A Probabilistic Generative Grammar for Semantic Parsing

Abulhair Saparov
Carnegie Mellon University
Machine Learning Department
Pittsburgh, P.A.
asaparov@cs.cmu.edu

Vijay Saraswat
IBM T.J. Watson
Research Center
Yorktown Heights, N.Y.
vijay@saraswat.org

Tom M. Mitchell
Carnegie Mellon University
Machine Learning Department
Pittsburgh, P.A.
tom.mitchell@cmu.edu

Abstract

We present a generative model of natural language sentences and demonstrate its application to semantic parsing. In the generative process, a logical form sampled from a prior, and conditioned on this logical form, a grammar probabilistically generates the output sentence. Grammar induction using MCMC is applied to learn the grammar given a set of labeled sentences with corresponding logical forms. We develop a semantic parser that finds the logical form with the highest posterior probability exactly. We obtain strong results on the GeoQuery dataset and achieve state-of-the-art F1 on Jobs.

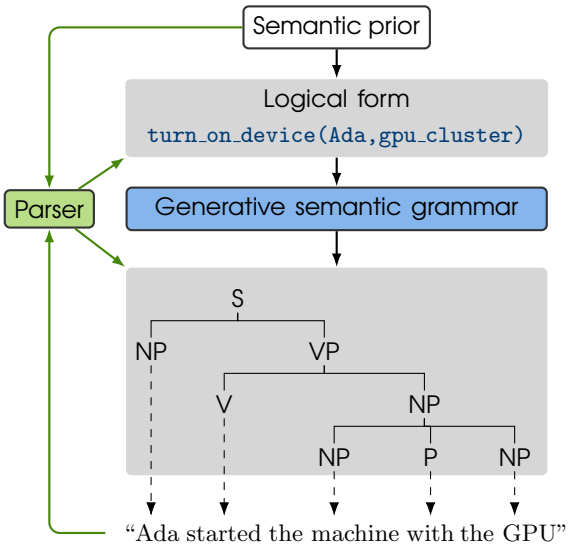

Figure 1: High-level illustration of the setting in which our grammar is applied in this paper. The dark arrows outline the generative process. During parsing, the input is the observed sentence, and we wish to find the most probable logical form and derivation given the training data under the semantic prior.

1 Introduction

Accurate and efficient semantic parsing is a long-standing goal in natural language processing. Existing approaches are quite successful in particular domains (Zettlemoyer and Collins, 2005, 2007; Wong and Mooney, 2007; Liang et al., 2011; Kwiatkowski et al., 2010, 2011, 2013; Li et al., 2013; Zhao and Huang, 2014; Dong and Lapata, 2016). However, they are largely domain-specific, relying on additional supervision such as a lexicon that provides the semantics or the type of each token in a set (Zettlemoyer and Collins, 2005, 2007; Kwiatkowski et al., 2010, 2011; Liang et al., 2011; Zhao and Huang, 2014; Dong and Lapata, 2016), or a set of initial synchronous context-free grammar rules (Wong and Mooney, 2007; Li et al., 2013). To apply the above systems to a new domain, additional supervision is necessary. When beginning to read text from a new domain, humans do not need to re-learn basic English grammar. Rather, they may encounter novel terminology. With this in mind, our approach is akin to that of (Kwiatkowski et al., 2013) where we provide domain-independent supervision to train a semantic parser on a new domain. More specifically, we restrict the rules that may be learned during training to a set that characterizes the general syntax of English. While we do not explicitly present and evaluate an open-domain semantic parser, we hope our work provides *a step* in that direction.

Knowledge plays a critical role in natural language understanding. Even seemingly trivial sentences may have a large number of ambiguous interpretations. Consider the sentence *"Ada started the machine with the GPU,"* for example. Without additional knowledge, such as the fact that "machine" can refer to computing devices that contain GPUs, or that computers generally contain devices such as

GPUs, the reader cannot determine whether the GPU is part of the machine or if the GPU is a device that is used to start machines. Context is highly instrumental to quickly and unambiguously understand sentences.

In contrast to most semantic parsers, which are built on discriminative models, our model is fully generative: To generate a sentence, the logical form is first drawn from a prior. A grammar then recursively constructs a derivation tree top-down, probabilistically selecting production rules from distributions that depend on the logical form (see Figure 1 for a high-level schematic diagram). The semantic prior distribution provides a straightforward way to incorporate background knowledge, such as information about the types of entities and predicates, or the context of the utterance. Additionally, our generative model presents a promising direction to *jointly* learn to understand and generate natural language.

This article describes the following contributions:

- In Section 2, we present our grammar formalism in its general form.
- Section 2.2 discusses aspects of the model in its application to the later experiments.
- In Section 3, we present a method to perform grammar induction in this model. Given a set of observed sentences and their corresponding logical forms, we apply Markov chain Monte Carlo (MCMC) to infer the posterior distributions of the production rules in the grammar.
- Given a trained grammar, we also develop a method to perform parsing in Section 4: to find the k-best logical forms for a given sentence, leveraging the semantic prior to guide its search.
- Using the GeoQuery and Jobs datasets, we demonstrate in Section 6 that this framework can be applied to create natural language interfaces for semantic formalisms as complex as Datalog/lambda calculus, which contain variables, scope ambiguity, and superlatives.

All code and datasets are available at `https://github.com/asaparov/parser`.

2 Semantic grammar

A grammar in our formalism operates over a set of nonterminals $\mathcal{N}$ and a set of terminal symbols $\mathcal{W}$. It can be understood as an extension of a context-free grammar (CFG) (Chomsky, 1956) where the generative process for the syntax is dependent on a logical form, thereby coupling syntax with semantics. In the top-down generative process of a derivation tree, a logical form guides the selection of production rules. Production rules in our grammar have the form $A \to B_1{:}f_1 \dots B_k{:}f_k$ where $A \in \mathcal{N}$ is a nonterminal, $B_i \in \mathcal{N} \cup \mathcal{W}$ are right-hand side symbols, and f_i are *semantic transformation functions*. These functions can encode how to "decompose" this logical form when recursively generating the subtrees rooted at each B_i. Thus, they enable semantic compositionality. An example of a grammar in this framework is shown in Figure 2, and a derivation tree is shown in Figure 3. Let $\boldsymbol{\mathcal{R}}$ be the set of production rules in the grammar and $\boldsymbol{\mathcal{R}}_A$ be the set of production rules with left-hand nonterminal symbol A.

