A General-Purpose Algorithm for Constrained Sequential Inference

Daniel Deutsch,[*†] **Shyam Upadhyay,**[*‡◇] **and Dan Roth**[†]
[†]Department of Computer and Information Science, University of Pennsylvania
[‡]Google, New York
{ddeutsch,danroth}@seas.upenn.edu
shyamupa@google.com

Abstract

Inference in structured prediction involves finding the best output structure for an input, subject to certain constraints. Many current approaches use sequential inference, which constructs the output in a left-to-right manner. However, there is no general framework to specify constraints in these approaches. We present a principled approach for incorporating constraints into sequential inference algorithms. Our approach expresses constraints using an *automaton*, which is traversed in lock-step during inference, guiding the search to valid outputs. We show that automata can express commonly used constraints and are easily incorporated into sequential inference. When it is more natural to represent constraints as a set of automata, our algorithm uses an *active set method* for demonstrably fast and efficient inference. We experimentally show the benefits of our algorithm on constituency parsing and semantic role labeling. For parsing, unlike unconstrained approaches, our algorithm always generates valid output, incurring only a small drop in performance. For semantic role labeling, imposing constraints using our algorithm corrects common errors, improving F_1 by 1.5 points. These benefits increase in low-resource settings. Our active set method achieves a 5.2x relative speedup over a naive approach.[1]

1 Introduction

The key challenge in structured prediction problems (like sequence tagging and parsing) is *inference* (also known as *decoding*), which involves identifying the best output structure $\mathbf{y}$ for an input instance $\mathbf{x}$ from an exponentially large search space $\mathcal{Y}$ (Taskar, 2004; Tsochantaridis et al., 2005). The search is restricted to set of valid $\mathbf{y} \in \mathcal{Y}_\mathbf{x} \subseteq \mathcal{Y}$ for $\mathbf{x}$ by imposing *constraints* during inference. Figure 1 shows examples of constraints used in Semantic Role Labeling (SRL) (Gildea and Jurafsky, 2002).

input:	Alice	Smith	gave	a	flower	to	Bob
gold tags:	B-A_0	I-A_0	O	O	B-A_1	O	B-A_2
predicate:	gave						
legal args:	A_0, A_1, A_2 (from PropBank)						
spans:	[Alice Smith] gave a [flower] to [Bob]						
invalid tags:	B-A_0	I-A_0	O	O	**B-A_0**	O	B-A_2
reason:	duplicate A_0						
invalid tags:	**B-A_5**	**I-A_5**	O	O	B-A_1	O	B-A_2
reason:	illegal arg A_5 for predicate "gave"						
invalid tags:	B-A_0	**O**	O	O	B-A_1	O	B-A_2
reason:	span [Alice Smith] should have single label						

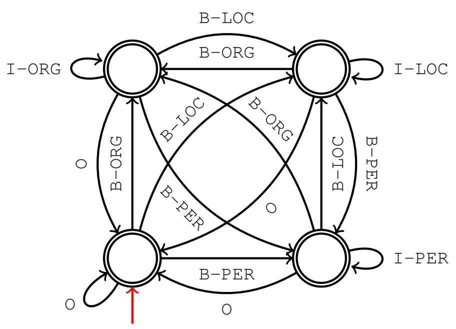
Figure 1: An example SRL instance (input, predicate, gold tags) with different invalid tag sequences and the constraints they violate (details in §4.2).

Currently, inference for most structured prediction problems is solved *sequentially*, predicting the output in a left-to-right manner (Sutskever et al., 2014; Luong et al., 2016). Such *sequential inference* approaches enforce constraints in different ways. For example, a shift-reduce parser consults a stack to determine which action sequences produce valid trees (Nivre et al., 2014), while an SRL model penalizes tag sequences which violate constraints during inference (Punyakanok et al., 2008; Täckström et al., 2015).

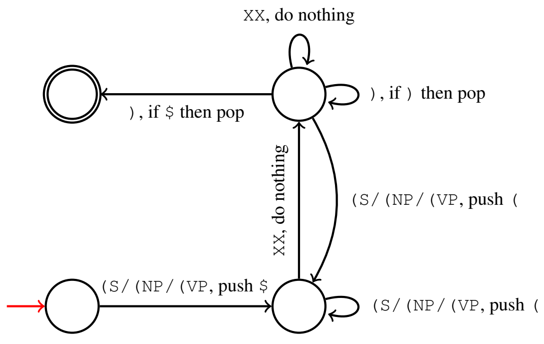
At present, inference algorithms are designed to handle task-specific constraints, and there is no general formulation for constrained sequential inference. This contrasts with the state of affairs in NLP before deep learning, when constrained inference approaches used general formulations like Integer Linear Programming (ILP) (Roth and Yih, 2004; Clarke and Lapata, 2008, inter alia).

We present a simple, general-purpose sequential inference algorithm that takes a model and

[1]All code available at https://cogcomp.seas.upenn.edu/page/publication_view/884
[*]Equal contribution, authors listed alphabetically
[◇]Work done while at University of Pennsylvania.



(a) A FSA for BIO tag sequences for NER.



(b) A PDA that accepts valid parse trees like `(S (NP XX) (VP XX XX))` and `(S (VP XX))`. The edge label "P/Q/R, A" denotes consuming either P, Q or R as input and changing the stack per A. "If T" indicates that edge is only valid if T is on top of the stack.

Figure 2: An example FSA and PDA. Red arrows mark start states, and double circles mark accepting states.

an *automaton* expressing the constraints as input, and outputs a structure that satisfies the constraints (§2.2). The automaton guides the inference to always produce a valid output by reshaping the model's probability distribution such that actions deemed invalid by the automaton are not taken.

In some situations, it is more natural to express the constraints as a *set* of automata. However, naively enforcing multiple automata by fully intersecting them is potentially expensive. Instead, our algorithm lazily intersects the automata using an efficient active set method, reminiscent of the cutting-plane algorithm (Tsochantaridis et al., 2005) (§2.4).

