# Constrained Markov Chains, Memory Augmentation, and Mediating Variables

**Anonymous authors**
**Paper under double-blind review**

## Abstract

Imposing constraints on a Markov chain can create global dependencies that violate the Markov property and render standard algorithms inapplicable. A classical solution is memory augmentation, which augments the chain with an auxiliary state that records the constraint-relevant history. However, existing memory-based methods are scattered across several disciplines and are typically specialized to particular constraints or tasks such as inference or sampling. We present a probabilistic formulation of memory augmentation over both discrete and continuous spaces, recasting memory augmentation as generating a common hidden Markov model. This perspective unifies various memory constructions in the literature and reduces several constrained tasks, including inference, learning, and sampling, to standard algorithms on the augmented model. It also yields a compositional framework for constructing and verifying which constraints admit tractable representations, covering common examples like hitting, precedence, and counting constraints.

## 1 Introduction

Markov chain models, such as hidden Markov models (HMMs), dynamic Bayesian networks, and state space models, are widely used across many fields including computational biology (Durbin et al., 1998), natural language processing (Manning & Schütze, 1999), and computer vision (Forsyth & Ponce, 2002). Their success is partly due to efficient algorithms supporting inference (Rabiner, 1989), learning (Baum et al., 1970), and sampling (Carter & Kohn, 1994; Frühwirth-Schnatter, 1994) that exploit the Markov property. Despite their flexibility in modeling real-world phenomena, some applications motivate placing constraints on the model's variables (Christiansen et al., 2010; Pachet et al., 2011). These constraints may introduce long-range dependencies that violate the Markov property, disabling the algorithms that make these models attractive and creating a need for alternative constrained analysis methods.

Many existing methods for constrained inference fall naturally into three families, each making a different compromise. Path-finding algorithms such as beam search (Daumé et al., 2009; Doppa et al., 2014) and penalized sequential methods (Roth & Yih, 2004; 2005; Deutsch et al., 2019) build the path one step at a time using only local information, an approach that has difficulty enforcing global constraints. For example, a constraint requiring a specific token subsequence to appear somewhere in the output cannot be checked until the sequence is complete, and no local penalty can guarantee that the constraint will be met (Cohen & Beck, 2019; Stahlberg & Byrne, 2019; Meister et al., 2020). Reformulations into equivalent problems in other domains, such as logic programs (Christiansen et al., 2010) and integer linear programs (Roth & Yih, 2004; 2005), are exact but treat the problem as a combinatorial optimization over the entire path space, with runtimes that can scale poorly in the time horizon (Karp, 1972; Globerson et al., 2015; Meshi et al., 2016). Monte Carlo and variational methods (Foti et al., 2014; Blei et al., 2017) approximate the constrained distribution rather than representing it exactly, which can be undesirable when the constraint must hold with certainty or when downstream decisions depend on the exact posterior.

Similarly, methods for constrained sampling suffer from similar tradeoffs. Naive rejection sampling can be inefficient when constraint satisfaction is a rare event (Biondini, 2015). Doob's h-transform sampling reweights the Markov kernel to achieve exact constrained sampling, but computing these weights can be

highly nontrivial when conditioning on certain events (Du et al., 2024). Approximate methods sample from the constrained chain using a variety of strategies such as approximating the h-function (Du et al., 2024; Schauer et al., 2017; Corstanje et al., 2023), importance weighting to encourage feasible paths (Aguilar & Gatto, 2024), and utilizing particle filters Moral (2012). However, such approximate sampling methods might be undesirable for the same reasons as in the constrained inference case.

**Memory augmentation:** The Markov property can fail when conditioning on a constraint whose satisfaction depends on the entire path $X_{1:T}$, as conditioning can couple all relevant time steps. However, for many constraints these global dependencies can be handled using a deterministic memory variable that updates with the chain. Consider a constraint requiring that a noncontiguous subsequence of states $(a, b, c)$ occurs somewhere in the trajectory. At any time, all we need to know about the past is which states in the subsequence have been visited already, and a single memory variable tracking this progress carries everything the constraint needs to know. By augmenting the constrained chain with this memory variable, the dependency is broken into local steps, restoring the Markov property.

Memory augmentation is a classical strategy for representing constrained Markov chains, with successful applications in many fields including pattern matching (Lladser et al., 2007; Nuel, 2008), computational biology (Tataru et al., 2013; Titsias et al., 2016), and most prominently model checking and formal verification Baier & Katoen (2008). However, prior work leveraging memory is fragmented across different disciplines, often specialized to specific constraint families, and limited to only a few tasks like inference or sampling. Our contribution is to provide a unified probabilistic account of memory augmentation, showing how generic memory constructions can be abstracted into a common probabilistic model. The benefit of this abstraction is collecting various constrained tasks, such as inference and sampling, under a single standard algorithmic workflow on this model. We also provide a formal framework for determining which constraints admit tractable representations, extending the scope of memory augmentation beyond the constraint families individually considered in prior works.

**Memory as probabilistic mediation:** Interpreted probabilistically, memory augmentation generates a sequence of random variables that, together with the original chain, form a new augmented process. This sequence of memory variables $\{M_t\}$ mediates between the chain $\{X_t\}$ and the constraint indicator $\mathcal{C}$ (Figure 1b). Our naming choice of 'mediation' rather than 'memory' echoes their functional role and highlights this probabilistic interpretation. This augmented process $\{(M_t, X_t)\}$, together with the constraint indicator $C_t = \text{eval}(M_t)$, is a hidden Markov model (HMM) in which constraint satisfaction is a single binary emission (Theorem 2.3). In other words, memory augmentation yields a common HMM structure that acts as a surrogate for the original constrained chain, transforming constrained tasks into posterior queries on an HMM. Immediately, this implies the entire HMM algorithmic toolkit can be directly applied to support a large suite of constrained tasks, providing a single systematic workflow and dispensing with the need to construct bespoke algorithms tailored to specific constraints or tasks.

**Our Contributions:** The classical procedure of memory augmentation can be recast probabilistically as inserting a sequence of mediating variables into a Markov chain. We show how this mediation variable perspective unifies various memory constructions under a single probabilistic model. This perspective has several payoffs, organized into the following key points:

1. **Single standard workflow** (Section 2). We show that a generic memory-augmented Markov chain is a HMM whose constrained measure agrees with the original chain's, transforming constrained queries into equivalent posterior queries on an HMM (Theorem 2.3). This allows the standard HMM algorithmic toolkit to be directly applied on the augmented model, such as forward-backward for constrained marginals (Bishop, 2007), Viterbi for constrained inference (Forney, 1973), Baum–Welch for constrained learning (Rabiner, 1989), and Forward-Filtering Backward-Sampling (FFBS) (Carter & Kohn, 1994; Frühwirth-Schnatter, 1994) for constrained sampling. This dispenses with the need to design constraint-specific procedures and systematizing various tasks - including constrained inference, learning, and sampling - under a single workflow.

2. **Compositional framework of representable constraints** (Section 4). We show that a rich class of constraints - including hitting, precedence, counting, dwell-time, and resource constraints - can all be represented as mediation variables and composed freely using union, intersection, precedence, counts, and other operations. This provides a practical framework for constructing and verifying when a constraint admits a tractable representation. Table 1 lists common examples from pattern matching, formal methods, and constrained sequence modeling, showing how each one is built from this framework.

3. **Algorithms for additional constrained queries** (Section 5). We show how the HMM structure facilitates additional constrained queries: posterior satisfaction probability (algorithm 2), the distribution of the satisfaction time (algorithm 1), and posterior sampling of constrained paths with random lengths determined by a stopping condition (algorithm 3). While these queries have been partially explored in prior work (satisfaction probability in model checking (Baier & Katoen, 2008), distribution of first-passage and changepoint times (Hobolth et al., 2024; Fearnhead & Rigaill, 2019)), our framework integrates them as additional capabilities of a single toolkit applicable to any constraint represented by mediation variables.

4. **Empirical capability demonstration and validation** (Section 6). We provide two case studies to demonstrate the breadth of tasks supported by the HMM structure. Firstly, a DNA example with heterogeneous constraints showcases the full algorithmic suite. Secondly, an energy grid example with a dynamic budget provides a continuous-state space demonstration using a time-varying representation. We provide runtime comparisons on constrained inference tasks against exact and inexact baselines to highlight problem regimes where memory augmentation can be an attractive solution, and we also look at how the runtime of constrained inference on the augmented model grows as the time horizon, state space size, and number of constraints grows.

**Related Works** Memory augmentation is a well-known construction in the statistics literature. For exact constrained inference, finite memory has been used to tackle a variety of constraints including run length and hit counts (Fu & Koutras, 1994; Koutras & Alexandrou, 1995) as well as pattern matching (Nicodème et al., 2002; Lladser et al., 2007; Nuel, 2008; Tataru et al., 2013). However, these works typically design algorithms tailored to particular constraints and, with some exceptions (Tataru et al., 2013), focus exclusively on inference. Constrained sampling methods have similarly utilized finite memory constructions, but they likewise remain limited to exclusively sampling with a specific constraint family in mind (Pachet et al., 2011; Papadopoulos et al., 2015). A notable exception is prior work on jump count constraints, which enables constrained inference, learning, and sampling using a memory-augmented model (Titsias et al., 2016), and our framework extends this broad algorithmic capability to constraint families beyond jump counts and supports additional constrained queries. Our contribution lies not in the memory construction itself but in providing an abstraction that unifies these various constructions. Along with our compositional framework, our mediation variable abstraction compiles prior work in memory augmentation - fragmented across disciplines, constraints, and tasks - under a single model that applies to a rich class of constraints and simultaneously supports constrained inference, learning, sampling, and other tasks.

As a special case of memory augmentation, augmenting a Markov chain with a finite-state automaton is a classical construction in probabilistic model checking (PMC) (Pnueli, 1977; Vardi, 1985; Hansson & Jonsson, 1994; Bianco & de Alfaro, 1995; Courcoubetis & Yannakakis, 1995; Baier & Katoen, 2008; Forejt et al., 2011; Katoen, 2016), where this automaton-and-chain construction is primarily used to answer verification and reachability queries of the form "does property $\phi$ hold with at least probability $p$?". For regular constraints over finite spaces, our regular mediation-variable representation coincides with this classical product construction. Our contribution is a probabilistic reinterpretation of this construction: we view the augmented system as an HMM and use this perspective to support posterior queries, such as latent-state inference, parameter learning, and exact path sampling under constraints, that are not typically considered in the PMC setting. Furthermore, our compositional framework includes several operators - such as precedence, counts, and never satisfied - that are nonstandard in the automata literature but are central to describing common constraint families. The continuous state space treatment sharpens the distinction. Continuous-state verification in PMC typically proceeds by first constructing a finite abstraction of the continuous chain and verifying constraints

on this abstraction (Abate et al., 2008; 2010; Soudjani et al., 2015; Haesaert et al., 2017; Lavaei et al., 2022; Badings et al., 2023b). By contrast, mediation variables provide an exact model-level representation of the constrained chain before discretization. When mediation variables are incorporated during discretization (Section 6.2), the resulting finite approximation remains sound with respect to the constraint: every path and its discretized image are both feasible or both infeasible (Appendix H).

A recent line of work in the deep learning community leverages similar automaton-and-chain constructions for controllable generation in large language models. Some works distill a language model into an automaton-augmented hidden Markov model, which is used to compute exact constrained marginals that help guide token generation toward feasible paths (Zhang et al., 2023; 2024). Other work takes a similar approach by using an automaton to mask invalid tokens during text generation (Koo et al., 2024). These methods must also scale to model sizes far larger than the typical statistical setting we consider, a challenge we do not address. Nonetheless, these methods focus exclusively on constrained sampling and, like the constrained sampling works mentioned above, employ special cases of a more general construction under our framework. Our framework identifies the augmented HMM as the central object that enables a broader suite of probabilistic queries, including constrained (stopped) sampling.

### 1.1 Paper Outline

Section 2 introduces the problem setting, defining constraints on Markov chains and mediation variable representations. Section 3 highlights connections between automata, language, and probability theory within mediating variables. Section 4 provides a framework of basic amenable constraint types and closure properties, with Table 1 listing common examples. Section 5 describes algorithms for computing additional constraint queries enabled by the full augmented HMM. Section 6 provides two capability demonstrations of our framework's breadth on DNA and energy grid examples, runtime comparisons on constrained inference tasks against exact (Integer Linear Programming) and inexact (penalized beam search) baselines, and scaling studies on memory augmentation with respect to the state space size, time horizon, and number of constraints.

## 2 Constraints and Mediation Variables

The introduction sketched the mediation variable idea informally through the subsequence example, and in this section we make these ideas precise. We define what a constraint is, define what it means to represent a constraint by mediation variables, and state the central HMM identification result (Theorem 2.3).

### 2.1 Technical Preliminaries

We work with a discrete-time, one-sided Markov chain $\{X_t\} = X_1, X_2, \cdots$ on a standard Borel space $(\mathcal{X}, \mathcal{F})$ with the canonical filtration $\mathcal{F}_t = \sigma(X_1, \cdots, X_t)$. Since $\{\mathcal{F}_t\}$ is canonical, we abuse notation slightly and treat a set $A \in \mathcal{F}_t$ as a measurable subset of $\mathcal{X}^t \equiv \mathcal{X}^{\otimes t}$. We write $X_{1:t}$ for $X_1, \cdots, X_t$ and $\mu(x)$ in place of $\mu(dx)$ where convenient. While the standard Borel setting is what lets the framework cover continuous state spaces, a reader who prefers concrete intuition can mentally substitute a finite state space throughout, in which case $\mathcal{F}_t$ is just the powerset of length-$t$ strings and a constraint is just a family of feasible sequence sets. This finite setting provides a simpler illustration of the core concepts with little loss of generality.

### 2.2 Mediation Variable Representations of Constraints

In many applications, Markov chains are used to model finite sequences of data, and constraints on the chain are understood as constraints on finite-length paths. Importantly, the length of a feasible path can vary. In the introduction's subsequence example, any finite path is feasible as long as the required subsequence occurs at some point, and it does not matter whether this path has thirty or three hundred steps. Hence, we define a constraint as a family of feasible finite-length paths $x_{1:t}$, indexed by their length $t$.

**Definition 2.1.** A **constraint** $\mathcal{C}$ on $\{X_t\}$ is a family of feasible sets:

$$\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \cdots\} \ ; \ \mathcal{C}_t \in \mathcal{F}_t$$

We also identify each set $\mathcal{C}_t$ with its indicator function and use the same symbol for both, with context distinguishing the two. Treating $\mathcal{C}_t$ as a function on $\mathcal{X}^t$, a constraint is equivalently a family of binary functions adapted to $\mathcal{F}_t$. For any time horizon $T$, a Markov chain specifies a distribution $\mu_T$ over finite paths $X_{1:T}$. Imposing the constraint amounts to restricting $\mu_T$ to $\mathcal{C}_T$ and renormalizing, producing the constrained measure $\mu_T^{\mathcal{C}} = P(X_{1:T}|\mathcal{C}_T = 1)$. We drop the time subscript when context allows.

(a) Original Markov Model

(b) Equivalent Hidden Markov Model

Figure 1: Memory augmentation as a mediating sequence of variables

A major difficulty with constrained Markov chains is that conditioning on a constraint generally couples the chain's variables (Figure 1a), violating the Markov property and disabling standard algorithms. The introduction's subsequence example illustrates the memory augmentation solution: add a progress tracker that contains all constraint-relevant past information. Augmenting the chain with a memory variable restores the Markov property by mediating between the chain and the constraint (Figure 1b). We formalize the notion of a generic memory construction by defining mediation variable representations.

**Definition 2.2.** A **mediation variable representation** (MVR) is a 5-tuple $(\mathcal{X}, \{\mathcal{M}_t\}, init, \{upd_t\}, \{eval_t\})$:

1. $\mathcal{X}$ is the state space.
2. $\{\mathcal{M}_t\}$ are finite spaces.
3. Measurable maps:
   (a) $init : \mathcal{X} \to \mathcal{M}_1$.
   (b) $upd_t : \mathcal{M}_{t-1} \times \mathcal{X} \to \mathcal{M}_t$ for $t > 1$.
   (c) $eval_t : \mathcal{M}_t \to \{0, 1\}$ for $t \geq 1$.

The maps in our definition generate a sequence of random variables $\{M_t\}$ that track constraint information. After initializing $M_1 = init(X_1)$, the memory state updates deterministically with the current state $X_t$: $M_t = upd_t(M_{t-1}, X_t)$ for $t > 1$. At any time $t$, we can determine feasibility of the current path by evaluating the memory state: $eval_t(M_t)$. We say that **a MVR represents the constraint** $\mathcal{C}$ if $eval_t(M_t) = \mathcal{C}_t$ for all $t$. The following result identifies every MVR-augmented chain (Figure 1b) as an HMM.

