Journal of Artificial Intelligence Research 69 (2020) 1203-1254 Submitted 04/2020; published 12/2020

Lifted Bayesian Filtering in Multiset Rewriting Systems

Stefan Lüdtke STEFAN.LUEDTKE2@UNI-ROSTOCK.DE
Thomas Kirste THOMAS.KIRSTE@UNI-ROSTOCK.DE
Institute of Visual & Analytic Computing
University of Rostock, Germany

Abstract

We present a model for Bayesian filtering (BF) in discrete dynamic systems where multiple entities (inter)-act, i.e. where the system dynamics is naturally described by a Multiset rewriting system (MRS). Typically, BF in such situations is computationally expensive due to the high number of discrete states that need to be maintained explicitly.

We devise a lifted state representation, based on a suitable decomposition of multiset states, such that some factors of the distribution are exchangeable and thus afford an efficient representation. Intuitively, this representation groups together *similar* entities whose properties follow an exchangeable joint distribution. Subsequently, we introduce a BF algorithm that works directly on lifted states, without resorting to the original, much larger ground representation.

This algorithm directly lends itself to approximate versions by limiting the number of explicitly represented lifted states in the posterior. We show empirically that the lifted representation can lead to a factorial reduction in the representational complexity of the distribution, and in the approximate cases can lead to a lower variance of the estimate and a lower estimation error compared to the original, ground representation.

1. Introduction

Modeling dynamic systems is fundamental for the understanding of complex phenomena in a variety of AI tasks. Many dynamic systems consist of multiple, interacting entities which can be grouped into species, like biochemical reactions (Barbuti et al., 2011), population dynamics in ecological studies (Pescini et al., 2006), or *human activity recognition* (HAR, Bulling et al., 2014), the main motivation of this paper: As a concrete example, suppose that multiple persons are present in an office environment where they move around and perform activities like working, chatting, or preparing coffee, and suppose that the environment is equipped with various sensors (e.g. presence sensors). We are interested in estimating the persons' fine-grained *activities* (e.g. walking, typing) and the environmental context state (e.g. location of persons and objects) for each point in time. Probabilistic *Multiset Rewriting Systems* (MRSs) provide a convenient mechanism to represent such dynamic *multi-entity systems*.

In HAR, one established method for deriving state distributions from sequences of observations is *Bayesian filtering* (BF, also called recursive Bayesian state estimation). In BF, the goal is to iteratively compute the posterior $p(X_t \mid y_{1:t})$ for time t from the previous posterior $p(X_{t-1} \mid y_{1:t-1})$ at time $t-1$ and an observation y_t. In this paper, we investigate BF for systems whose *dynamics* $p(X_t \mid X_{t-1})$ is described by a probabilistic MRS (specifically, we use a general MRS formalism where multiple actions can occur simultaneously (Barbuti et al., 2011), as introduced in Section 2). Using such symbolic probabilistic models of dynamic systems provides several advantages over connectionist systems, like deep neural networks (Wang et al., 2019): Models can be constructed based on domain knowledge instead of extensive amounts of training data, the same symbolic model can be used for different sensor modalities, and context information can be inferred directly (Chen et al., 2012).

Unfortunately, a system whose dynamics is represented by an MRS can easily have a very large number of discrete states: For the person tracking domain, when there are n agents and m rooms, there are already n^m system states only for describing the location of each person. When the multisets x_t are treated as a single, discrete random variable, the categorical distribution $p(X_t \mid y_{1:t})$ has a large support (i.e. many assignments with non-zero probability), and thus, BF in such systems is computationally very expensive.

However, the system states often have a symmetrical structure: For example, in the person tracking scenario above, we might not always be able to discriminate the persons based on the sensor data, such that all states that are identical with respect to the *number* of persons per room are assigned the same probability. Thus, the joint probability of persons' names will be exchangeable. A number of algorithms that exploit such exchangeability for efficient inference have been devised (known as *lifted probabilistic inference*, e.g. Niepert & Van den Broeck, 2014; Poole, 2003). Unfortunately, existing lifted inference methods cannot be applied directly: Naively, the multiset state x is treated as an atomic random variable, and the resulting univariate, categorical distribution does not exhibit any structure that can be exploited. Furthermore, existing lifted inference algorithms have not been devised for BF tasks where the system dynamics is described symbolically, as in MRSs.

The central technical contribution of this paper is to introduce a suitable decomposition of multisets x into a *multiset structure* s and a *value tuple* $\mathbf{v}$, such that distributions over x can be defined via distributions over s and $\mathbf{v}$ (Section 4). This decomposition directly leads to several desirable properties of the distributions. For the distibution over tuples $\mathbf{v}$, standard mechanisms for efficient representations of distributions can be used: The distribution can be *factorized* to exploit independence, and factors can sometimes be represented parametrically (this is an instance of Rao-Blackwellization, Doucet et al., 2000). When a factor is exchangeable (which naturally occurs due to the fact that the value sequence represents *multisets*, where entities are not ordered), it can be represented by sufficient statistics (Niepert & Van den Broeck, 2014). In the remaining multiset structure, entities whose values follow the same distribution are grouped together, and thus the number of multiset structures is substantially smaller than the number of original multisets. Due to the relationship to lifted inference, we call this more efficient representation the *lifted* representation.

As shown below, the prediction and update steps of BF can be performed directly on the lifted representation, without resorting to the original, much larger *ground representation* (shown in Section 5). When the system dynamics breaks the exchangeability, the state representation is automatically adapted by *splitting* operations.

We empirically show on three different application domains that in the best case (when the distribution is fully exchangeable), our approach leads to a factorial reduction in representational complexity, in comparison to complete enumeration (Section 6). Additionally, we present an approximation strategy for this approach, which allows efficient inference even in cases where exact exchangeability is not present, or vanishes over time. We empirically show that in the approximate case, the lifted representation leads to a lower variance of the estimate, and a lower estimation error.

To summarize our contributions, we show (i) how to provide BF for systems with MRS dynamics, (ii) devise a lifted representation of distributions over multisets, and (iii) show how to perform prediction and update directly on that lifted representation.

2. Introduction to Multiset Rewriting Systems

In the following, we provide an introduction to Multiset Rewriting Systems (MRSs). MRSs are mainly investigated in two research areas: As a formalism for modeling and simulation of, for example, cell-biological systems (Danos et al., 2007; Faeder et al., 2009; John et al., 2011), and as a theoretical model of computation (in a research area called *membrane computing*, Paun, 2012). We start by providing the basic concepts of MRS in Section 2.1, then continue by presenting several extensions of MRS that have been proposed in either of the two research communities, which are necessary for using MRS in a BF context: *Structured entities* (which allow more flexibility and expressiveness) are introduced in Section 2.2, *maximally parallel* MRS (where multiple actions can be performed in parallel) are introduced in Section 2.3, and in Section 2.4, probabilistic maximally parallel MRS (PMPMRS) are presented, which explicitly model *distributions* over possible actions.

2.1 Multiset Rewriting Systems

In MRSs, states of a dynamic system are multisets, describing which "things" (*entities*[1]) and how many of them exist at a specific point in time.

Definition 1 (Multiset)**.** Let $\mathcal{E}$ be a set of *entities.* A multiset $x \in \mathcal{X}$ (over $\mathcal{E}$) is a map (partial function) $x : \mathcal{E} \nrightarrow \mathbb{N}$ from entities to natural numbers (called *multiplicities* in this context). ○

For entities $e_1, \ldots, e_k \in \mathcal{E}$ and multiplicities $n_1, \ldots, n_k \in \mathbb{N}$, we write $[\![\, n_1 e_1, \ldots, n_k e_k \,]\!]$ to denote a multiset, where the multiplicity of e_i is n_i and the multiplicities of all entities not listed is zero. We write $x\#e$ to denote the multiplicity of e in x. Let x, x' be two multisets. We assume that multiset union $x \uplus x'$, multiset difference $x \,\dot{\cup}\, x'$, and multiset subset relation $x \sqsubseteq x'$ are defined as usual (Blizard et al., 1988). A multiset represents a state of the dynamic system we are considering. We call such a multiset x a *ground state.*

The system dynamics is described by *rewriting rules*, or *actions*[2]. For now, it is sufficient to consider actions as triples $(l, r, \kappa) \in \mathcal{X} \times \mathcal{X} \times \mathbb{R}_{\geq 0}$, where l and r are multisets called *reactands* and *products*, respectively, and κ is the kinetic constant, or *weight* (that is later used for defining probabilities of actions). An action $a = (l, r, \kappa)$ is *compatible* to a state x when the reactands are contained in x, i.e. $l \sqsubseteq x$, and is *applied* to x by removing the reactands, and adding the products: $x' = (x \,\dot{\cup}\, l) \uplus r$. We denote actions $a = (l, r, \kappa)$ as

[1] Note that we use the term *entity* to refer both to a specific type of object (this is typically called a *species* in the MRS community), as well as specific instances of that object (e.g. the elements occurring in a multiset), as such a distinction is not relevant for our purposes.

[2] In the planning community, a rewriting rule would be called *action schema*.

$l \xrightarrow{\kappa} r$. An MRS is a triple $(\mathcal{E}, A, x_0)$, where $\mathcal{E}$ is the set of entities, A is the set of actions, and x_0 is the initial state.

Example 1. Consider the MRS $(\mathcal{E}, A, x_0)$ with $\mathcal{E} = \{A, B\}$, $A = \{[\![\, 1X \,]\!] \xrightarrow{\kappa_1} [\![\ ]\!], [\![\, 1X, 1Y \,]\!] \xrightarrow{\kappa_2} [\![\, 2X \,]\!], [\![\, 2Y \,]\!] \xrightarrow{\kappa_3} [\![\, 3Y \,]\!]\}$ and $x_0 = [\![\, 1X, 1Y \,]\!]$, which models a simple predator-prey system where X are predators and Y are prey, κ_1 is the rate of death of predators, κ_2 is the consumption rate, and κ_3 is the reproduction rate of prey. The initial state x_0 has the successor states $[\![\, 2X \,]\!]$ (by applying the first action) and $[\![\, 1Y \,]\!]$ (by applying the second action). The third action is not compatible to x_0 (there are not enough Y entities in x_0). ○

Using this formalism, it is straightforward to *sample* a trajectory (i.e. a sequence of states) as follows: Given a state, test which actions are compatible, select one of the actions with probability that is proportional to its weight, compute the successor state, and repeat. By repeatedly sampling trajectories, insights about properties of the system can be obtained (e.g. by using the stochastic simulation algrithm, Gillespie, 1977). Note that we are only discussing *discrete-time* MRSs here, as only this type of MRS is relevant later on for BF (where we assume that observations are obtained at discrete – typically fixed – time intervals).

2.2 MRS with Structured Entities

The simple formalism described above is already powerful and can model a number of interesting systems. However, there are many domains where entities are actually a collection of of multiple *properties*. For example, suppose we want to model a multi-agent activity recognition scenario, consisting of agents that can move around multiple rooms, and agents can pick up and manipulate objects. In this case, each combination of properties of an agent (name, location, handled objects, current goal, ...) has to be modeled as a separate entity. This does not only result in a large set $\mathcal{E}$ of entities, but also in a large set A of actions, as separate actions are necessary for each of the entities. Properties with *continuous* domains lead to an infinite number of actions, making representation and simulation of the MRS complicated.

A more elegant way to model such situations is to explicitly encode the structure of the entities, and adapt the definitions of actions so that they can work directly with such structured entities. Entities with attributes and corresponding schematic actions can be found in many rule-based languages in systems biology, e.g. in the systems by Danos et al. (2007), Faeder et al. (2009), John et al. (2011).

Here, we describe a generalization of those approaches: Instead of fixing the number of properties per entity type in advance (as usual for those approaches), we allow changes in the number of properties. Furthermore, entities do not need to be identified via a name, which makes the approach flexible and expressive. Specifically, we assume that entities are *property-value maps*, and use a constraint-based mechanism for describing possible reactands of actions.