The choice of using automata to express constraints has several benefits. First, automata are capable of expressing constraints used in a wide variety of NLP tasks. Indeed, in §3, we show that task-specific constrained inference approaches implicitly use an automaton. Second, automata can be naturally incorporated into any sequential inference algorithm such as beam search. Finally, automata make enforcing multiple constraints straightforward — only the automata for individual constraints need to be specified, which are then intersected at inference time.

Our algorithm is a principled approach for enforcing constraints and has many desirable properties. It decouples the constraints from the inference algorithm, making it generally applicable to many problems. Further, it guarantees valid output and allows for the seamless addition of constraints at inference time without modifying the inference code. We experimentally demonstrate the benefits of our algorithm on two structured prediction tasks, constituency parsing (§5.1) and semantic role labeling (§5.2). Our results in constituency parsing show that our algorithm always outputs valid parses, incurring only a small drop in F_1. In SRL, constrained inference using our algorithm corrects common errors produced by unconstrained inference, resulting in a 1.5 F_1 improvement. This increase in performance is more prominent in low-resource settings. Finally, the active set method for enforcing multiple constraints achieves a 5.2x speed-up over the naive approach of fully intersecting the relevant automata.

2 Constrained Inference with Automata

We briefly review automata that we use for representing constraints in our algorithm.

2.1 Brief Review of Automata Theory

For the purposes of this work, an automaton is a (possibly weighted) directed graph that compactly encodes a set of strings, known as its *language*. The two types of automata used in this work are finite-state automata (FSA) and push-down automata (PDA).[2]

In an FSA, each edge is labeled with a symbol y from an alphabet Σ, and any traversal from the starting state to the final state(s) represents a unique string $\mathbf{y}$ in the language. A PDA is an extension of an FSA in which the traversal can maintain a stack that is used for computation, and the edges may manipulate or examine the state of the stack. Any language that can be expressed using regular expressions or context-free grammars

[2]We only consider deterministic automata.

has an equivalent accepting FSA or PDA, respectively (Sipser, 1997). An example of each type of automata is depicted in Figure 2.

Our inference algorithm views an automaton as an abstract stateful function, denoted as $\mathcal{A}$, which accepts strings from its language $L(\mathcal{A})$. After consuming the prefix $\mathbf{y}_{1:i}$ of a string $\mathbf{y}$, $\mathcal{A}$ provides a score $\mathcal{A}(y_{i+1} \mid \mathbf{y}_{1:i})$ for every symbol $y \in \Sigma$. Invoking $\mathcal{A}$.accepts(s) tests if a string s is in $L(\mathcal{A})$.

2.2 Sequential Inference

The traditional inference problem for structured prediction can be formalized as solving

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathcal{Y}_\mathbf{x}}{\operatorname{argmax}} \log p_\theta(\mathbf{y} \mid \mathbf{x}) \tag{1}$$

where $\mathbf{x}$ and $\mathbf{y}$ are the input and output structures, $\mathcal{Y}_\mathbf{x}$ is the set of valid output structures for $\mathbf{x}$, and θ are the parameters of the model $p_\theta(\mathbf{y} \mid \mathbf{x})$. A common way to solve this problem is to decompose the objective as follows:

$$\hat{\mathbf{y}} = \underset{\mathbf{y} \in \mathcal{Y}_\mathbf{x}}{\operatorname{argmax}} \sum_i \log p_\theta(y_i \mid \mathbf{x}, \mathbf{y}_{1:i-1}) \tag{2}$$

This decomposition is adopted by popular approaches, such as seq2seq models, SEARN (Daumé et al., 2009), etc. Under this decomposition, the inference problem is often solved by an inexact sequential inference algorithm, such as beam search.

2.3 Imposing Constraints with Automata

We are interested in versions of Equation 2 in which the output space is described by the language of an automaton. Formally,

$$\hat{\mathbf{y}} = \underset{\mathbf{y}}{\operatorname{argmax}} \sum_i \log p_\theta(y_i \mid \mathbf{x}, \mathbf{y}_{1:i-1}) \quad \text{such that} \quad \mathbf{y} \in L(\mathcal{A}_\mathbf{x}) \tag{3}$$

where $L(\mathcal{A}_\mathbf{x})$ is the language of an automaton $\mathcal{A}_\mathbf{x}$ describing the valid output space for instance $\mathbf{x}$. This framework is capable of expressing many constraints which are common in NLP applications, described in detail in §3.

Equation 3 can be rewritten as:

$$\hat{\mathbf{y}} = \underset{\mathbf{y}}{\operatorname{argmax}} \sum_i \log \tilde{p}_\theta(y_i \mid \mathbf{x}, \mathbf{y}_{1:i-1}) \tag{4}$$

$$\log \tilde{p}_\theta(y_i \mid \mathbf{x}, \mathbf{y}_{1:i-1}) \triangleq \log p_\theta(y_i \mid \mathbf{x}, \mathbf{y}_{1:i-1}) + \mathcal{A}_\mathbf{x}(y_i \mid \mathbf{y}_{1:i-1})$$

Algorithm 1 Constrained Sequential Greedy Inference

```
Input: Model $p_\theta$, automaton $\mathcal{A}_\mathbf{x}$, sequence $\mathbf{x}$
Output: Prediction $\hat{\mathbf{y}} \in L(\mathcal{A}_\mathbf{x})$
1: procedure CONSTRAINED-SEARCH($p_\theta$, $\mathcal{A}_\mathbf{x}$, $\mathbf{x}$)
2:     $\hat{\mathbf{y}} \leftarrow$ [BOS]                ▷ Beginning of sequence token
3:     while $\hat{\mathbf{y}}$ is not finished do
4:         $\log \tilde{p}_\theta(y' \mid \mathbf{x}, \hat{\mathbf{y}}) = \log p_\theta(y' \mid \mathbf{x}, \hat{\mathbf{y}}) + \mathcal{A}_\mathbf{x}(y' \mid \hat{\mathbf{y}})$
5:         $y \leftarrow \arg\max_{y'} \log \tilde{p}_\theta(y' \mid \mathbf{x}, \hat{\mathbf{y}})$
6:         $\hat{\mathbf{y}} \leftarrow \hat{\mathbf{y}} + y$             ▷ Extend current sequence
7:     return $\hat{\mathbf{y}}$
```