$\mathrm{S} \to \mathrm{N}{:}\texttt{select\_arg1}\ \ \mathrm{VP}{:}\texttt{delete\_arg1}$
$\mathrm{VP} \to \mathrm{V}{:}\texttt{identity}\ \ \mathrm{N}{:}\texttt{select\_arg2}$
$\mathrm{VP} \to \mathrm{V}{:}\texttt{identity}$
$\mathrm{N} \to$ "tennis" $\quad \mathrm{V} \to$ "swims"
$\mathrm{N} \to$ "Andre Agassi" $\quad \mathrm{V} \to$ "plays"
$\mathrm{N} \to$ "Chopin"

Figure 2: Example of a grammar in our framework. This grammar operates on logical forms of the form *predicate(first argument, second argument)*. The semantic function `select_arg1` returns the first argument of the logical form. Likewise, the function `select_arg2` returns the second argument. The function `delete_arg1` removes the first argument, and `identity` returns the logical form with no change. In our use of the framework, the interior production rules (the first three listed above) are examples of rules that we specify, whereas the terminal rules and the posterior probabilities of *all* rules are learned via grammar induction. We also use a richer semantic formalism than in this example. Section 2.2 provides more detail.

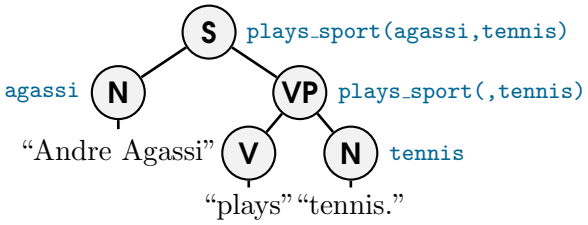

Figure 3: Example of a derivation tree under the grammar given in Figure 2. The logical form corresponding to every node is shown in blue beside the respective node. The logical form for V is `plays_sport(,tennis)` and is omitted above to reduce clutter.

2.1 Generative process

A *parse tree* (or *derivation*) in this formalism is a tree where every interior node is labeled

with a nonterminal symbol, every leaf is labeled with a terminal, and the root node is labeled with the root nonterminal S. Moreover, every node in the tree is associated with a logical form: let $x^{\boldsymbol{n}}$ be the logical form assigned to the tree node $\boldsymbol{n}$, and $x^{\mathbf{0}} = x$ for the root node $\mathbf{0}$.

The generative process to build a parse tree begins with the root nonterminal S and a logical form x. We *expand* S by randomly drawing a production rule from $\boldsymbol{\mathcal{R}}_S$, *conditioned* on the logical form x. This provides the first level of child nodes in the derivation tree. So if, for example, the rule $S \rightarrow B_1{:}f_1 \ldots B_k{:}f_k$ were drawn, the root node would have k child nodes, $\boldsymbol{n}_1, \ldots, \boldsymbol{n}_k$, respectively labeled $B_1, \ldots, B_k$. The logical form associated with each node is determined by the semantic transformation function: $x^{\boldsymbol{n}_i} = f_i(x)$. These functions describe the relationship between the logical form at a child node and that of its parent node. This process repeats recursively with every right-hand side nonterminal symbol, until there are no unexpanded nonterminal nodes. The sentence is obtained by taking the *yield* of the terminals in the tree (a concatenation).

The semantic transformation functions are specific to the semantic formalism and may be defined as appropriate to the application. In our parsing application, we define a domain-independent set of transformation functions (e.g., one function selects the left n conjuncts in a conjunction, another selects the n^{th} argument of a predicate instance, etc).

2.2 Selecting production rules

In the above description, we did not specify the distribution from which rules are selected from $\boldsymbol{\mathcal{R}}_A$. There are many modeling options available when specifying this distribution. In our approach, we choose a hierarchical Dirichlet process (HDP) prior (Teh et al., 2006). Every nonterminal in our grammar $A \in \mathcal{N}$ will be associated with an HDP hierarchy. For each nonterminal, we specify a sequence of *semantic feature functions*, $\{g_1, \ldots, g_m\}$, each of which return a discrete feature (such as an integer) of an input logical form x. We use this sequence of feature functions to define the hierarchy of the HDP: starting with the root node, we add a child node for every possible value of the first feature function g_1. For each of these child nodes, we add a grandchild node for every possible value of the second feature function g_2, and so forth. The result is a complete tree of depth m. Each node $\boldsymbol{n}$ in this tree is assigned a distribution $G^{\boldsymbol{n}}$ as follows:

$$G^{\mathbf{0}} \sim \mathrm{DP}(\alpha^{\mathbf{0}}, H), \qquad (1)$$
$$G^{\boldsymbol{n}} \sim \mathrm{DP}(\alpha^{\boldsymbol{n}}, G^{\pi(\boldsymbol{n})}),$$

where $\mathbf{0}$ is the root node, $\pi(\boldsymbol{n})$ is the parent of $\boldsymbol{n}$, α are a set of concentration parameters, and H is a base distribution over $\boldsymbol{\mathcal{R}}_A$. This base distribution is independent of the logical form x.

To select a rule in the generative process, given the logical form x, we can compute its feature values $(g_1(x), \ldots, g_m(x))$ which specify a unique path in the HDP hierarchy to a leaf node G^x. We then draw the production rule from G^x. The specified set of production rules and semantic features are included with the code package. The specified rules and features do not change across our experiments.

Take, for example, the derivation in Figure 3. In the generative process where the node VP is expanded, the production rule is drawn from the HDP associated with the nonterminal VP. Suppose the HDP was constructed using a sequence of two semantic features: (`predicate`, `arg2`). In the example, the feature functions are evaluated with the logical form `plays_sport(,tennis)` and they return the sequence (`plays_sport`, `tennis`). This sequence uniquely identifies a path in the HDP hierarchy from the root node $\mathbf{0}$ to a leaf node $\boldsymbol{n}$. The production rule VP $\rightarrow$ V N is drawn from this leaf node $G^{\boldsymbol{n}}$, and the generative process continues recursively.

In our implementation, we divide the set of nonterminals $\mathcal{N}$ into two groups: (1) the set of "interior" nonterminals, and (2) preterminals. The production rules of preterminals are restricted such that the right-hand side contains only terminal symbols. The rules of interior nonterminals are restricted such that only nonterminal symbols appear on the right side.

1. For **preterminals**, we set H to be a distribution over sequences of terminal symbols as follows: we generate each token in the sequence i.i.d. from a uniform distribution over a finite set of terminals and a special *stop* symbol with probability ϕ_A. Once the stop symbol is drawn, we have finished gen-

erating the rule. Note that we do not specify a set of domain-specific terminal symbols in defining this distribution.