**Theorem 2.3.** *Let $\mathcal{C}$ be a constraint on $\{X_t\}$ with MVR $\{M_t\}$. For any Markov measure $\mu$ on $\{X_t\}$ and any time horizon $T$:*

*(a) The augmented sequence $\{(M_t, X_t)\}$ is a Markov chain, and $\{(M_t, X_t), C_t\}$ is a hidden Markov model.*

*(b) The constrained measure over the augmented chain $\tilde{\mu}^{\mathcal{C}} = P(m_{1:T}, x_{1:T} \mid \mathcal{C}_T = 1)$ is Markov.*

*(c) Over its support, $\tilde{\mu}^{\mathcal{C}}$ depends only on $x_{1:T}$:*

$$\hat{\mu}^{\mathcal{C}} \equiv \tilde{\mu}^{\mathcal{C}}|_{supp(\tilde{\mu}^{\mathcal{C}})} = \tilde{\mu}^{\mathcal{C}}(m(x_{1:T}), x_{1:T})$$

*(d) Regarding $\hat{\mu}^{\mathcal{C}}$ as a measure on $X_{1:T}$, $\hat{\mu}^{\mathcal{C}}$ coincides with the original chain's constrained measure:*

$$\hat{\mu}^{\mathcal{C}}(x_{1:T}) = P(x_{1:T} \mid \mathcal{C}_T = 1)$$

In plain terms, the augmented process with binary emissions $\mathcal{C}_t = eval(M_t)$ is a HMM (a), conditioning on $\mathcal{C}_T = 1$ keeps the augmented chain Markov (b), the augmented chain's conditional measure is degenerate in the $M$ component since $M_{1:T}$ is a deterministic function of $X_{1:T}$ (c), and projecting back down to $X_{1:T}$ recovers exactly the original chain's constrained measure (d).

The upshot with MVR augmentation is that working with a constrained Markov chain reduces to working with an HMM conditioned on a single emission. Standard HMM algorithms apply directly for posterior queries like constrained inference, sampling, and learning. Theorem 2.3 dispenses with the need to design algorithms for specific queries or tailored to specific constraints. Once an MVR is constructed for a constraint, the standard HMM toolkit does the rest.

### 2.3 Constraints on Subsequences

In some applications a constraint applies only to a contiguous subsequence rather than the full path, and the setup extends to this case with no real change. A **subsequence constraint $(\mathcal{C}, [t_1, t_2])$** asks that $x_{t_1:t_2} \in \mathcal{C}_{t_2-t_1+1}$, and a **MVR of $(\mathcal{C}, [t_1, t_2])$** initializes at time $t_1$ via $M_{t_1} = init(X_{t_1})$ and evaluates at time $t_2$ via $\mathcal{C}_{t_2} = eval_{t_2-t_1+1}(M_{t_2})$. Theorem 2.3 extends to this setting and to finite collections of such constraints.

**Corollary 2.4.** *Let $\{(\mathcal{C}^i, [t_1^i, t_2^i])\}$ be a finite set of regular subsequence constraints, and let $\{\tilde{X}_t\}$ be the augmented chain obtained by attaching mediation variables for each constraint. $\{\tilde{X}_t\}$ has the same properties as in Theorem 2.3a-d.*

### 2.4 Regular MVRs and Regular Constraints

Our definition of mediation variables (Definition 2.2) permits the memory space $M_t$, update rules $upd_t$, and evaluation rules $eval_t$ to vary with time. While we used this level of generality in our definition to show that Theorem 2.3 holds for generic memory constructions, it allows for representations that are not practical. For example, we could represent any constraint by making the memory state track the entire history. This gives a practically infeasible representation whose memory space explodes exponentially in time. The natural restriction is to consider time-homogeneous representations, where the memory space, update rule, and evaluation rule are all fixed. This is the class we work with for most of the paper, and it is exactly the class that connects to classical automata theory in the next section.

**Definition 2.5.** A **regular MVR** is a time-homogeneous MVR. A **regular constraint** is a constraint that admits a regular MVR.

For concrete examples, Table 1 and Section 4 discuss common classes of constraints that are regular. Although we focus on regular MVRs, time-varying representations may be practically useful. For example, our energy grid example in Section 6.2 has a dynamic resource constraint that is more compactly represented by a time-varying MVR.

## 3 Connections to Automata and Language Theory

A reader familiar with formal languages will already have noticed that a regular MVR over a finite state space is essentially a deterministic finite automata (DFA), with $\mathcal{M}$ playing the role of the automaton's internal states, $upd$ playing the role of the transition function, and $eval^{-1}(1)$ playing the role of the accepting states.[1] This is more than a passing similarity: the two constructions are equivalent over finite spaces. In this section, we cover several connections to automaton and formal language theory that will be relevant to the compositional framework of Section 4 and algorithms of Section 5.

### 3.1 Regular Language and Constraints

As we noted earlier, over finite spaces regular MVRs and DFAs are the same construction. This equivalence lets us import a body of classical results into the framework.

[1] We defer formal definitions of finite automata, regular expressions, and related results to Appendix A.2.

**Lemma 3.1.** *For finite $\mathcal{X}$, there is a bijection between regular MVRs and DFAs.*

Similarly, regular languages and regular constraints over finite state spaces are equivalent. A **language** $L$ in alphabet $\Sigma$ is a collection of finite strings $L \subseteq \Sigma^*$, where $*$ is the Kleene operator. A language can be viewed as a constraint by identifying subsets of $\mathcal{X}^t$ with members of $\mathcal{F}_t$, provided every length-$t$ subset is measurable. For finite alphabets, a **regular language** is one accepted by a DFA, and combining this with Lemma 3.1 gives three equivalent characterizations of constraint regularity.

**Proposition 3.2.** *For a constraint $\mathcal{C}$ on finite $\mathcal{X}$, the following are equivalent:*

*(a) $\mathcal{C}$ is regular.*

*(b) $\mathcal{C}$ forms a regular language.*

*(c) $\mathcal{C}$ is generated by a regular expression.*

The third characterization is particularly useful in practice. Formal regular expressions and the regular expressions one writes in a programming language are different objects but share enough operations that they often coincide (Zheng et al., 2021). If the feasible sequences can be described with a regex, that is good evidence the constraint is regular and admits an MVR.

### 3.2 Prefix-Free Constraints and Stopping Times

The class of prefix-free constraints turns out to coincide with the classical notion of a stopping time. This bijection is the technical infrastructure underlying Algorithm 1 and Algorithm 3, for the satisfaction-time distribution and random-length sampling respectively. We make the equivalence precise here.

We begin with the two notions side by side. A **prefix-free language** $L$ is one in which $x \in L$ implies $xv \notin L$ for any non-empty extension $v$. Equivalently, no feasible string admits a feasible proper extension. A **stopping time** is a random variable $\tau : \mathcal{X}^{\mathbb{N}^+} \to \mathbb{N}^+ \cup \{\infty\}$ satisfying $\{\tau = t\} \in \mathcal{F}_t$ for every $t \geq 1$. The defining property is that the first $t$ coordinates of the sequence are sufficient to determine whether $\tau = t$, with no dependence on future coordinates.

To compare the two objects, we examine $\{\tau = t\}$ as a subset of the sample space $\mathcal{X}^{\mathbb{N}^+}$. Since $\{\tau = t\} \in \mathcal{F}_t$, this event is determined entirely by the first $t$ coordinates and consequently is a cylinder set $A_t \times \mathcal{X}^{\mathbb{N}^+}$ for some measurable $A_t \subseteq \mathcal{X}^t$. Therefore, we view each $\{\tau = t\}$ equivalently as a set of length-$t$ strings $A_t$. The events $\{\tau = t\}$ are pairwise disjoint, since $\tau$ is a well-defined function. Using the above identification, this implies the family $\{A_t\}_{t\geq 1}$ is prefix-free since $\tau$ takes exactly one value on each infinite path. Every stopping time therefore generates a prefix-free constraint $\{\tau < \infty\} = \bigcup_{t\geq 1}\{\tau = t\}$. Similarly, in the reverse direction we can convert a prefix-free constraint into a stopping time via the identification $\{\tau = t\} \equiv \mathcal{C}_t \in \mathcal{F}_t$ and $\{\tau = \infty\} = (\bigcup \mathcal{C}_t)^c$. By the same argument, $\tau$ is a well-defined function on $\mathcal{X}^{\mathbb{N}^+}$ by the prefix-free property.

**Proposition 3.3.** *There is a bijection between stopping times and prefix-free constraints.*

Importantly, every constraint $\mathcal{C}$ has a canonical stopping time: its satisfaction time $Sat_{\mathcal{C}}$. In light of the above equivalence, we may think of $Sat_{\mathcal{C}}$ as a 'prefix-free version' of $\mathcal{C}$. While not every constraint is prefix-free, every constraint is canonically associated to a prefix-free constraint via its satisfaction time.

**Definition 3.4.** The **satisfaction time** of constraint $\mathcal{C}$ is

$$Sat_{\mathcal{C}} = \min_t \{t \mid \mathcal{C}_t = 1\}.$$

**Proposition 3.5.** *Every satisfaction time is a stopping time and equivalent to a prefix-free constraint.*

The above equivalence also allows us to transport notions of regularity to stopping times. We therefore define a **regular stopping time** as one whose equivalent prefix-free constraint (Proposition 3.5) is regular. Moreover, regularity is preserved when taking the satisfaction time: a regular constraint generates a regular satisfaction time. We use this fact in the derivations of Algorithm 1 and Algorithm 3.

**Proposition 3.6.** *If $\mathcal{C}$ is regular, then $Sat_{\mathcal{C}}$ is regular.*

Table 1: Common regular constraint types as well as operations for combining regular constraints. Appendix E discusses how each constraint type is regular and provides examples from other literature.

| Examples of Regular Constraints | |
|---|---|
| Type | Example |
| Hitting State/Sequence | *state a must occur at some time* |
| Unlocked Transitions | *transition* $b \to c$ *can happen only after hitting state a* |
| Spaced Sequences | *states* $a, b, c$ *occur sequentially at some times* |
| Stay Time | *stay in each state for at least time* $\delta$ |
| Return Time | *time gap of at least/most* $\delta$ *between visits to state a* |
| Visit/Pattern Counts | *exactly(at least/most)* $k$ *occurrence of patterns* $p \in P$ |
| Jump Counts | *exactly(at least/most)* $k$ *transitions to distinct states* |
| Resource Constraint | *the resource* $B_t = \sum_{s=1}^{t} score(x_s)$ *cannot exceed a limit* |

| Regularity-Preserving Operations | | |
|---|---|---|
| Operation | Description | Notation |
| Union/Intersection | *either/both of* $\mathcal{C}^1, \mathcal{C}^2$ *are satisfied* | $\mathcal{C}^1 \bigcup \mathcal{C}^2$ / $\mathcal{C}^1 \bigcap \mathcal{C}^2$ |
| Complement | *complement of feasible sets* | $\mathcal{C}^c$ |
| Concatenation | $x$ *feasible if* $x = wv$, $w \in \mathcal{C}^1$ *and* $v \in \mathcal{C}^2$ | $\mathcal{C}^1 \cdot \mathcal{C}^2$ |
| Set Difference | *sequences exclusive to* $\mathcal{C}^1$ | $\mathcal{C}^1 \setminus \mathcal{C}^2$ |
| Reversal | *reversing feasible sequences* | $R(\mathcal{C}^1)$ |
| Kleene Star | *arbitrary concatenation of feasible strings* | $\mathcal{C}^*$ |
| Prefix | $x$ *feasible w.r.t.* $\mathcal{C}$ *with no strict prefix that is feasible* | $Sat_{\mathcal{C}}$ |
| Prior/Never Satisfied | $\mathcal{C}$ *has already/never been satisfied* | $\mathcal{C}^{\leq}, \mathcal{C}^{>}$ |
| Precedence | *constraint* $\mathcal{C}^1$ *satisfied before constraint* $\mathcal{C}^2$ | $Sat_{\mathcal{C}^1} < Sat_{\mathcal{C}^2}$ |
| Counts | $\mathcal{C}$ *has been satisfied exactly(at least/most)* $k$ times | $Ct(\mathcal{C}, k)$ |

## 4 Regular Constraints and Closure Properties

Theorem 2.3 reduces constrained analysis to analyzing an augmented HMM, but the reduction is only useful if the constraints we care about admit tractable MVRs. This section establishes a framework for building and verifying regular constraints with Table 1 providing a comprehensive list. We exhibit basic regular constraints, prove that several natural operations preserve regularity, and use these results to show that the constraint types appearing across the literature on constrained Markov chains all fall within the regular class.

**Closure under set and string operations.** Regular constraints inherit all the standard closure properties of regular languages over finite alphabets.

**Proposition 4.1.** *Regular constraints are closed under union, intersection, complement, concatenation, set difference, reversal, and Kleene star* $*$.

**Temporal modifiers.** Two natural temporal modifications of a regular constraint $\mathcal{C}$ are the prior-satisfied version $\mathcal{C}^{\leq}$ and the never-satisfied version $\mathcal{C}^{>}$. $\mathcal{C}^{\leq}$ contains sequences in which $\mathcal{C}$ has been satisfied at some point during the trajectory. $\mathcal{C}^{>}$ contains sequences in which $\mathcal{C}$ has not been satisfied at any point. We note that $\mathcal{C}^{>}$ is generally distinct from the complement $\mathcal{C}^c$. A sequence may fail $\mathcal{C}$ at its final length yet have satisfied $\mathcal{C}$ earlier in the trajectory, in which case it lies in $\mathcal{C}^c$ but not in $\mathcal{C}^{>}$.

**Proposition 4.2.** *If* $\mathcal{C}$ *is regular, then* $\mathcal{C}^{\leq}$ *and* $\mathcal{C}^{>}$ *are regular.*

**Hitting simple events.** Many constraints reduce to asking that a finitely-described event occur somewhere in the trajectory. We formalize this as a **simple event**, which is any event of the form $B = \bigcup_{i=1}^{n} B_{i,1} \times \cdots \times B_{i,k}$ where each $B_{i,j} \in \mathcal{F}$ and $n, k$ are finite. For a finite state space, every measurable subset of $\mathcal{X}^k$ is automatically a simple event. Some examples of simple events include hitting any distinguished state(s) ($k = 1$, $n \geq 1$) and observing a specific transition $a \to b$ ($k = 2$, $n = 1$, with $B_{1,1} = \{a\}$ and $B_{1,2} = \{b\}$). By taking complements

(Proposition 4.1), constraints asking that a simple event *never* occur are also regular, covering forbidden states, transitions, and patterns.

**Proposition 4.3.** *Let $\mathcal{C}^B$ be the constraint that the simple event $B$ occurs at least once. Then $\mathcal{C}^B$ is regular.*

**Precedence between satisfaction times.** Comparisons between satisfaction times yield a further class of regular constraints, with simple-event precedence as a useful special case. Common examples that reduce to precedence over simple events include hitting-time comparisons, unlocking-transition constraints (e.g., $b \to c$ permitted only after first reaching $a$), and spaced-sequence constraints (e.g., $a$, $b$, $c$ occurring sequentially with arbitrary gaps).

**Proposition 4.4.** *Let $\mathcal{C}^1, \mathcal{C}^2$ be regular constraints. The precedence constraints $Sat_{\mathcal{C}^1} < Sat_{\mathcal{C}^2}$ and $Sat_{\mathcal{C}^1} \leq Sat_{\mathcal{C}^2}$ are regular.*

**Corollary 4.5.** *For simple events $A, B$, the constraints $\mathcal{C}^{A<B}$ and $\mathcal{C}^{A\leq B}$ asking $A$ to occur (strictly) before $B$ are regular.*

**Counting non-overlapping occurrences.** A final family of constraints controls how often $\mathcal{C}$ is satisfied within a sequence, in the non-overlapping sense from the pattern-matching literature (Lladser et al., 2007; Nuel, 2008; Tataru et al., 2013). A finite path $w$ contains exactly $k$ non-overlapping occurrences of $\mathcal{C}$ if it admits a decomposition

$$w = w_1 \cdots w_k \, v,$$

where each $w_i$ is a feasible string for $\mathcal{C}$ and no prefix of $v$, including $v$ itself, is feasible. The trailing $v$ records the (possibly empty) tail of the path after the last completed occurrence. We denote this constraint $Ct(\mathcal{C}, k)$, with $Ct(\mathcal{C}, [j, k])$ collecting the sequences having between $j$ and $k$ occurrences and $Ct(\mathcal{C}, > k)$ collecting those having strictly more than $k$. Appendix E verifies that this notion agrees with the non-overlapping pattern count studied in Lladser et al. (2007); Nuel (2008); Tataru et al. (2013). Stay-time and return-time constraints, jump counts, and additive resource budgets fall within this framework as well. Appendix E works through these in detail with explicit MVR constructions.

**Proposition 4.6.** *If $\mathcal{C}$ is regular, then $Ct(\mathcal{C}, k)$, $Ct(\mathcal{C}, [j, k])$, and $Ct(\mathcal{C}, > k)$ are regular for all $0 \leq j \leq k$.*

## 5 Algorithms for Additional Constraint Queries

Under the mediation variable perspective, generic memory augmentation produces an equivalent HMM, where one can apply standard algorithms for inference, learning, and sampling. Since mediation variables carry a constraint's satisfaction history in the form of the HMM's emissions, this same HMM supports additional constraint queries that are not part of the standard toolkit. We provide three additional algorithms that compute the satisfaction time distribution (Algorithm 1), posterior satisfaction probability (Algorithm 2), and stopped constrained sampling (Algorithm 3). Derivations and pseudocode can be found in Appendix F.