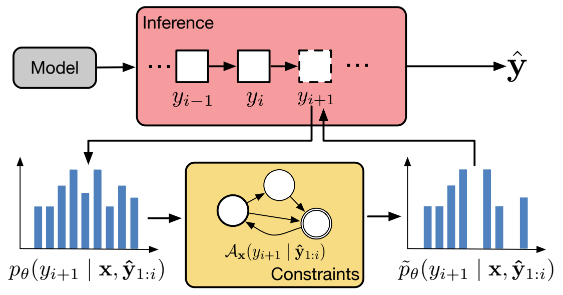

Model
Inference
y_{i-1}
y_i
y_{i+1}
$\hat{\mathbf{y}}$
$p_\theta(y_{i+1} \mid \mathbf{x}, \hat{\mathbf{y}}_{1:i})$
$\mathcal{A}_\mathbf{x}(y_{i+1} \mid \hat{\mathbf{y}}_{1:i})$
Constraints
$\tilde{p}_\theta(y_{i+1} \mid \mathbf{x}, \hat{\mathbf{y}}_{1:i})$

Figure 3: At each time step, the automata $\mathcal{A}_\mathbf{x}$ reshapes the model's probability distribution p_θ to $\tilde{p}_\theta$, from which the next output label y_{i+1} is determined.

where $\mathcal{A}_\mathbf{x}$ reshapes p_θ to $\tilde{p}_\theta$ at each time step. To impose hard constraints, we set this score to $-\infty$ for invalid y_i and constant for all valid y_i.[3] Equation 4 allows natural extensions of sequential inference algorithms, where an automaton is traversed in lock-step to guide the search to always output a valid structure. The necessary extension of greedy search is presented in Algorithm 1. Concretely, when predicting y_i, the inference selects the most probable action under the reshaped probability distribution $\tilde{p}_\theta$ (line 5). It is straightforward to similarly extend other sequential inference algorithms.

2.4 Imposing Multiple Constraints

The formulation above assumes that the constraints are described using a single automaton. However, in some scenarios, it is more natural to impose multiple constraints by representing them as a set of automata.

A Motivating Example. Consider Figure 1 which illustrates the SRL problem. The following constraints should be obeyed by the tag sequence: the predicate cannot have multiple arguments of the same type (A0–A5), each predicate may only accept a certain set of argument types, and certain spans derived from a constituency parse should re-

[3] Soft constraints can also be imposed, but are not explored in this work.

ceive the same label.[4] Each of these constraints can be easily expressed using a separate automaton. In contrast, directly writing a single automaton to enforce these constraints simultaneously could be impractical and difficult.

Issues with Naive Approaches. Naively extending sequential inference algorithms to impose a set of constraints by traversing multiple automata in parallel fails. There is no guarantee that a valid structure will be found, even if the intersection of the automata's languages is non-empty (a proof by counter-example is provided in Appendix A). The alternative solution of intersecting the automata into a single automaton may be intractable, as the size of the intersected automaton grows exponentially in the number of constraints (Hopcroft and Ullman, 1979).

An Active Set Method. Intuitively, intersecting all of the automata may not be necessary because it is possible for a constraint to be satisfied without it being enforced. This is the basis for *active set* methods (such as the cutting-plane algorithm (Kelley, 1960; Tsochantaridis et al., 2005)), which maintain a set during inference that contains currently active (i.e., enforced) constraints. When a constraint is violated by the current output, it enters (i.e., is added to) the active set.

We present an active set method for imposing multiple constraints represented by a set of automata $\mathcal{S}_\mathbf{x}$ in Algorithm 2. Our algorithm is inspired by the active set algorithm of Tromble and Eisner (2006) for finite-state transducers.

For an instance $\mathbf{x}$, Algorithm 2 maintains a active set[5] $\mathcal{W}$ corresponding to all violated constraints so far. $\mathcal{W}$ is represented by the intersection $\mathcal{A}_\mathcal{W}$ of the relevant automata, which is initialized with an automata Σ^* that accepts any sequence (line 1). On each iteration, the algorithm runs a constrained inference algorithm (such as Algorithm 1) that uses $\mathcal{A}_\mathcal{W}$ (line 3) to find an output $\hat{\mathbf{y}}$. Then, FIND-VIOLATION checks if $\hat{\mathbf{y}}$ violates any of the constraints that are not currently in the active set, $\mathcal{S}_\mathbf{x} \setminus \mathcal{W}$ (line 4). If $\hat{\mathbf{y}}$ is accepted by all of the automata (line 5), it is valid and subsequently returned (line 6). Otherwise, the first violated constraint is added to $\mathcal{W}$ (line 8), its automaton $\mathcal{A}'$ intersected with $\mathcal{A}_\mathcal{W}$ (line 9), and constrained inference is re-run (line 3).

[4]These constraints are described in detail in §4.
[5]Also known as a *working set*.