Definition 2 (Entity)**.** Let $\mathcal{P}$ and $\mathcal{V}$ be two sets. We call elements from $\mathcal{P}$ *property names* and elements from $\mathcal{V}$ *values*. An *entity* $e \in \mathcal{E}$ is a partial function $e: \mathcal{P} \nrightarrow \mathcal{V}$, i.e. a map of property names $\mathcal{P}$ to values $\mathcal{V}$. ○

The following example illustrates how structured entities are used to define the state in complex domains[3].

Example 2. We are modeling a person tracking and activity recognition task (Schröder et al., 2017): Multiple persons move in an office environment. Each agent is characterized by a *name* and their current location (other aspects, like objects in the environment that can be picked up by the agents, and so on are not modeled in this simple example). Suppose there are two locations "Door" and "Table", and three agents "Alice" (A), "Bob" (B) and "Charlie" (C). Let $\mathcal{P} = \{\text{Loc}, \text{Name}\}$ and $\mathcal{V} = \{\mathit{Door}, \mathit{Table}, A, B, C\}$. A state of the system where two agents are at the door, and one is at the table, is described by the following multiset:

$$x = [\![\, 1\langle \text{Name: } A, \text{Loc: } \mathit{Table}\rangle, 1\langle \text{Name: } B, \text{Loc: } \mathit{Door}\rangle, 1\langle \text{Name: } C, \text{Loc: } \mathit{Door}\rangle \,]\!] \qquad \circ$$

Note that neither $\mathcal{P}$ nor $\mathcal{V}$ needs to be finite. For example, the location could be described by elements from $\mathbb{R}^2$. When either $\mathcal{P}$ or $\mathcal{V}$ is infinite, the set $\mathcal{E}$ of entity types is also infinite. To efficiently handle such structured entities, we define actions in terms of *preconditions* (that describe which constraints a structured entity must satisfy such that the action can be applied) and *effects* (that describe how the state changes, with respect to the entities that are used for satisfying the preconditions).

Definition 3 (Action)**.** Let $c \in \mathcal{C}$ be a sequence of boolean functions of entities, i.e. $c : \langle \mathcal{E} \to \{true, false\}\rangle$, called *preconditions*, and let $f \in \mathcal{F}$ be a function that manipulates a state, given a sequence of entities, i.e. $\mathcal{F} := \langle \mathcal{E} \rangle \times \mathcal{X} \to \mathcal{X}$. (called *effect*). The *weight* κ of an action is a positive real number. An *action* $a \in \mathcal{A}$ is a triple $a = (c, f, \kappa) \in \mathcal{C} \times \mathcal{F} \times \mathbb{R}_{>0}$. $\circ$

Before we can given an example of an action, we need to introduce some convenient notation for some *simple* effects: Replacing a property value with a new value, adding (or overwriting) a property-value pair to an entity, and adding an entity to the state.

- Consider an effect f that appends the property-value pair (k, v) to an entity e, or, if a property k is already present in e, overwrites the value of k to v, i.e.

$$f(\langle \dots, e, \dots \rangle, x) = (x \uplus [\![\, 1e \,]\!]) \uplus [\![\, 1e' \,]\!], \text{ where}$$
$$e' = e \oplus \langle \text{k: } v\rangle$$

  We will denote such an effect as "$e(k) \leftarrow v$".
- Consider an effect f that adds an entity e^* to the state, i.e. $f(e, x) = x \uplus [\![\, 1e^* \,]\!]$. We denote such an effect as "$+e^*$".
- The *composition* of two effects f_1 and f_2 is defined as $(f_1 \circ f_2)(i_1 \oplus i_2, x) = f_1(i_1, f_2(i_2, x))$.

[3] We use $f = \langle k_1 \colon v_1, \dots, k_n \colon v_n\rangle$ to denote the partial function f where $f(k_1) = v_1, \dots, f(k_n) = v_n$. Sequences (partial functions of indices to elements) are written as $\langle x_1, x_2, \dots\rangle$, to denote the sequence that has x_1 at position 1, x_2 at position 2 and so on. See Appendix A for a complete overview of the notation.

Example 3. In the office domain (Example 2), agents can move between locations. One of the actions – that describes movement between the door and the table – is the action *move-d-t* $= (c, f, \kappa)$, that is defined as follows[4]:

$$\begin{aligned} c(e) &= (e(\mathit{Loc}) == \mathit{Door}) \\ f(\langle e \rangle, x) &= e(\mathit{Loc}) \leftarrow \mathit{Table} \end{aligned} \qquad \circ$$

To apply an action a to a state x, the action is *instantiated*: For each precondition of a, an entity from x is selected that satisfies that precondition. The effect then manipulates the state based on these entities – they are used as parameters of the effect function. We call such a pair of action and a sequence of entities an *action instance.*

Definition 4 (Action Instance)**.** An *action instance* is a pair $(a, i) \in \mathcal{A} \times \langle \mathcal{E} \rangle$ where $a = (c, f, \kappa)$ is an action and i is a sequence of entities. An action instance is *compatible* to a state x if the following conditions hold:
(i) There is a corresponding entity for each constraint, i.e. $|i| = |c|$.
(ii) Each precondition in c is satisfied by its corresponding entity. That is, $\forall j\ : i_j \models c_j$.
(iii) The multiset of the entities in e is contained in x, i.e. $\text{items}(i)^5 \sqsubseteq x$.
An action instance $alpha = ((c, f, \kappa), e)$ is applied to a state x by applying the effect function to the state and the bound entities, i.e. $x' = f(e, x)$. ○

It is important to note that preconditions and bound entities have a sequential order (instead of being a multiset, as before), and thus the effect can depend on which entity is bound to which *position.* For example, consider an action *eats* with effect $f(\langle e_1, e_2 \rangle, x) = x \uplus e_2$. In this case, it obviously makes a difference in which order the entities are bound to the preconditions, as this defines *which* of the entities gets eaten by the other one.

Example 4. Consider the state

$$x = [\![\, 1e_A, 1e_B, 1e_C \,]\!],$$

where $e_A = \langle \text{Name: } A, \text{Loc: } \mathit{Table} \rangle$, $e_B = \langle \text{Name: } B, \text{Loc: } \mathit{Door} \rangle$ and $e_C = \langle \text{Name: } C,$ Loc: $\mathit{Door} \rangle$. The action *move-d-t* from Example 3 can be applied to all entities where $\mathit{Loc} == \mathit{Door}$, and thus, the state x has two compatible action instances

$$\begin{aligned} \alpha_1 &= (\text{move-d-t}, e_B), \\ \alpha_2 &= (\text{move-d-t}, e_C). \end{aligned}$$

Applying these action instances leads to successor states x_1 or x_2, where

$$\begin{aligned} x_1 &= [\![\, 1e_A, 1e'_B, 1e_C \,]\!], \\ x_2 &= [\![\, 1e_A, 1e_B, 1e'_C \,]\!] \end{aligned}$$

with $e'_B = \langle \text{Name: } B, \text{Loc: } \mathit{Table} \rangle$ and $e'_C = \langle \text{Name: } C, \text{Loc: } \mathit{Table} \rangle$. ○

[4] Note that the expression $e(\mathit{Loc}) == \mathit{Door}$ is a Boolean expression in the constraint language (in this case, an equality constraint).

[5] The function items(i) returns the multiset of elements in the sequence i, in which each element a appears exactly as often as a apprears in i

The set of all action instances of an action that are compatible with a given state can be computed by backtracking. Usually (for the scenarios we are concerned with), the number of *different* entities (i.e. the number of species), and thus the number of action instances of each action will be small, such that enumeration of all action instances is feasible.

2.3 Maximally Parallel MRS

We are interested in modeling systems where between observations, more than one action can be applied. Formally, this is expressed by *maximally parallel* multiset rewriting systems (MPMRS) (Barbuti et al., 2011). In such systems, each state transition consists of a parallel execution of a multiset of action instances, called *compound action*. Such maximally parallel rule application is also typically used in P systems (Paun, 2012) (a model of computation based on MRS), motivated by the fact that in cell-biological systems, multiple reactands can interact at the same time.

Definition 5 (Applicable and Maximal Compound Action (AMCA))**.** A *compound action* $k \in \mathcal{K}$ is a multiset of action instances, i.e. $k \in [\![\, \mathcal{A} \times \langle \mathcal{E} \rangle \,]\!]$. We call a compound action *applicable* to a state x if each action instance is compatible with x, and the multiset of all bindings of the action instances is contained in x, i.e. $\biguplus_{(a,i)\in k} \text{items}(i) \sqsubseteq x$ (each entity in a state is bound at most once). We call a compound action k *maximal* with respect to a state x if no action can be added to k such that the resulting compound action is still applicable to x. The set of applicable and maximal compound actions (AMCAs) of a state x is denoted as K_x. ○

In the following, we are mostly concerned with the applicable and maximal compound actions (AMCAs). The effect of a compound action is the composition of the individual action instances' effects. As the order of the individual actions of a compound action is arbitrary, we require that the order in which the effects are applied can also be arbitrary, i.e. the individual effects must be commutative. The *simple* effects introduced above are always commutative (given that they cannot operate on the same entity, which is the case when the compound action is *applicable*).

Definition 6 (Compound Action Effect, Successor State)**.** Let k be a compound action, and let the effects of all actions in k be commutative. The *effect* of a compound action is the composition of all individual action instances' effects, i.e.

$$f_k = \bigcirc_{((c,f,\kappa),i)\in k} f$$

We call a state $x' = f_k(x)$ a *successor state* of x. ○

Example 5. Consider the state

$$x = [\![\, 2\langle \text{Name: } A, \text{Loc: } \textit{Table}\rangle, 1\langle \text{Name: } B, \text{Loc: } \textit{Door}\rangle \,]\!]$$

and the two actions *move-d-t* (a_m), as defined in Example 4) and *stay* (a_s), which has a precondition that is always true, and its effect is the identity. For readability, we denote $e_A = \langle \text{Name: } A, \text{Loc: } \textit{Table}\rangle$ and $e_B = \langle \text{Name: } B, \text{Loc: } \textit{Door}\rangle$. There is just a single

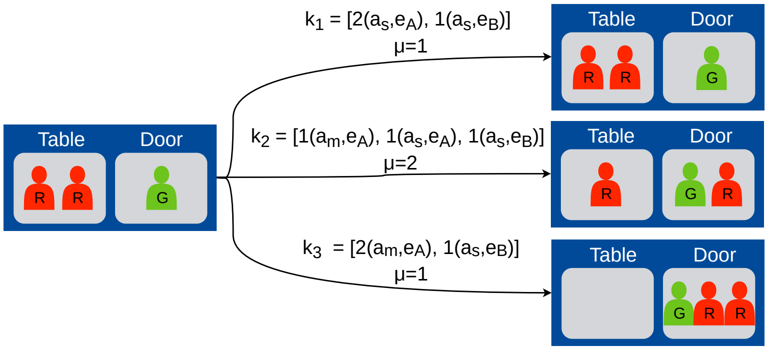


Table
Door
R
R
G
k1 = [2(as,eA), 1(as,eB)]
μ=1
k2 = [1(am,eA), 1(as,eA), 1(as,eB)]
μ=2
k3 = [2(am,eA), 1(as,eB)]
μ=1
Table
Door
R
R
G
Table
Door
R
G
R
Table
Door
G
R
R

Figure 1: Compound actions and successor states for Example 5: Either both red entities (entities with name A) stay at their location, one of them moves, or both move. The compound action k_2, where one entity moves, has multiplicity 2, as either of the two entities could have moved.

compatible action instance for a_m: (a_m, e_B). For a_s, there are two compatible action instances: (a_s, e_A) and (a_s, e_B). These three action instances allow for three AMCAs: $k_1 = \llbracket\, 2(a_s, e_A), 1(a_s, e_B) \,\rrbracket$, $k_2 = \llbracket\, 1(a_m, e_A), 1(a_s, e_A), 1(a_s, e_B) \,\rrbracket$, and $k_3 = \llbracket\, 2(a_m, e_A), 1(a_s, e_B) \,\rrbracket$ – either both agents at the table move, just one of them moves or both stay where they are. The situation is shown in Figure 1. ○

Note that maximally parallel MRSs are a strict generalization of conventional MRSs (where a single action is executed at each step): Single-action application can be modeled in a maximally parallel MRS by introducing an additional *mutex* entity to the state that is required by each action. Then, each AMCA has a cardinality of one.