2. For **interior nonterminals**, we specify H as a discrete distribution over a domain-independent set of production rules. This requires specifying a set of nonterminal symbols, such as S, NP, VP, etc. Since these production rules contain semantic transformation functions, they are specific to the semantic formalism.

We emphasize that only the prior is specified here, and we will use grammar induction to infer the posterior. In principle, a more relaxed choice of H may enable grammar induction without pre-specified production rules, and therefore without dependence on a particular semantic formalism or natural language, if an efficient inference algorithm can be developed in such cases.

3 Induction

We describe grammar induction independently of the choice of rule distribution. Let $\boldsymbol{\theta}$ be the random variables in the grammar: in the case of the HDP prior, $\boldsymbol{\theta}$ is the set of all distributions $G^{\boldsymbol{n}}$ at every node in the hierarchies. Given a set of sentences $\boldsymbol{y} \triangleq \{y_1, \ldots, y_n\}$ and corresponding logical forms $\boldsymbol{x} \triangleq \{x_1, \ldots, x_n\}$, we wish to compute the posterior $p(\boldsymbol{t}, \boldsymbol{\theta} | \boldsymbol{x}, \boldsymbol{y})$ over the unobserved variables: the grammar $\boldsymbol{\theta}$ and the latent derivations/parse trees $\boldsymbol{t} \triangleq \{t_1, \ldots, t_n\}$. This is intractable to compute exactly, and so we resort to Markov chain Monte Carlo (MCMC) (Gelfand and Smith, 1990; Robert and Casella, 2010). To perform blocked Gibbs sampling, we pick initial values for $\boldsymbol{t}$ and $\boldsymbol{\theta}$ and repeat the following:

1. For $i = 1, \ldots, n$, sample $t_i | \boldsymbol{\theta}, x_i, y_i$.
2. Sample $\boldsymbol{\theta} | \boldsymbol{t}$.

However, since the sampling of each tree t depends on $\boldsymbol{\theta}$, and we need to resample all n parse trees before sampling $\boldsymbol{\theta}$, this Markov chain can be slow to mix. Thus, we employ collapsed Gibbs sampling by integrating out $\boldsymbol{\theta}$. In this algorithm, we repeatedly sample from $t_i | \boldsymbol{t}_{-i}, x_i, y_i$ where $\boldsymbol{t}_{-i} = \boldsymbol{t} \setminus \{t_i\}$.

$$p(t_i | \boldsymbol{t}_{-i}, x_i, y_i) = \tag{2}$$
$$\mathbb{1}\{\text{yield}(t_i) = y_i\} \prod_{A \in \mathcal{N}} p\left(\bigcap_{\substack{\{\boldsymbol{n} \in t_i \,:\, \boldsymbol{n} \\ \text{has label } A\}}} r^{\boldsymbol{n}} \;\middle|\; \boldsymbol{t}_{-i}, x_i \right),$$

where the intersection is taken over tree nodes $\boldsymbol{n} \in t_i$ labeled with the nonterminal A, $r^{\boldsymbol{n}}$ is the production rule at node $\boldsymbol{n}$, and $\mathbb{1}\{\cdot\}$ is 1 if the condition is true and zero otherwise. With $\boldsymbol{\theta}$ integrated out, the probability does not necessarily factorize over rules. In the case of the HDP prior, selecting a rule will increase the probability that the same rule is selected again (due to the "rich get richer" effect observed in the Chinese restaurant process). We instead use a Metropolis-Hastings step to sample t_i, where the proposal distribution is given by the fully factorized form:

$$p(t_i^* | \boldsymbol{t}_{-i}, x_i, y_i) = \tag{3}$$
$$\mathbb{1}\{\text{yield}(t_i^*) = y_i\} \prod_{\boldsymbol{n} \in t_i^*} p\left(r^{\boldsymbol{n}} \;\middle|\; \boldsymbol{t}_{-i}, x_i^{\boldsymbol{n}}\right).$$

After sampling t_i^*, we choose to accept the new sample with probability

$$\frac{\prod_{\boldsymbol{n} \in t_i} p(r^{\boldsymbol{n}} | x^{\boldsymbol{n}}, \boldsymbol{t}_{-i})}{p\left(\bigcap_{\boldsymbol{n} \in t_i} r^{\boldsymbol{n}} | x, \boldsymbol{t}_{-i}\right)} \frac{p\left(\bigcap_{\boldsymbol{n} \in t_i^*} r^{\boldsymbol{n}} | x, \boldsymbol{t}_{-i}\right)}{\prod_{\boldsymbol{n} \in t_i^*} p(r^{\boldsymbol{n}} | x^{\boldsymbol{n}}, \boldsymbol{t}_{-i})},$$

where t_i, here, is the old sample, and t_i^* is the newly proposed sample. In practice, this acceptance probability is very high. This approach is very similar in structure to that in Johnson et al. (2007); Blunsom and Cohn (2010); Cohn et al. (2010).

If an application requires posterior samples of the grammar variables $\boldsymbol{\theta}$, we can obtain them by drawing from $\boldsymbol{\theta} | \boldsymbol{t}$ after the collapsed Gibbs sampler has mixed. Note that this algorithm requries no further supervision beyond the utterances $\boldsymbol{y}$ and logical forms $\boldsymbol{x}$. However, it is able to exploit additional information such as supervised derivations/parse trees. For example, a lexicon can be provided where each entry is a terminal symbol y_i with a corresponding logical form label x_i. We evaluate our method with and without such a lexicon.

Refer to Saparov and Mitchell (2016) for details on HDP inference and computing $p(r^{\boldsymbol{n}} | x^{\boldsymbol{n}}, \boldsymbol{t}_{-i})$.

3.1 Sampling t_i^*

To sample from equation (3), we use inside-outside sampling (Finkel et al., 2006; Johnson et al., 2007), a dynamic programming approach, where the inside step is implemented using an agenda-driven chart parser (Indurkhya and Damerau, 2010). The algorithm fills a chart, which has a cell for every

nonterminal A, sentence start position i, end position j, and logical form x. The algorithm aims to compute the *inside probability* of every chart cell: that is, for every cell (A, i, j, x), we compute the probability that t_i^* contains a subtree rooted with the nonterminal A and logical form x, spanning the sentence positions (i, j). Let $I_{(A,i,j,x)}$ be the inside probability at the chart cell (A, i, j, x):

$$I_{(A,i,j,x)} = \sum_{A \to B_1 : f_1 \ldots B_K : f_K} \sum_{i=l_1 < \ldots < l_{K+1} = j} \prod_{u=1}^{K} I_{(B_u, l_u, l_{u+1}, f_u(x))}. \quad (4)$$

Each item in the agenda represents a snapshot of the computation of this expression for a single rule $A \to B_1 : f_1 \ldots B_K : f_K$. The agenda item stores the current position in the rule k, the set of sentence spans that correspond to the first k right-hand side symbols $l_1, \ldots, l_{k+1}$, the span of the rule (i, j), the logical form x, and the inside probability of the portion of the rule computed so far. At every iteration, the algorithm pops an item from the agenda and adds it to the chart, and considers the next right-hand side symbol B_k.