Algorithm 2 An Active Set Method for Multiple Constraints

```
Input: Model $p_\theta$, set of automata $\mathcal{S}_\mathbf{x}$, sequence $\mathbf{x}$
Output: Prediction $\hat{\mathbf{y}} \in \bigcap_{\mathcal{A} \in \mathcal{S}_\mathbf{x}} L(\mathcal{A})$
 1: $\mathcal{W} \leftarrow \emptyset$, $\mathcal{A}_\mathcal{W} \leftarrow \Sigma^*$
 2: while True do
 3:     $\hat{\mathbf{y}} \leftarrow$ CONSTRAINED-SEARCH($p_\theta$, $\mathcal{A}_\mathcal{W}$, $\mathbf{x}$)
 4:     $\mathcal{A}' \leftarrow$ FIND-VIOLATION($\mathcal{S}_\mathbf{x} \setminus \mathcal{W}$, $\hat{\mathbf{y}}$)
 5:     if $\mathcal{A}'$ is null then            ▷ No constraint is violated
 6:         return $\hat{\mathbf{y}}$                 ▷ return current output
 7:     else
 8:         $\mathcal{W} \leftarrow \mathcal{W} \cup \{\mathcal{A}'\}$            ▷ update working set
 9:         $\mathcal{A}_\mathcal{W} \leftarrow \mathcal{A}_\mathcal{W} \cap \mathcal{A}'$          ▷ automata intersection
10: procedure FIND-VIOLATION(K, $\mathbf{y}$)
11:     for each $\mathcal{A}$ in K do
12:         if not $\mathcal{A}$.accepts($\mathbf{y}$) then
13:             return $\mathcal{A}$           ▷ the first violated constraint
14:     return null
```

Algorithm 2 is guaranteed to terminate with a valid output. In the worst case, all of the constraints will be eventually enter the active set and inference will run with a fully intersected automata. Although the cost of this worst case is exponential in the number of constraints, this occurs infrequently in practice. Moreover, we found that Algorithm 2 is faster than naively computing the full intersection despite running inference multiple times.

3 Representing Constraints as Automata

We now illustrate the expressibility of automata by showing how they can represent commonly used constraints in various NLP applications.

Text Generation. Text generation tasks like image captioning, machine translation, sentence simplification, etc., often require that the output must contain specific words or phrases (Anderson et al., 2017; Hokamp and Liu, 2017; Post and Vilar, 2018; Zhang et al., 2017) in order to incorporate prior knowledge (e.g., the caption must have the word "chair" as the object was detected in the image). Similarly, constraints can disallow invalid sequences, such as words which do not rhyme (Ghazvininejad et al., 2016, 2018) or do not appear in a dictionary (Deutsch et al., 2018). These constraints can be represented as FSAs where all paths to an accepting state contain the required sequences and do not contain any disallowed sequences. Note that the size of the automata is not a function of the output vocabulary but the vocabulary that participates in the constraints.

Sequence Tagging. Many sequence tagging problems (such as NER, shallow parsing, etc.) require that the output tags are valid under the specific tagging scheme, such as BIO, which marks each token as beginning, inside, or outside of a phrase. One can easily write an FSA that recognizes the language of valid tag sequences for these schemes, such as the automaton in Figure 2a.

Other constraints commonly applied to sequence tagging are specific to the particular task. In SRL, each argument type can appear exactly one time. For instance, for the label $\mathtt{A}_0$, an FSA with 3 states (before-$\mathtt{A}_0$, emitting-$\mathtt{A}_0$ and after-$\mathtt{A}_0$) can be written to enforce this constraint. See Tromble and Eisner (2006) for examples.

Syntactic Parsing. Syntactic parsing (dependency or constituency) tasks require that the output forms a valid tree, a constraint commonly enforced using the shift-reduce algorithm (Zhu et al., 2013; Nivre et al., 2014; Dyer et al., 2016). Shift-reduce inference inspects the state of a stack to decide which next actions are valid (e.g., if the stack is empty, reduce is invalid). Shift-reduce inference is implicitly using an automaton which is the intersection of a PDA (that counts how many shift and reduce actions have occurred) and an FSA (that restricts the maximum number of actions based on the input sentence length).

Semantic Parsing and Code Generation. In semantic parsing and code generation, constraints ensure both the syntactic validity and the executability of the output. For instance, for the predicate IsAdjacent (which compares two countries for adjacency) the syntactic constraint ensures the predicate receives two arguments, whereas the executability ensures they are properly typed (e.g., that they are countries). Krishnamurthy et al. (2017) use a type-constrained grammar to ensure the output logical form is well-typed. Similarly, Ling et al. (2016) use a stack while decoding to ensure the validity of the generated code. Both these constraints can be encoded in a PDA derived from the grammar of the language (Sipser, 1997).

4 Experimental Setup

We elaborate on two constrained inference tasks from the previous section, constituency parsing and semantic role labeling, which serve as case studies for showing the applicability and versatility of our approach. Our goal is not to beat the state-of-the-art, but to illustrate the practicality and benefits of our approach.

4.1 Models, Datasets, and Evaluation

Constituency Parsing. We follow the experimental setup of Vinyals et al. (2015) and train a seq2seq model with attention using standard splits on the Penn Treebank. This setup was chosen to ensure that no constraints are built into the model for a truly unconstrained baseline. The input is a sequence of tokens, and the model outputs a linearized parse tree ("gold parse" in Figure 4).

We compare our approach (Constrained) to unconstrained inference (Unconstrained), which runs beam search and selects the highest-scoring output. We also compare to Vinyals et al. (2015)'s inference approach (henceforth PostHoc). Vinyals et al. (2015) removes parses without the correct number of preterminals at the end of beam search and adds parentheses to almost-valid parses.[6] The algorithms are evaluated using F_1 as reported by EVALB, a standard evaluation toolkit.[7] However, because EVALB ignores invalid trees during evaluation (which can artificially improve the performance of models which violate constraints), we also report coverage, the percentage of valid outputs. We use a beam size of 10.

Semantic Role Labeling. For SRL, the unconstrained model is an off-the-shelf implementation of He et al. (2017). The input to the model is the sentence and the predicate for which the arguments need to be identified, and the model outputs a tag sequence (such as Figure 1). To isolate the effect of constraints, we assume the gold predicates are available, unlike He et al. (2017). We use the standard train and development splits from the CoNLL 2005 shared task and the same model parameters as He et al. (2017). We report the CoNLL F_1 score[8] computed for the core arguments (namely $\mathtt{A}_0$–$\mathtt{A}_5$) and use greedy inference.