### 5.1 Algorithm 1: Satisfaction Time Distribution

For the satisfaction time $Sat_{\mathcal{C}}$ of constraint $\mathcal{C}$, the event $\{Sat_{\mathcal{C}} = t\}$ has an equivalent characterization in terms of the full augmented HMM's emissions (Figure 7):

$$\{Sat_{\mathcal{C}} = t\} = \{\mathcal{C}_1 = 0, \cdots, \mathcal{C}_{t-1} = 0, \mathcal{C}_t = 1\}$$

This emission characterization is what makes the satisfaction time distribution computable by a forward-backward variant. By recursively computing the 'not-yet' filtering quantity $P(\mathcal{C}_{1:t-1} = 0, x_{1:t-1}, y_{1:t-1})$, at time $t$ we can easily compute the quantities $P(\mathcal{C}_{1:t-1} = 0, \mathcal{C}_t = b, x_{1:t}, y_{1:t})$ for the 'satisfied at $t$' event ($b = 1$) and 'not-yet' event ($b = 0$). Marginalizing over the latent paths $x_{1:T}$ gives the required quantities $P(Sat_{\mathcal{C}} = t, y_{1:T})$. For finite $\mathcal{X}$, this yields an efficient message-passing algorithm (Algorithm 1) that computes the posterior distribution $P(Sat_{\mathcal{C}} = t | y, Sat_{\mathcal{C}} \leq T)$. Here, $y$ represents any additional information about the model's variables, such as their emissions or other enforced constraints. The case study of section 6.1 shows how this can be used to estimate the location of a disease-causing mutation in a DNA sequence.

### 5.2 Algorithm 2: Posterior Satisfaction Probability

Similarly, we might wish to compute the probability that a constraint is satisfied given only partial information about the model's variables. Since satisfaction is a terminal emission of the augmented HMM, Algorithm 2 is forward-message filtering procedure that efficiently computes the posterior satisfaction probability given additional evidence $y$: $P(\mathcal{C}_T = 1|y)$. We note that this is a simpler computation than the satisfaction time, since we do not track the intermediate emissions. The DNA example of Section 6.1 demonstrates how this can be used to infer whether an individual has a disease-causing mutation.

### 5.3 Algorithm 3: Constrained Stopped Sampling

Finally, Algorithm 1 can be combined with the Forward-Filtering Backward-Sampling (FFBS) algorithm to exactly sample feasible paths with length determined by a stopping condition, up to a maximum length $T$. More precisely, we can sample exactly from the posterior $P(X_{1:\tau}|y, \mathcal{C}_\tau = 1, \tau \leq T)$, where $\tau$ is a regular stopping time. Evidence $y$ includes any path constraints as well as observed emissions. Note that $y$ can be empty, allowing for exact stopped sampling with any regular stopping time.

We emphasize that even for simple stopping conditions, constrained stopped sampling can be highly nontrivial. In the DNA study of Section 6.1, the stopping condition is simply hitting a reserved terminal state: $\tau = Hit_{end}$. In the unconstrained case, exactly sampling $X_{1:\tau}$ can be easily done by forward sampling until hitting this terminal state. However, imposing other constraints adds global dependencies that make forward sampling inexact, since the path must also be feasible when we stop (Appendix G). In our DNA example, the constraint that the spaced sequence $(pro, dis, enh)$ must occur is one example of a global constraint. Our algorithm resolves this difficulty by splitting these two issues - stopping and satisfaction - into sequential tasks: we first sample the time horizon $\tau$ using Algorithm 1 before running FFBS to sample a constrained path. This allows us to perform constrained stopped sampling from the exact posterior, applicable to any regular stopping time and regular constraints.

**Algorithm 3** Constrained Stopped Sampling

**Input:** Markov chain $\{\rho, \pi\}$, $MVR_\tau$, maximum length $T$, optional evidence/constraints $y$
**Output:** Feasible stopped path $x_{1:\tau}$

**Sample the stopping time**
$\tau \leftarrow$ SatTime $1(\{\rho, \pi\}, y, MVR_\tau, T)$

**Run Forward-Filtering Backward Sampling**
$y \leftarrow y \bigcup MVR_\tau$
$x_{1:\tau} \leftarrow FFBS(\{\rho, \pi\}, y, \tau)$

## 6 Experiments

In this section, we provide two capability demonstrations showcasing the breadth of tasks supported by our framework. The DNA example features a discrete-space model with multiple heterogeneous constraints - dwell-time, hitting, counts, and three-state precedence - and showcases the full algorithmic suite. The energy grid example uses a Gaussian state-space model to capture log-changes in energy demand, subject to a time-varying resource constraint. This latter example highlights the benefits of having an exact constrained representation prior to discretization, and it gives an example of a constraint that is more compactly represented by a non-regular, dynamic MVR. We also include runtime comparisons against exact (integer linear programming) and inexact (penalized beam search) baselines that highlight regimes where memory augmentation can be an attractive strategy. We conclude with scaling studies on how constrained inference on the MVR-augmented model scales with the time horizon, state space size, and number of constraints.

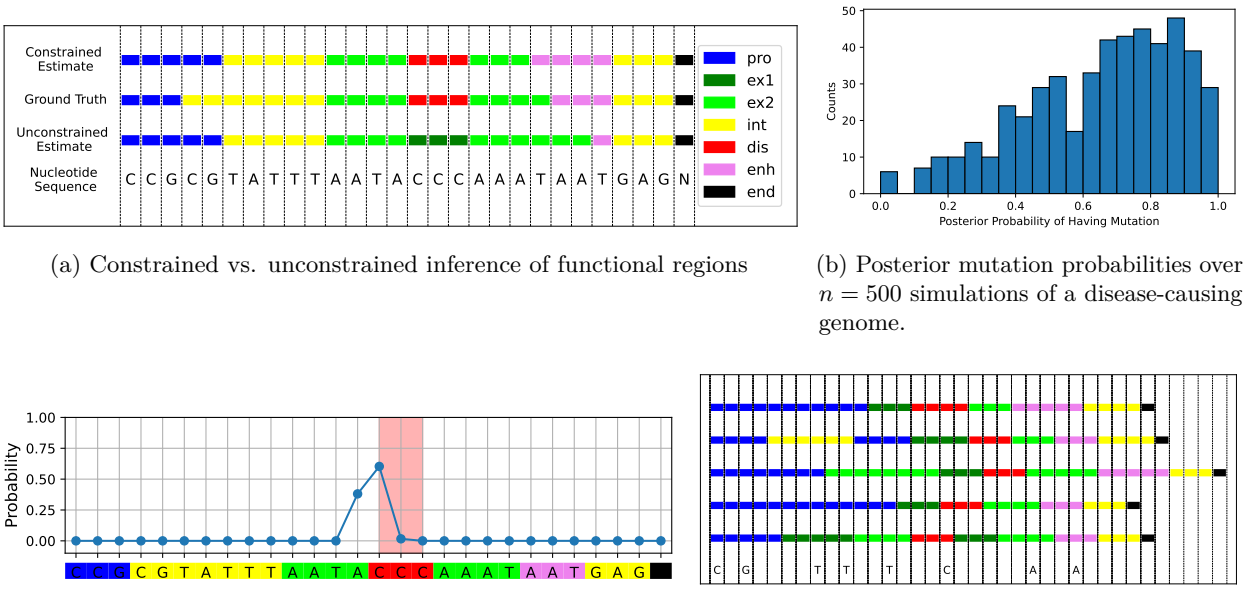
### 6.1 DNA Analysis

We model DNA sequences containing a rare disease-causing mutation using an HMM similar to those found in previous works (Yoon, 2009; Ma et al., 2026). Examples of such single-gene disorders include Duchenne muscular dystrophy, cystic fibrosis, and hemophilia (Blencowe et al., 2018). The HMM has 7 hidden states $X \in \{pro, ex1, ex2, dis, int, enh, end\}$ that represent different functional regions: upstream promoter, two different exons, disease-causing mutation, intron, downstream enhancer, and a special end-of-sequence state respectively. The HMM's emissions are the four nucleotides $Y \in \{A, T, C, G\}$, with end-of-sequence state *end* having a reserved emission $N$ to signal termination. The emission probabilities reflect typical nucleotide frequencies for each region. Notably, many regions have similar nucleotide frequencies, with the mutation region *dis* frequency being an equal mixture of the two exons' frequencies. Our goal is to infer the functional regions from a given nucleotide sequence. Model and experiment details can be found in appendix I.1.

We enforce four biologically-motivated constraints:

1. $\mathcal{C}^1$: Each functional region has a minimum length of 3.
2. $\mathcal{C}^2$: Promoter *pro* occurs in the first 10 nucleotides.
3. $\mathcal{C}^3$: There is exactly one *dis* mutation region.
4. $\mathcal{C}^4$: $pro, dis, enh$ must occur at some point, in that order.

Note that the second constraint is a subsequence constraint on $X_{1:10}$. All four constraints are examples from Table 1: hitting, stay, visit count, and precedence respectively.




(a) Constrained vs. unconstrained inference of functional regions

(b) Posterior mutation probabilities over $n = 500$ simulations of a disease-causing genome.

(c) Distribution of the mutation's location via the satisfaction time

(d) Sampling plausible functional sequences from partial nucleotide information

Figure 2: Constrained analysis: inference, sampling, satisfaction time, and satisfaction probability.

Figure 2a compares unconstrained versus constrained inference of the latent functional regions from a nucleotide sequence. Unsurprisingly, the unconstrained estimate is missing a mutation region and has an enhancer region below the minimum length: several functional regions have similar nucleotide frequencies, and adding information from constraints improves inference of the correct regions. Figure 2c shows an application of the satisfaction time algorithm (Algorithm 1) to estimate the mutation's position, given by the hitting time $Hit_{dis}$. We compute the posterior distribution of $Hit_{dis}$ conditioned on both the observed

Table 2: DNA: unconstrained vs. constrained learning over $n = 25$ trials using EM with a max iteration limit of 100 and stopping tolerance of $1 \times 10^{-3}$. For each trial, we sampled 10 nucleotide sequences of minimum length 30 from the DNA HMM with constraints $\mathcal{C}^1, \mathcal{C}^2, \mathcal{C}^3, \mathcal{C}^4$. Note that since the constrained and unconstrained HMMs are distinct probabilistic models, their log-likelihoods are not directly comparable.

| | Unconstrained | Constrained |
|---|---|---|
| Steps to convergence | $100.000 \pm 0.000$ | $59.440 \pm 23.977$ |
| Initial vector $L^2$ error | $0.898 \pm 0.233$ | $0.767 \pm 0.224$ |
| Transition matrix $L^2$ error | $1.680 \pm 0.173$ | $1.783 \pm 0.040$ |
| Emissions matrix $L^2$ error | $1.722 \pm 0.302$ | $1.142 \pm 0.208$ |
| Log-likelihood | $-630.801 \pm 88.037$ | $-647.696 \pm 89.002$ |

nucleotide sequence $y_{1:T}$ and the other constraints $\mathcal{C}^1, \mathcal{C}^2, \mathcal{C}^4$: $P(Hit_{dis} = t | y_{1:T}, \mathcal{C}^{(1,2,4)} = 1, Hit_{dis} \leq T)$. Figure 2d shows an application of constrained stopped sampling (Algorithm 3) to sample feasible functional sequences given partial nucleotide information. Note that while the stopping condition is simply hitting *end*, the global dependencies introduced by our constraints make exact stopped sampling highly nontrivial. These two simultaneous issues - stopping and feasibility - make devising an exact sampling scheme highly nontrivial. For example, a forward sampling procedure could easily stop as soon as *end* is sampled, but ensuring this stopped sequence also satisfies the other constraints in our study would require additional modifications of the sampling process, with the added risk of these bespoke modifications distorting the sampling distribution. In contrast, we emphasize that our sampling algorithm performs constrained stopped sampling from the exact posterior.

We can also infer from an individual's nucleotide sequence whether they have a disease-causing mutation by computing the posterior satisfaction probability of the mutation-hit constraint $\mathcal{C}^5$. This posterior probability is conditioned on stay $\mathcal{C}^1$ and promoter $\mathcal{C}^2$ constraints, in addition to the modified subsequence $\mathcal{C}^6$ and uniqueness $\mathcal{C}^7$ constraints below:

5. $\mathcal{C}^5$: *dis* must occur at least once.
6. $\mathcal{C}^6$: $pro, enh$ must occur in that order.
7. $\mathcal{C}^7$: There is at most one *dis* mutation region.

The modified constraint $\mathcal{C}^6$ omits *dis* from the precedence relation to allow for sequences without mutations, and constraint $\mathcal{C}^7$ enforces that an individual has at most one mutation region. Figure 2b shows a histogram of the posterior mutation probability over $n = 500$ simulations of a disease-causing genome drawn from our constrained HMM, with an average probability of $.651 \pm .230$. On average, the posterior probability of the hitting constraint $\mathcal{C}^5$ correctly suggests a mutation is more likely to be present.

Finally, we also compared constrained and unconstrained learning methods applied to our DNA model. Over $n = 25$ trials, for each trial we randomly initialized a new HMM, sampled 10 nucleotide sequences of minimum length 30 from the original HMM (with constraints $\mathcal{C}^{1-4}$), and applied learning methods using this batch of 10 sequences. Both methods were expectation maximization (EM) using the Baum-Welch algorithm, with a maximum iteration limit of 100 and stopping tolerance of $1 \times 10^{-3}$. For the constrained case, Baum-Welch was run over an augmented HMM with constraints $\mathcal{C}^{1-4}$, and we marginalized the augmented moments $\gamma(m, x, m', x'), \xi(m, x)$ to recover the original chain's moments $\gamma(x, x'), \xi(x)$. Since the augmented measure is degenerate in the mediation component (Theorem 2.3), marginalization recovers the correct moments. Note that while typically constrained learning is understood as placing constraints on the learned parameters, we impose constraints on the path space and incorporate this information during learning. Table 2 summarizes the results of this experiment. Unconstrained EM did not converge within the iteration budget in any trial, whereas constrained EM on average converged in $\sim 59$ iterations; we do not have a clear explanation for this gap, and due to this convergence gap comparisons of parameter error are inconclusive. We do note that, because the sequences are generated from the constrained model, constrained EM fits the correct generative model while unconstrained EM fits a misspecified one. We also note that imposing path constraints during

learning can in principle raise identifiability concerns, since eliminating infeasible paths may leave multiple parameter settings equally consistent with the feasible set. This is a tradeoff outside the scope of the paper that we do not explore further. Finally, while we report the log-likelihood of each method for completeness, the constrained and unconstrained HMMs are distinct probabilistic models, and their log-likelihoods are not directly comparable.

### 6.2 Resource Constrained Energy Grid

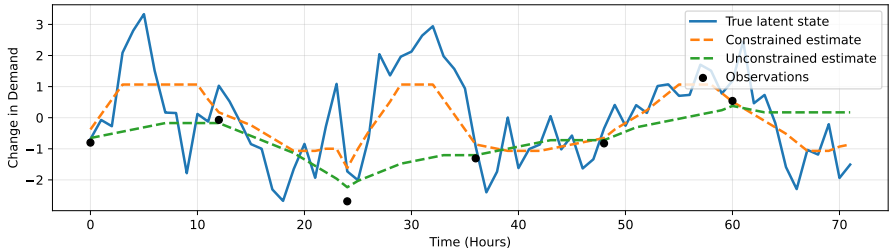
We model hourly fluctuations in energy demand using a state space model:

$$X_t = .9X_{t-1} + W_t \ ; \ X_1 \sim N\left(0, \frac{1}{1-.9^2}\right)$$
$$Y_t = X_t + V_t$$

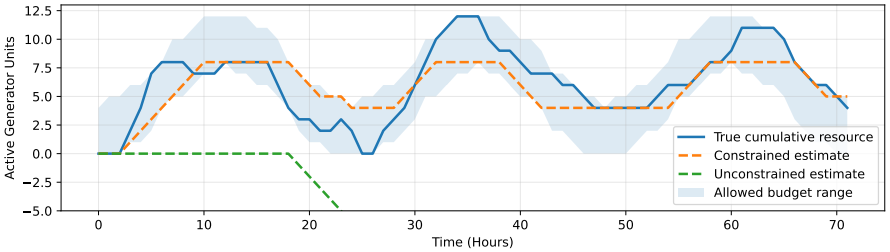
where $X_t$ is the log change-in-demand, $Y_t$ is a noisy observation, and $W_t, V_t$ are independent Gaussian noise processes with unit variance. Depending on the current change-in-demand $X_t$, the grid operator must actively intervene to balance the load by (de)activating generator blocks. A resource function $r$

$$r(x) = sgn(x) * min(floor(|x|), 3)$$

maps fluctuations $X_t$ to the number of generator blocks that need to be (de)activated, with the operator being able to adjust up to three blocks in a hour. The cumulative sum $b_t = \sum_{s=1}^{t} r_s$ is the current number of active generators. Due to safety and reliability regulations, the operator is required to maintain this number within time-varying bounds $l_t \leq b_t \leq u_t$ depending on the time of day and anticipated demand.




(a) Change-in-demand $X_t$ estimates: constrained versus unconstrained




(b) Time series of active generator units from inferred paths

Figure 3: Inference of change-in-demand from partial noisy observations

In our problem, we receive noisy emissions $Y_t$ about the change-in-demand $X_t$ every six hours, and we know *a priori* that the operator did not breach the operating bounds on the generator blocks. Our goal is to infer the full hourly changes $X_t$ over a three-day period, incorporating the time-varying limit on active generator blocks as an additional constraint. Figure 3 shows both constrained and unconstrained path estimates from the noisy partial observations $Y_t$. Incorporating the budget constraint provides additional information over the no-emission periods, helping guide the constrained estimate to more plausible trajectories. On the other hand, the unconstrained path very quickly implies an impossible negative number of active generators.