2.4 Probabilistic Maximally Parallel MRS

In general, given a state x, more than one compound action can be applicable and maximal, as shown in Example 5. Thus, a mechanism is necessary to decide which of those AMCAs will be chosen. For example, classical P systems (Paun, 2012) use a given priority relation of actions for this task. Here, we are considering *probabilistic* systems, for which we need to define some mechanism that assigns probabilities to compound actions (and thus a probabilistic transition model) – similar to the way probabilities have been devised for single-action MRSs in Section 2.1.

Note that up front, *any* function from the AMCAs to positive real numbers which integrates to one is a valid definition of these probabilities. Each definition might be plausible for different domains, depending on the underlying "physics" (i.e. action selection mechanism): A world where entities can independently choose which action to participate in requires a different definition of AMCA probabilities than a world where entities *cooperate* to reach a common goal.

Here, we use the probabilities that arise when entities choose which action to participate in without coordinating. This is the intended semantics for the multi-agent scenarios we are concerned with. To calculate this probability, we count the number of ways specific entities from a state x can be chosen to be assigned to the action instances in the compound action.

This concept is closely related to the MPMRS of Barbuti et al. (2011) – except that due to the fact that we use positional preconditions, the counting process is slightly different. We start by defining the *multiplicity of an action instance* α as the number of ways the bindings of α can be chosen from the entities of a state x.

Definition 7 (Multiplicity of an Action Instance)**.** Let α be an action instance, let $i = \langle i_1, \dots, i_n \rangle$ be the entities bound in α, and let x be a state. The multiplicity μ of i in x is

$$\mu(i = \langle i_1, \dots, i_n \rangle, x) = \begin{cases} x\#i_1 * \mu(\langle i_2, \dots, i_n \rangle, x \uplus [\![\, 1 i_1 \,]\!]) & \text{if } |i| > 0 \\ 1 & \text{otherwise} \end{cases}$$

Furthermore, let the multiplicity $\mu_x(\alpha)$ of action instance $\alpha = (a, i)$ in x be $\mu_x(\alpha) = \mu(i, x)$. ○

The multiplicity of a compound action k in state x is in principle just the product of the component action instances' multiplicities regarding the corresponding remaining state. However, when doing so, the multiplicity of the compound action is overestimated: For example, consider the state $x = [\![\, 2y \,]\!]$ and the compound action $k = [\![\, 2(a, y) \,]\!]$. Although the multiplicity $\mu_x(a, y)$ is 2, the multiplicity of k should be 1, as there is only a single way to assign the entities in x to the compound action, as the order of the action instances in the compound action is not relevant. To obtain the correct multiplicity, we therefore need to divide it by the number of permutations of identical action instances.

Definition 8 (Multiplicity of a Compound Action)**.** Let k be a compound action, and let x be a state. The uncorrected multiplicity μ' of k in x is

$$\mu'(k, x) = \begin{cases} 1 & \text{if } k = [\![\;]\!] \\ \mu_x(\alpha) * \mu'(k \uplus [\![\, 1\alpha \,]\!], x \uplus \text{items}(i)), \text{ where } \alpha = (a, i) \in \text{dom}(k) & \text{otherwise} \end{cases}$$

The multiplicity μ of k in x is obtained by dividing μ' by the product of the number of permutations of identical action instances in k:

$$\mu_x(k) = \frac{\mu'(k, x)}{z(k)}, \text{ where } z(k) = \prod_{\alpha \in \text{dom}(k)} k\#\alpha!$$ ○

Note the close relationship to the multinomial coefficient $\binom{n}{k\#\alpha_1,\ k\#\alpha_2, \dots}$. Finally, the probability of a compound action is its normalized multiplicity, multiplied by the product of the individual actions' weights.

Definition 9 (Weight, Probability of an AMCA)**.** Let K_x be the set of AMCAs of the state x. The *weight* $v_x(k)$ of an AMCA $k_i \in K_x$ is

$$v_x(k_i) = \mu_x(k_i) \prod_{\alpha \in \text{dom}(k_i)} \kappa_a^{k_i\#\alpha}. \tag{1}$$

The probability of an AMCA $k_i \in K$, given x, is its normalized weight:

$$p(k \mid x) = \frac{v_x(k)}{Z}, \text{ where } Z = \sum_{k_i \in K} v_x(k_i) \tag{2}$$

○

The AMCAs shown in Example 5 have multiplicities $\mu_x(k_1) = \mu_x(k_3) = 1$ and $\mu_x(k_2) = 2$ (see Figure 1). Thus, assuming that the actions a_m and a_s have equal weight, the probabilities of the AMCAs are $p(k_1 \mid x) = p(k_3 \mid x) = 0.25$ and $p(k_2 \mid x) = 0.5$.

Computing the weight of an AMCA is closely related to the weighted model counting problem (WMC) (Chavira & Darwiche, 2008): In WMC, we are given a propositional theory Δ and a weight for each literal (which induce a weight of each model). The goal is to compute the summed weight of all models that satisfy Δ. Here, the conjunction of all constraints of the actions (as well as the other requirements of AMCAs, see Definition 5) define a propositional theory, and each AMCA is a model of that theory. In this case, the weight of each AMCA can be computed directly via Equation 1.

To be able to sample from the distribution of AMCAs (Equation 2), we do not only need to compute the WMC for a single AMCA k, but for *all* AMCAs (to obtain the normalization factor Z) – in WMC terminology, we need to first *generate* all theories that have a non-zero model count, and then perform WMC for each theory. An algorithm for generating all AMCAs that has linear runtime in the number of AMCAs is described in Appendix B. Using this algorithm, it is easy to draw sample trajectories: Given a state x, calculate all compound actions and their probabilities, sample one of them, compute the successor state, and iterate the process.

As the number of AMCAs can easily become very large (see Appendix B), enumerating all compound actions can be computationally expensive. An approximate, MCMC-based algorithm that samples AMCAs directly has been presented by Lüdtke et al. (2018b).

3. Bayesian Filtering in Multiset Rewriting Systems

As described above, MRSs are typically used for *simulation*, i.e. sampling trajectories. Instead, in this paper, the goal is *Bayesian filtering* (BF) in MRS: Estimate the state of a system that is described by a MRS, given a sequence of noisy observations.

First, we provide a brief introduction to BF in Section 3.1, then describe how the prediction (Section 3.2) and update (Section 3.3) steps of BF can be realized for probabilistic maximally parallel MRS (PMPMRS), and then describe why naively proceeding like this is infeasible (Section 3.4), which leads to the need for a more efficient representation.

3.1 Bayesian Filtering

The goal of BF is to estimate the (hidden) state sequence $x_{1:t}$, based on a sequence of noisy observations $y_{1:t}$. Usually, this is done in an iterative process that, given the *prior distribution* $p(X_t \mid y_t)$, calculates the posterior distribution $p(X_{t+1} \mid y_{t+1})$. This computation can be decomposed into the *prediction step*

$$p(X_{t+1} \mid y_{1:t}) = \sum_{x_t \in \mathcal{X}} p(x_t \mid y_{1:t})\, p(X_{t+1} \mid x_t) \tag{3}$$

and the *update* step

$$p(X_{t+1} \mid y_{1:t+1}) = \frac{p(y_{t+1} \mid X_{t+1})\, p(X_{t+1} \mid y_{1:t})}{p(y_{t+1} \mid y_{1:t})}. \tag{4}$$

Algorithm 1 Prediction.

```
• Input: Actions $A$, categorical distribution $p(X_t|y_{1:t})$, represented as $\{(x_t^{(i)}, p_t^{(i)}\}_{i=1}^N$
• For $i = 1, \ldots, N$
    – Let $AI_x = \textsc{enum-ai}(x, A)$ be the action instances of $x_t^{(i)}$
    – Let $K_x = \textsc{enum-ca}(x, AI_x)$ be the AMCAs of $x_t^{(i)}$ (see Algorithm 6)
    – Compute successor states and multiply prior:
      Let $P_i = \left\{ \left( f_k(x_t^{(i)}), p_t^{(i)} * p(k \mid x_t^{(i)}) \right) \mid k \in K_x \right\}$
• Let $P = \bigcup_{i=1}^N P_i$
• Let $x_{t+1}^{(1)}, \ldots, x_{t+1}^{(M)}$ be the set of unique states in P
• Marginalize prior, by summing weights of identical states:
  For $i = 1, \ldots, M$: $p_{t+1}^{(i)} = \sum_{\{(x,p) \in P \mid x = x_{t+1}^{(i)}\}} p$
• Return $\{(x_{t+1}^{(i)}, p_{t+1}^{(i)}\}_{i=1}^M$
```

We call $p(X_{t+1} \mid X_t)$ *transition model*, $p(Y_{t+1} \mid X_{t+1})$ *observation model* and $p(X_t \mid y_{1:t})$ the *filtering distribution* at time t. Common approaches to perform this recursive estimation are Hidden Markov Models (where states are categorical, the state space is finite, and the transition model is represented as a matrix of transition probabilities) and Particle Filters (which represent the filtering distribution by weighted samples).

BF is a fundamental task in artificial intelligence, that arises, for example, in speech recognition (Rabiner, 1989), handwriting recognition (Plötz & Fink, 2009), gene prediction (Stanke & Waack, 2003), robot localization and mapping (Montemerlo et al., 2002), or Human Activity Recognition (Wilson & Atkeson, 2005). Here, we are concerned with the case where the states x_t are *multisets*, and the transition model $p(X_{t+1} \mid X_t)$ is given by a PMPMRS. This task arises, for example, in sensor-based Human Activity Recognition for multiple agents, or when the state of multiple *objects* needs to be estimated.

3.2 Prediction

The distribution of AMCAs $p(k \mid x_t)$ (Equation 2) directly induces a transition model $p(x_{t+1} \mid x_t)$: Intuitively, the probability of a posterior state x_{t+1} is the summed probability of all AMCAs that lead to x_{t+1}. More formally, the distribution is computed by

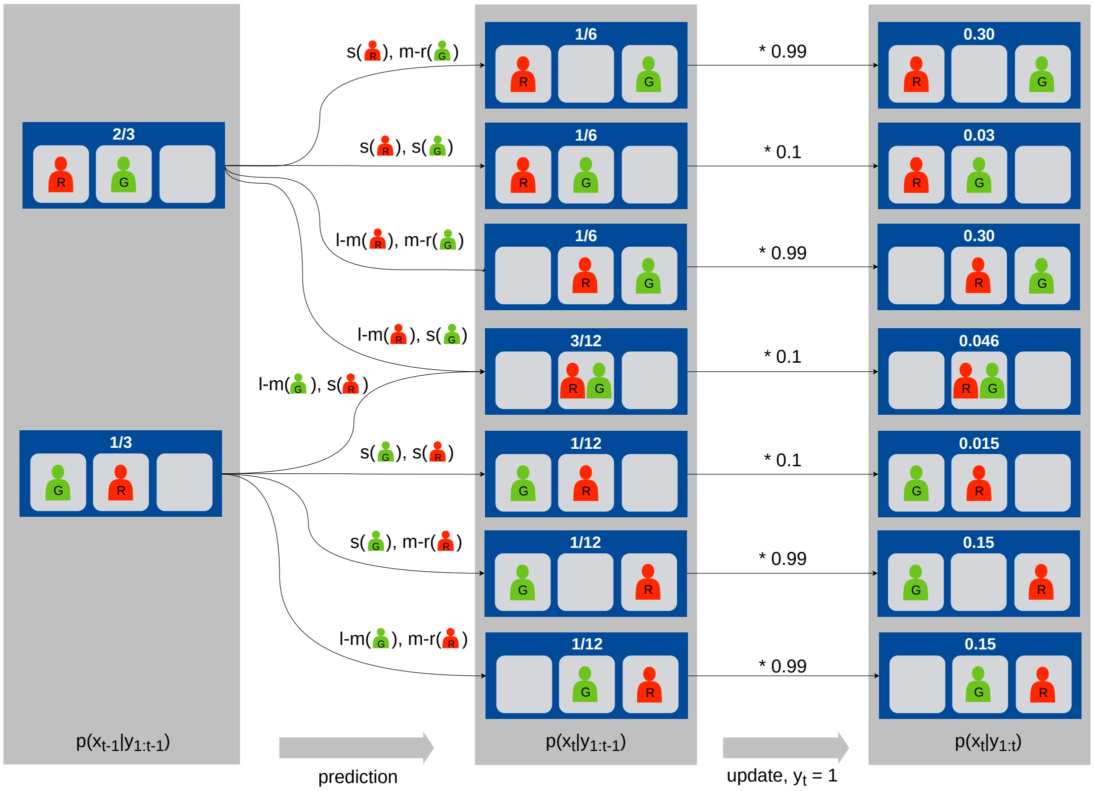


Figure 2: Example of prediction and update for the office scenario (see Examples 6 and 7).