4.2 Constraints

Constituency Parsing. The constraints used in parsing disallow invalid parses, such as the examples in Figure 4. The constraints ensure that **(i)** the parentheses are balanced (Bal), **(ii)** there is exactly one preterminal per input token

[6] personal communication with Vinyals et al. (2015)

[7] `nlp.cs.nyu.edu/evalb/`

[8] `www.lsi.upc.es/~srlconll/srl-eval.pl`

input:	John kissed Mary
gold parse:	`(S (NP XX)(VP XX (NP XX)))`
invalid parse:	`(S (NP )(VP XX XX (NP XX)))`
reason:	empty phrase
invalid parse:	`(S (VP XX (NP XX)))`
reason:	incorrect number of preterminals
invalid parse:	`(S (NP XX)(VP XX (NP XX))))`
reason:	unbalanced parentheses

Figure 4: An example parsing instance with different invalid parses and the constraints they violate (details in §4.2). XX denotes POS tags.

(#PRETERM), and **(iii)** every phrase contains at least one preterminal (NONEMPTY).

Semantic Role Labeling. The constraints used in SRL disallow invalid sequences, such as the ones in Figure 1. The constraints ensure that **(i)** a predicate cannot have multiple arguments of the same type (NODUP). This constraint is expressed using one FSA per argument label for a total of 5 automata; **(ii)** a predicate can only take a certain set of argument types (LEGALARGS), according to PropBank v3.1 (Palmer et al., 2005). This constraint is expressed using one FSA; **(iii)** certain spans in the sentence should be assigned the same argument label type (SPANLABEL). These spans are identified from the predicate and the constituency parse of the sentence using the heuristic of Xue and Palmer (2004).[9] This constraint is expressed using one FSA.

Ease of Implementation. All of the constraints were expressed using an FSA, with the exception of BAL which requires a PDA. The automata were implemented using Pynini (Gorman, 2016). We use the same inference code for both tasks, except for the automata generation.

5 Experimental Results

We show the practicality of our approach and the benefits over unconstrained inference in the experiments below. In order to illustrate the benefits of constrained inference in low-resource settings, we simulate different levels of supervision.

5.1 Constituency Parsing

We experimentally show the need for constraints and compare inference strategies for parsing.

Necessity of Constraints. In order to demonstrate that constraints are necessary to guarantee

[9]We use Kitaev and Klein (2018)'s parser.

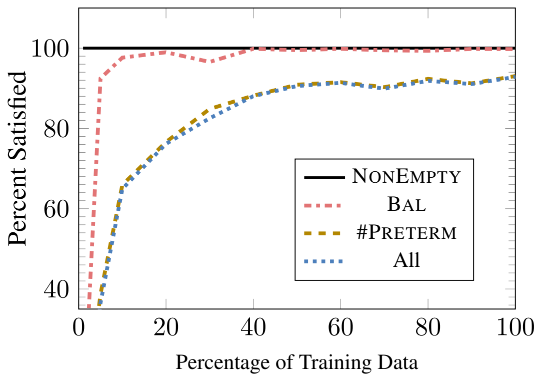


Figure 5: Percentage of output test parses that satisfy the constituency parsing constraints in §4.2 when using unconstrained inference as the amount of training data is increased. "All" measures the percent of parses which satisfy all of the constraints. Even with 100% of the training data, the model fails to always output the correct number of preterminals.

structurally valid output, we vary the amount of training data for a seq2seq model and measure how frequently the output is an invalid parse tree.

The results in Figure 5 show that while the model learns some constraints with little data, others are frequently violated even with full supervision. For instance, the model outputs parse trees with balanced parentheses for over 94% of instances, after training on as little as 5% of the training data. However, the model frequently violates the #PRETERM constraint, outputting an incorrect number of preterminals on about 7% of parses even after training on 100% of the data. Therefore, we cannot expect seq2seq models to learn to produce valid outputs, even when it has access to 40k labeled parse trees, let alone in low-resource settings.

Comparing Inference Approaches. Figure 6 shows the performance of all the three inference techniques discussed in §4.1, namely, UNCONSTRAINED, POSTHOC, and CONSTRAINED.

At first glance, it appears that UNCONSTRAINED is slightly better than both POSTHOC and CONSTRAINED at 100% supervision. However, EVALB ignores any invalid parse trees when computing F_1, and therefore it is necessary to take coverage (the percent of valid output parses) into account when comparing inference approaches. It is evident from Figure 6 (left) that CONSTRAINED always ensures 100% coverage, and POSTHOC only reaches near-100% coverage with full supervision. In contrast, UNCONSTRAINED

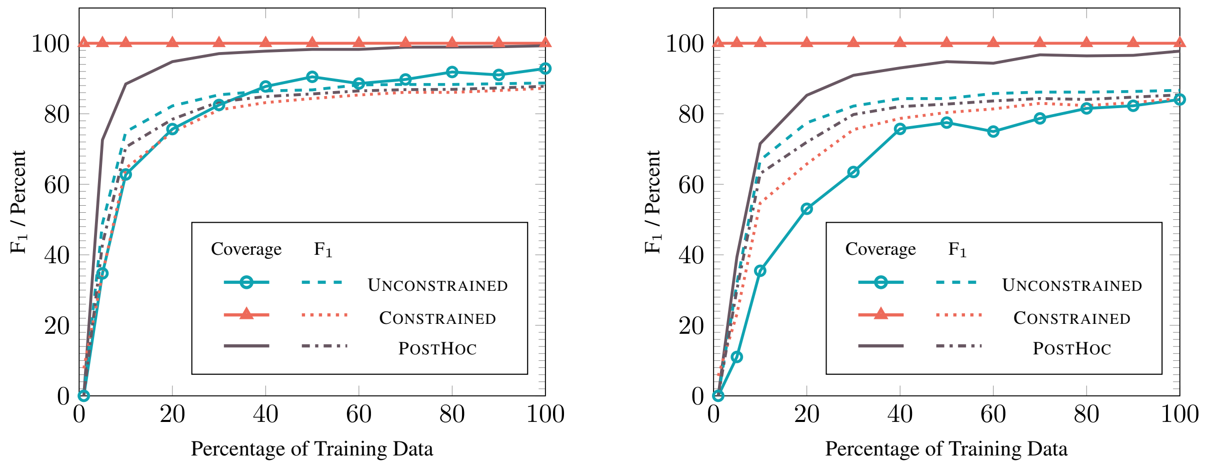


Figure 6: F_1 and coverage (# valid output parses / # test instances) for UNCONSTRAINED, POSTHOC, and CONSTRAINED on the entire test data (**left**) and test sentences of length $\geq$30 (**right**). The UNCONSTRAINED and POSTHOC inference algorithms have a harder time producing valid parse trees for the longer input sentences.

achieves consistently low coverage, with the best model reaching 7% lower coverage than the CONSTRAINED.