- If B_k is a terminal, it will match it against the input sentence. If the terminal does not match the sentence, this agenda item is discarded and the algorithm continues to the next iteration. If the terminal does match, the algorithm *increments* the rule. That is, for each possible value of l_{k+2}, the algorithm constructs a new agenda item containing the same contents as the old agenda item, but with rule position $k + 1$.
- If B_k is a nonterminal, the algorithm will *expand* it (if it was not previously expanded at this cell). The algorithm considers every production rule of the form $B_k \to \beta$, and every possible end position for the next nonterminal $l_{k+2} = l_{k+1} + 1, \ldots, j - 1$, and enqueues a new agenda item with rule $B_k \to \beta$, rule position set to 1, span set to (l_k, l_{k+1}), logical form set to $f_k(x)$, and inside probability initialized to 1. The original agenda item is said to be "waiting" for B_k to be completed later on in the algorithm.
- If the rule is *complete* (there are no subsequent symbols in the rule of this agenda item), we can compute its inner probability $p(A \to B_1 : f_1 \ldots B_K : f_K | x, \boldsymbol{t}_{-i})$. First, we record that this rule was used to complete the left-hand nonterminal A at the cell (A, i, j, x). Then, we consider every agenda item in the chart that is currently "waiting" for the left-hand nonterminal A at this sentence span. The search *increments* each "waiting" item, adding a new item to the agenda for each, whose log probability is the sum of the log probability of the old agenda item and the log probability of the completed rule.

We prioritize items in the agenda by $i - j$ (so items with smaller spans are dequeued first). This ensures that whenever the search considers expanding B_k, if B_k was previously expanded at this cell, its inside probability is fully computed. Thus, we can avoid re-expanding B_k and directly increment the agenda item. The algorithm terminates when there are no items in the agenda.

All that remains is the outside step: to sample the tree given the computed inside probabilities. To do so, we begin with the chart cell $(S, 0, |y_i|, x_i)$ where $|y_i|$ is the length of sentence y_i, and we consider all completed rules at this cell (these rules will be of the form $S \to \beta$). Each rule will have a computed inside probability, and we can sample the rule from the categorical distribution according to these inside probabilities. Then, we consider the right-hand side nonterminals in the selected rule, and continue sampling recursively. The end result is a tree sampled from equation (3).

4 Parsing

For a new sentence y_*, we aim to find the logical form x_* and derivation t_* that maximizes

$$p(x_*, t_* | y_*, \boldsymbol{\theta}) \propto p(x_*) p(y_* | t_*) p(t_* | x_*, \boldsymbol{\theta}),$$
$$= \mathbb{1}\{\text{yield}(t_*) = y_*\} p(x_*) \prod_{\boldsymbol{n} \in t_*} p(r^{\boldsymbol{n}} | x_*^{\boldsymbol{n}}, \boldsymbol{\theta}). \quad (5)$$

Here, $\boldsymbol{\theta}$ is a point estimate of the grammar, which may be obtained from a single sample, or from a Monte Carlo average over a finite set of samples.

To perform parsing, we first describe an algorithm to compute the derivation t_* that maximizes the above quantity, given the logical form x_* and input sentence y_*. We will later demonstrate how this algorithm can be used to find the optimal logical form and

derivation x_*, t_*. To find the optimal t_*, we again use an agenda-driven chart parser to perform the optimization, with a number of important differences. Each agenda item will keep track the derivation tree completed so far.

The algorithm is very similar in structure to the inside algorithm described above. At every iteration of the algorithm, an item is popped from the agenda and added to the chart, applying one of the three operations available to the inside algorithm. The algorithm begins by expanding the root nonterminal S at $(0, |y_*|)$ with the logical form x_*.

4.1 Agenda prioritization

The most important difference from the inside algorithm is the prioritization of agenda items. For a given agenda item with rule $A \rightarrow B_1 : f_1 \dots B_K : f_K$ with logical form x at sentence position (i, j), we aim to assign as its priority an upper bound on equation (5) for any derivation that contains this rule at this position. To do so, we can split the product in the objective $\prod_{\boldsymbol{n} \in t_*} p(r^{\boldsymbol{n}} | x_*^{\boldsymbol{n}}, \boldsymbol{\theta})$ into a product of two components: (1) the *inner probability* is the product of the terms that correspond to the subtree of t_* rooted at the current agenda item, and (2) the *outer probability* is the product of the remaining terms, which correspond to the parts of t_* outside of the subtree rooted at the agenda item. A schematic decomposition of a derivation tree is shown in Figure 4.

We define an upper bound on the log inner probability $I_{(A,i,j)}$ for any subtree rooted at nonterminal A at sentence span (i, j).

$$I_{(A,i,j)} \triangleq \tag{6}$$
$$\max_{A \rightarrow B_1 \dots B_K} \left(\max_{x'} \log p(A \rightarrow B_1, \dots, B_K | x', \boldsymbol{\theta}) + \max_{l_2 < \dots < l_K} \sum_{k=1}^{K} I_{(B_k, l_k, l_{k+1})} \right),$$

where $l_1 = i$, $l_{K+1} = j$. Note that the left term is a maximum over all logical forms x', and so this upper bound only considers syntactic information. The right term can be maximized using dynamic programming in $\mathcal{O}(K^2)$. As such, classical syntactic parsing algorithms can be applied to compute I for every chart cell in $\mathcal{O}(n^3)$. For any terminal symbol T, we define $I_{(T,i,j)} = 0$.

We similarly define $O_{(A,i,j,x)}$ representing a bound on the outer probability at every cell.

$$O_{(A,i,j,x)} \triangleq \max_{\{t : \text{yield}(t) = y_*\}} \left(\log p(x) + \log p(t_L | x, \boldsymbol{\theta}) + \sum_{(A', i', j') \in r(t_R)} I_{(A', i', j')} \right), \tag{7}$$

where the maximum is taken over t which is a derivation containing a subtree rooted at A at sentence position (i, j). In this expression, t_L is the outer-left portion of the derivation tree t, t_R is the outer-right portion, and $r(t_R)$ is the set of root vertices of the trees in t_R.