marginalizing over the AMCAs of x_t as follows:

$$\begin{aligned} p(x_{t+1} \mid x_t) &= \sum_{k \in K_{x_t}} p(x_{t+1}, k \mid x_t) \\ &= \sum_{k \in K_{x_t}} p(x_{t+1} \mid x_t, k)\, p(k \mid x_t) \\ &= \sum_{k \in K_{x_t}} \mathbb{1}(f_k(x_t) = x_{t+1})\, p(k \mid x_t). \end{aligned} \tag{5}$$

The filtering distribution $p(X_t \mid y_{1:t})$ is a *categorical* distribution in this case, i.e. we maintain a set of pairs (x, p_x), where x is a possible state, and p_x is its probability (we only need to store the states which have non-zero probability). Algorithm 1 shows how a prediction step of BF (Equation 3) is performed in this approach, given a set of actions A, and a prior state distribution $p(X_t \mid y_{1:t})$ as a set of pairs $\{(x_t^{(i)}, p_t^{(i)})\}_{i=1}^N$): For each state $x_t(i)$, we enumerate all action instances and AMCAs, compute their probabilities (Equation 2), compute successor states and their probabilities (Equation 5), and finally perform the prediction by multiplying with the prior and marginalizing over X_t (Equation 3).

Example 6. Consider the following variation of the office scenario: There are three rooms (L, M, R), and two agents (red and green). Agents can move between adjacent rooms or stay at their current room (i.e. there are five actions: l-m, m-l, m-r, r-m, s). Suppose that at some point during BF, the filtering distribution looks as follows:

$$\begin{aligned}
&p(x_1) = 1/3,\ p(x_2) = 2/3,\\
&x_1 = [\![\, 1\langle \text{N: Red}, \text{L: L}\rangle, 1\langle \text{N: Green}, \text{L: M}\rangle \,]\!]\\
&x_2 = [\![\, 1\langle \text{N: Red}, \text{L: M}\rangle, 1\langle \text{N: Green}, \text{L: L}\rangle \,]\!]
\end{aligned}$$

Each state has four AMCAs: Either no agent moves, the red agent moves, the green agent moves, or both move. Suppose that all actions have equal weight, which leads to all AMCAs having equal weight. After applying the AMCAs and summing the probability of identical predicted states, there are 7 states with non-zero probability, as shown in Figure 2 (center column). ○

3.3 Update

As we saw above, PMPMRSs directly induce a transition model $p(x_t \mid x_{t-1})$. However, existing MRS formalisms do not consider observations, and therefore, new concepts are required to specify observations models for MRSs. Such observation models have to account for two considerations:

- When the states x_t are multisets, defining the distribution $p(Y_t \mid X_t)$ is not straightforward, as the domain of X_t will typically be very large, and thus storing one distribution over Y_t for each possible value of X_t is infeasible.

- For MRSs, it is natural to allow observations that to not depend on any *specific* entity, but rather on *features* of the state x_t, like existence of entities with certain properties. For example, a room presence sensor might observe whether *any* person is at a specific location, but not *which* one. Such *non-identifying* observations will also turn out to be convenient later when using a more efficient representation for the filtering distribution, because they do not break the symmetry of the entities.

In the following, we describe a *constraint-based* formalism that accounts for both of these considerations. The idea is to partition the states into groups that behave identical with respect to observations (i.e. that are weighted with the same likelihood). These partitions are defined via features of the states.

Here, we use *counting constraints* on states x to express this feature selection: These constraints are true when there are exactly n (at least n, at most n) entities in x that satisfy an entity constraint c. Counting constraints are used here because of their flexibility, i.e. they allow to model a wide range of possible observation situations.

Counting constraints where exactly n, at least n or at most n entities in x satisfy an entity constraint c are denoted by $\#_c^{=n}$, $\#_c^{\geq n}$, and $\#_c^{\leq n}$. The observation distribution $p(y \mid x)$ then depends on which counting constraint can be satisfied in x (and of course on the value of y). The constraints are constructed in such a way that exactly a single constraint is satisfied in any x, given y. This way, the constraints *partition* the state space, i.e. each constraint represents a different characteristic of the state with unique distribution of observations.

Definition 10 (Observation Model)**.** Let $c \in \mathcal{CN} = \mathcal{X} \to \{true, false\}$ be a counting constraint, and let $y \in \mathcal{Y}$ be an *observation*. We call $o \in \mathcal{Y} \times \mathcal{CN} \to \mathbb{R}$ *observation model*, if for any state x and observation y, exactly a single constraint c with $(y, c) \in \mathrm{dom}(o)$ can be satisfied in x. An observation model o specifies the distribution $p(y \,|\, x)$ of observing y in state x by $p(y \,|\, x) = o(y, c)$, where c is the constraint satisfied in x. ○

Example 7. Suppose that one of the rooms in the office is equipped with a presence sensor that is active when at least one agent is in the room. The sensors have a false positive rate of 0.1 and a false negative rate of 0.01. Specifically, we assume that the right location is equipped with such a sensor. Thus, the observation distribution looks as follows:

$$p(y = 1 \,|\, x) = \begin{cases} 0.99 & \text{if at least one agent in } x \text{ is at right location} \\ 0.1 & \text{otherwise} \end{cases}$$

This observation model can be formalized using the constraint-based formalism: The entity constraint $c(e) = (e(Loc) == \mathrm{Right})$ tests whether the agent e is at the right location, and the counting constraints $\#_c^{=0}$ and $\#_c^{\geq 1}$ test whether none (or at least one) agent is at the right location in a state x. Thus, the observation model is given by:

y	c	p
0	$\#_c^{=0}$	0.9
1	$\#_c^{=0}$	0.1
0	$\#_c^{\geq 1}$	0.01
1	$\#_c^{\geq 1}$	0.99

Consider the prediction distribution shown in Figure 2 (middle), and suppose that we observe $y = 1$. Each state where at least one entity is at the right location is weighted with 0.99, and each state where no entity is at the right location is weigh with 0.1. Afterwards, the distribution is re-normalized. ○

3.4 Marginal Filtering and Problem Statement

We now have all components that we need for performing exact BF in systems with probabilistic maximally parallel MRS dynamics. Unfortunately, doing this turns out to be effectively infeasible: In MRS, the number of possible systems states (i.e. the support of the filtering distribution) is typically very large: When n is the total number of entities in a state, and m is the number of possible *different* entities, there are at most $\binom{m+n-1}{n} = \frac{(m+n-1)!}{n!\,(m-1)!}$ multisets. Therefore, exact BF quickly becomes infeasible due to the large number of pairs $(x_t^{(i)}, p_t^{(i)})$ that need to be maintained to represent the filtering distribution $p(x_t \,|\, y_{1:t})$.

Thus, the naive approach of representing $p(x_t \,|\, y_{1:t})$ by a set of pairs $(x_t^{(i)}, p_t^{(i)})$ is insufficient, and we require a more efficient way to represent that distribution. However, when the multisets x_t are treated as atomic terms, the filtering distribution $p(x_t \,|\, y_{1:t})$ is *categorical* and *univariate*, and thus cannot be factorized, nor can methods for efficiently representing continuous distributions (parametric representations, variational methods) be used directly. The classical approach in such cases is to employ a sampling-based approximation called *particle filter* (PF) (Arnaud et al., 2001), where the distribution is approximated by a set of weighted samples $(x_t^{(i)}, w_t^{(i)})$ (particles):

$$p(x_t) \approx \tilde{p}(x_t) = \sum_{i=1}^{N} w_t^{(i)} \, \delta_{x_t^{(i)}}. \tag{6}$$

However, particle filtering is unsuitable for large, categorical state spaces (Nyolt et al., 2015) (like the multiset states that need to be handled here): As no metric is defined on the categorical states, resampling (which needs to be done to prevent weight degeneration) leads to many duplicate particles that represent the same system state, instead of smoothly approximating the distribution around that state, as for continuous distributions. Thus, the filtering distribution is effectively represented by particle numbers instead of weights, and particles are utilized poorly.

To circumvent these problems, the *marginal filter* (MF) (Nyolt et al., 2015) has been proposed, that is specifically suited for categorical state spaces: The general idea is to directly collapse identical particles, by summing their weight. Specifically, in the marginal filter, the prediction step is performed exactly. That is, for each particle $(x_t^{(i)}, w_t^{(i)})$, the posterior distribution $p(x_{t+1} \mid x^{(i)})$ is computed by enumerating all posterior states x_{t+1} (which is possible in the MRS dynamics described here, as each state can only have a finite number of successor states). The algorithm then marginalizes the prior states x_t, by summing the probabilities of identical successor states. This can be realized efficiently by representing the filtering distribution $p(X_t \mid y_{1:t})$ as a *map* $\mathcal{X} \nrightarrow \mathbb{R}_{\geq 0}$ of states to probabilities: This way, each state is represented at most once in the MF, and map insertion can be redefined to directly perform the summation of probabilities.

Although this computation is exact, it is based on the approximation $\tilde{p}(x_t)$ from the previous time step. The approximation that is performed for each time step is to limit the number of particles that represent $p(x_{t+1})$ by an operation called *pruning* (see Nyolt & Kirste, 2015 for a discussion of pruning strategies). The MF provides the fundamental algorithmic idea to allow approximate inference in very large categorical state spaces. For example, the algorithm has been used for inference in a cooking activity recognition task with 146 million discrete states (Nyolt et al., 2015).

Still, MF can require a large number of particles for an accurate representation of the filtering distribution, especially when the true distribution has a large support (as typically the case for the PMPMRS we are concerned with). The main shortcoming of MF in MRS domains is that states are handled as atomic terms without considering their internal structure.

However, for multiset states, this is not the case. Intuitively, in MRSs, a certain symmetry in the states can arise due to the transition semantics, such that multiple states are identical except for permutation of values.

Example 8. From the system shown in Figure 2, consider the states

$$x_1 = \llbracket\, 1\langle \text{L: Left}, \text{N: Red}\rangle, 1\langle \text{L: Middle}, \text{N: Green}\rangle \,\rrbracket$$
$$x_2 = \llbracket\, 1\langle \text{L: Left}, \text{N: Green}\rangle, 1\langle \text{L: Middle}, \text{N: Red}\rangle \,\rrbracket.$$

The states are identical, except for the permutation of agent *names*. ○

If we could make use of these symmetries – by summarizing such similar states – it would be possible to reduce the space complexity. Fleshing out this idea is the main technical contribution of this paper, and the topic of the next section.

4. Factorizing Distributions of Multisets

In this section, we present the main technical contribution of this paper: An efficient representation of distributions $p(X)$ of $\mathcal{X}$-valued random variables (RVs), where $\mathcal{X}$ is the set of possible multisets over structured entities. Initially, as discussed above, multisets are atomic terms, thus $p(X)$ is a univariate, categorical distribution that does not directly afford any efficient representation. Our goal here is to *unfold* the structure that is present in the multisets into a form for which we can readily devise a distribution that can be represented efficiently. It is important to note that this task of describing a distribution over a complex data structure (like multiset states) via distributions over simpler objects (like tuples) is not straightforward: For example, Flach and Lachiche (2000) describe a distribution over sets and multisets, which requires to explicitly marginalize over the serializations of the set.