The increased coverage explains the apparent drop in F_1 for constrained approaches. Intuitively, longer sentences are inherently harder to parse. CONSTRAINED and POSTHOC produce a valid, but potentially incorrect parse for these sentences and get penalized. In contrast, UNCONSTRAINED is more likely to produce invalid parses for these sentences, and is thus effectively evaluated on a test set containing short sentences. To verify this, we re-evaluated the inference approaches on test sentences which are $\geq$30 tokens (Figure 6, right). Under this setting, every approach has worse F_1 and coverage at all levels of supervision (Figure 6 left vs right).

5.2 Semantic Role Labeling

We show that we can improve the performance of a trained model by incorporating constraints at inference time which address common errors (Daza and Frank, 2018) made by SRL models. This experiment also shows the efficiency of the active set method in enforcing multiple constraints.

Correcting Common Errors. We now show how some common SRL errors like the ones discussed in Figure 1, can be corrected using constraints added at inference time. Starting with an unconstrained baseline model (UNCONSTRAINED), we successively add the NODUP, LEGALARGS and SPANLABEL constraints, in that order.

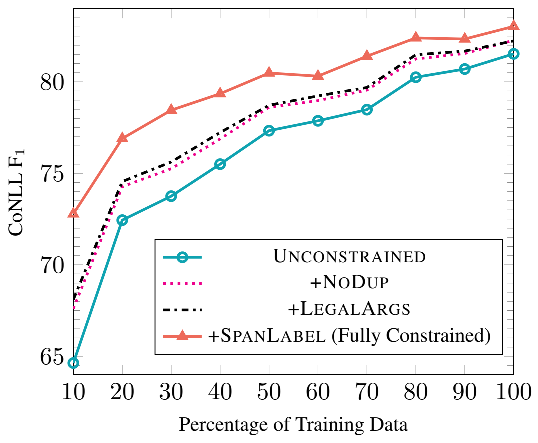


Figure 7: Learning curve for SRL. Starting with unconstrained inference, we incrementally add NODUP, LEGALARGS and SPANLABEL constraints. The constraints improve performance at all levels of supervision, with the largest improvements in low supervision settings.

Figure 7 shows that using the constraints improves performance over UNCONSTRAINED in all settings. Constraints like NODUP and SPANLABEL give significant gains, demonstrating their value. At 100% supervision, fully constrained inference improves by 1.5 F_1 points over unconstrained inference (83.03 vs 81.53). In fact, constrained inference at only 50% data achieves similar F_1 score as UNCONSTRAINED at 100%, with larger gains in low supervision regimes ($\leq$40%).

Active Set Size and Efficiency. For SRL, the maximum possible size of the active set $\mathcal{W}$ is 7 for any test instance. Intersecting all 7 automata would lead to an automaton with 1043 states and

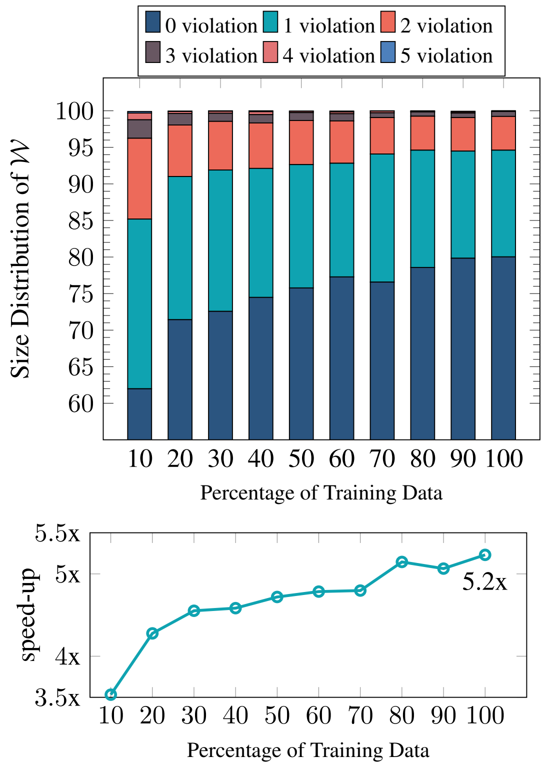


Figure 8: **Top**: Size distribution of the final working set for the active set method. With a model trained on 30% of the data, 72% of the test instances have an empty final working set (i.e., with 0 violations). **Bottom**: Relative speed-up in decoding time using the active set method over naively computing the full intersection.

2022 arcs. We now measure the size of the active set observed in practice.

Figure 8 shows the size distribution of the final $\mathcal{W}$ under different amounts of supervision. With a fully supervised model, 80% of the test instances had an empty $\mathcal{W}$ when inference terminated (i.e., no constraints entered $\mathcal{W}$). Under the same setting, $\mathcal{W}$ contains 1 and 2 constraints for 10.6% and 8.6% of the instances, respectively. Under any setting, $\mathcal{W}$ was empty for $>$60% of the instances. On average, the active set automaton had 34 states and 66 arcs, a 30x reduction over the full intersection.

To evaluate the efficiency of Algorithm 2, Figure 8 (bottom) plots the relative decoding times of the active set method over naively computing the full intersection. The active set method is consistently faster, with the largest speed-up (5.2x) at the highest level of supervision as the average active set size is the smallest at this setting.

