CoNLL*, 2008.

[Clark and Curran, 2007] Stephen Clark and James R. Curran. Wide-coverage efficient statistical parsing with ccg and log-linear models. *Computational Linguistics*, 33(4), 2007.

[Dupont *et al.*, 1994] Pierre Dupont, Laurent Miclet, and Enrique Vidal. What is the search space of the regular inference? In R. C. Carrasco and J. Oncina, editors, *Proceedings of the Second International Colloquium on Grammatical Inference (ICGI-94)*, volume 862, pages 25–37, Berlin, 1994. Springer.

[Ge and Mooney, 2005] Ruifang Ge and Raymond J. Mooney. A statistical semantic parser that integrates syntax and semantics. In *Proceedings of CoNLL-2005*, 2005.

[He and Young, 2006] Yulan He and Steve Young. Spoken language understanding using the hidden vector state model. *Speech Communication Special Issue on Spoken Language Understanding in Conversational Systems*, 48(3-4), 2006.

[Hockenmaier and Steedman, 2002] Julia Hockenmaier and Mark Steedman. Generative models for statistical parsing with combinatory categorial grammar. In *ACL '02: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 335–342, 2002.

[Hockenmaier and Steedman, 2007] Julia Hockenmaier and Mark Steedman. Ccgbank: A corpus of ccg derivations and dependency structures extracted from the penn treebank. *Computational Linguistics*, 33(3):355–396, 2007.

[Joshi and Schabes, 1997] Aravind Joshi and Yves Schabes. Tree-Adjoining Grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, chapter 2, pages 69–124. Springer, Berlin,New York, 1997.

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

[Kietz and Džeroski, 1994] Jörg-Uwe Kietz and Sašo Džeroski. Inductive logic programming and learnability. *ACM SIGART Bulletin.*, 5(1):22–32, 1994.

[Kwiatkowksi *et al.*, 2010] Tom Kwiatkowksi, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. Inducing probabilistic ccg grammars from logical form with higher-order unification. In *Proceedings of EMNLP*, 2010.

[Lehmann *et al.*, 1996] Sabine Lehmann, Stephan Oepen, Sylvie Regnier-Prost, Klaus Netter, and et al. Tsnlp - test suites for natural language processing. In *Proceedings of COLING*, 1996.

[Lloyd, 2003] John W. Lloyd. *Logic for Learning: Learning Comprehensible Theories from Structured Data*. Springer, Cognitive Technologies Series, 2003.

[Marcus *et al.*, 1994] Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. The penn treebank: annotating predicate argument structure. In *HLT '94: Proceedings of the workshop on Human Language Technology*, pages 114–119, 1994.

[Muresan and Rambow, 2007] Smaranda Muresan and Owen Rambow. Grammar approximation by representative sublanguage: A new model for language learning. In *Proceedings of ACL*, 2007.

[Muresan, 2006] Smaranda Muresan. Learning constraint-based grammars from representative examples: Theory and applications. Technical report, PhD Thesis, Columbia University, 2006.

[Muresan, 2008] Smaranda Muresan. Learning to map text to graph-based meaning representations via grammar induction. In *Coling 2008: Proceedings of the 3rd Textgraphs Workshop*, pages 9–16, Manchester, UK, 2008.

[Muresan, 2010] Smaranda Muresan. A learnable constraint-based grammar formalism. In *Proceedings of COLING*, 2010.

[Neumann and van Noord, 1994] Günter Neumann and Gertjan van Noord. Reversibility and self-monitoring in natural language generation. In Tomek Strzalkowski, editor, *Reversible Grammar in Natural Language Processing*, pages 59–96. Kluwer Academic Publishers, Boston, 1994.

[Pollard and Sag, 1994] Carl Pollard and Ivan Sag. *Head-Driven Phrase Structure Grammar*. University of Chicago Press, Chicago, Illinois, 1994.

[Poon and Domingos, 2009] Hoifung Poon and Pedro Domingos. Unsupervised semantic parsing. In *Proceedings of EMNLP'09*, 2009.

[Quirk *et al.*, 1972] Randolph Quirk, Sidney Greenbaum, Geoffrey Leech, and Jan Svartvik. *A Grammar of Contemporary English*. Longman, 1972.

[Shieber *et al.*, 1983] Stuart Shieber, Hans Uszkoreit, Fernando Pereira, Jane Robinson, and Mabry Tyson. The formalism and implementation of PATR-II. In Barbara J. Grosz and Mark Stickel, editors, *Research on Interactive Acquisition and Use of Knowledge*, pages 39–79. SRI International, Menlo Park, CA, 1983.

[Steedman, 1996] Mark Steedman. *Surface Structure and Interpretation*. The MIT Press, 1996.

[Wong and Mooney, 2007] Yuk Wah Wong and Raymond Mooney. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL-2007)*, 2007.

[Zettlemoyer and Collins, 2005] Luke S. Zettlemoyer and Michael Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of UAI-05*, 2005.