We propose to use a decomposition function ϕ (introduced in Section 4.1), that maps states (i.e. multisets) x to pairs $(s, \mathbf{v})$, where s is the *multiset structure* (which and how many entities exist), and $\mathbf{v}$ is a sequence of *values* of the entities. Then, we can decompose the distribution as $p(X) = p(S, V) = p(S)\, p(V|S)$.

For distributions over s and $\mathbf{v}$, we can use standard mechanisms for representing distributions more efficiently. Specifically, as we show in Section 4.2, we can assume that the distribution $p(V \mid S)$ exhibits independence and exchangeability, due to the regular structure of the multisets. The distribution $p(S)$ is a categorical distribution with substantially smaller support – and therefore much more compact representation – than $p(X)$. Interestingly, BF can be performed directly on this efficient representation: Multiset rewriting is performed on the structures s, and the corresponding value distributions are only inspected and manipulated when necessary (as outlined in Section 5).

4.1 Decomposing Multisets of Structured Entities

To keep the presentation simple, in the following, we consider entities that contain information about the factor of the distribution that its values have been drawn from (we call this information the *distribution type*). This is not strictly necessary, but will be convenient later on by making the factorization structure explicit.

Definition 11 (Typed Entity, Typed State)**.** A *typed entity* $e_d \in \mathcal{E}_d$ is a partial function $e_d :\ \mathcal{P} \nrightarrow (\mathcal{D} \times \mathcal{V})$, i.e. a map of property names $\mathcal{P}$ to pairs of *distribution types* $\mathcal{D}$ and values $\mathcal{V}$. A multiset of typed entities is called *typed state.* ○

We write the distribution type as indices. For example, the typed state

$$x = [\![\, 1\langle \mathrm{N}_{\mathbf{N}}\colon A, \mathrm{L}_{\mathbf{L_1}}\colon 1\rangle, 1\langle \mathrm{N}_{\mathbf{N}}\colon B, \mathrm{L}_{\mathbf{L_2}}\colon 2\rangle \,]\!]$$

consists of two entities with a property N that is drawn from a distribution with type $\mathbf{N}$, one of those has a property L that is drawn from $\mathbf{L_1}$, and the other one has a property L drawn from $\mathbf{L_2}$. The intuition here is that the values of the properties have been drawn from a factorized representation consisting of three factors: One joint factor describing the distribution of the N properties, and two independent factors describing the distribution of the L properties.

To achieve such a factorized representation, we first introduce a mapping between multisets (i.e. abstract objects) to syntactic structures (*terms*), such that (i) the distribution over terms can be represented more efficiently (e.g. it factorized due to independence assumptions), and (ii) this distribution over terms directly induces a distribution $p(X)$ over multisets. The obvious choice is to transform the multisets into a *tuple*, as for distributions over tuples, existing methods for efficiently representing distributions, like graphical models, can be used. Unfortunately, in contrast to tuples, elements in multisets do not have an order, and thus there are multiple tuples that can represent the same multiset, so there is no straightforward bijection between multisets and tuples.

To derive a suitable bijection, we proceed in two steps: First, we define the canonical (ordered) *serializations* σ of a multiset, in which entities are arranged sequentially (and repeated as many times as its multiplicity indicates), as well as the key-value-pairs that each entity consists of. Then, we define a bijective *decomposition function* that decomposes a serialization σ into a pair $(s, \mathbf{v})$ of *multiset structure* s (where multiset rewriting can be applied) and *value sequence* $\mathbf{v}$ (where distributions over $\mathbf{v}$ can be represented more efficiently).

Definition 12 (Entity Serialization, Canonical State Serialization)**.** Let $<_{\mathcal{P}} \subseteq \mathcal{P} \times \mathcal{P}$ be a total order of property names, $<_{\mathcal{D}} \subseteq \mathcal{D} \times \mathcal{D}$ be a total order of distribution types, and $<_{\mathcal{V}} \subseteq \mathcal{V} \times \mathcal{V}$ be a total order of values. Let e be a typed entity. We call $\varsigma(e) = \langle k, e(k) \rangle_{k \in \mathrm{dom}(e)}$, where the order of the pairs follows $<_{\mathcal{P}}$ (major order), $<_{\mathcal{D}}$, and $<_{\mathcal{V}}$ (minor order) the *canonical serialization of* e.

Let x be a typed state. We call the sequence σ that contains $x\#e$ copies of the ordered serialization of all entities e in x, and is ordered according to $<_{\mathcal{P}}$ (major order), $<_{\mathcal{D}}$, and $<_{\mathcal{V}}$ (minor order) the *canonical serialization of* x. Formally, for a state x, the canonical serialization σ of x has the form

$$\langle \underbrace{\varsigma(e_1), \ldots, \varsigma(e_1)}_{x\#e_1 \text{ times}}, \ldots, \underbrace{\varsigma(e_n), \ldots, \varsigma(e_n)}_{x\#e_n \text{ times}} \rangle,$$

where $\{e_1, \ldots, e_n\} = \mathrm{dom}(x)$. ∘

Example 9. The state

$$x = [\![\, 1\langle \mathrm{N}_{\mathbf{N}}\colon A, \mathrm{L}_{\mathbf{L_1}}\colon 1 \rangle, 1\langle \mathrm{N}_{\mathbf{N}}\colon B, \mathrm{L}_{\mathbf{L_2}}\colon 2 \rangle \,]\!]$$

has the canonical serialization

$$\sigma = \langle \langle \langle N, \langle \mathbf{N}, A \rangle \rangle, \langle L, \langle \mathbf{L_1}, 1 \rangle \rangle \rangle,\ \langle \langle N, \langle \mathbf{N}, B \rangle \rangle, \langle L, \langle \mathbf{L_2}, 2 \rangle \rangle \rangle \rangle$$ ∘

Next, we define the decomposition function, that extracts the *structure* and the *value list* from the serialization. The intuition is that s is identical to x, except that the values are removed, and $\mathbf{v}$ is the sequence of values.

Definition 13 (Decomposition Function)**.** Let x be a typed state, and let σ be the canonical serialization of x.

- The *structure* $\mathrm{se}(e) \in \mathcal{E}_\mathcal{D}$ of a typed entity $e \in x$ is identical to e, except that the values are removed, i.e. $\mathrm{se}(\langle k_1 : (d_1, v_1), \ldots, k_i : (d_i, v_i)\rangle) = \langle k_1 : d_1, \ldots, k_i : d_i\rangle$.
- The *structure* $s \in \mathcal{S}$ of σ (where x is the state corresponding to σ) is the multiset of entity structures: $s = [\![\, \mathrm{se}(e) \mid e \in x \,]\!]$ (multiplicities are added when entities are mapped to the same entity structure).
- The *value sequence* $\mathbf{v} \in \mathcal{V}$ of σ is obtained by selecting all values in σ from left to right. That is, the value sequence of the serialization $\sigma = \langle k_1, \langle d_1, v_1\rangle, \ldots, k_n, \langle d_n, v_n\rangle\rangle$ is $\mathbf{v} = \langle v_1, \ldots, v_n\rangle$.

Finally, the *decomposition function* is defined as $\phi(x) = (s, \mathbf{v})$. ○

The association between entities in x and values in $\mathbf{v}$ is maintained by the order of the elements in $\mathbf{v}$. Thus, the decomposition function $\phi(x)$ is *bijective*, i.e. there is an inverse function $\phi^{-1}(s, \mathbf{v}) = x$ (that works by "inserting" the values at the corresponding positions).

For example, the decomposition $\phi(x)$ of x shown in Example 9 is

$$\begin{aligned} &\phi(x) = (s, \mathbf{v}), \text{ where} \\ &\quad s = [\![\, 1\langle \text{N: } \mathbf{N}, \text{L: } \mathbf{L_1}\rangle, 1\langle \text{N: } \mathbf{N}, \text{L: } \mathbf{L_2}\rangle \,]\!] \\ &\quad \mathbf{v} = \langle A, 1, B, 2\rangle. \end{aligned}$$

4.2 Distributions of Value Sequences

We started this section with the goal of deriving an efficient representation of the distribution $p(x)$. Now, via the bijection $\phi(x) = (s, \mathbf{v})$, a given distribution $p(s, \mathbf{v})$ induces a distribution $p(x)$. In the following, we change the perspective and discuss ways of compactly representing $p(s, \mathbf{v}) = p(\mathbf{v} \mid s)\, p(s)$ – which then directly leads to an efficient representation of $p(x)$ via ϕ.

Specifically, we discuss how $p(\mathbf{v} \mid s)$ can be maintained efficiently, for which we make two assumptions:

(i) *Independence* of values belonging to different distribution types (situations where independence does not hold can be represented by mixtures, shown below), and

(ii) *exchangeability* of values corresponding to the same entity structure (due to the fact that the entities in the multiset are not ordered, so all orders of values correspond to the same state).

4.2.1 Independence

We assume that $p(\mathbf{v} \mid s)$ factorizes into independent factors according to the *distribution types* (i.e. RVs with different distribution type are independent). The distribution type for each value is given by s (by generating a sequence of types from s, in the same way as $\mathbf{v}$ has been extracted from x). In the following, we write $\mathbf{v}^{(d)}$ to denote the sub-sequence of values with distribution type d. Subsequently, the distribution factorizes as follows:

$$p(\mathbf{v} \mid s) = \prod_d p(\mathbf{v}^{(d)} \mid s)$$

We can think about the relationship between $p(x)$, $p(s)$ and $p(\mathbf{v} \,|\, s)$ via the following sampling semantics: Given the factors of a distribution $p(\mathbf{v} \,|\, s)$ and a (categorical) distribution $p(s)$, a sample of x is obtained by (i) sampling a structure s from $p(S)$, (ii) sampling a sub-sequence $\mathbf{v}^{(d)}$ from $p(V^{(d)} \,|\, s)$ for each d, (iii) construct the sequence $\mathbf{v}$ from these sub-sequences, and (iv) apply the inverse function $\phi^{-1}(s, \mathbf{v})$. Steps (iii) and (iv) can also be understood as "inserting" the values $\mathbf{v}^{(d)}$ directly into s, at the positions indicated by the distribution types d.

The following example illustrates how a distribution over x can be decomposed into a a distribution over s, and a (factorized) distribution $p(\mathbf{v} \,|\, s)$.