Factors Affecting Speed-up. In general, the amount of speed-up provided by the active set method depends on several factors, including the number of constraints, the size of the constraint automata, and the cost of computing the softmax during inference. The largest gains will come when the former two factors are most expensive, as the active set will only incur the intersection cost as needed. If the output vocabulary is large, softmax computation may outweigh the cost of fully intersecting the constraint automata.

6 Related Work

Traditional Constrained Inference. Traditional constrained inference approaches enforced constraints using general combinatorial optimization frameworks, for instance linear (Taskar et al., 2004) and integer linear programs (Roth and Yih, 2005, 2007; Clarke and Lapata, 2008; Martins et al., 2009), SAT solvers (Richardson and Domingos, 2006; Poon and Domingos, 2008), etc. Unlike these approaches which find the best output that satisfies a set of constraints, our work attempts to provide a similar general framework for sequential inference algorithms that will find the approximate-best output.

Unconstrained Data-driven Approaches. Many sequential inference approaches do not enforce constraints at all, in the hope that they will be learned from data (Lample et al., 2016; Choe and Charniak, 2016; Suhr et al., 2018). While the model can potentially "impose" some constraints which are well-represented in the data, there is no guarantee that the output structure will be valid. In contrast, our work guarantees valid output.

Post-Hoc Constraint Satisfaction. Some approaches first run unconstrained inference to find the top-k structures and then identify valid structures in a post-hoc manner (Andreas et al., 2013; Vinyals et al., 2015; Kiddon et al., 2016; Upadhyay et al., 2018). Such techniques also cannot guarantee validity of the output structure when all top-k structures are invalid, whereas our model ensures all top-k are valid.

Our Work. We draw from work in NLP that uses automata (Mohri, 1997; Karttunen, 2000, inter alia) and recent work in constrained text generation (discussed in §3). Our work is also related to Anderson et al. (2017) who impose lexical constraints for image captioning by maintaining a beam for *each* state in an FSA, an impractical strategy for automata with thousands of states (like those in §5.2). Tromble and Eisner (2006) was the inspiration for the active set method. A very recent work, Lee et al. (2019), also impose constraints in sequential inference by including them in the objective function and modifying the model's weights until the constraints are satisfied.

7 Conclusion and Future Work

We presented a principled, general-purpose constrained sequential inference algorithm. Key to our algorithm is using automata to represent constraints, which we showed are capable of expressing popularly used constraints in NLP. Our approach is an attractive alternative to task-specific constrained inference approaches currently in use. Using a fast active set method, we can seamlessly incorporate multiple constraints at inference time without modifying the inference code. The experimental results showed the value of our approach over unconstrained inference, with the gains becoming more prominent in low-resource settings.

Acknowledgments

The authors would like to thank Sampath Kannan, Caleb Stanford, Jordan Kodner, and the anonymous reviewers for their useful comments and suggestions.

This work was supported by Contract HR0011-15-C-0113 and Contract HR0011-18-2-0052 with the US Defense Advanced Research Projects Agency (DARPA). Approved for Public Release, Distribution Unlimited. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

References

Peter Anderson, Basura Fernando, Mark Johnson, and Stephen Gould. 2017. Guided Open Vocabulary Image Captioning with Constrained Beam Search. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 936–945, Copenhagen, Denmark. Association for Computational Linguistics.

Jacob Andreas, Andreas Vlachos, and Stephen Clark. 2013. Semantic Parsing as Machine Translation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 47–52, Sofia, Bulgaria. Association for Computational Linguistics.

Do Kook Choe and Eugene Charniak. 2016. Parsing as Language Modeling. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2331–2336, Austin, Texas. Association for Computational Linguistics.

James Clarke and Mirella Lapata. 2008. Global inference for sentence compression: An integer linear programming approach. *Journal of Artificial Intelligence Research*, 31:399–429.

Hal Daumé, John Langford, and Daniel Marcu. 2009. Search-based Structured Prediction. *Machine learning*, 75(3):297–325.

Angel Daza and Anette Frank. 2018. A Sequence-to-Sequence Model for Semantic Role Labeling. In *Proceedings of The Third Workshop on Representation Learning for NLP*, pages 207–216. Association for Computational Linguistics.

Daniel Deutsch, John Hewitt, and Dan Roth. 2018. A Distributional and Orthographic Aggregation Model for English Derivational Morphology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1938–1947. Association for Computational Linguistics.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 199–209, San Diego, California. Association for Computational Linguistics.

Marjan Ghazvininejad, Yejin Choi, and Kevin Knight. 2018. Neural Poetry Translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 67–71, New Orleans, Louisiana. Association for Computational Linguistics.

Marjan Ghazvininejad, Xing Shi, Yejin Choi, and Kevin Knight. 2016. Generating Topical Poetry. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1183–1191, Austin, Texas. Association for Computational Linguistics.

Daniel Gildea and Daniel Jurafsky. 2002. Automatic Labeling of Semantic Roles. *Computational Linguistics*, 28(3):245–288.

Kyle Gorman. 2016. Pynini: A Python Library for Weighted Finite-state Grammar Compilation. In *Proceedings of the SIGFSM Workshop on Statistical NLP and Weighted Automata*, pages 75–80, Berlin, Germany. Association for Computational Linguistics.

Luheng He, Kenton Lee, Mike Lewis, and Luke Zettlemoyer. 2017. Deep Semantic Role Labeling: What Works and What's Next. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages

473–483. Association for Computational Linguistics.

Chris Hokamp and Qun Liu. 2017. Lexically Constrained Decoding for Sequence Generation Using Grid Beam Search. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1535–1546, Vancouver, Canada. Association for Computational Linguistics.

John E Hopcroft and Jeffrey D Ullman. 1979. *Introduction to automata theory, languages, and computation*. Addison-Wesley.

Lauri Karttunen. 2000. Applications of finite-state transducers in natural language processing. In *International Conference on Implementation and Application of Automata*, pages 34–46. Springer.