Table 3: Energy grid: unconstrained vs. constrained inference over $n = 100$ trials

| | Unconstrained | Constrained |
|---|---|---|
| Path log-probability | $-266.576 \pm 0.793$ | $-268.706 \pm 1.342$ |
| % unfeasible paths | 100% | 0% |
| Time-averaged budget excess | $5.192 \pm 4.420$ | 0 |

We also compared constrained versus unconstrained path estimates of the latent demand over $n = 100$ simulations of our energy grid over a 72-hour period. Table 3 summarizes the results, showing that every single unconstrained estimate leads to infeasible paths with a time-averaged budget excess of $5.192 \pm 4.420$. In addition to the table, we directly computed the log-probability gap between estimates for each trial, with an average constrained-unconstrained gap of $-2.131 \pm 0.740$. Considering inference is done over a time horizon of 72 steps, we find that the constrained method gives paths that are both roughly comparable in likelihood to the unconstrained method and also fully conformant to budget restrictions.

This problem naturally fits a memory augmentation approach where one tracks the budget state $b_t$ over time. However, a regular MVR with fixed mediation space would require tracking all possible values that $b_t$ can take, which can potentially be extremely large. Instead, if the feasible range $[l_t, u_t]$ is small for all times $t$, a time-varying representation tracking just the current feasible range (plus an additional fail state) would be much more compact. This is the approach we take: we define a MVR where $\mathcal{M}_t = [l_t, \cdots, u_t, f]$ with $f$ an absorbing fail state. The augmented process, with discrete memory and continuous latent space, is a switching-state model, and inference with these mixed models is typically done by discretizing the continuous portion before running a discrete inference algorithm like Viterbi (Ghahramani & Hinton, 2001). Theoretically, the MVR-augmented process is an exact representation of the constrained model. One benefit of this is that the mediation states naturally partition the state space via the pre-images of *init* and $upd_t$. By incorporating these partitions into the discretization, we get a discretized representation that is sound in the constraint: the feasibility of every path agrees with the feasibility of its discretized image (Appendix H).

### 6.3 Comparison to Integer Linear Programming

The main benefit of memory augmentation methods is that they recover the Markov property in the augmented model. This enables the use of dynamic-programming (DP) algorithms that scale linearly with the time horizon, at the cost of increasing the state space. This contrasts with combinatorial methods like integer linear programming (ILP), which do not expand the state space but search over the entire path space at once. While ILP methods scale linearly in time for unconstrained inference (Roth & Yih, 2005), constrained inference can be nonlinear in the problem size; in general the resulting ILP is NP-hard (Koller & Friedman, 2009) with a worst-case runtime exponential in the time horizon. To demonstrate the tradeoffs between these approaches, we compare them on a constrained inference task as the time horizon grows.

We used a HMM as our Markov chain model with $K = 20$ hidden states and $E = 10$ emission states, with the HMM parameters randomly drawn from a standard Gaussian before normalizing using a softmax. We imposed a single subsequence hitting constraint of length $L = 10$, where the contiguous subsequence $(s_0, s_1, \cdots, s_9)$ must appear in the path at some point. For each time horizon $T \in \{50, 100, \cdots, 300\}$, we ran 25 trials with independently drawn observation sequences and compared runtimes. The ILP was implemented in `pyomo` following a flow-based formulation (Roth & Yih, 2004; Koller & Friedman, 2009) and optimized with the industry-standard Gurobi solver (Gurobi Optimization, LLC, 2026). We used a regular MVR tracking progress through the subsequence, with a memory space of size $|\mathcal{M}| = 11$, and we used GPU-accelerated Viterbi on the augmented model for inference. This experiment was run on a system with a 32-core AMD Zen3 processor, 512GB of DDR4 RAM, and a NVIDIA A100 GPU.

Figure 4 shows that GPU-accelerated Viterbi on the memory-augmented model scales favorably with the time horizon in our experiments. This is consistent with the main algorithmic advantage of memory augmentation: at the cost of increasing the state space, inference can be carried out by DP algorithms that scale linearly in the time horizon. Although ILP avoids state-space expansion, it solves a global constrained optimization problem whose runtime increases with the time horizon. These results highlight that memory augmentation

can be an attractive alternative when analyzing problems with long time horizons, and they suggest that the computational benefits of recovering a Markov structure can outweigh the cost of augmentation when constraints admit a compact memory representation. Similarly, standard algorithms on the augmented model, like Viterbi, admit vectorized implementations that can leverage GPU acceleration - this is another strength of memory augmentation highlighted in the next comparison to beam search.

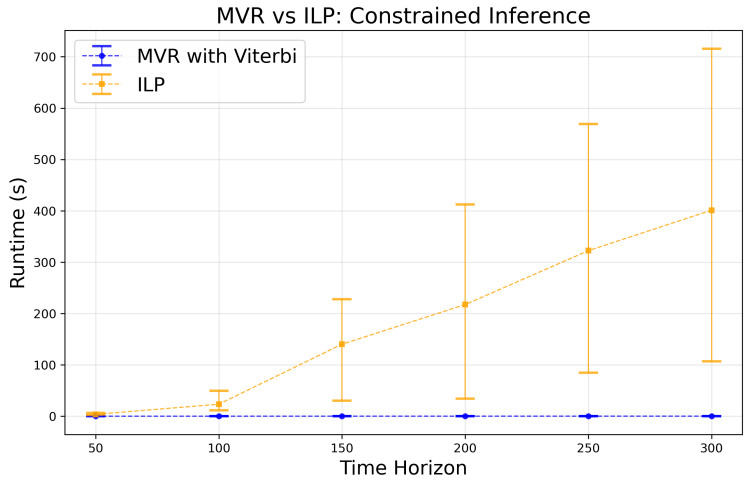



Figure 4: Average runtimes over $n = 25$ trials (with 95% bands) per time horizon comparing ILP vs. memory methods on a constrained inference task, with a subsequence hitting constraint of length $L = 10$. The base model was a HMM with $K = 20$ hidden states and $E = 10$ emissions.

### 6.4 Comparison to Penalized Beam Search

In addition to the ILP comparison, we compare Viterbi run on the MVR-augmented model with an approximate method: penalized beam search. Beam search builds path estimate(s) by greedily extending each path one state at time. For a fixed number $B$ of beams $\{\hat{x}_{1:t}\}$ of length $t$, beam search considers all possible one-step extensions of these beams. It then picks the $B$ extensions with the highest probabilities as the new estimates and repeats this greedy extension until a set of full path estimates is built. The best full path is returned as the constrained path estimate. To enforce the subsequence constraint, we modify the extension step to incentivize state transitions $(s_t, s_{t+1})$ occurring in the required subsequence $(s_1, \cdots, s_L)$:

$$score(\hat{x}_{1:t-1} \cdot x) = score(\hat{x}_{1:t-1}) + \log p(\hat{x}_{t-1}, x_t) + g(\hat{x}_{t-1}, x_t)$$

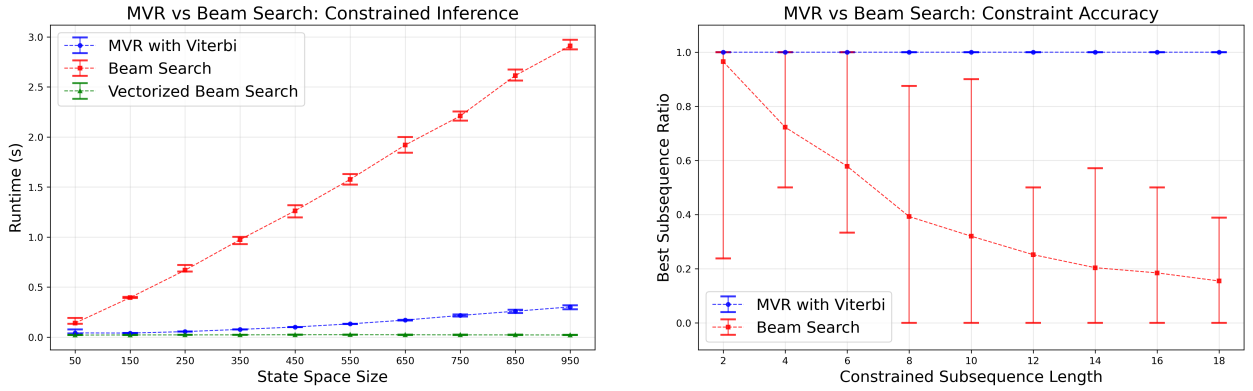
$$g(x, x') = \begin{cases} 3 & (x, x') = (a_t, a_{t+1}) \text{ for some } t \\ 0 & \text{otherwise} \end{cases}$$

We implemented both a serial beam search implementation, as well as a vectorized version of penalized beam search that recasts the above branching step as a sum-then-max tensor operation, so that it could leverage GPU-acceleration like our Viterbi implementation.

The main contrast between penalized beam search and Viterbi is that penalized beam search incorporates the constraint as an additional penalty during its branching step, without augmenting the state space. While this approach incurs no additional overhead, the tradeoff is that this greedy sequential approach can distort the posterior distribution (Appendix G), and it can fail to enforce constraints with soft penalties.

Figure 5a shows the runtimes of Viterbi run on the MVR-augmented model as well as both versions of penalized beam search. We compared runtimes as a function of the number of hidden states $K = \{50, 150, \cdots, 950\}$. We fixed the number of emissions $E = 10$, the constrained subsequence length $L = 10$, and the time horizon $T = 100$. For each $K$, we ran $n = 25$ trials for independently drawn observation sequences. Our beam search algorithm used $B = 5$ beams.

Figure 5a shows that vectorized beam search can outperform vectorized Viterbi on a MVR-augmented model. However, we do not expect this runtime advantage to hold uniformly across constraints or penalties. The Viterbi algorithm has a standard vectorized implementation, and once a constraint is represented by an MVR this same implementation can be applied to the corresponding augmented model. In contrast, the efficiency of penalized beam search depends on the form of the penalty function. Some penalties, such as those that adaptively weight transitions based on the beam's history, may be difficult to express as simple tensor operations and may require constraint-specific engineering. This illustrates another strength of memory augmentation: standard HMM algorithms, as well as those from Section 5, admit implementations that can leverage GPU acceleration.




(a) Average runtimes over $n = 25$ trials (with 95% bands) per state size. Fixed parameters of $B = 5,\ T = 100,\ E = 10, L = 10$.

(b) Average accuracy over $n = 100$ (with 95% bands. Fixed parameters of $B = 5,\ K = 50,\ T = 100,\ E = 10$.

Figure 5: Comparison with penalized beam search on a constrained inference task involving an HMM with a single subsequence hitting constraint.

Memory augmentation always enforces the constraint by construction, but heuristic methods like beam search can fail to reliably enforce constraint feasibility especially for highly constrained problems. Thus, we also compared the accuracy of the two methods in enforcing the constraint, where accuracy is defined as the ratio of the longest sequence $(s_1, \cdots, s_m)$ in the path estimate relative to the required length $L$. We grew the required subsequence length $L \in \{2, 4, \cdots, 18\}$ and looked at the average accuracy over $n = 100$ trials for each length. We fixed the number of hidden states $K = 50$, emissions $E = 10$, and time horizon $T = 100$. Our beam search algorithm used $B = 5$ beams. Figure 5b shows that the accuracy of beam search rapidly drops with the subsequence length: for harder constraints, satisfaction becomes an increasingly rarer event, and heuristic methods using soft penalties have difficulty reliably enforcing rare constraints. On the other hand, incorporating hard penalties into sequential methods like beam search often distorts the posterior, since it requires knowledge of future information such as the time horizon (Appendix G). Memory augmentation offers an alternative: if a constraint admits a tractable representation, exact constrained inference can be done on the augmented model.

### 6.5 Scaling with Number of Constraints

In the previous experiments, we looked at the average runtime of Viterbi on the MVR-augmented model as a function of time horizon (Section 6.3) and state space size (Section 6.4). In this section we look at the average runtime as a function of the number of constraints. For a naive product of memory constructions, the size of the augmented state space grows exponentially with the number of constraints. However, for many applications the number of constraints does not grow without bound, and at most a modest number of constraints are enforced at any time. In this regime, we vary the number of hitting constraints and look at its impact on the average runtime.

We fixed the number of hidden states $K = 100$, emissions $E = 10$, and time horizon $T = 100$. Each hitting constraint enforces that a particular state $a_i$ is visited by the path, and for $C$ hitting constraints the size of the augmented state space is $K2^C$. The number of hitting constraints was capped at $C = 8$ since adding more constraints would result in an augmented model too large to fit on the GPU, which highlights a key limitation of memory augmentation. Within the regime $C \leq 8$, Figure 6 shows that the empirical runtime scales exponentially with the number of constraints. However, note that even the worst-case runtime for $C = 8$ constraints is under two seconds. These scaling studies suggest that the primary bottleneck of memory augmentation is being able to fit the augmented model in memory. If this can be accomplished, then standard HMM algorithms with GPU-acceleration can have practically feasible runtimes.

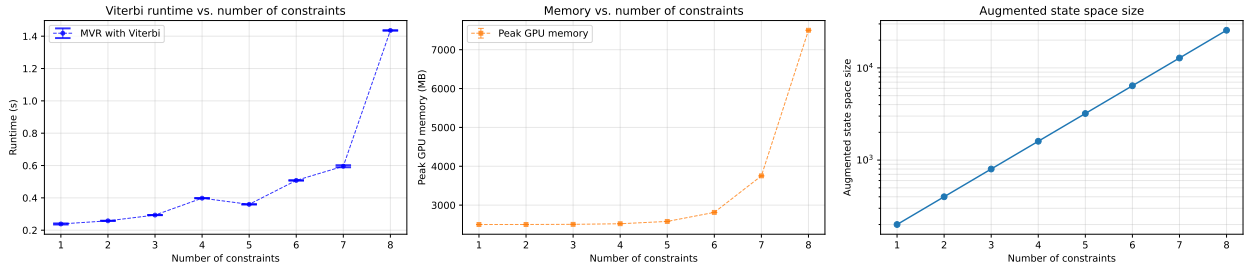



Figure 6: Left: average runtimes over $n = 25$ trials (with 95% bands) per number of unique hitting constraints. Center: Peak GPU memory usage. Right: size of the augmented model's state space.

## 7 Discussion

**Limitations** The main limitation of memory augmentation, and hence mediation variables, is the growth of the augmented state space. As a product construction, the size of the augmented state space can grow exponentially with the number of constraints. Furthermore, not all constraints admit practical representations: even a regular constraint might be too large in practice, such as one composed from many other constraints. Therefore, determining whether a constraint admits a compact representation, and constructing that representation, is overhead inherent to memory-based methods.

These limitations are mitigated by several factors. Deterministic memory often induces a sparse transition structure, with sparse representations and numerical methods potentially reducing the practical cost of state expansion. Many practically relevant constraints - such as pattern counts, budgets, and visit precedence - admit compact representations, and in many applications the number of active constraints is modest. Furthermore, as memory augmentation recovers the Markov property, this enables the use of DP algorithms whose complexity scales linearly with the time horizon and admit vectorized implementations that can leverage GPU-acceleration: both of these factors can help offset the cost of state expansion. Finally, our compositional framework reduces some of the modeling burden by providing constructive recipes for building constraint representations from simple atomic classes.

**Concluding remarks** We recast memory augmentation as probabilistically inserting a mediating sequence of random variables, providing a unified account of memory-based methods that have appeared across several literatures. The central consequence of this reframing is that a memory-augmented Markov chain has the structure of a hidden Markov model. This allows standard HMM algorithms to be applied for constrained inference, learning, and sampling, while also supporting additional queries like satisfaction-time estimation, posterior satisfaction probabilities, and constrained stopped sampling. By drawing connections to formal language theory, we also obtain a compositional framework for regular constraints, showing how many common constraint families admit tractable representations. Our experiments demonstrate the breadth of this workflow and highlight problem regimes where the benefits of enabling linear-in-time DP algorithms can outweigh the cost of state-space expansion.

## References


Alessandro Abate, Maria Prandini, John Lygeros, and Shankar Sastry. Probabilistic reachability and safety for controlled discrete time stochastic hybrid systems. *Automatica*, 44(11):2724–2734, 2008. doi: 10.1016/j.automatica.2008.03.027.

Alessandro Abate, Joost-Pieter Katoen, John Lygeros, and Maria Prandini. Approximate model checking of stochastic hybrid systems. *European Journal of Control*, 16(6):624–641, 2010. doi: 10.3166/ejc.16.624-641.

Javier Aguilar and Riccardo Gatto. Unified perspective on exponential tilt and bridge algorithms for rare trajectories of discrete markov processes. *Phys. Rev. E*, 109:034113, Mar 2024. doi: 10.1103/PhysRevE.109.034113. URL `https://link.aps.org/doi/10.1103/PhysRevE.109.034113`.

Eitan Altman. *Constrained Markov Decision Processes*. CRC Press, Boca Raton, FL, 1999.

Thom Badings, Licio Romao, Alessandro Abate, and Nils Jansen. Probabilities are not enough: Formal controller synthesis for stochastic dynamical models with epistemic uncertainty. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 14701–14710, 2023a.