Example 10. For the office domain (Example 2), suppose we need to represent the situation "two of the three agents with names A, B, and C are in room 1, and the other one is at room 2, and we have no information about which specific agent is at which location" (such situations naturally arise during filtering in MRS, when some of the entities' properties cannot be observed directly). This situation is represented by the following uniform distribution of three states:

$$\begin{aligned}
&p(x_1) = p(x_2) = p(x_3) = 1/3, \text{ where} \\
&x_1 = [\![\, 1\langle \mathrm{N}_{\mathbf{N}}\colon A, \mathrm{L}_{\mathbf{L_1}}\colon 1\rangle, 1\langle \mathrm{N}_{\mathbf{N}}\colon B, \mathrm{L}_{\mathbf{L_2}}\colon 2\rangle, 1\langle \mathrm{N}_{\mathbf{N}}\colon C, \mathrm{L}_{\mathbf{L_2}}\colon 2\rangle \,]\!] \\
&x_2 = [\![\, 1\langle \mathrm{N}_{\mathbf{N}}\colon B, \mathrm{L}_{\mathbf{L_1}}\colon 1\rangle, 1\langle \mathrm{N}_{\mathbf{N}}\colon A, \mathrm{L}_{\mathbf{L_2}}\colon 2\rangle, 1\langle \mathrm{N}_{\mathbf{N}}\colon C, \mathrm{L}_{\mathbf{L_2}}\colon 2\rangle \,]\!] \\
&x_3 = [\![\, 1\langle \mathrm{N}_{\mathbf{N}}\colon C, \mathrm{L}_{\mathbf{L_1}}\colon 1\rangle, 1\langle \mathrm{N}_{\mathbf{N}}\colon A, \mathrm{L}_{\mathbf{L_2}}\colon 2\rangle, 1\langle \mathrm{N}_{\mathbf{N}}\colon B, \mathrm{L}_{\mathbf{L_2}}\colon 2\rangle \,]\!].
\end{aligned}$$

The decompositions $(s, \mathbf{v}) = \phi(x)$ of those states all have the identical structure

$$s = [\![\, 1\langle \mathrm{N}\colon \mathbf{N}, \mathrm{L}\colon \mathbf{L_1}\rangle, 2\langle \mathrm{N}\colon \mathbf{N}, \mathrm{L}\colon \mathbf{L_2}\rangle \,]\!]$$

and value sequences

$$\begin{aligned}
\mathbf{v}_1 &= \langle A, 1, B, 2, C, 2\rangle \\
\mathbf{v}_2 &= \langle B, 1, A, 2, C, 2\rangle \\
\mathbf{v}_3 &= \langle C, 1, A, 2, B, 2\rangle
\end{aligned}$$

By assumption, the distribution $p(\mathbf{v} \,|\, s)$ factorizes into one factor per type, i.e.

$$p(\mathbf{v} \,|\, s) = p(\mathbf{v}^{(\mathbf{N})} \,|\, s)\, p(\mathbf{v}^{(\mathbf{L_1})} \,|\, s)\, p(\mathbf{v}^{(\mathbf{L_2})} \,|\, s),$$

where $p(\mathbf{v}^{(\mathbf{L_1})} \,|\, s) \sim \delta_1$, $p(\mathbf{v}^{(\mathbf{L_2})} \,|\, s) \sim \delta_{(2,2)}$ and

$$p(\mathbf{v}^{(\mathbf{N})} \,|\, s) = \begin{cases} 1/3 & \text{if } \mathbf{v}^{(\mathbf{N})} = \langle A, B, C\rangle \\ 1/3 & \text{if } \mathbf{v}^{(\mathbf{N})} = \langle B, A, C\rangle \\ 1/3 & \text{if } \mathbf{v}^{(\mathbf{N})} = \langle C, A, B\rangle \end{cases} \qquad \circ$$

4.2.2 Exchangeability

Next, we discuss how the factors $p(\mathbf{v}^{(d)} \,|\, s)$ can be represented more efficiently than by complete enumeration (as done in the example above), by exploiting *exchangeability*.

First, note that the sequences in the domain of each factor $p(\mathbf{v}^{(d)} \,|\, s)$ adhere to a certain order. More specifically, each sub-sequence of values that is associated with the same entity structure in s follows $<_{\mathcal{V}}$, which is due to the fact that in the serialization process, the values corresponding to identical entity structures are ordered according to $<_{\mathcal{V}}$. For example, in the factor $p(\mathbf{v}^{(\mathbf{N})} \,|\, s)$, only the sequences $\langle A, B, C \rangle$, $\langle B, A, C \rangle$ and $\langle C, A, B \rangle$ have non-zero probability: These are exactly the sequences where the second and third value (which correspond to the entity structure $\langle$X: $\mathbf{N}$, Y: $\mathbf{L_2}\rangle$ with multiplicity 2 in s) are ordered. We call the set of value sequences with this property the *canonical sequences* of type d according to structure s, and denote them by $V_s^{(d)}$.

The main insight that allows to represent the factors $p(\mathbf{v}^{(d)} \,|\, s)$ more efficiently is to note that *any* distribution over arbitrary (non-canonical) sequences $\tilde{p}(\mathbf{v}^{(d)} \,|\, s)$ can be used to define a distribution $p(\mathbf{v}^{(d)} \,|\, s)$ over canonical sequences: The distribution $p(\mathbf{v}^{(d)} \,|\, s)$ is obtained by marginalizing over all sequences in $\tilde{p}(\mathbf{v}^{(d)} \,|\, s)$ that are projected to the same canonical sequence (by ordering each sub-sequence that corresponds to the same entity structure), i.e.

$$p(\mathbf{v}_*^{(d)} \,|\, s) = \sum_{\{\mathbf{v}^{(d)} | \pi_s(\mathbf{v}^{(d)}) = \mathbf{v}_*^{(d)}\}} \tilde{p}(\mathbf{v}^{(d)} \,|\, s), \tag{7}$$

where π_s is a function that maps sequences $\mathbf{v}^{(d)}$ to their corresponding canonical sequence $\mathbf{v}_*^{(d)} \in V_s$ by ordering the sub-sequences for each entity.

When all RVs in $\tilde{p}(\mathbf{v}^{(d)} \,|\, s)$ are *exchangeable*, these factors can be represented much more efficiently than by complete enumeration (Diaconis & Freedman, 1980): For instance, an exchangeable distribution of n boolean RVs can be represented by $n + 1$ parameters rather than requiring 2^n parameters as in the naive representation. Furthermore, Equation 7 becomes particularly simple in this case: As all permutations of a sequence $\mathbf{v}^{(d)}$ have the same probability, it is sufficient to compute the probability $\tilde{p}$ of a single sequence (say, the canonical sequence $\mathbf{v}_*^{(d)}$), and multiply that probability by the number of sequences that are mapped to the canonical sequence $\mathbf{v}_*^{(d)}$, i.e.

$$p(\mathbf{v}_*^{(d)} \,|\, s) = \alpha_d \, \tilde{p}(\mathbf{v}_*^{(d)} \,|\, s), \tag{8}$$

where $\alpha_d = |\{\mathbf{v}^{(d)} | \pi_s(\mathbf{v}^{(d)}) = \mathbf{v}_*^{(d)}\}|$.

Finally, the factor α_d is the product of the number of permutations of the sub-sequences corresponding to each entity structure with values of type d. For example, when all values in each of the sub-sequences are unique, and the sub-sequences for each entity have length $k_1, k_2, \ldots, k_n$ (i.e. the entity structures in s with values of type d have multiplicities $k_1, k_2, \ldots, k_n$), then $\alpha_d = \prod_{i=1}^n k_i!$. Thus, Equation 8 can be calculated without explicitly enumerating all sequences $\mathbf{v}^{(d)}$.

Example 11. Consider $\tilde{p}(\mathbf{v}^{(\mathbf{N})} \,|\, s) \sim \mathcal{U}(A, B, C)$, where $\mathcal{U}(A, B, C)$ represents a uniform distribution of the six permutations of A, B, C (which is obviously exchangeable). Via Equation 8, this distribution represents the distribution $p(\mathbf{v}^{(N)} \,|\, s)$ shown in Example 10. For example, the probability $p(B, A, C \,|\, s)$ is computed as

$$p(B, A, C \,|\, s) = \alpha_{\mathbf{N}} \, \tilde{p}(B, A, C \,|\, s) = 1! * 2! * 1/6 = 1/3. \qquad \circ$$

Here, we have assumed that the complete factor $\tilde{p}$ is exchangeable, which is a strong assumption that allows an efficient representation, but not all distributions $p(\mathbf{v} \mid s)$ can be modeled this way directly. However, this assumptions holds in many practically relevant scenarios (for example, when some of the properties cannot be observed directly, as in Example 10). Distributions where this assumption does not hold can be decomposed into *mixtures* where all factors can be assumed to be exchangeable, as described in see Section 4.3.

Note that we do not attempt to *find* such a decomposition into exchangeable factors for a given distribution $p(x)$[6]. Instead, the idea is that an abstract (*lifted*) representation of the probabilistic model – in the form of $p(v \mid s)$ and $p(s)$ – is given directly by the description of the application domain (see the next section for such a representation), and the goal is to *maintain* that structure during inference[7].

4.3 Lifted States

In the previous section, we showed that by making independence and exchangeability assumptions for $p(\mathbf{v} \mid s)$, the distribution can be represented via a set of exchangeable factors for each s. In the following, we present an efficient representation for the distribution $p(s, \mathbf{v})$ that makes use of this insight, and will enable us to perform multiset rewriting directly on that representation (which is shown in Section 5).

Here, it is useful to distinguish between a distribution, and the *representation* of that distribution. The representation can be a table, the set of parameters of a parametric distribution, or sufficient statistics. For example, a uniform distribution of permutations of of the three elements A, B and C can be represented by the string "$\mathcal{U}(A, B, C)$", the normal distribution with $\mu = 0$ and $\sigma^2 = 1$ can be represented by the string "$\mathcal{N}(0, 1)$". We call $\rho \in \mathcal{R}$ the *representation* of the distribution p. Given a representation ρ, we write p_ρ for the distribution that is represented by ρ.

The idea to represent $p(\mathbf{v} \mid s)$ is to maintain a representation ρ for each factor $p(\mathbf{v}^{(d)} \mid s)$. As we have seen above, we can instead represent the *exchangeable* factors $\tilde{p}(\mathbf{v}^{(d)} \mid s)$, from which the factors $p(\mathbf{v} \mid s)$ can be directly computed via Equation 8. Technically, this is realized as a map from distribution types d to representations ρ of exchangeable factors. We call this representation the *context* of s.

Definition 14 (Context)**.** A *context* $\gamma \in \Gamma$ is a map from distribution types to representations of exchangeable distributions, i.e. $\Gamma = \mathcal{D} \nrightarrow \mathcal{R}$. The distribution of canonical value sequences that is induced by a context γ and a structure s is

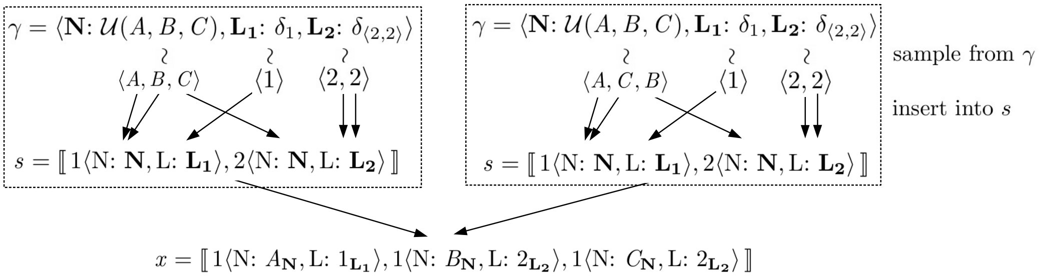
$$p(\mathbf{v} \mid s, \gamma) = \prod_{(d,\rho) \in \gamma} p_\rho(\mathbf{v}^{(d)} \mid s). \tag{9}$$

○

Thus, the distribution $p(\mathbf{v} \mid s)$ is represented on the parametric rather than the instances level. This technique is known as *Rao-Blackwellization*, as used in the Rao-Blackwellized particle filter (Doucet et al., 2000). Intuitively, Rao-Blackwellization leads to more accurate estimates of the filtering distribution, because each particle represents a complete region of

[6] This task is pursued in *bottom-up lifted inference*, e.g. Lifted Belief Propagation (Kersting et al., 2009).

[7] Similar to what is done in *top-down lifted inference*, e.g. First-Order Variable Elimination (Poole, 2003).



$\gamma = \langle \mathbf{N}: \mathcal{U}(A,B,C), \mathbf{L_1}: \delta_1, \mathbf{L_2}: \delta_{\langle 2,2\rangle}\rangle$
$\langle A,B,C\rangle$
$\langle 1\rangle$
$\langle 2,2\rangle$
$s = [\![1\langle \text{N: } \mathbf{N}, \text{L: } \mathbf{L_1}\rangle, 2\langle \text{N: } \mathbf{N}, \text{L: } \mathbf{L_2}\rangle]\!]$
$\gamma = \langle \mathbf{N}: \mathcal{U}(A,B,C), \mathbf{L_1}: \delta_1, \mathbf{L_2}: \delta_{\langle 2,2\rangle}\rangle$
$\langle A,C,B\rangle$
$\langle 1\rangle$
$\langle 2,2\rangle$
$s = [\![1\langle \text{N: } \mathbf{N}, \text{L: } \mathbf{L_1}\rangle, 2\langle \text{N: } \mathbf{N}, \text{L: } \mathbf{L_2}\rangle]\!]$
sample from γ
insert into s
$x = [\![1\langle \text{N: } A_{\mathbf{N}}, \text{L: } 1_{\mathbf{L_1}}\rangle, 1\langle \text{N: } B_{\mathbf{N}}, \text{L: } 2_{\mathbf{L_2}}\rangle, 1\langle \text{N: } C_{\mathbf{N}}, \text{L: } 2_{\mathbf{L_2}}\rangle]\!]$

Figure 3: Example illustrating the sampling semantics of lifted states: Here, two samples are drawn from $l = (s, \gamma)$, each by sampling a value from each factor in γ, and inserting into the places specified by the distribution types. In this case, both samples lead to the same ground state x

the state space, instead of only a single instance. Here, in comparison to the conventional Rao-Blackwellized particle filter, the partitioning of variables represented on the instance and on the parameter level is not fixed, but can change over time (due to *splitting*, see Section 5.2).