James E Kelley, Jr. 1960. The Cutting-Plane Method for Solving Convex Programs. *Journal of the society for Industrial and Applied Mathematics*, 8(4):703–712.

Chloé Kiddon, Luke Zettlemoyer, and Yejin Choi. 2016. Globally Coherent Text Generation with Neural Checklist Models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 329–339, Austin, Texas. Association for Computational Linguistics.

Nikita Kitaev and Dan Klein. 2018. Constituency Parsing with a Self-Attentive Encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2676–2686. Association for Computational Linguistics.

Jayant Krishnamurthy, Pradeep Dasigi, and Matt Gardner. 2017. Neural semantic parsing with type constraints for semi-structured tables. In *EMNLP*.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural Architectures for Named Entity Recognition. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 260–270, San Diego, California. Association for Computational Linguistics.

Jay Yoon Lee, Sanket Vaibhav Mehta, Michael Wick, Jean-Baptiste Tristan, and Jaime G. Carbonell. 2019. Gradient-based inference for networks with output constraints. In *AAAI 2019*.

Wang Ling, Phil Blunsom, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiský, Fumin Wang, and Andrew Senior. 2016. Latent predictor networks for code generation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 599–609. Association for Computational Linguistics.

Minh-Thang Luong, Quoc V Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. 2016. Multi-task Sequence to Sequence Learning. In *Proc. of ICLR*.

André FT Martins, Noah A. Smith, and Eric P. Xing. 2009. Concise Integer Linear Programming Formulations for Dependency Parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Mehryar Mohri. 1997. Finite-state Transducers in Language and Speech Processing. *Computational linguistics*, 23(2):269–311.

Joakim Nivre, Yoav Goldberg, and Ryan McDonald. 2014. Constrained Arc-Eager Dependency Parsing. *Computational Linguistics*, 40(2).

Martha Palmer, Daniel Gildea, and Paul Kingsbury. 2005. The proposition bank: An annotated corpus of semantic roles. *Computational linguistics*, 31(1):71–106.

Hoifung Poon and Pedro Domingos. 2008. Joint Unsupervised Coreference Resolution with Markov Logic. In *Proceedings of the Conference on Empirical Methods for Natural Language Processing (EMNLP)*, pages 650–659.

Matt Post and David Vilar. 2018. Fast Lexically Constrained Decoding with Dynamic Beam Allocation for Neural Machine Translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1314–1324. Association for Computational Linguistics.

Vasin Punyakanok, Dan Roth, and Wen tau Yih. 2008. The Importance of Syntactic Parsing and Inference in Semantic Role Labeling. *Computational Linguistics*, 34(2).

Matthew Richardson and Pedro Domingos. 2006. Markov Logic Networks. *Machine Learning Journal*, 62(1-2):107–136.

Dan Roth and Scott Wen-tau Yih. 2004. A linear programming formulation for global inference in natural language tasks. In *Proc. of the Conference on Computational Natural Language Learning (CoNLL)*, pages 1–8. Association for Computational Linguistics.

Dan Roth and Scott Wen-tau Yih. 2005. Integer Linear Programming Inference for Conditional Random Fields. In *Proc. of the International Conference on Machine Learning (ICML)*, pages 737–744.

Dan Roth and Wen-tau Yih. 2007. Global Inference for Entity and Relation Identification via a Linear Programming Formulation.

Michael Sipser. 1997. *Introduction to the Theory of Computation*, volume 2. PWS Publishing.

Alane Suhr, Srinivasan Iyer, and Yoav Artzi. 2018. Learning to Map Context-Dependent Sentences to Executable Formal Queries. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2238–2249. Association for Computational Linguistics.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Advances in neural information processing systems*, pages 3104–3112.

Oscar Täckström, Kuzman Ganchev, and Dipanjan Das. 2015. Efficient inference and structured learning for semantic role labeling. *Transactions of the Association for Computational Linguistics*, 3:29–41.

B. Taskar. 2004. *Learning Structured Prediction Models: A Large Margin Approach*. Ph.D. thesis, Stanford University. Computer Science TR-280.

B. Taskar, C. Guestrin, and D. Koller. 2004. Max-margin markov networks. In *The Conference on Advances in Neural Information Processing Systems (NIPS)*.

Roy Tromble and Jason Eisner. 2006. A fast finite-state relaxation method for enforcing global constraints on sequence decoding. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 423–430, New York City, USA. Association for Computational Linguistics.

I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. 2005. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484.

Shyam Upadhyay, Jordan Kodner, and Dan Roth. 2018. Bootstrapping Transliteration with Constrained Discovery for Low-Resource Languages. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 501–511. Association for Computational Linguistics.

Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a Foreign Language. In *Advances in Neural Information Processing Systems*, pages 2773–2781.

Nianwen Xue and Martha Palmer. 2004. Calibrating Features for Semantic Role Labeling. In *Proceedings of the Conference on Empirical Methods for Natural Language Processing (EMNLP)*, pages 88–94, Barcelona, Spain.

Yaoyuan Zhang, Zhenxu Ye, Yansong Feng, Dongyan Zhao, and Rui Yan. 2017. A Constrained Sequence-to-Sequence Neural Model for Sentence Simplification. *CoRR*, abs/1704.02312.

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and accurate shift-reduce constituent parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 434–443, Sofia, Bulgaria. Association for Computational Linguistics.

A Counter Example for Lock-Step Traversal of Multiple Automata

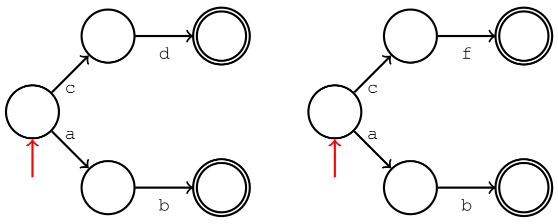


Figure 9: Counter example showing that traversing two automaton in parallel may not find any string in the intersection, even if one exists. The two automaton above have a non-empty intersection (the string `ab`), but if the search algorithm takes the edge `c`, then inference will not find any valid string.