Thom S. Badings, Licio Romao, Alessandro Abate, David Parker, Hasan A. Poonawala, Mariëlle Stoelinga, and Nils Jansen. Robust control for dynamical systems with non-Gaussian noise via formal abstractions. *Journal of Artificial Intelligence Research*, 76:341–391, 2023b. doi: 10.1613/jair.1.14253.

Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, Cambridge, MA, 2008.

Leonard E Baum, Ted Petrie, George Soules, and Norman Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The Annals of Mathematical Statistics*, 41 (1):164–171, 1970.

Andrea Bianco and Luca de Alfaro. Model checking of probabilistic and nondeterministic systems. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 1026 of *Lecture Notes in Computer Science*, pp. 499–513. Springer, 1995.

Gino Biondini. Chapter 2 - an introduction to rare event simulation and importance sampling. In Venu Govindaraju, Vijay V. Raghavan, and C.R. Rao (eds.), *Big Data Analytics*, volume 33 of *Handbook of Statistics*, pp. 29–68. Elsevier, 2015. doi: https://doi.org/10.1016/B978-0-444-63492-4.00002-2.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1 edition, 2007. ISBN 0387310738.

David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, April 2017. doi: 10.1080/01621459.2017.1285773.

Hannah Blencowe, Sophia Moorthie, Maria Petrou, and et al. Rare single gene disorders: estimating baseline prevalence and outcomes worldwide. *Journal of Community Genetics*, 9(4):397–406, 2018. doi: 10.1007/s12687-018-0376-2.

Craig Boutilier, Ronen Brafman, and Matthieu Geist. Budget allocation using weakly coupled, constrained markov decision processes. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 52–61. AUAI Press, 2016.

Chris K. Carter and Robert Kohn. On Gibbs sampling for state space models. *Biometrika*, 81(3):541–553, 1994. doi: 10.1093/biomet/81.3.541.

Henning Christiansen, Christian Theil Have, Ole Torp Lassen, and Matthieu Petit. Inference with constrained hidden Markov models in PRISM. In *International Conference on Logic Programming*, pp. 40–55. Springer, 2010.

Eldan Cohen and Christopher Beck. Empirical analysis of beam search performance degradation in neural sequence models. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 1290–1299. PMLR, 09–15 Jun 2019.

Marc Corstanje, Frank van der Meulen, and Moritz Schauer. Conditioning continuous-time markov processes by guiding. *Stochastics*, 95(6):963–996, 2023. doi: 10.1080/17442508.2022.2150081. URL `https://doi.org/10.1080/17442508.2022.2150081`.

Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995. doi: 10.1145/210332.210339.

Hal Daumé, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine Learning*, 75(3): 297–325, March 2009. doi: 10.1007/s10994-009-5106-x.

Daniel Deutsch, Shyam Upadhyay, and Dan Roth. A general-purpose algorithm for constrained sequential inference. In Mohit Bansal and Aline Villavicencio (eds.), *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pp. 482–492, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/K19-1045.

Janardhan Rao Doppa, Alan Fern, and Prasad Tadepalli. Structured prediction via output space search. *Journal of Machine Learning Research*, 15(38):1317–1350, 2014.

Yuanqi Du, Michael Plainer, Rob Brekelmans, Chenru Duan, Frank Noé, Carla P. Gomes, Alán Aspuru-Guzik, and Kirill Neklyudov. Doob's lagrangian: A sample-efficient variational approach to transition path sampling. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 65791–65822. Curran Associates, Inc., 2024. doi: 10.52202/079017-2103. URL `https://proceedings.neurips.cc/paper_files/paper/2024/file/78fa1d5646bc285940281cfc1386efae-Paper-Conference.pdf`.

Richard Durbin, Sean R Eddy, Anders Krogh, and Graeme Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge University Press, 1998.

Paul Fearnhead and Guillem Rigaill. Changepoint detection in the presence of outliers. *Journal of the American Statistical Association*, 114(525):169–183, 2019. doi: 10.1080/01621459.2017.1385466.

Vojtěch Forejt, Marta Kwiatkowska, Gethin Norman, and David Parker. Automated verification techniques for probabilistic systems. In *Formal Methods for Eternal Networked Software Systems (SFM)*, volume 6659 of *Lecture Notes in Computer Science*, pp. 53–113. Springer, 2011.

G David Forney. The Viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

Patrick Forré and Joris M. Mooij. Markov properties for graphical models with cycles and latent variables, 2017. URL `https://arxiv.org/abs/1710.08775`.

David A. Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference, 2002.

Nicholas J. Foti, Jason Xu, Dillon Laird, and Emily B. Fox. Stochastic variational inference for hidden markov models. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

Sylvia Frühwirth-Schnatter. Data augmentation and dynamic linear models. *Journal of Time Series Analysis*, 15(2):183–202, 1994. doi: 10.1111/j.1467-9892.1994.tb00184.x.

J. C. Fu and M. V. Koutras. Distribution theory of runs: A markov chain approach. *Journal of the American Statistical Association*, 89(427):1050–1058, September 1994. doi: 10.1080/01621459.1994.10476841.

Zoubin Ghahramani and Geoffrey E. Hinton. Variational learning for switching state-space models. In Michael I. Jordan and Terrence J. Sejnowski (eds.), *Graphical Models: Foundations of Neural Computation*. The MIT Press, 10 2001. ISBN 9780262291200. doi: 10.7551/mitpress/3349.003.0016. URL https://doi.org/10.7551/mitpress/3349.003.0016.

Amir Globerson, Tim Roughgarden, David Sontag, and Cafer Yildirim. How hard is inference for structured prediction? In Francis Bach and David Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 2181–2190, Lille, France, 07–09 Jul 2015. PMLR.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2026. URL https://www.gurobi.com.

Sofie Haesaert, Sadegh Esmaeil Zadeh Soudjani, and Alessandro Abate. Verification of general Markov decision processes by approximate similarity relations and policy refinement. *SIAM Journal on Control and Optimization*, 55(4):2333–2367, 2017. doi: 10.1137/16M1079397.

Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994. doi: 10.1007/BF01211866.

Asger Hobolth, Iker Rivas-González, Mogens Bladt, and Andreas Futschik. Phase-type distributions in mathematical population genetics: An emerging framework. *Theoretical Population Biology*, 157:14–32, 2024. ISSN 0040-5809. doi: https://doi.org/10.1016/j.tpb.2024.03.001. URL https://www.sciencedirect.com/science/article/pii/S0040580924000212.

Olav Kallenberg. *Foundations of modern probability*. Probability and its Applications (New York). Springer-Verlag, New York, second edition, 2002. doi: 10.1007/978-1-4757-4015-8.

Richard M. Karp. *Reducibility among Combinatorial Problems*, pp. 85–103. Springer US, 1972. doi: 10.1007/978-1-4684-2001-2_9.

Joost-Pieter Katoen. The probabilistic model checking landscape. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pp. 31–45, 2016. doi: 10.1145/2933575.2934574.

D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. Adaptive computation and machine learning. MIT Press, 2009. ISBN 9780262013192.

Terry Koo, Frederick Liu, and Luheng He. Automata-based constraints for language model decoding. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=BDBdblmyzY.

M. V. Koutras and V. A. Alexandrou. Runs, scans and urn model distributions: A unified markov chain approach. *Annals of the Institute of Statistical Mathematics*, 47(4):743â€“766, December 1995. doi: 10.1007/bf01856545.

Abolfazl Lavaei, Sadegh Soudjani, Alessandro Abate, and Majid Zamani. Automated verification and synthesis of stochastic hybrid systems: A survey. *Automatica*, 146:110617, 2022. doi: 10.1016/j.automatica.2022.110617.

Jason Xinyu Liu, Saumya Banerjee, Ziyi Chen, Mahyar Fazlyab, and George J. Pappas. Skill transfer for temporal task specification. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

Manuel Lladser, M. D. Betterton, and Rob Knight. Multiple pattern matching: A markov chain approach. *Journal of Mathematical Biology*, 56:51 – 92, 2007.

Bin Ma, John Tromp, and Ming Li. Patternhunter: faster and more sensitive homology search. *Bioinformatics*, 18(3):440–445, 2002. doi: 10.1093/bioinformatics/18.3.440.

Yingnan Ma, Haiyan Chen, Jingxuan Kang, Xuying Guo, Chen Sun, Jing Xu, Junxian Tao, Siyu Wei, Yu Dong, Hongsheng Tian, Wenhua Lv, Zhe Jia, Shuo Bi, Zhenwei Shang, Chen Zhang, Hongchao Lv, Yongshuai Jiang, and Mingming Zhang. The hidden markov model and its applications in bioinformatics analysis. *Genes & Diseases*, 13(1):101729, 2026. ISSN 2352-3042. doi: 10.1016/j.gendis.2025.101729.

Christopher D Manning and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT Press, 1999.

Izumi Masubuchi, Shunsuke Koga, and Atsumi Ohara. Constraint satisfaction for switched linear systems with restricted dwell-time. In *IEEE Conference on Decision and Control (CDC)*, pp. 2267–2272. IEEE, 2017.

Clara Meister, Ryan Cotterell, and Tim Vieira. If beam search is the answer, what was the question? In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 2173–2185. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.emnlp-main.170.

Ofer Meshi, Mehrdad Mahdavi, Adrian Weller, and David Sontag. Train and test tightness of lp relaxations in structured prediction. In Maria Florina Balcan and Kilian Q. Weinberger (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1776–1785, New York, New York, USA, 20–22 Jun 2016. PMLR.

P.D. Moral. *Feynman-Kac Formulae: Genealogical and Interacting Particle Systems with Applications*. Probability and Its Applications. Springer New York, 2012. ISBN 9781468493931. URL `https://books.google.com/books?id=2IHdBwAAQBAJ`.

Pierre Nicodème, Bruno Salvy, and Philippe Flajolet. Motif statistics. *Theoretical Computer Science*, 287(2): 593–617, September 2002. doi: 10.1016/s0304-3975(01)00264-x.

Grégory Nuel. Pattern markov chains: Optimal markov chain embedding through deterministic finite automata. *Journal of Applied Probability*, 45(1):226–243, March 2008. doi: 10.1239/jap/1208358964.

François Pachet, Pierre Roy, and Gabriele Barbieri. Finite-length markov processes with constraints. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume One*, IJCAI'11, pp. 635–642. AAAI Press, 2011.

Alexandre Papadopoulos, François Pachet, Pierre Roy, and Jason Sakellariou. Exact sampling for regular and markov constraints with belief propagation. In Gilles Pesant (ed.), *Principles and Practice of Constraint Programming*, pp. 341–350, Cham, 2015. Springer International Publishing. ISBN 978-3-319-23219-5.

Judea Pearl. Causal diagrams for empirical research. *Biometrika*, 82(4):669–688, 12 1995. doi: 10.1093/biomet/82.4.669.

Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 46–57. IEEE, 1977. doi: 10.1109/SFCS.1977.32.

Giordano Pola, Manuela L. Bujorianu, John Lygeros, and Maria Domenica Di Benedetto. Stochastic hybrid models: An overview. *IFAC Proceedings Volumes*, 36(6):45–50, 2003. doi: 10.1016/S1474-6670(17)36405-4.

Lawrence R Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

Dan Roth and Wen-tau Yih. A linear programming formulation for global inference in natural language tasks. In *Proceedings of the Eighth Conference on Computational Natural Language Learning (CoNLL)*, pp. 1–8, Boston, Massachusetts, USA, May 6 - May 7 2004. Association for Computational Linguistics.

Dan Roth and Wen-tau Yih. Integer linear programming inference for conditional random fields. In *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, pp. 736–743, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1595931805. doi: 10.1145/1102351.1102444.

Moritz Schauer, Frank van der Meulen, and Harry van Zanten. Guided proposals for simulating multi-dimensional diffusion bridges. *Bernoulli*, 23(4A):2917 – 2950, 2017. doi: 10.3150/16-BEJ833. URL `https://doi.org/10.3150/16-BEJ833`.

Michael Sipser. *Introduction to the Theory of Computation*. Course Technology, Boston, MA, third edition, 2013. ISBN 113318779X.

Sadegh Esmaeil Zadeh Soudjani, Caspar Gevaerts, and Alessandro Abate. FAUST$^2$: Formal abstractions of uncountable-state stochastic processes. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 9035 of *Lecture Notes in Computer Science*, pp. 272–286. Springer, 2015.

Felix Stahlberg and Bill Byrne. On NMT search errors and model errors: Cat got your tongue? In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3356–3362, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1331.

Paula Tataru, Andreas Sand, Asger Hobolth, Thomas Mailund, and Christian N. S. Pedersen. Algorithms for hidden markov models restricted to occurrences of regular expressions. *Biology*, 2(4):1282–1295, 2013. doi: 10.3390/biology2041282.

Michalis K. Titsias, Christopher C. Holmes, and Christopher Yau. Statistical inference in hidden markov models using k-segment constraints. *Journal of the American Statistical Association*, 111(513):200–215, 2016. doi: 10.1080/01621459.2014.998762.

Moshe Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 327–338. IEEE, 1985.

Thomas Verma and Judea Pearl. Causal networks: Semantics and expressiveness. In Ross D. Shachter, Tod S. Levitt, Laveen N. Kanal, and John F. Lemmer (eds.), *Uncertainty in Artificial Intelligence*, volume 9 of *Machine Intelligence and Pattern Recognition*, pp. 69–76. North-Holland, 1990. doi: 10.1016B978-0-444-88650-7.50011-1.

Byung-Jun Yoon. Hidden markov models and their applications in biological sequence analysis. *Current Genomics*, 10(6):402–415, September 2009. doi: 10.2174/138920209789177575.

Shun-Zheng Yu. *Hidden Semi-Markov Models: Theory, Algorithms and Applications.* Academic Press, 2010.

Honghua Zhang, Meihua Dang, Nanyun Peng, and Guy Van Den Broeck. Tractable control for autoregressive language generation. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

Honghua Zhang, Po-Nien Kung, Masahiro Yoshida, Guy Van den Broeck, and Nanyun Peng. Adaptable logical control for large language models. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 115563–115587. Curran Associates, Inc., 2024. doi: 10.52202/079017-3670.

L. X. Zheng, S. Ma, Z. X. Chen, et al. Ensuring the correctness of regular expressions: A review. *International Journal of Automation and Computing*, 18:521–535, 2021. doi: 10.1007/s11633-021-1301-4.

# A Technical Details and Definitions

Here, we give a more detailed exposition of the technical setting discussed in Section 2.1. A **standard Borel space** is a measurable space $(\mathcal{X}, \mathcal{F})$ that is isomorphic to a Borel subset of $([0,1], \mathcal{B}[0,1])$, and some examples include the standard real space $(\mathbb{R}, \mathcal{B}(\mathbb{R}))$ and any countable space with the discrete $\sigma$-algebra. Standard Borel spaces include most continuous state spaces while retaining nice properties, including existence of conditional measures and applicability of $d$-separation.

For a Markov chain $\{X_t\}$ on state space $(\mathcal{X}, \mathcal{F})$, we distinguish between the chain's measurable space $(\mathcal{X}^{\mathbb{N}^+}, \mathcal{F}_\infty)$ and the Markov measure $\mu = (\rho, \{\pi_t\})$. The sequence set $\mathcal{X}^{\mathbb{N}^+}$ consists of all infinite sequences $(x_1, x_2, \cdots)$ in $\mathcal{X}$. Each $X_t$ is viewed as the canonical projection from $\mathcal{X}^{\mathbb{N}^+}$, and we implicitly use this interpretation of $X_t$ as a measurable function without a specified distribution. Since we work with the canonical filtration $\mathcal{F}_t = \sigma(X_1, \cdots, X_t)$, each member $A \in \mathcal{F}_t$ has the form $A = A_t \times \mathcal{X}^{\mathbb{N}^+}$, where $A_t \subseteq \mathcal{X}^t$ is a member of the product $\sigma$-algebra $\mathcal{F}^{\otimes t}$. This justifies our abuse of notation in considering a member $A \in \mathcal{F}_t$ as also representing a measurable subset $A \subseteq \mathcal{X}^t$. $\mathcal{F}_\infty$ is the $\sigma$-algebra generated by the canonical filtration. Finally, a Markov measure $\mu = (\rho, \{\pi_t\})$ is any measure on sequence space $(\mathcal{X}^{\mathbb{N}^+}, \mathcal{F}_\infty)$ that is generated from: (1) initial measure $\rho$ on state space $(\mathcal{X}, \mathcal{F})$, (2) a sequence of Markov kernels $\pi_t(dx, x) : \mathcal{F} \times \mathcal{X} \to [0,1]$ dictating transitions $X_t \to X_{t+1}$. It can be shown that when $\{X_t\}$ is a Markov chain, there is an (almost sure) unique decomposition of the chain measure $\mu$ into an initial measure and a sequence of Markov kernels Kallenberg (2002).

## A.1 $d$-Separation in Graphical Models

$d$-separation is a method for assessing conditional independence between sets of variables in a directed graphical model Verma & Pearl (1990); Pearl (1995). While originally developed for finite random variables, $d$-separation still holds when the random variables take values in standard Borel spaces (Forré & Mooij, 2017, Theorem 3.2.1). A **path** from variable $a$ to variable $b$ is any sequence of variables $(x_1, \cdots, x_n)$ such that:

1. $x_1 = a$ and $x_n = b$
2. $(x_{i+1}, x_i)$ or $(x_i, x_{i+1})$ is an edge for each $1 \leq i < n$.