Using these two upper bounds, we define the priority of any agenda item with rule $A \rightarrow B_1 : f_1 \dots B_K : f_K$ at rule position k, with log probability score ρ, and logical form x as:

$$\rho + \max_{l_{k+2} < \dots < l_{K+1}} \sum_{u=k}^{K} I_{(B_u, l_u, l_{u+1})} + O_{(A,i,j,x)}. \tag{8}$$

Thm 1. *If the priority of agenda items is computed as in equation (8), then at every iteration of the chart parser, the priority of new agenda items will be at most the priority of the current item.*

Proof. See supplementary material A.

Thus, the search is *monotonic*[1]. That is, the maximum priority of items in the agenda never increases.

This property allows us to compute the outer probability bound $O_{(A,i,j,x)}$ for free. Computing it directly is intractable. Consider the expansion step for an agenda item with rule $A \rightarrow B_1 : f_1 \dots B_K : f_K$ at rule position k, with log probability score ρ, and logical form x. The nonterminal B_k is expanded next at sentence position (l_k, l_{k+1}), and its outer probability is simply

$$O_{(B_k, l_k, l_{k+1}, f_k(x))} = \rho + \max_{l_{k+2} < \dots < l_{K+1}} \sum_{u=k+1}^{K} I_{(B_u, l_u, l_{u+1})} + O_{(A,i,j,x)}. \tag{9}$$

The monotonicity of the search guarantees that any subsequent expansion of B_k at (l_k, l_{k+1}) will not yield a more optimal bound.

Monotonicity also guarantees that when the algorithm completes a derivation for the root nonterminal S, it is optimal (i.e. the Viterbi

[1] In the presentation of the algorithm as an A* search, the heuristic is *consistent*.

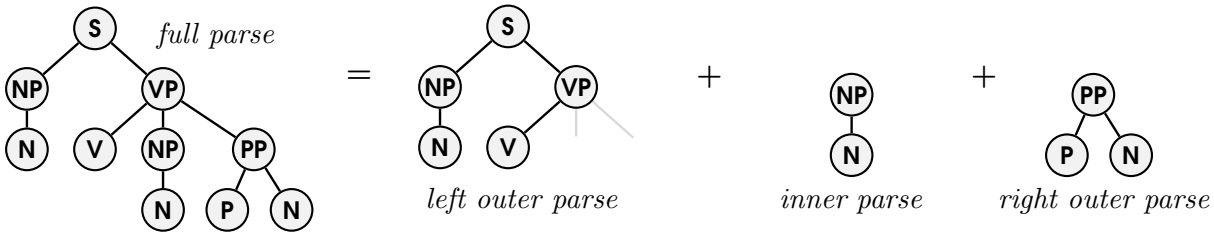

Figure 4: Decomposition of a parse tree into its left outer parse, inner parse, and its right outer parse. This is one example of such a decomposition. For instance, we may similarly produce a decomposition where the prepositional phrase is the inner parse, or where the verb is the inner parse. The terminals are omitted and only the syntactic portion of the parse is displayed here for conciseness.

parse). In this way, we can continue execution to obtain the k-best parses for the given sentence.

4.2 Optimization over logical forms

The above algorithm finds the optimal derivation t_*, given a sentence y_*, logical form x_*, and grammar $\boldsymbol{\theta}$. To jointly optimize over both the derivation and logical form, given $\boldsymbol{\theta}$, imagine running the above algorithm repeatedly for *every* logical form. This approach, implemented naively, is clearly infeasible due to the sheer number of possible logical forms. However, there is a great deal of overlap across the multiple runs, which corresponds to shared substructures across logical forms, which we can exploit to develop an efficient and exact algorithm. At the first step of every run, the root nonterminal is expanded for every logical form. This would create of a new agenda item for every logical form, which are identical in every field except for the logical form (and therefore, its prior probability). Thus, we can represent this set of agenda items as a single agenda item, where instead of an individual logical form x, we store a logical form set X. The outer probability bound is now defined over sets of logical forms: $O_{(A,i,j,X)} \triangleq \max_{x \in X} O_{(A,i,j,x)}$. We can use this quantity in equation (8) to compute the priority of these "aggregated" agenda items. Thus, this algorithm is a kind of branch-and-bound approach to the combinatorial optimization problem. A sparse representation of a set of logical forms is essential for efficient parsing.

Another difference arises after completing the parsing of a rule $A \rightarrow B_1 : f_1 \ldots B_K : f_K$ with a set of logical forms X, where we need to compute $\log p(A \rightarrow B_1 : f_1 \ldots B_K : f_K | x, \boldsymbol{\theta})$. In the inside algorithm, this was straightforward since there was only a single logical form. But in the parsing setting, X is a set of logical forms, and the aforementioned probability can vary across instances within this set (for the HDP prior, for example, the set may correspond to multiple distinct paths in the HDP hierarchy). Therefore, we divide X into its equivalence classes. More precisely, consider the set of disjoint subsets of $X = X_1 \bigcup \ldots \bigcup X_m$ where $X_i \bigcap X_j = \varnothing$ for $i \neq j$, such that $p(A \rightarrow B_1 : f_1 \ldots B_K : f_K | x', \boldsymbol{\theta})$ is the same for every $x' \in X_i$. For each equivalence class X_i, we create a "completed nonterminal" item with the appropriate parse tree, log probability, and logical form set X_i. With these, we continue inspecting the chart for search states "waiting" for the nonterminal A.

The *increment* operation is also slightly different in the parser. When we increment a rule $A \rightarrow B_1 : f_1 \ldots B_K : f_K$ after completing parsing for the symbol B_k with logical form set X, we create a new agenda item with the same contents as the old item, but with the rule position increased by one. The log probability of the new agenda item is the sum of the log probabilities of the old agenda item and the completed subtree. Similarly the logical form set of the new agenda item will be the intersection of $\{f_k^{-1}(x) : x \in X\}$ and the logical forms in the old agenda item.

Our implementation is available for reference at https://github.com/asaparov/grammar and https://github.com/asaparov/parser.