We do not attempt to decompose the distribution over *structures* s any further, and simply represent $p(S)$ as a categorical distribution. Thus, overall, we need to maintain a categorical distribution $p(S)$ of structures, and for each structure, a *context* γ that represents $p(V \mid s)$. This is equivalent to directly maintaining a categorical distribution over pairs (s, γ).

Definition 15 (Lifted State)**.** We call a pair $l = (s, \gamma) \in \mathcal{S} \times \Gamma$ a *lifted state.* ○

Intuitively, a lifted state represents a distribution over ground states that all have the structure of s, and whose values follow the distribution induced by γ. That is, the distribution of ground states, given a lifted state $l = (s, \gamma)$ is

$$p(x \mid s, \gamma) = \mathbb{1}(s_x = s)\, p(\mathbf{v} \mid s, \gamma), \tag{10}$$

where $\phi(x) = (s_x, \mathbf{v})$. This distribution can also be described by the following sampling procedure: Draw a sample of each factor in γ, and insert the values into s at the positions indicated by the distribution types (see Figure 3 for an example).

Example 12. Consider the lifted state $l = (s, \gamma)$ with

$$s = [\![1\langle \text{N: } \mathbf{N}, \text{L: } \mathbf{L_1}\rangle, 2\langle \text{N: } \mathbf{N}, \text{L: } \mathbf{L_2}\rangle]\!] \text{ and}$$
$$\gamma = \langle \mathbf{N}: \mathcal{U}(A,B,C), \mathbf{L_1}: \delta_1, \mathbf{L_2}: \delta_{\langle 2,2\rangle}\rangle.$$

This lifted state represents exactly the distribution of ground states shown in Example 10, which can be seen via the sampling semantics (see Figure 3). The probability of each ground state can also be computed in closed form via Equation 10. For example, the probability of the ground state

$$x = [\![1\langle \text{N: } A_{\mathbf{N}}, \text{L: } 1_{\mathbf{L_1}}\rangle, 1\langle \text{N: } B_{\mathbf{N}}, \text{L: } 2_{\mathbf{L_2}}\rangle, 1\langle \text{N: } C_{\mathbf{N}}, \text{L: } 2_{\mathbf{L_2}}\rangle]\!]$$

in l is computed as follows: Applying the decomposition function to x leads to $\phi(x) = (s, \mathbf{v})$ with

$$s = [\![\, 1\langle \mathrm{N}\colon \mathbf{N}, \mathrm{L}\colon \mathbf{L_1}\rangle, 2\langle \mathrm{N}\colon \mathbf{N}, \mathrm{L}\colon \mathbf{L_2}\rangle \,]\!]$$
$$\mathbf{v}^{(\mathbf{L_1})} = 1,\ \mathbf{v}^{(\mathbf{L_2})} = \langle 2, 2\rangle,\ \mathbf{v}^{(\mathbf{N})} = \langle A, B, C\rangle.$$

The context γ represents the distributions $\tilde{p}(\mathbf{v}^{(\mathbf{L_1})} \mid s) \sim \delta_1$, $\tilde{p}(\mathbf{v}^{(\mathbf{L_2})} \mid s) \sim \delta_{\langle 2,2\rangle}$ and $\tilde{p}(\mathbf{v}^{(\mathbf{N})} \mid s) \sim \mathcal{U}(A, B, C)$. Furthermore, $\alpha(1) = 1$, $\alpha(2, 2) = 1$ and $\alpha(A, B, C) = 2$. Thus, according to Equation 10, the probability of x in l is computed as

$$\begin{aligned} p(x \mid s, \gamma) &= \alpha(1)\,\tilde{p}(1)\,\alpha(2,2)\,\tilde{p}(2,2)\,\alpha(A,B,C)\,\tilde{p}(A,B,C) \\ &= 1 * 1 * 1 * 1 * 2 * 1/6 = 1/3. \end{aligned} \qquad \circ$$

In the following, for readability we omit delta distributions and the corresponding types, and instead write the corresponding value directly into the structure. This way, the lifted state l in Example 12 above can by written as

$$s = [\![\, 2\langle \mathrm{X}\colon \mathbf{N}, \mathrm{Y}\colon 1\rangle, 1\langle \mathrm{X}\colon \mathbf{N}, \mathrm{Y}\colon 2\rangle \,]\!]$$
$$\gamma = \langle \mathbf{N}\colon \mathcal{U}(A, B, C)\rangle$$

Finally, a *distribution* of lifted states $p(L)$ defines a distribution of ground states via

$$p(x) = \sum_{l=(s,\gamma)} p(l)\, p(x \mid s, \gamma) \tag{11}$$

Note that in a distribution over lifted states $l = (s, \gamma)$, the structures s need not be distinct. This way, the case where the value distribution is a *mixture* of multiple components can be represented.

To summarize the results so far, we showed how to efficiently represent distributions over structured multisets, by exploiting exchangeability. Technically, this was achieved by decomposing multisets into a structure and a sequence of values, such that the distribution over values factorizes into exchangeable factors. In the following, we show how to use these constructs for efficient BF, by applying multiset rewriting directly to the structure part of the lifted states, and manipulating the value distributions only when necessary.

5. Bayesian Filtering for Lifted States

In this section, we present *Lifted Marginal Filtering* (LiMa), a BF algorithm that works directly on the lifted state representation. The system dynamics is defined in terms of a PMPMRS, as introduced in Sections 2 and 3. The key insight of this section is that the prediction and update steps can be performed directly on lifted states, which is equivalent to performing the same transformation to all ground states that are represented by the lifted state (described in Section 5.1).

Of course, this is not directly possible when different ground states that are represented by a lifted state allow *different* actions to be applied, or when they need to be weighted by different observation likelihoods. In this case, *splitting* (introduced in Section 5.2) needs to be applied first, which transforms a lifted state into an equivalent set of lifted states that each permit uniform application of actions or observations.

5.1 Applying Constraints and Effects to Lifted States

A fundamental task for multiset rewriting (and for performing the update using the observation model described in Section 3.3) is to test whether a constraint c is satisfied for an entity e, i.e. whether $e \models c$. The goal here is to test constraints directly for *entity structures*, the elements contained in the structure s of a lifted state $l = (s, \gamma)$.

To formalize this, we need the concept of a *region* of a lifted state l, which is the set of all ground states that are assigned a non-zero probability by l.

Definition 16 (Region)**.** Let $l = (s, \gamma)$ be a lifted state and e be an entity structure in s. We call the set

$$\text{region}(l) = \{x \,|\, p(x \,|\, s, \gamma) > 0\}$$

the *region* of l, and the set

$$\text{region}_l(e) = \{e_x \,|\, e = \text{se}(e_x), e_x \in x, x \in \text{region}(l)\}$$

the *region* of e regarding l. ○

We say that an entity structure e satisfies a constraint c when all groundings of e satisfy c, i.e. when $\forall e_x \in \text{region}_l(e) : e_x \models c$, and does not satisfy c when none of the groundings satisfy c, i.e. when $\forall e_x \in \text{region}_l(e) : e_x \not\models c$. The constraint is indeterminate for e when it is satisfied for some groundings of e and not satisfied for other groundings of e. This latter case is handled by splitting (Section 5.2).

The algorithm needs to be able to test this property *without* generating all ground entities first. When only considering a simple constraint language (that only allows testing whether a property of an entity is equal to a given constant value, and conjunctions of those tests), this task is trivial, as illustrated in the following example.

Example 13. Consider the lifted state $l = (s, \gamma)$ where

$$s = \llbracket\, 2\langle \text{X: } \mathbf{N}, \text{Y: } 1\rangle, 1\langle \text{X: } \mathbf{N}, \text{Y: } 2\rangle \,\rrbracket, \gamma = \langle \mathbf{N}\text{: } \mathcal{U}(A, B, C)\rangle$$

and the constraints $c_1(e) = e(Y) == 1$ and $c_2(e) = e(N) == A$. The constraint c_1 is satisfied for $\langle \text{X: } \mathbf{N}, \text{Y: } 1\rangle$ and not satisfied for $\langle \text{X: } \mathbf{N}, \text{Y: } 2\rangle$. The constraint c_2 is indeterminate for each of the entities in l, as it is satisfied for only some of the groundings of each entity. ○

To allow multiset rewriting on lifted states, the algorithm also needs to be able to apply an effect function directly to a lifted state l, such that the result is equivalent to applying the effect to all groundings $x \in \text{region}(l)$. More specifically, the resulting successor state l' needs to describe the same ground distribution as the ground distribution resulting from applying the effect to all groundings of l.

However, for the simple effect functions introduced in Section 2.2 (replace a value of an entity, add a new property-value-pair to an entity, add a new entity), this is also trivial: As all manipulated properties are constants, manipulations of a lifted state are directly equivalent to manipulations of its groundings.

Example 14. Consider the effect $f = e(Y) \leftarrow 2$, and suppose that the the effect is applied to the entity structure $e = \langle \text{X: } \mathbf{N}, \text{Y: } 1\rangle$ in the lifted state $l = (s, \gamma)$ where

$$s = [\![\, 2\langle \text{X: } \mathbf{N}, \text{Y: } 1\rangle, 1\langle \text{X: } \mathbf{N}, \text{Y: } 2\rangle \,]\!], \gamma = \langle \mathbf{N}\text{: } \mathcal{U}(A, B, C)\rangle.$$

The successor state is $l' = (s', \gamma)$ with

$$s' = [\![\, 1\langle \text{X: } \mathbf{N}, \text{Y: } 1\rangle, 2\langle \text{X: } \mathbf{N}, \text{Y: } 2\rangle \,]\!]. \qquad \circ$$

Thus, constraints can be tested and effects can be applied directly on lifted states, and subsequently, BF can be performed directly on the lifted representation, using the PMPMRS-based transition model (Section 2.4) and the constraint-based observation model (Section 3.3). Specifically, the Marginal Filtering algorithm (Section 3.4) can be applied. The resulting *Lifted* Marginal Filtering (LiMa) algorithm is presented in more detail in Section 5.3.

5.2 Splitting

Before we can describe the overall Lifted Marginal Filtering algorithm, we need to discuss a problem that can occur when testing constraints on lifted states: A constraint can be satisfied for some groundings of an entity structure, and not satisfied for other groundings (as illustrated in Example 13). More generally, the ground states $x \in \text{region}(l)$ form *partitions* based on *how many entities* in x satisfy c. We need to represent each of those regions by a separate lifted state, because a *different* set of AMCAs can be applicable for each partition (and to compute the lifted successor states of l, it is necessary that a fixed set of AMCAs is applicable for all groundings of l).

We want to compute lifted states that describe these partitions of l without requiring a complete enumeration of all ground states first. This is done by manipulating l by an operation called *splitting*. A split is an operation that decomposes a lifted state $l = (s, \gamma)$ into a set $L = \{(l_i, w_i)\}$. We call a split *correct*, when (i) L describes the same distribution of ground states as l, i.e.

$$\sum_i w_i \, p(x \mid s_i, \gamma_i) = p(x \mid s, \gamma), \tag{12}$$

and (ii) for each $l_i \in L$, all ground states $x \in \text{region}(l_i)$ lie in the same partition regarding c[8], i.e. $\forall l_i \forall x \in \text{region}(l_i) : x \models \#_c^{=i}$.