In other words, a path can traverse edges of the graphical model in either direction. Segments of a path fall into three categories depending on the orientation of the arrows: unidirectional $x_{t-1} \to x_t \to x_{t+1}$, diverging $x_{t-1} \leftarrow x_t \to x_{t+1}$ , and converging $x_{t-1} \to x_t \leftarrow x_{t+1}$. In a converging segment, we say that $x_t$ is a **collider** for variables $x_{t-1}$ and $x_{t+1}$.

In the unconditional case, converging segments act as blockers, whereas unidirectional and diverging segments are unblocked segments. If we condition on a set of variables $C$, unidirectional and diverging segments become blockers if the central node is in $C$, whereas converging segments become unblocked if the collider or any of its descendants are in $C$. Thus, we say a path from $a$ to $b$ conditioned on set $C$ is **unblocked** if every segment is an unblocked segment, and the path is **blocked by** $C$ otherwise.

$C$ $d$-**separates** $A$ and $B$ if every path from any $a \in A$ to any $b \in B$ is blocked by conditioning on $C$, and it implies the conditional independence relation $A \perp_C B$.

## A.2 Definitions and Results in Language and Automata Theory

In this section, we fill in some definitions that were deferred in the paper. Note that we always assume a finite alphabet.

**Definition A.1.** A **deterministic finite automata** is a 5-tuple $(\Sigma, Q, q_0, A, \delta)$:

1. $\Sigma$ is a finite input alphabet.
2. $Q$ is a finite set of states.

3. $q_0 \in Q$ is the unique starting state.
4. $A \subseteq Q$ is a subset of accepting states.
5. $\delta : Q \times \Sigma \to Q$ is a deterministic (partial) function.

A DFA processes strings in its input alphabet. Always initializing its internal state to the starting state $m_0$, it reads a string $w \in \Sigma^*$ by ingesting one letter at a time. The DFA's internal states change according to the transition function $\delta$ as letters are ingested, and after ingesting the entire string it checks if the internal state is an accepting state. We say a DFA **accepts a string** $w$ if it ends on an accepting state after processing $w$. We say a DFA **accepts a language** $L$ if $L$ is exactly the set of all strings that are accepted by it. A language is **regular** if it is accepted by a DFA. Similarly, regular expressions are formal descriptions of sets of strings, defined recursively:

**Definition A.2.** $R$ is a **regular expression** if it is:

1. $\epsilon$ the empty string,
2. $\sigma \in \Sigma$,
3. $\emptyset$,
4. $R_1 \cup R_2$ where $R_1, R_2$ are regular expressions,
5. $R_1 \cdot R_2$ where $R_1, R_2$ are regular expression,
6. $R_1^*$ where $R_1$ is a regular expression

In the above definition, $\cdot$ is the concatenation operator and $*$ is the Kleene operator. For finite alphabets, we have the following equivalence between regular languages and languages described by regular expressions.

**Theorem A.3** (Theorem 1.54 of (Sipser, 2013))**.** *A language (over a finite alphabet) is regular iff some regular expression describes it.*

# B Proofs in Section 2

## B.1 Proof of Theorem 2.3

*Proof.* (a) As $M_t = upd(M_{t-1}, X_t)$, conditioning on $(M_{t-1}, X_t)$ makes $M_t$ conditionally independent from all other variables. A similar observation holds for $C_t = eval(M_t)$ conditioned on $M_t$. Therefore, Figure 7 is a graphical model of the joint distribution $P(X_1, M_1, C_1, \cdots, X_T, M_T, C_T)$. In the graph, $(M_{t-1}, X_{t-1})$ blocks all paths between the past $(X_t, M_t)$'s and the future $(X_t, M_t)$'s. Therefore, by $d$-separation (Appendix A.1) the augmented chain $\{(M_t, X_t)\}$ is Markov. Since each $\mathcal{C}_t$ is an emission of $(X_t, M_t)$, $\{(X_t, M_t), \mathcal{C}_t\}$ is a hidden Markov model.

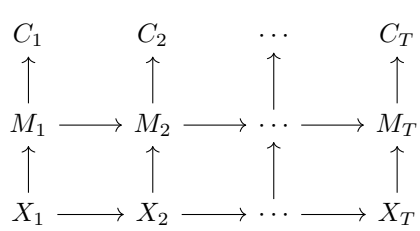



Figure 7: Full Augmented Hidden Markov Model

(b) It is well-known that the hidden sequence of a HMM, conditioned on its emissions, is a Markov chain.

(c) We disintegrate the conditional augmented measure as:

$$\tilde{\mu}^{\mathcal{C}}(m_{1:T}, x_{1:T}) = p(m_{1:T}|x_{1:T}, \mathcal{C}=1)p(x_{1:T}|\mathcal{C}=1) = \mathbb{1}_{M(x_{1:T})}(m_{1:t})p(x_{1:T}|\mathcal{C}=1) \ a.s.$$

The second equality follows from $M_{1:T}$ being a function of $X_{1:T}$, so the conditional measure $p(m_{1:T}|x_{1:T}, \mathcal{C}=1)$ is a point mass at the value $M(x_{1:T})$; note that this decomposition holds almost surely even if $\{\mathcal{C}=1\} \cap \{M^{-1}(m_{1:T})\} = \emptyset$. Thus, $supp(\tilde{\mu}^{\mathcal{C}}) \subseteq (\mathcal{M} \times \mathcal{X})^T$ is a subset of the graph of $M : \mathcal{X}^T \to \mathcal{M}^T; x_{1:T} \mapsto M(x_{1:T})$.

(d) We define $\hat{\mu}^{\mathcal{C}}(\cdot) \equiv \tilde{\mu}^{\mathcal{C}}(\mathcal{M}^T \times \cdot)$, and from the above computation we see that $\hat{\mu}^{\mathcal{C}}(\cdot) = p(\cdot|\mathcal{C}=1)$.

□

### B.2 Proof of Corollary 2.4

*Proof.* Figure 8 shows the augmented chain for a subsequence constraint $\mathcal{C}$ on $[t_1, t_2]$. We repeat the same arguments as in the proof of Theorem 2.3, and Theorem 2.3a allows us to add finitely many subsequence constraints. □

$$\begin{tikzcd}
 & & & C_{t_2} & \\
 & M_{t_1} \arrow[r] & \cdots \arrow[r] & M_{t_2} \arrow[u] & \\
X_{t_1-1} \arrow[r] & X_{t_1} \arrow[r] \arrow[u] & \cdots \arrow[r] \arrow[u] & X_{t_2} \arrow[r] \arrow[u] & X_{t_2+1}
\end{tikzcd}$$

Figure 8: Augmented Chain for Subsequence Constraint

## C Proofs in Section 3

### C.1 Proof of Lemma 3.1

*Proof.* MVR to DFA. Given an MVR $(\mathcal{X}, \mathcal{M}, init, upd, eval)$, the equivalent DFA is $(\mathcal{X}, \mathcal{M} \cup \{m_0\}, m_0, eval^{-1}(1), \delta)$. We add a start symbol $m_0$ to the DFA's internal states, and the accepting states $eval^{-1}(1) \subseteq \mathcal{M}$ are those that get mapped to 1. We define the transition function $\delta$:

$$\delta(m, x) = \begin{cases} upd(m, x) & m \neq m_0 \\ init(x) & m = m_0 \end{cases}$$

DFA to MVR. Conversely, given DFA $(\mathcal{X}, Q, q_0, A, \delta)$ we construct the MVR $(\mathcal{X}, Q \setminus \{q_0\}, init, upd, eval)$. We remove the starting state $q_0$ from the internal space and define the relevant functions:

$$\begin{aligned} init(x) &= \delta(q_0, x) \\ upd(q, x) &= \delta(q, x) \\ eval(q) &= \mathbb{1}_A(q) \end{aligned}$$

These two construction methods are inverses of each other, establishing the equivalence. □

### C.2 Proof of Proposition 3.2

*Proof.* The equivalences follow from Lemma 3.1 and Theorem A.3. □

### C.3 Proof of Proposition 3.5

*Proof.* $\{Sat_{\mathcal{C}} = t\} = \{C_1 = 0, \cdots, C_{t-1} = 0, C_t = 1\} \in \mathcal{F}_t$ since $\mathcal{C}_s \in \mathcal{F}_s$ and $\mathcal{F}_s \subseteq \mathcal{F}_t$ for all $s \leq t$. □

### C.4 Proof of Proposition 3.3

*Proof.* First, we show a prefix-free constraint generates a stopping time. A prefix-free constraint $\mathcal{C}$ is one where if $x$ is feasible then $xv$ cannot be feasible for any nonempty extension $v \in \mathcal{X}^*$. As the filtration is canonical, the feasible sets take the form $\mathcal{C}_t = A_t \times \mathcal{X}^{\mathbb{N}^+}$ for some measurable $A_t \subseteq \mathcal{X}^t$. By the prefix property, if $x \in A_t$ then $xv \notin A_{t+len(v)}$ for any nonempty extension $v$, and $\mathcal{C}_t$ is disjoint from all $\mathcal{C}_s$ with $s > t$. By the same argument, no prefix of $x$ can be in $A_s$ for $s < t$, and $\mathcal{C}_t$ is disjoint from $\mathcal{C}_s$ for $s < t$. Thus, for prefix-free constraints the $\mathcal{C}_t$'s are disjoint sets. We may define a stopping time $\tau_{\mathcal{C}}$ with $\{\tau_{\mathcal{C}} = t\} \equiv \mathcal{C}_t$ and $\{\tau_{\mathcal{C}} = \infty\} \equiv (\bigcup_{t \geq 1} \mathcal{C}_t)^c$.

We now show every stopping time generates a prefix-free constraint. Given a stopping time $\tau$, we define the constraint $\mathcal{C}^\tau$ with $\mathcal{C}^\tau_t \equiv \{\tau = t\} \in \mathcal{F}_t$. Since the sets $\{\tau = t\}$ are disjoint, this implies the prefix property using the above argument. Our constructions give a bijection between the sets $\mathcal{C}_t$ and $\{\tau = t\}$. Since $\{\tau = \infty\}$ is completely determined by the sets $\{\tau = 1\}, \{\tau = 2\}, \cdots$, this establishes the equivalence. □

### C.5 Proof of Proposition 3.6

*Proof.* For regular $\mathcal{C}$, let $(\mathcal{X}, \mathcal{M}, init, upd, eval)$ be a MVR of $\mathcal{C}$. We create a new MVR $(\mathcal{X}, \mathcal{M}', init', upd', eval')$ taking values in $\mathcal{M}' = \mathcal{M} \times \{0, 1, e\}$ with new functions $init', upd', eval'$ defined as:

$$
\begin{aligned}
init' : \mathcal{X} \to \mathcal{M}' \quad &; \quad init'(x) = (init(x), eval \circ init(x)) \\
upd' : \mathcal{M}' \times \mathcal{X} \to \mathcal{M}' \quad &; \quad upd((m, b), x) = \begin{cases} (m, e) & b \in \{1, e\} \\ upd(m, x) & \text{otherwise} \end{cases} \\
eval : \mathcal{M}' \to \{0, 1\} \quad &; \quad eval((m, b)) = \mathbb{1}_{b=1}(b)
\end{aligned}
$$

This new MVR modifies the original one by adding an absorbing non-accepting state $e$ [2], and once the MVR hits an accepting state it transitions to $e$ and stays there. It is easily seen that this new MVR represents the prefix-free constraint of $Sat_{\mathcal{C}}$. □

## D Proofs in Section 4

### D.1 Proof of Proposition 4.1

*Proof.* We will show that for a regular constraint $\mathcal{C}$ on $\mathcal{X}$ with MVR $MV$, there exists a measurable map into a finite space $g : \mathcal{X} \to \mathcal{Y}$ and a DFA $F$ over $\mathcal{Y}$ such that $x$ feasible iff $g(x)$ is accepted by $F$. For pairs of constraints $\mathcal{C}^1, \mathcal{C}^2$, we show that there exists a common map $g$ with equivalent DFAs $F_1, F_2$ accepting strings $g(x)$ for $\mathcal{C}^1, \mathcal{C}^2$ respectively. Once we show this, we may apply standard closure properties of regular languages over finite alphabets Sipser (2013).

Let $MV = (\mathcal{X}, \mathcal{M}, init, upd, eval)$ be an MVR of regular constraint $\mathcal{C}$. We define the sets:

$$
\begin{aligned}
B_{m_1, m_2} &= \{x \in \mathcal{X} |\ upd(m_1, x) = m_2\} \qquad m_1, m_2 \in \mathcal{M} \\
B_m &= \{x \in \mathcal{X} |\ init(x) = m\} \qquad m \in \mathcal{M}
\end{aligned}
$$

Note that $B_{m_1, m_2}, B_m \in \mathcal{F}$ for all $m_1, m_2, m \in \mathcal{M}$. Fixing $m_1$, the collection $\{B_{m_1, m_2}\}_{m_2 \in \mathcal{M}}$ forms a finite partition of $\mathcal{X}$, and similarly the collection $\{B_m\}_{m \in \mathcal{M}}$ is also a finite partition. Thus, we have $|\mathcal{M}| + 1$ different partitions of $\mathcal{X}$, each partition having at most $|\mathcal{M}|$ non-empty members. Taking finitely many intersections, we create a refinement $\{B_y\}_{y \in \mathcal{Y}}$ of the partitions with finite index set $\mathcal{Y}$. As each $B_y$ is a finite intersection of measurable sets, we have $B_y \in \mathcal{F}$. We define the embedding $g : \mathcal{X} \to \mathcal{Y}$ by sending $x \mapsto y$ if $x \in B_y$. Since the $B_y$ partition $\mathcal{X}$ into measurable sets, $g$ is a well-defined measurable function.

[2] Since MVRs are essentially DFAs, by accepting states we mean states in $\mathcal{M}$ that map to 1 under *eval*, and vice versa.

Similarly, we construct the DFA $F$ over $\mathcal{Y}$ in two steps. We first construct a MVR $MV'$ over $\mathcal{Y}$ and then convert it into the desired DFA. We construct $MV' = (\mathcal{Y}, \mathcal{M}, init', upd', eval)$ by reusing the same internal space $\mathcal{M}$. We then define $init'(y) = init(x)$ for some $x \in B_y$. Since $B_y$ is a refinement of $\{B_m\}$, the choice of $x \in B_y$ does not matter. We similarly define $upd'$: $upd'(m, y) = upd(m, x),\ x \in B_y$. We reuse the evaluation function $eval$. This fully defines the MVR $MV'$ over $\mathcal{Y}$, and we convert this into a DFA $F$ using Proposition 3.1. By construction, a string $x = x_1 x_2 \cdots x_k$ is feasible iff $g(x) = g(x_1)g(x_2)\cdots g(x_k) \in \mathcal{Y}^k$ is accepted by $F$.

For pairs of regular constraint $\mathcal{C}^1, \mathcal{C}^2$ with MVRs $MV^1, MV^2$, we generate the sets $B^i_{m_1,m_2}, B^i_m$ for $i = 1, 2$ using the above procedure. Taking a refinement of these $2|\mathcal{M}| + 2$ finite partitions, we build the common finite embedding $g$ and corresponding DFAs $F_1, F_2$ following the above procedure.

□

### D.2 Auxiliary Lemmas

**Lemma D.1.** *Let $\tau$ be a stopping time. The following are equivalent:*

*(a) $\tau$ is regular.*

*(b) $\mathcal{C}^{\tau\leq} = \bigcup_{t\geq 1}\{\tau \leq t\}$ is regular.*

*(c) $\mathcal{C}^{\tau>} = \bigcup_{t\geq 1}\{\tau > t\}$ is regular.*

*Proof.*

1. $(a) \implies (b)$. If $\tau$ is regular, then its associated prefix-free constraint $\mathcal{C}^\tau$ (Proposition 3.3) is regular by definition. $\mathcal{C}^{\tau\leq}$ is the set of all feasible strings with a prefix in $\mathcal{C}^\tau$. To construct a MVR for $\mathcal{C}^{\tau\leq}$, we use the same strategy as the proof in Section C.5 to build one from the MVR of $\mathcal{C}^\tau$, except now $e$ is an absorbing accepting state. This new MVR runs the MVR of $\mathcal{C}^\tau$ until it hits an accepting state. Then, it transitions to $e$ and stays there, so subsequent $M_t$'s take values in $\mathcal{M} \times \{e\}$. This means that $eval_t = 1$ for $\tau \leq t$, and this is the desired MVR of $\mathcal{C}^{\tau\leq}$.

2. $(b) \implies (a)$. Since $\tau$ is the satisfaction time of $\mathcal{C}^{\tau\leq}$, this follows from Proposition 3.6.

3. $(b) \iff (c)$. Regarding $\{\tau \leq t\}$ as a measurable subset of $\mathcal{X}^t$, we see $\{\tau > t\} = \{\tau \leq t\}^c$ is also a measurable subset of $\mathcal{X}^t$. $\mathcal{C}^{\tau\leq}$ and $\mathcal{C}^{\tau>}$ are complements of each other, and regularity follows from Proposition 4.1.