5 Semantic prior

The modular nature of the semantic prior allows us to explore many different models of logical forms. We experiment with a fairly straightforward prior: Predicate instances are generated left-to-right, conditioned only on the last predicate instance that was sampled for each variable. When a predicate instance is sampled, its predicate, arity, and "direc-

Method	Additional Supervision	GeoQuery			Jobs		
		P	R	F1	P	R	F1
WASP (Wong and Mooney, 2006)	1,2	87.2	74.8	80.5			
λ-WASP (Wong and Mooney, 2007)	1,2,6	92.0	86.6	89.2			
Extended GHKM (Li et al., 2013)	2,6	93.0	87.6	90.2			
Zettlemoyer and Collins (2005)	3,5,6	**96.3**	79.3	87.0	97.3	79.3	87.4
Zettlemoyer and Collins (2007)	3,5,6	91.6	86.1	88.8			
UBL (Kwiatkowski et al., 2010)	5	94.1	85.0	89.3			
FUBL (Kwiatkowski et al., 2011)	5	88.6	88.6	88.6			
TISP (Zhao and Huang, 2014)	5,6	92.9	**88.9**	**90.9**	85.0	**85.0**	85.0
GSG − lexicon − type-checking	4	86.9	75.7	80.9	89.5	67.1	76.7
GSG + lexicon − type-checking	4,5	88.4	81.8	85.0	91.4	75.7	82.8
GSG − lexicon + type-checking	4,6	89.3	77.9	83.2	93.2	69.3	79.5
GSG + lexicon + type-checking	4,5,6	90.7	83.9	87.2	**97.4**	81.4	**88.7**

Legend for sources of additional supervision are:

1. Training set containing 792 examples,
2. Domain-specific set of initial synchronous CFG rules,
3. Domain-independent set of lexical templates,
4. Domain-independent set of interior production rules,
5. Domain-specific initial lexicon,
6. Type-checking and type specification for entities.

Figure 5: The methods in the top part of the table were evaluated using 10-fold cross validation, whereas those in the bottom part were evaluated with an independent test set.

Logical form:	`answer(A,smallest(A,state(A)))`	`answer(A,largest(B,(state(A),population(A,B))))`
Test sentence:	"Which state is the smallest?"	"Which state has the most population?"
Generated:	"What state is the smallest?"	"What is the state with the largest population?"

Figure 6: Examples of sentences generated from our trained grammar on logical forms in the GeoQuery test set. Generation is performed by computing $\arg\max_{y_*} p(y_*|x_*, \boldsymbol{\theta})$.

tion"[2] are simultaneously sampled from a categorical distribution. Functions like `largest`, `shortest`, etc, are sampled in the same process. We again use an HDP to model the discrete distribution conditioned on a discrete random variable.

We also follow Wong and Mooney (2007); Li et al. (2013); Zhao and Huang (2014) and experiment with type-checking, where every entity is assigned a type in a type hierarchy, and every predicate is assigned a functional type. We incorporate type-checking into the semantic prior by placing zero probability on type-incorrect logical forms. More precisely, logical forms are distributed according to the original prior, conditioned on the fact that the logical form is type-correct. Type-checking requires the specification of a type hierarchy. Our hierarchy contains 11 types for GeoQuery and 12 for Jobs. We run experiments with and without type-checking for comparison.

6 Results

To evaluate our parser, we use the GeoQuery and Jobs datasets. Following Zettlemoyer and Collins (2007), we use the same 600 GeoQuery sentences for training and an independent test set of 280 sentences. On Jobs, we use the same 500 sentences for training and 140 for testing. We run our parser with two setups: (1) with no domain-specific supervision, and (2) using a small domain-specific lexicon and a set of beliefs (such as the fact that Portland is a city). For each setup, we run the experiments with and without type-checking, for a total of 4 experimental setups. A given output logical form is considered correct if it is semantically equivalent to the true logical form.[3] We measure the precision and recall of our method, where precision is the number of correct parses divided by the number of sentences for which our parser provided output, and recall is the number of correct parses divided by the total number of sentences in each dataset. Our results are shown compared against many other semantic parsers in Figure 5. Our method is labeled GSG for "generative semantic grammar." The numbers for the baselines were copied from their respective papers, and so their specified lexicons/type hierarchies may differ slightly.

Many sentences in the test set contain tokens previously unseen in the training set. In such cases, the maximum possible recall is 88.2 and 82.3 on GeoQuery and Jobs, re-

[2] `size(A)`, `size(A,B)`, vs `size(B,A)`, etc.

[3] The result of execution of the output logical form is identical to that of the true logical form, for any grounding knowledge base/possible world.

spectively. Therefore, we also measure the effect of adding a domain-specific lexicon, which maps semantic constants like `maine` to the noun "maine" for example. This lexicon is analogous to the string-matching and argument identification steps previous parsers. We constructed the lexicon manually, with an entry for every city, state, river, and mountain in GeoQuery (141 entries), and an entry for every city, company, position, and platform in Jobs (180 entries).

Aside from the lexicon and type hierarchy, the only training information is given by the set of sentences $\boldsymbol{y}$, corresponding logical forms $\boldsymbol{x}$, and the domain-independent set of interior production rules, as described in section 2.2. In our experiments, we found that the sampler converges rapidly, with only 10 passes over the data. This is largely due to our restriction of the interior production rules to a domain-independent set.

We emphasize that the addition of type-checking and a lexicon are mainly to enable a fair comparison with past approaches. As expected, their addition greatly improves parsing performance. Our method achieves state-of-the-art F1 on the Jobs dataset. However, even without such domain-specific supervision, the parser performs reasonably well.

7 Related work

Our grammar formalism can be related to synchronous CFGs (SCFGs) (Aho and Ullman, 1972) where the semantics and syntax are generated simultaneously. However, instead of modeling the joint probability of the logical form and natural language utterance $p(x, y)$, we model the factorized probability $p(x)p(y|x)$. Modeling each component in isolation provides a cleaner division between syntax and semantics, and one half of the model can be modified without affecting the other (such as the addition of new background knowledge, or changing the language/semantic formalism). We used a CFG in the syntactic portion of our model (although our grammar is not context-free, due to the dependence on the logical form). Richer syntactic formalisms such as combinatory categorial grammar (Steedman, 1996) or head-driven phrase structure grammar (Pollard and Sag, 1994) could replace the syntactic component in our framework and may provide a more uniform analysis across languages. Our model is similar to lexical functional grammar (LFG) (Kaplan and Bresnan, 1995), where f-structures are replaced with logical forms. Nothing in our model precludes incorporating syntactic information like f-structures into the logical form, and as such, LFG is realized in our framework. Our approach can be used to define new generative models of these grammatical formalisms. We implemented our method with a particular semantic formalism, but the grammatical model is agnostic to the choice of semantic formalism or the language. As in some previous parsers, a parallel can be drawn between our parsing problem and the problem of finding shortest paths in hypergraphs using A* search (Klein and Manning, 2001, 2003; Pauls and Klein, 2009; Pauls et al., 2010; Gallo et al., 1993).

8 Discussion

In this article, we presented a generative model of sentences, where each sentence is generated recursively top-down according to a semantic grammar, where each step is conditioned on the logical form. We developed a method to learn the posterior of the grammar using a Metropolis-Hastings sampler. We also derived a Viterbi parsing algorithm that takes into account the prior probability of the logical forms. Through this semantic prior, background knowledge and other information can be easily incorporated to better guide the parser during its search. Our parser provides state-of-the-art results when compared with past approaches.

As a generative model, there are promising applications to interactive learning, caption generation, data augmentation, etc. Richer semantic priors can be applied to perform ontology learning, relation extraction, or context modeling. Applying this work to semi-supervised settings is also interesting. The avenues for future work are numerous.

Acknowledgments

We thank the anonymous reviewers for their helpful feedback, and we also thank Emmanouil A. Platanios for insightful discussion and comments. This research is based upon work supported in part by the Office of the Director of National Intelligence (ODNI),

IARPA, and by DARPA under contract number FA8750-13-2-0005. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied of ODNI, IARPA, DARPA, or the US government. The US Government is authorized to reproduce and distribute the reprints for governmental purposed notwithstanding any copyright annotation therein.

References

Albert V. Aho and Jeffery D. Ullman. 1972. *The Theory of Parsing, Translation, and Compiling*, volume 1. Prentice-Hall, Englewood Cliffs, NJ.

Phil Blunsom and Trevor Cohn. 2010. Inducing synchronous grammars with slice sampling. In *HLT-NAACL*. The Association for Computational Linguistics, pages 238–241.

Noam Chomsky. 1956. Three models for the description of language. *IRE Transactions on Information Theory* 2:113–124.

Trevor Cohn, Phil Blunsom, and Sharon Goldwater. 2010. Inducing tree-substitution grammars. *Journal of Machine Learning Research* 11:3053–3096.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. *CoRR* abs/1601.01280.

Jenny Rose Finkel, Christopher D. Manning, and Andrew Y. Ng. 2006. Solving the problem of cascading errors: approximate bayesian inference for linguistic annotation pipelines. In *EMNLP '06: Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Morristown, NJ, USA, pages 618–626.

Giorgio Gallo, Giustino Longo, and Stefano Pallottino. 1993. Directed hypergraphs and applications. *Discrete Applied Mathematics* 42(2):177–201.

Alan E. Gelfand and Adrian F. M. Smith. 1990. Sampling-based approaches to calculating marginal densities. *Journal of the American Statistical Association* 85(410):398–409.

Nitin Indurkhya and Fred J. Damerau, editors. 2010. *Handbook of Natural Language Processing, Second Edition*. Chapman and Hall/CRC.

M. Johnson, T. L. Griffiths, and S. Goldwater. 2007. Bayesian inference for PCFGs via Markov chain Monte Carlo. In *Proceedings of the North American Conference on Computational Linguistics (NAACL '07)*.

Ronald M. Kaplan and Joan Bresnan. 1995. Lexical-functional grammar: A formal system for grammatical representation.

Dan Klein and Christopher D. Manning. 2001. Parsing and hypergraphs. In *IWPT*. Tsinghua University Press.

Dan Klein and Christopher D. Manning. 2003. A* parsing: Fast exact viterbi parse selection. In *HLT-NAACL*.

Tom Kwiatkowski, Eunsol Choi, Yoav Artzi, and Luke S. Zettlemoyer. 2013. Scaling semantic parsers with on-the-fly ontology matching. In *EMNLP*. ACL, pages 1545–1556.

Tom Kwiatkowski, Luke S. Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2010. Inducing probabilistic ccg grammars from logical form with higher-order unification. In *EMNLP*. ACL, pages 1223–1233.

Tom Kwiatkowski, Luke S. Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2011. Lexical generalization in ccg grammar induction for semantic parsing. In *EMNLP*. ACL, pages 1512–1523.

Peng Li, Yang Liu, and Maosong Sun. 2013. An extended ghkm algorithm for inducing lambda-scfg. In Marie desJardins and Michael L. Littman, editors, *AAAI*. AAAI Press.

Percy Liang, Michael I. Jordan, and Dan Klein. 2011. Learning dependency-based compositional semantics. *CoRR* abs/1109.6841.

Adam Pauls and Dan Klein. 2009. K-best a* parsing. In Keh-Yih Su, Jian Su, and Janyce Wiebe, editors, *ACL/IJCNLP*. The Association for Computer Linguistics, pages 958–966.

Adam Pauls, Dan Klein, and Chris Quirk. 2010. Top-down k-best a* parsing. In *Proceedings of the ACL 2010 Conference Short Papers*. Association for Computational Linguistics, Stroudsburg, PA, USA, ACLShort '10, pages 200–204.

Carl Pollard and Ivan A. Sag. 1994. *Head-driven phrase structure grammar*. University of Chicago Press, Chicago.

C. P. Robert and G. Casella. 2010. *Monte Carlo Statistical Methods*. Springer, New York, NY.

Abulhair Saparov and Tom M. Mitchell. 2016. A probabilistic generative grammar for semantic parsing. In *arXiv:1606.06361*.

Mark Steedman. 1996. *Surface structure and interpretation*. Linguistic inquiry monographs, 30. MIT Press.

Yee Whye Teh, Michael I. Jordan, Matthew J. Beal, and David M. Blei. 2006. Hierarchical dirichlet processes. *Journal of the American Statistical Association* 101(476):1566–1581.

Yuk Wah Wong and Raymond J. Mooney. 2006. Learning for semantic parsing with statistical machine translation. In Robert C. Moore, Jeff A. Bilmes, Jennifer Chu-Carroll, and Mark Sanderson, editors, *HLT-NAACL*. The Association for Computational Linguistics.

Yuk Wah Wong and Raymond J. Mooney. 2007. Learning synchronous grammars for semantic parsing with lambda calculus. In John A. Carroll, Antal van den Bosch, and Annie Zaenen, editors, *ACL*. The Association for Computational Linguistics.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *UAI*. AUAI Press, pages 658–666.

Luke S. Zettlemoyer and Michael Collins. 2007. Online learning of relaxed ccg grammars for parsing to logical form. In Jason Eisner, editor, *EMNLP-CoNLL*. ACL, pages 678–687.

Kai Zhao and Liang Huang. 2014. Type-driven incremental semantic parsing with polymorphism. *CoRR* abs/1411.5379.