□

**Lemma D.2.** *If $\tau_1, \tau_2$ are regular stopping times, then $\tau = \tau_2 \circ \theta^{\tau_1} + \tau_1$ is also a regular stopping time, where $\theta$ is the shift operator.*

*Proof.* Unpacking definitions, $\{\tau = t\} = \bigcup_{1\leq j<t}\{\tau_1 = j,\ \tau_2 \circ \theta^{\tau_1} = t - j\}$, and any sequence in $x \in \{\tau = t\}$ consists of two contiguous segments $x = wv$ with $w \in \{\tau_1 = j\}$ and $v \in \{\tau_2 = t - j\}$. Thus, we see that the associated prefix-free constraint $\mathcal{C}^\tau$ is the concatenation $\mathcal{C}^{\tau_1} \cdot \mathcal{C}^{\tau_2}$. By Proposition 4.1, $\mathcal{C}^\tau$ is a regular constraint, and $\tau$ is a regular stopping time. □

**Lemma D.3.** *The constant stopping times are regular.*

*Proof.* For constant stopping time $\tau \equiv k$, the corresponding MVR essentially counts time until it hits $k$ before transitioning to an absorbing non-accepting state. That is, $\mathcal{M} = \{1, \cdots, k, e\}$, and $upd(j, x) = j + 1$ for $j < k$ and $upd(j, x) = e$ for $j \in \{k, e\}$. We have $eval(j) = \mathbb{1}_k(j)$. □

### D.3 Proof of Proposition 4.2

*Proof.* This follows from Lemma D.1 applied to $Sat_{\mathcal{C}}$. □

### D.4 Proof of Proposition 4.3

*Proof.* We first prove the result for events of the form $B = B_1 \times \cdots B_k$ with $B_i \in \mathcal{F}$. Let $\tau_B$ be the hitting time of $B$. We show that $\tau_B$ is regular by giving a sketch of its MVR. The indicator array $B_{1:k} : \mathcal{X} \to \{0,1\}^k;\ B_{1:k}(x) = (\mathbb{1}_{B_1}(x), \cdots, \mathbb{1}_{B_k}(x))$ is a measurable function. We then define an increment function $inc : [0,k] \times \{0,1\}^k \to [0,k]$ where $inc(j-1,b) = j$ if the $j^{th}$ entry $b_j$ is 1 and $inc(j,b) = b_1$ otherwise. The increment function processes the output of $B_{1:k}$, checking that we hit the sets $B_i$ in sequence and restarting the counter if we transition out. When $inc$ outputs $k$ signifying $B$ has occurred, the evaluation function $eval$ returns a 1. After this, the MVR will transition to an absorbing, non-accepting state as in the proof of Section $C.5$. This is the desired MVR of the hitting time $\tau_B$.

For simple events of the form $B = \bigcup_{i=1}^n B_{i_1} \times \cdots \times B_{i_k}$, we apply the above procedure to each term $B_i = B_{i_1} \times \cdots B_{i_k}$ to show that $\tau_{B_i}$ is regular. Let $\mathcal{C}^{\tau_{B_i}}$ be its associated regular constraint. It is easily seen that the prefix-free constraint $\mathcal{C}^{\tau_B}$ of $\tau_B$ is the union $\bigcup_{i=1}^n \mathcal{C}^{\tau_{B_i}}$, and by Proposition 4.1 $\tau_B$ is a regular stopping time.

Finally, as the feasibility of constraint $\mathcal{C}$ only depends on the occurrence of simple event $B$ at some point, we see that $\mathcal{C}$ is exactly the set of strings $x$ where $\tau_B \leq |x|$. Therefore, $\mathcal{C}^B = \mathcal{C}^{\tau_B \leq}$ is regular by Lemma D.1. □

### D.5 Proofs of Proposition 4.4 and Corollary 4.5

*Proof.* We give a sketch of the MVR for $Sat_{C_1} < Sat_{C_2}$. By Proposition 3.6 the satisfaction times $Sat_{C_1}, Sat_{C_2}$ are regular, and let $MV^1, MV^2$ be MVRs of their respective prefix free constraints. Our new MVR runs both $MV^1, MV^2$ and tracks $eval_t^1, eval_t^2$. It has one binary memory variable $\{mem_t\}$ that tracks if $\mathcal{C}^1$ has been satisfied yet: $mem_t = \mathbb{1}_{Sat_{\mathcal{C}^1 \leq t}}$ with updates $mem_t = max(mem_{t-1}, eval_t^1)$. To enforce precedence the MVR has another binary variable $\{prec_t\}$ that tracks if $\mathcal{C}^2$ has occurred before $\mathcal{C}^1$ at any time: $prec_t = prec_{t-1} \cdot [1 - (1 - mem_{t-1}) \cdot eval_t^2]$. At time $t$, $prec_t = 1$ as long as either $\mathcal{C}^2$ has not been satisfied yet or $\mathcal{C}^1$ has been previously satisfied. This MVR is the desired representation of $Sat_{C_1} < Sat_{C_2}$.

For $Sat_{C_1} \leq Sat_{C_2}$, we modify the precedence tracker $prec_t = prec_{t-1} \cdot [1 - (1 - mem_t) \cdot eval_t^2]$ to include the case of both constraints being satisfied at the same time.

The corollary follows from $\tau_A, \tau_B$ being regular for simple events (Propositions 4.3 and 3.6) and Proposition 4.4. □

### D.6 Proof of Proposition 4.6

*Proof.* Define $\tau^k$ recursively with $\tau^1 = \tau$ and $\tau^k = \tau \circ \theta^{\tau^{k-1}} + \tau^{k-1}$. Let $\tau_{\mathcal{C}} \equiv \tau_{Sat_{\mathcal{C}}}$ be shorthand for the satisfaction time. We see $\tau_{\mathcal{C}}^k$ describes the strings composed of exactly $k$ contiguous, non-overlapping feasible substrings. Since $\mathcal{C}$ and hence $\tau_{\mathcal{C}}$ is regular, repeated applications of Lemma D.2 show that $\tau_{\mathcal{C}}^k$ is regular. Hence, the constraint $\mathcal{C}^{\tau_{\mathcal{C}}^k \leq}$ specifies all strings that begin with $k$ contiguous feasible substrings. We have $Ct(\mathcal{C}, k) = \mathcal{C}^{\tau_{\mathcal{C}}^k \leq} \cap (\mathcal{C}^{\tau_{\mathcal{C}}^{k+1} \leq})^c$, which is regular by Proposition 4.1.

Similarly, Proposition 4.1 can also be used to show that $Ct(\mathcal{C}, [l,k]) = \bigcup_{j=l}^{k} Ct(\mathcal{C}, j)$ and $Ct(\mathcal{C}, > k) = [\bigcup_{j=0}^{k} Ct(\mathcal{C}, j)]^c$ are both regular, with $Ct(\mathcal{C}, 0) = \mathcal{C}^{>}$. □

## E Discussion of Constraint Examples in Section 4

We briefly sketch how regularity of examples listed in Section 4 and Table 1 can be established.

Hitting constraints for states or finite sequence Pnueli (1977); Baier & Katoen (2008) depend on a simple event and are hence regular (Proposition 4.3). Similarly, an unlocked transition Liu et al. (2024); Baier & Katoen (2008) ($a$ must happen before $b \to c$) corresponds to a precedence constraint between the hitting time

of $a$ and the hitting time of the sequence $(b, c)$, which are both satisfaction times of simple events (Proposition 4.3).

Spaced sequence constraints Ma et al. (2002), where states $a, b, c$ must happen in sequence with arbitrary gaps, and similar temporal logic properties are fundamental to formal methods (Pnueli, 1977; Baier & Katoen, 2008; Liu et al., 2024), with applications with timing requirements such as manufacturing processes and clinical protocols (Masubuchi et al., 2017). The regularity of the spaced sequence constraint $a < b < c$ can be shown with the following argument: let $\mathcal{C}^a, \mathcal{C}^b, \mathcal{C}^c$ be the constraints where we hit $a, b, c$ at least once respectively (Lemma 4.3). Similarly, let $\mathcal{C}^{a<b}, \mathcal{C}^{b<c}$ be the respective precedence constraint (Corollary 4.5). We then see that $\mathcal{C} = [\mathcal{C}^a \cap \mathcal{C}^b \cap \mathcal{C}^c] \bigcap [\mathcal{C}^{a<b} \cap \mathcal{C}^{b<c}]$ is the desired spaced sequence constraint.

Assuming finite $\mathcal{X}$, the hitting time of a jump is the complement of the union of self-transition hitting times: $\mathcal{C}^{jump} = (\bigcup_{x \in \mathcal{X}} \mathcal{C}^{(x,x)})^c$. The jump count constraint is the counting constraint generated from $\mathcal{C}^{jump}$.

Similarly for finite $\mathcal{X}$, we show that the staying time constraint Pola et al. (2003); Yu (2010) is regular, where we stay in each state for at least time $k$. Let $\mathcal{C}^{jump>k} = \mathcal{C}^{jump} \cap \mathcal{C}^{>k}$ be the set of strings where the first jump to a distinct state occurs after time $k$. $\mathcal{C}^{>k}$ is the set of all strings of length greater than $k$, which is regular by Propositions D.3 and D.1. Applying the Kleene star, we see $(\mathcal{C}^{jump>k})^*$ is the set of all strings where we stay in each state for at least time $k$. We then concatenate this with the constant sequence set of at most length $k$, $\mathcal{C}' = \bigcup_{x \in \mathcal{X}} \bigcup_{j \leq k} \{x\}^j$. This gives the desired staying time constraint: $\mathcal{C}^{stay} = (\mathcal{C}^{jump>k})^* \cdot \mathcal{C}'$.

For a resource constraint Boutilier et al. (2016); Altman (1999), by assumption the budget is a sum over time of individual state scores: $B_t = \sum_{s \leq t} score(x_s)$. This may be viewed as a particular case of a mediation variable, as $B_t = B_{t-1} + score(X_t)$. We can then define the *eval* function appropriately to ensure $B_t$ stays under $K$: $eval(b) = \mathbb{1}_{b \leq K}(b)$. If we want to ensure that the budget never exceeds $K$, we take the complement of the sequences where the budget exceeds $K$ (changing the above *eval* function) and the perform the "never" operation of Proposition 4.2.

A return time constraint Masubuchi et al. (2017) for visits to state $a$ can be cast as the union of two constraints: (1) the strings where $a$ never happens (2) the strings where $a$ happens at least once. The former is the "never" constraint of the hitting time $\mathcal{C}^{a>}$ (Lemma D.1). The latter can be represented as the concatenation of: (1) strings in the hitting time $\tau_a$, (2) strings in the Kleene star of $\tau_a$ having length at least $k$: $\mathcal{C}^a \cdot (\mathcal{C}^a \bigcap \mathcal{C}^{\geq k})^*$. This specifies a constraint where we must wait at least time $\delta$ before returning, and taking complements gives us the constraint where the gap is at most $\delta$.

### E.1 Connection to pattern matching

In the pattern matching literature Tataru et al. (2013); Nuel (2008); Ma et al. (2002), one has a set of patterns $P$ specified by some regular expression over a finite alphabet, and in the non-overlapping case one counts disjoint occurrences $p \in P$ in a string. In this scenario, there might be arbitrary gaps between patterns, and we show how our definitions subsume this setting. Firstly, by the equivalence between regular languages and regular expressions, there exists some DFA accepting pattern set $P$. Since $\mathcal{X}^*$ is trivially regular, the set of all strings ending in $P$ is regular, denoted as $\mathcal{C}' = \mathcal{X}^* \cdot P$. Let $\mathcal{C}''$ be the prefix-free version of $\mathcal{C}'$, generated from the satisfaction time of $\mathcal{C}'$. The constraint $\mathcal{C}''$ consists of all strings where, reading left-to-right, we stop as soon we encounter a pattern $p \in P$. Sequences that contain $k$ counts of $P$ coincide with $Ct(\mathcal{C}'', k)$.

## F Derivations of Algorithms in Section 5

Both algorithms are modifications of the forward-backward (FB) procedure used in the Baum-Welch algorithm Baum et al. (1970) for HMMs. The FB procedure computes the following forward messages $\alpha$ and backward messages $\beta$ given the observed emission sequence $y_{1:T}$:

$$\alpha_k^t = P(y_{1:t}, X_t = k)$$
$$\beta_k^t = P(y_{t+1:T} | X_t = k)$$

Letting $\pi_{jk} = P(X_t = k|X_{t-1} = j)$ and $E_k^t = P(y_t|X_t = k)$, the FB equations are:

$$\alpha_k^t = \sum_j \alpha_j^{t-1} \pi_{jk} E_k^t$$
$$\beta_k^t = \sum_j \beta_j^{t+1} \pi_{kj} E_j^{t+1}$$

After computing these messages, summing them at any time-index gives the probability of the observed sequence:

$$\forall t, \qquad P(y_{1:T}) = \sum_k \alpha_k^t \beta_k^t$$

### F.1 Derivation of Algorithm 1: Satisfaction Time Estimation

To compute the distribution of the satisfaction time $Sat_{\mathcal{C}}$, we work with the full augmented HMM (Figure 7). Since $\{Sat_C = t\} = \{C_1 = 0, \cdots, C_{t-1} = 0, C_t = 1\}$, we have:

$$\begin{aligned} P(y_{1:T}, Sat_{\mathcal{C}} = t) &= \sum_{x_t, m_t} P(y_{1:T}, Sat_C = t, x_t, m_t) \\ &= \sum_{x_t, m_t} P(y_{1:t}, C_{1:t-1} = 0, C_t = 1, x_t, m_t) P(y_{t+1:T}|x_t, m_t) \\ &= \sum_{x_t, m_t} P(y_{1:t}, C_{1:t-1} = 0, C_t = 1, x_t, m_t) P(y_{t+1:T}|x_t) \end{aligned}$$

For the last equality, inspection of Figure 7 shows that $M_{t+1}$ is an unconditioned collider, so conditioning on $X_t$ blocks all paths from $M_t$ to future $X_{t+1:T}$ and hence future emissions $Y_{t+1:T}$. Therefore, by $d$-separation $Y_{t+1:T} \perp_{X_t} M_t$ and the second term in the product simplifies to the original backward message $\beta(x_t) = P(y_{t+1:T}|x_t)$. Similarly, we can decompose the first term in the product as:

$$\begin{aligned} P(y_{1:t}, C_{1:t-1} = 0, C_t = 1, x_t, m_t) &= \sum_{x_{t-1}, m_{t-1}} P(y_{1:t}, C_{1:t-1} = 0, C_t = 1, x_{t-1}, m_{t-1}, x_t, m_t) \\ &= \sum_{x_{t-1}, m_{t-1}} P(y_{1:t-1}, C_{1:t-1} = 0, x_{t-1}, m_{t-1}) P(C_t = 1, x_t, m_t|x_{t-1}, m_{t-1}) \\ &= \sum_{x_{t-1}, m_{t-1}} P(y_{1:t-1}, C_{1:t-1} = 0, x_{t-1}, m_{t-1}) P(x_t|x_{t-1}) upd(m_t|m_{t-1}, x_t) eval(C_t = 1|m_t) \end{aligned}$$

A similar derivation holds for the $\mathcal{C}_t = 1$ case. Therefore, for the forward $\alpha$ messages we track not only the original state but also the mediation variable(s) and current emission. The forward messages $\alpha$ and backward messages $\beta$ are:

$$\alpha_{krb}^t = P(y_{1:t}, C_{1:t-1} = 0, C_t = b, X_t = k, M_t = r)$$
$$\beta_k^t = P(y_{t+1:T}|X_t = k)$$

Let $\rho_k = P(X_1 = k)$, $eval_r^b = \mathbb{1}_{eval(r)=b}$, $init_{kr} = \mathbb{1}_{init(k)=r}$, and $upd_{r|ks} = \mathbb{1}_{upd(s,k)=r}$. Following the same computations as the original forward-backward procedure, we derive the following initialization and update

equations:

$$\alpha^1_{krb} = \rho_k E^1_k init_{kr} eval^b_r$$
$$\beta^1_k = 1$$
$$\alpha^t_{krb} = \sum_{js} \alpha^{t-1}_{js0} \pi_{jk} upd_{r|ks} E^t_k eval^b_r$$
$$\beta^t_k = \sum_j \beta^{t+1}_j \pi_{kj} E^{t+1}_j$$
$$P(y_{1:T}, Sat_{\mathcal{C}} = t) = \sum_{k,r} \alpha^t_{kr1} \beta^t_k$$

Summing the appropriate $\alpha$ and $\beta$ messages, we get the quantities $P(y_{1:T}, Sat_{\mathcal{C}} = t)$ which specifies the distribution of $Sat_{\mathcal{C}}$. One can normalize this distribution to get the posterior distribution $P(Sat_{\mathcal{C}} = t | Sat_{\mathcal{C}} \leq T, y_{1:T})$. Similarly, one can compute $P(y_{1:T})$ via another run of the FB procedure on the original model, and dividing by $P(y_{1:T})$ gives the related posterior $P(Sat_{\mathcal{C}} = t | y_{1:T})$.

Finally, note that this algorithm extends to cases where we include other extraneous constraints $\mathcal{C}'$ by applying it to the augmented HMM of $\mathcal{C}'$. This allows us to compute the satisfaction time distribution of a regular constraint $\mathcal{C}$ while conditioning on other constraints $\mathcal{C}'$. Section 6.1 provides an example of this computation.

**Algorithm 1** Forward-Backward Procedure for Satisfaction Time Distribution

**Input:** Markov chain $\{\rho, \pi\}$, $MVR_{\mathcal{C}}$ $\{upd, init, eval\}$, optional evidence/constraints $y$

**Initialize messages and emission weights**

$E^t = \text{evidenceToWeights}(y)$ ▷ convert evidence to state weights

$\alpha^1_{krb} \leftarrow \rho_k E^1_k init_{kr} eval^b_r$

$\beta^T_k \leftarrow 1$

**Forward and Backward Pass**

**for** $t = 2$ to $T$ **do**

  $\alpha^t_{krb} \leftarrow \sum_{j,s} \alpha^{t-1}_{js0}\ \pi_{jk}\ upd_{r|ks}\ E^t_k\ eval^b_r$

**end for**

**for** $t = T - 1$ to $1$ **do**

  $\beta^t_k \leftarrow \sum_j \beta^{t+1}_j\ \pi_{kj}\ E^{t+1}_j$

**end for**

**Compute Distribution**

$p^t \leftarrow \sum_{k,r} \alpha^t_{kr} \beta^t_k$

$p^t \leftarrow p^t / \sum_t p^t$

### F.2 Derivation of Algorithm 2: Posterior Satisfaction Probability

A similar message-passing procedure exists for computing the posterior probability that a constraint is satisfied: $P(\mathcal{C}_T | y_{1:T})$. Using the same notation in the previous derivation, we compute the forward messages

$\alpha^t_{kr} = P(y_{1:t}, X_t = k, M_t = r)$, incorporating the evaluation emission at only the last time $T$.

$$\alpha^1_{kr} = \rho^1_k E^1_k init_{kr}$$
$$\alpha^t_{kr} = \sum_{js} \alpha^{t-1}_{js} \pi_{jk} E^t_k upd_{r|ks} \quad ; \quad 1 < t < T$$
$$\alpha^T_{krb} = \sum_{js} \alpha^{T-1}_{js} \pi_{jk} E^T_k upd_{r|ks} eval^b_r$$
$$P(\mathcal{C}_T = b, y_{1:T}) = \sum_{kr} \alpha^T_{krb}$$

After normalization, this provides the required posterior distribution $P(\mathcal{C} = b|y_{1:T})$. As in the satisfaction time algorithm, this algorithm can also incorporate other constraints $\mathcal{C}'$ by applying it to the appropriate augmented HMM. Section 6.1 provides a demonstration of this algorithm as well.

**Algorithm 2** Forward Algorithm for Computing Posterior Probability of Satisfaction

**Input:** Markov chain $\{\rho, \pi\}$, $MVR_{\mathcal{C}}$ $\{upd, init, eval\}$, time horizon $T$, optional evidence/constraints $y$

**Initialization**
$E^t = \text{evidenceToWeights}(y)$ ▷ convert evidence to state weights
$\alpha^1_{kr} \leftarrow \rho^1_k E^1_k init_{kr}$

**Forward Pass**
**for** $t = 2$ to $T$ **do**
  **if** $t < T$ **then**
    $\alpha^t_{kr} \leftarrow \sum_{j,s} \alpha^{t-1}_{js} \pi_{jk} E^t_k upd_{r|ks}$
  **else**
    $\alpha^T_{krb} \leftarrow \sum_{j,s} \alpha^{T-1}_{js} \pi_{jk} E^T_k upd_{r|ks} eval^b_r$
  **end if**
**end for**

**Compute Posterior Probability**
$p \leftarrow \sum_{k,r} \alpha^T_{krb}$
$p \leftarrow p / \sum_b p_b$

### F.3 Derivation of Algorithm 3: Constrained Stopped Sampling

The decomposition

$$P(X_{1:\tau}|y, \tau \leq T) = P(X_{1:\tau}|\tau, y) P(\tau|y, \tau \leq T)$$

naturally motivates our procedure: first sample $\tau$ from its posterior, then sample $X_{1:\tau}$ using the standard Forward-Filtering Backward-Sampling (FFBS) algorithm. In our notation, $y$ represents all path constraints and any additional evidence like observed chain emissions. Once the time horizon $\tau$ is fixed, FFBS samples from the exact posterior of an HMM. Since mediation variables interprets constraint satisfaction as binary emissions of the augmented HMM (Figure 1b), the FFBS algorithm samples from the exact constrained posterior of the original chain. Thus, Algorithm 3 accomplishes the first sampling task for regular stopping time $\tau$. Standard FFBS on the augmented chain, incorporating the desired constraints, achieves the second. Note that the MVR of $\tau$ does not generally enter the FFBS step, since $\tau$ is interpreted as a stopping time and not a path constraint.

## G Decoding via At-Time Steering is Generally Approximate

Several approaches to constrained sequential inference use an auxiliary score function that steers the path estimate $\hat{x}_{1:T}$ to feasible trajectories Deutsch et al. (2019); Roth & Yih (2004; 2005). While these algorithms

can be more compact, they cannot exactly represent some constraints. These algorithms all sequentially build the path estimate $\hat{x}_{1:T}$ by optimizing the following objective (possibly as a subroutine):

$$\hat{x}_t = \arg\max_x \log p(x|\hat{x}_{1:t-1}, Y) + g(x|\hat{x}_{1:t-1}, Y)$$
$$\hat{x}_{1:t} = \hat{x}_{1:t-1} \cdot \hat{x}_t$$

where $Y$ represents some general evidence, $\hat{x}_{1:t-1}$ is the current path estimate, and $g : \mathcal{X} \to \overline{\mathbb{R}}$ is a score function. Note that if we were to ignore $g$, the above procedure would be a subroutine of the Viterbi algorithm which tracks the best path to each state at each time. The score weights the next possible states, ensuring that the estimate remains feasible as we extend it one state at a time. As the notation suggests, $g$ can depend on both the current estimate $\hat{x}_{1:t-1}$ and evidence $Y$. In particular, while prior greedy approaches Deutsch et al. (2019) also use automata to enforce feasibility, this automaton $A$ is instantiated as a score function $A_y(x|x_{1:t-1})$, limiting its expressivity relative to our approach.

Importantly, the score function $g(x|\hat{x}_{1:t-1}, Y)$ does not have access to future information, limiting the types of constraints these approaches can exactly represent. For example, consider a Markov chain over state space $\mathcal{X} = (a, b)$ with the constraint that we visit $b$ an odd number of times. Let $g(x|x_{1:t-1}) \in \{-\infty, 0\}$ be a hard scoring function. If $g$ rejects substrings that contain even visits to $b$, it forces initialization to $b$ and the only allowed trajectory is $\hat{x}_{1:t+1} = b \cdot a_{1:t}$ where $a_{1:t}$ is a sequence of $a$'s. On the other hand, suppose $g(x|x_{1:t-1})$ rejects extensions that cannot become a feasible full path. As every strict subsequence can become feasible for our odd count constraint, $g = 0$ for $t < T$. Equip the chain with the following homogeneous measure:

$$\rho = (.5,\ .5) \qquad \pi = \begin{bmatrix} .9 & .1 \\ .2 & .8 \end{bmatrix}$$

Let $T$ be large enough (say $T = 30$) so that the best unconstrained paths ending at $X_{T-1} = a$ and $X_{T-1} = b$ are $a_{1:T-1}$ and $a_{1:T-2} \cdot b$ respectively. As every subsequence has a feasible extension, these are the same estimates we would get using $g$ to constrain trajectories. At time $T$, the only feasible extensions of these two paths are $a_{1:T-1} \cdot b$ and $a_{1:T-2} \cdot b \cdot a$ respectively. In the first path,the last $a \to b$ transition incurs a multiplicative penalty of 0.1. In the second path, the last $a \to b \to a$ transitions incurs a multiplicative penalty of 0.02. However, the true optimal path is $b \cdot a_{1:T-1}$, which incurs only a penalty of 0.2 for the first $b \to a$ transition. The problem here is that the algorithm finds the best subpaths over sets that are too coarse, and for exact inference it should further partition paths by whether $b$ has been visited an even or odd number of times. In that case, $b \cdot a_{1:T-2}$ is the optimal length $T - 1$ path that ends at $a$ and visits $b$ an odd number of times.

In general, constraints can exhibit complicated future dependencies, and approaches that rely on at-time steering via score functions can only approximately capture these types of constraints. This shows why tracking $M_t$ is necessary for dynamic-programming algorithms like Viterbi: $M_t$ appropriately partitions the path space so, once feasibility is finally determined, the best constrained path is the optima of some partition.

## H Soundness of MVR-Aware Discretization

One approach to analyzing a continuous space Markov chain is to discretize its state space before applying typical discrete-space algorithms like Viterbi. This is the approach largely taken in the PMC literature, where one discretizes a continuous space model into a finite abstraction before enforcing constraints on the abstraction (Abate et al., 2008; Soudjani et al., 2015; Haesaert et al., 2017; Lavaei et al., 2022; Badings et al., 2023a). However, one risk to this approach is that it introduces spurious paths: if the discretization is too coarse, there might be infeasible paths that map to feasible discrete paths or vice versa. For example, consider a real-valued Markov chain with the simple constraint that the chain never visits the interval $(-.1, .1)$, and we discretize the state space into unit intervals $(n-1, n]$. Depending on whether we mark the cells $(-1, 0]$ and $(0, 1]$ as forbidden or allowed in the discretization, this will give spurious paths of either flavor.

We say a discretization is **sound in constraint $\mathcal{C}$** if it preserves feasibility: each path and its image under discretization are both feasible or both infeasible. Note that there are two separate but related constraints:

one constraint on the original continuous chain, and another on the discretized chain. For a continuous space Markov chain $\{X_t\}$ and constraint $\mathcal{C}$ with MVR $\{M_t\}$, consider any finite length path $x_{1:T}$. Since the MVR is an exact representation of $\mathcal{C}$, calling *eval* on the mediation state $M(x_{1:T})$ is guaranteed to correctly identify the feasibility of $x_{1:T}$. Let $P$ be a finite partition of $\mathcal{X}$, representing our discretization. By taking the refinement of $P$ with the partitions $\{B^t_{m_1,m_2}\}_t \bigcup \{B_m\}$ from the proof of Proposition 4.1 in Appendix D [3], we can construct an equivalent MVR $M'_t$ on the discretized space $\mathcal{Y}$ by defining its maps - $init, upd, eval$ - on the partition cells $p$ according to which members of $\{B^t_{m_1,m_2}\}_t \bigcup \{B_m\}$ that $p$ falls in (see the proof of Proposition 4.1 for more details). Let $D : \mathcal{X}^* \to \mathcal{Y}^*$ be the discretization operator mapping finite-length paths in $\mathcal{X}$ to paths in $\mathcal{Y}$. By construction, we have $eval(M(x_{1:T})) = eval'(M'(D(x_{1:T})))$; using the MVR to both refine the partition and build an equivalent MVR on the discretized chain, we can guarantee the discretization is sound in constraint.

# I Experiment Details

Here, we provide more details on the implementations of the ILP and MVR methods in Section 6.

## I.1 DNA Details

Figure 9 specifies the parameters of our (unconstrained) DNA HMM. Nucleotide sequences were drawn from the constrained HMM via a rejection sampling, with sequence having random length determined by hitting the terminal state *end*. The parameters of the DNA HMM are shown below.

(a) Transition Matrix: $P(X_t \mid X_{t-1})$

| | pro | ex1 | ex2 | int | dis | enh | end |
|---|---|---|---|---|---|---|---|
| pro | .6 | .1 | .1 | .2 | 0 | 0 | 0 |
| ex1 | 0 | .4 | .3 | .2 | .1 | 0 | 0 |
| ex2 | 0 | .3 | .4 | 0 | .2 | .1 | 0 |
| int | .1 | .1 | .1 | .4 | 0 | .1 | .2 |
| dis | 0 | $\frac{1}{3}$ | $\frac{1}{3}$ | 0 | $\frac{1}{3}$ | 0 | 0 |
| enh | 0 | .25 | .25 | .25 | 0 | .25 | 0 |
| end | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

(b) Emission Matrix: $P(Y_t \mid X_t)$

| | A | T | C | G | END |
|---|---|---|---|---|---|
| pro | .1 | .1 | .4 | .4 | 0 |
| ex1 | .1 | .1 | .7 | .1 | 0 |
| ex2 | .7 | .1 | .1 | .1 | 0 |
| int | .25 | .25 | .25 | .25 | 0 |
| dis | .5 | 0 | .5 | 0 | 0 |
| enh | .4 | .4 | .1 | .1 | 0 |
| end | 0 | 0 | 0 | 0 | 1 |

(c) Initial Distribution: $P(X_1)$

| state | $P(X_1)$ |
|---|---|
| pro | .5 |
| ex1 | 0 |
| ex2 | 0 |
| int | .5 |
| dis | 0 |
| enh | 0 |
| end | 0 |

Figure 9: Parameters of DNA sequence HMM

[3]The extra superscript $t$ is used to accommodate time-varying MVRs.

### I.2 ILP Comparison Details

We provide more details on the exact ILP and MVR formulations used in Section 6.3. Up to re-indexing the state space, WLOG we assume that the require subsequence $(a_1, \cdots, a_L) = (0, \cdots, L-1)$. Our ILP formulation is directly based on the formulation found in previous work Roth & Yih (2005); Koller & Friedman (2009):

$$
\begin{aligned}
\max_{x,z,a} \quad & \sum_{k=1}^{K} (\log \rho_k + \log E_{k,y_1})\, x_{1,k} + \sum_{t=2}^{T} \sum_{k=1}^{K} \log E_{k,y_t}\, x_{t,k} \\
& \quad + \sum_{t=1}^{T-1} \sum_{i=1}^{K} \sum_{j=1}^{K} \log \pi_{ij}\, z_{t,i,j} \\
\text{s.t.} \quad & \sum_{k=1}^{K} x_{t,k} = 1 && \forall t = 1, \ldots, T, \\
& \sum_{j=1}^{K} z_{t,i,j} = x_{t,i} && \forall t = 1, \ldots, T-1,\ \forall i = 1, \ldots, K, \\
& \sum_{i=1}^{K} z_{t,i,j} = x_{t+1,j} && \forall t = 1, \ldots, T-1,\ \forall j = 1, \ldots, K, \\
& a_s \leq x_{s+i-1,i} && \forall s = 1, \ldots, T-L+1,\ \forall i = 1, \ldots, N, \\
& a_s \geq \sum_{i=1}^{L} x_{s+i-1,i} - L + 1 && \forall s = 1, \ldots, T-L+1, \\
& \sum_{s=1}^{T-L+1} a_s \geq 1, \\
& x_{t,k} \in \{0,1\} && \forall t = 1, \ldots, T,\ \forall k = 1, \ldots, K, \\
& z_{t,i,j} \in \{0,1\} && \forall t = 1, \ldots, T-1,\ \forall i, j = 1, \ldots, K, \\
& a_s \in \{0,1\} && \forall s = 1, \ldots, T-L+1.
\end{aligned}
$$

where $\rho$, $\pi$, and $E$ are the HMM's initial vector, transition matrix, and emissions matrix respectively and $y_t$ denotes the observed emissions. The above ILP creates binary trackers for the states $x_{tk}$ and transitions $z_{tij}$, with the inequalities ensuring that: (1) exactly one value $x_{tk}$ is chosen for every time $t$, (2) the state and transition assignments are consistent. The inequalities involving only $x$'s and $z$'s form a standard flow-based representation of an unconstrained Markov chain.

To enforce our subsequence constraint, we introduce additional binary variables $a_s$ for $1 \leq s \leq T-L+1$ that track whether the required subsequence has occurred starting at time $t$. The first two inequalities ensure $s_t = 1$ exactly whenever the required sequence $(0, 1, \cdots, L-1)$ occurs starting at $t$, and the last inequality ensures we encounter this sequence at least once.

### I.3 MVRs for Subsequence Constraints

As in the ILP, WLOG we assume this subsequence is $(0, 1, \cdots, L-1)$. We sketch the MVR that tracks this sequence. The mediation space $\mathcal{M}$ consists of the integers $\mathcal{M} = \{0, 1 \cdots, L-1\}$, with our $init, upd, eval$

functions defined as:

$$init(x) = \begin{cases} 1 & x = 0 \\ 0 & \text{otherwise} \end{cases}$$

$$upd(m, x) = \begin{cases} m + 1 & x = m \text{ and } m < L \\ L - 1 & m = L - 1 \\ init(x) & \text{otherwise} \end{cases}$$

$$eval(m) = \mathbb{1}_{L-1}(m)$$

In other words, our mediation space $\mathcal{M}$ tracks how far in the sequence $(0, \cdots, L-1)$ we currently are. We increment the counter if we encounter the next state in the sequence and reset otherwise. Note that the counter also represents the next expected state. The size of $\mathcal{M}$ equals the length of the required sequence, and so the augmented HMM's hidden space grows as $O(KL)$ for state space $K$ and sequence length $L$.

Note that this formulation assumes that the elements of the required sequence are all unique. In the general case, we can instantiate counters $b_i$ for each unique element $a_i$, where $b_i \in \{0, \cdots, mult(a_i)\}$ with $mult(a_i)$ being the multiplicity of $a_i$ in the sequence. The size of the mediation space is $|\mathcal{M}| = \sum_i mult(a_i)$, summing over all unique entries $a_i$.