A Proof of Thm 1

Induct on the iteration of the algorithm. Let a be the current agenda item, where the production rule of the item is $A \to B_1{:}f_1 \dots B_K{:}f_K$, the rule position is k, the set of sentence spans that correspond to the first k right-hand side symbols is $l_1, \dots, l_{k+1}$, the span of the rule is (i, j), the logical form is x, and the log probability of the first k right-hand side symbols is ρ. For notational brevity, we let $c(a)$ denote the priority of the agenda item a as defined in equation (8). There are three cases:

Case 1 If B_k is a terminal, a new agenda item is created only if the terminal matches the token in the input sentence. Then for each possible value of l_{k+2}, a new agenda item a' is created, where ρ is the same as that in the old agenda item a. Since the logical form is the same, and

$$I_{(B_k,l_k,l_{k+1})} + \max_{l_{k+3}<\dots<l_{K+1}} \sum_{u=k+1}^{K} I_{(B_u,l_u,l_{u+1})} \leq \max_{l_{k+2}<\dots<l_{K+1}} \sum_{u=k}^{K} I_{(B_u,l_u,l_{u+1})},$$

the priority of the new agenda item is at most that of the old item $c(a') \leq c(a)$.

Case 2 If B_k is a nonterminal, it proceeds to "expand" it. Suppose to the contrary that we create an agenda item whose priority is greater than that of the current item. This implies that there exists a production rule $B_k \to C_1 \dots C_{K'}$ and a set of sentence spans $l'_2 < \dots < l'_{K'+1}$ such that

$$\begin{aligned} c(a) = & \\ & \rho + \max_{l_{k+2}<\dots<l_{K+1}} \sum_{u=k}^{K} I_{(B_u,l_u,l_{u+1})} + O_{(A,i,j,x)}, \\ & < \sum_{u=1}^{K'} I_{(C_u,l'_u,l'_{u+1})} + O_{(B_k,l_k,l_{k+1},f_k(x))} \\ & \leq I_{(B_k,l_k,l_{k+1})} + O_{(B_k,l_k,l_{k+1},f_k(x))}, \end{aligned}$$

where $l'_1 = l_k$ and $l_{K'+1} = l_{k+1}$. By definition of the bound on the outer probability, the above inequality implies that there exists a derivation t' and logical form x' such that $f_k(x') = f_k(x)$ and

$$\begin{aligned} c(a) < & \log p(x') + \log p(t'_L|x', \boldsymbol{\theta}) \\ & + I_{(B_k,l_k,l_{k+1})} + \sum_{(A',i',j')\in r(t'_R)} I_{(A',i',j')}. \end{aligned} \quad (10)$$

Let $D \to E_1 \dots E_{K''}$ be the production rule in t' that contains the B_k nonterminal at sentence span (l_k, l_{k+1}), and so for some m, $E_m = B_k$. In addition, let s'_i be the sibling derivation tree rooted at E_i for all $i = 1, \dots, m-1$. Therefore, there must exist an agenda item a' with the same production rule, with rule position m, (l_k, l_{k+1}) as the sentence spans of D_m, log probability of the first m right-hand side symbols $\sum_{i=1}^{m-1} \log p(s'_i|x', \boldsymbol{\theta})$, and logical form x'. The priority of this rule state is

$$\begin{aligned} c(a') = & \sum_{i=1}^{m-1} \log p(s'_i|x', \boldsymbol{\theta}) \\ & + \max_{l_{m+2}<\dots<l_{K''+1}} \sum_{u=m}^{K''} I_{(E_u,l_u,l_{u+1})} + O_{(D,i',j',x')}. \end{aligned}$$

By definition of the outer probability bound, we have $O_{(D,i',j',x')} \geq \log p(x') + \log p(t''_L|x', \boldsymbol{\theta}) + \sum_{(A',i',j')\in r(t''_R)} I_{(A',i',j')}$ where t''_L is the outer-left portion of the derivation tree t' not containing the subtree rooted at the nonterminal C, and t''_R is similarly the outer-right portion. Thus, we can lower

bound $c(a')$

$$c(a') \geq$$

$$\log p(x') + \log p(t''_L|x', \boldsymbol{\theta}) + \sum_{i=1}^{m-1} \log p(s'_i|x', \boldsymbol{\theta})$$

$$+ \max_{l_{m+2}<\ldots<l_{K''+1}} \sum_{u=m}^{K''} I_{(D_u,l_u,l_{u+1})} + \sum_{(A',i',j')\in r(t''_R)} I_{(A',i',j')},$$

which is at least the quantity in equation (10), since

$$\log p(t'_L|x', \boldsymbol{\theta}) =$$

$$\log p(t''_L|x', \boldsymbol{\theta}) + \sum_{i=1}^{m-1} \log p(s'_i|x', \boldsymbol{\theta}),$$

$$\sum_{(A',i',j')\in r(t'_R)} I_{(A',i',j')} \leq$$

$$\max_{l_{m+2}<\ldots<l_{K''+1}} \sum_{u=m+1}^{K''} I_{(D_u,l_u,l_{u+1})} + \sum_{(A',i',j')\in r(t''_R)} I_{(A',i',j')}.$$

Thus, $c(a') > c(a)$, and so by the inductive hypothesis, a' was processed by the algorithm at an earlier iteration. However, this would mean that the nonterminal B_k is expanded more than once at the sentence location (l_k, l_{k+1}), which is disallowed.

Case 3 If the rule is complete, the algorithm looks for previously-processed agenda items that are "waiting" for a derivation of B_k at positions (l_k, l_{k+1}). The algorithm will combine the completed derivation with the waiting state and create a new agenda item where the rule position is incremented by 1. Suppose to the contrary that the new agenda item a' has priority greater than the current item a. Let the production rule of a' be $D \rightarrow E_1 : f_1 \ldots E_{K'} : f_{K'}$ where $E_m = A$ for some m, and so m is the rule position of a'. Additionally, let the sentence spans of the first $m-1$ right-hand symbols be $l'_1, \ldots, l'_{m-1}$, and ρ' is the log probability of the first $m-1$ right-hand symbols, and the logical form is x' where $f_m(x') = x$. Therefore, we can write

$$c(a) = \log p(t'_A|x, \boldsymbol{\theta}) + O_{(A,i,j,x)},$$

$$< \rho' + \log p(t'_A|x, \boldsymbol{\theta}) + O_{(D,i',j',x')}$$

$$+ \max_{l_{m+3}<\ldots<l_{K'}} \sum_{u=m+1}^{K'} I_{(D_u,l'_u,l'_{u+1})} = c(a').$$

This expression implies

$$O_{(A,i,j,x)} < \rho' +$$

$$\max_{l_{m+3}<\ldots<l_{K'}} \sum_{u=m+1}^{K'} I_{(D_u,l'_u,l'_{u+1})} + O_{(C,i',j',x')},$$

but this is not possible by the definition of $O_{(A,i,j,x)}$ in equation (7), as we would have found a derivation with a strictly better objective. ■