Example 15. Consider the lifted state $l = (s, \gamma)$ with

$$s = [\![\, 2\langle \text{X: } \mathbf{N}, \text{Y: } 1\rangle, 2\langle \text{X: } \mathbf{N}, \text{Y: } 2\rangle \,]\!], \qquad \gamma = \langle \mathbf{N}\text{: } \mathcal{U}(3A, 2B)\rangle$$

and the constraint $c(e) = (e(X) == A) \wedge (e(Y) == 1)$. This constraint is indeterminate in l, as it is satisfied zero times, once, or twice for different groundings of l. The splitting

[8] Note that for counting constraints $\#_c^{=n}$, as used in the observation model, it would in principle be sufficient to create two partitions: One where the counting constraint is satisfied (i.e. where exactly n entities satisfy c), and one where the counting constraint is not satisfied (i.e. where c is satisfied for a different number $m \neq n$ of entities). However, the latter case can often not be described by a single lifted state (as shown below), and instead, partitions for each n need to be created anyways.

algorithm (which will be described in detail below) will create three lifted states $l_1 = (s_1, \gamma_1)$, $l_2 = (s_2, \gamma_2)$ and $l_3 = (s_3, \gamma_3)$ that describe exactly those partitions. These split results are identical to l, except that instead of entity structures $\langle \text{X: } \mathbf{N}, \text{Y: } 1\rangle$, they contain groundings of that entity structure (i.e. where $e(X) = A$ or $e(X) = B$):

$$\begin{aligned}
s_1 &= [\![\, 2\langle \text{X: } A, \text{Y: } 1\rangle, 2\langle \text{X: } \mathbf{N}, \text{Y: } 1\rangle \,]\!], & \gamma_1 &= \langle \mathbf{N}\text{: } \mathcal{U}(1A, 2B)\rangle,\\
s_2 &= [\![\, 1\langle \text{X: } A, \text{Y: } 1\rangle, 1\langle \text{X: } B, \text{Y: } 1\rangle, 2\langle \text{X: } \mathbf{N}, \text{Y: } 2\rangle \,]\!], & \gamma_2 &= \langle \mathbf{N}\text{: } \mathcal{U}(2A, 1B)\rangle\\
s_3 &= [\![\, 2\langle \text{X: } B, \text{Y: } 1\rangle, 2\langle \text{X: } \mathbf{N}, \text{Y: } 2\rangle \,]\!], & \gamma_3 &= \langle \mathbf{N}\text{: } \mathcal{U}(3A)\rangle
\end{aligned}$$

When the weights $w_1 = 0.3$, $w_2 = 0.6$ and $w_3 = 0.1$ are assigned to the states, they describe exactly the same distribution over ground states as l (which can be seen by computing the probability of each ground state via Equation 12). Furthermore, the constraint c is now satisfied for exactly two of the entities in l_1, for exactly one of the entities in l_2 and for none of the entities in l_3. ○

Next, we show how such splits can be computed systematically. We start with a general, high-level description of the splitting strategy, and afterwards show how this strategy is instantiated for different parametric forms of the factor that is split.

5.2.1 General Splitting Strategy

In the following, we discuss a general strategy for splitting a lifted state $l = (s, \gamma)$. Let c be a constraint of the form $q == v^*$, where q is a property name and v^* is a value[9]. Let e with $e(q) = d$ be the entity structure for which the constraint c needs to be tested, and let $\gamma(d) = \rho$ be the distribution representation that needs to be split. For splitting, we focus on the factor $p_\rho(V^{(d)})$ that is represented by ρ. To keep the notation uncluttered, in the following we omit the subscripts and superscripts and write $p(V)$ for that factor.

The general strategy for splitting is to partition the possible values of the random variable V into subsets $\mathcal{V}_i, \ldots, \mathcal{V}_n$. For each subset $\mathcal{V}_i$, we create a new factor $p_i(V)$ which has non-zero support only for the values $\mathcal{V}_i$. The probability of an assignment $p_i(V{=}\mathbf{v})$ is the re-normalized probability of the assignment in the original factor $p(V{=}\mathbf{v})$:

$$p_i(\mathbf{v}) = \begin{cases} \frac{p(\mathbf{v})}{\sum_{\mathbf{v}' \in \mathcal{V}_i} p(\mathbf{v}')} & \text{if } \mathbf{v} \in \mathcal{V}_i \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

The weights of the split results are set to $w_i = \sum_{\mathbf{v} \in \mathcal{V}_i} p(\mathbf{v})$. From these definitions, it directly follows that

$$p(V) = \sum_i w_i \, p_i(V). \tag{14}$$

For each factor $p_i(V)$, the splitting algorithm creates a split result l_i, which is identical to l except that the representation ρ of $p(V)$ is replaced by a representation of $p_i(V)$. Due to Equation 14 and the fact that all other factors of l and all spit results l_i are identical, this splitting procedure satisfies Equation 12 above, i.e. correctness requirement (i). To satisfy correctness requirement (ii) – the constraint c must be satisfied for a fixed number

[9] Conjunctions of those constraints can be handled consecutively by multiple splits.

Algorithm 2 In lifted state l with weight p, split entity e on constraint $q == v^*$, distributed according to **multinomial** distribution ρ.

```
function SPLIT-MULTINOMIAL($l{=}(s,\gamma)$, $p$, $q$, $v^*$, $e$, $\rho$)
    $k \leftarrow s\#e$;  $P' \leftarrow \emptyset$;  $\mathrm{d} \leftarrow e(q)$
    Create new distribution type $d_{v^*}$
    $\rho' \leftarrow \rho$ without $v^*$ ▷ Multinomial with parameters $\frac{p_1}{1-p_{v^*}}, \dots, \frac{p_m}{1-p_{v^*}}$, see Equation 18
    $\gamma' \leftarrow \gamma \oplus \langle \mathrm{d}\colon \rho', \mathrm{d}_{v^*}\colon \delta_{v^*} \rangle$    ▷ Add changed representations to context
    $e' \leftarrow e \oplus \langle \mathrm{q}\colon d_{v^*} \rangle$    ▷ New entity with constant value $v^*$ at $q$
    for $i = 0, \dots, k$ do
        $s' \leftarrow s \uminus [\![ (k-i)\, e ]\!] \uplus [\![ i\, e' ]\!]$    ▷ New structure where $i$ entities have value $v^*$
        $p' \leftarrow \binom{k}{i}\, p_{v^*}^{i}\, (1-p_{v^*})^{k-i}\, p$    ▷ Probability of $l'$, see Equation 17
        $P' \leftarrow P' \cup \{((s',\gamma'),p')\}$    ▷ Collect all split results
    return $P'$
```

of entities in all groundings of l_i – the subsets $\mathcal{V}_i$ need to be chosen appropriately. When this is the case, the general splitting strategy is correct.

In the following, we show how this general strategy is instantiated for different parametric forms of the factor $p(V)$. Specifically, we discuss how the subsets $\mathcal{V}_i$ can be chosen so that they satisfy correctness requirement (ii).

In the experiments (Section 6), we will mostly be concerned with the case where the factors $p(V)$ are uniform distributions over permutations (which arises when entities have unique identifies, like names). Thus, we will describe splitting procedures for this case, and the related, but more general cases of multivariate hypergeometric distributions and multinomial distributions (i.e. urns with and without replacement).

5.2.2 Multinomial Distribution

We first consider *sampling with replacement* from an urn, which has m possible values $v_1, \dots, v_m$, and probabilities $p_1, \dots, p_m$ of drawing each of the values. The probability of drawing k samples from the urn, where k_j samples have value v_j and $\sum_{j=1}^{m} k_j = k$ is described by the multinomial distribution:

$$p_\rho(k_1, \dots, k_m) = \frac{k!}{\prod_{n=1}^{m} k_n!} \prod_{n=1}^{m} p_n^{k_n}. \tag{15}$$

We start by giving an example of the split procedure, and describe the general algorithm afterwards.

Example 16. Consider the lifted state[10]

$$l = ([\![\, 3\langle \mathrm{N}\colon \mathbf{N}, \mathrm{L}\colon X \rangle, 2\langle \mathrm{N}\colon \mathbf{N}, \mathrm{L}\colon Y \rangle \,]\!], \langle \mathbf{N}\colon \mathcal{M}(1/2\,A, 1/3\,B, 1/6\,C) \rangle).$$

We want to split l based on the constraint $c(e) = (e(N) == A)$ for the entity structure $\langle \mathrm{N}\colon \mathbf{N}, \mathrm{L}\colon X \rangle$. The split procedure generates four split results – one for each number of

[10] We use $\mathcal{M}$ to indicate a multinomial distribution, i.e. an urn with replacement.

times the value A can be sampled from the multinomial distribution. More precisely, split result l_i contains $i-1$ times the entity structure $\langle \text{N: } A, \text{L: } X\rangle$ (which has taken value A), and $k-i-1$ times the entity structure $\langle \text{N: } \mathbf{N_1}, \text{L: } X\rangle$ (which has not taken the value A):

$$\begin{aligned}
l_1 &= (\llbracket\, 3\langle \text{N: } \mathbf{N_1}, \text{L: } X\rangle, 2\langle \text{N: } \mathbf{N_2}, \text{L: } Y\rangle \,\rrbracket && \langle \mathbf{N_1}\text{: } \mathcal{M}(2/3\,B, 1/3\,C), \mathbf{N_2}\text{: } \mathcal{M}(1/2\,A, 1/3\,B, 1/6\,C)\rangle \\
l_2 &= (\llbracket\, 1\langle \text{N: } A, \text{L: } X\rangle, 2\langle \text{N: } \mathbf{N_1}, \text{L: } X\rangle, 2\langle \text{N: } \mathbf{N_2}, \text{L: } Y\rangle \,\rrbracket, && \langle \mathbf{N_1}\text{: } \mathcal{M}(2/3\,B, 1/3\,C), \mathbf{N_2}\text{: } \mathcal{M}(1/2\,A, 1/3\,B, 1/6\,C)\rangle \\
l_3 &= (\llbracket\, 2\langle \text{N: } A, \text{L: } X\rangle, 1\langle \text{N: } \mathbf{N_1}, \text{L: } X\rangle, 2\langle \text{N: } \mathbf{N_2}, \text{L: } Y\rangle \,\rrbracket, && \langle \mathbf{N_1}\text{: } \mathcal{M}(2/3\,B, 1/3\,C), \mathbf{N_2}\text{: } \mathcal{M}(1/2\,A, 1/3\,B, 1/6\,C)\rangle \\
l_4 &= (\llbracket\, 3\langle \text{N: } A, \text{L: } X\rangle, 2\langle \text{N: } \mathbf{N}, \text{L: } Y\rangle \,\rrbracket, && \langle \mathbf{N}\text{: } \mathcal{M}(1/2\,A, 1/3\,B, 1/6\,C)\rangle
\end{aligned}$$

The weight of split result l_i is defined by the (marginal) probability of sampling the value A $i-1$ times from the multinomial distribution (see Equation 17 below). For example, $w_3 = \binom{3}{2}\,(\frac{1}{2})^2\,(1-\frac{1}{2})^1 = 0.375$. ○

Next, we describe the splitting algorithm more formally. Here, we only consider *equality constraints*, which test whether a property q of an entity structure $e \in \text{dom}(s)$ has a specific value v_j. We assume that q is distributed according to a multinomial distribution.

In this case, we generate $k+1$ split results, where $k = s\#e$ is the multiplicity of e in s. The set $\mathcal{V}_{i+1}$ that is used to define split results contains all assignments where the value v_j has been sampled i times from the urn, i.e. where $K_j = i$. That is, in split result l_{i+1}, i entity structures have value v_j, and $k-i$ entity structures have values distributed according to the "remaining" urn without v_j. To see how the parameters of that urn, and the weight of l_i need to be chosen so that the split is correct, we rewrite the multinomial distribution as

$$p(K_1, \dots, K_j{=}i, \dots, K_n) = p(K_j{=}i)\, p(K_1, \dots, K_{j-1}, K_{j+1}, \dots, K_n \mid K_j{=}i) \tag{16}$$

From this equation, it is easy to see that this splitting strategy is an instance of the general strategy defined above: The conditional distribution $p(K_1, \dots, K_{j-1}, K_{j+1}, \dots, K_n \mid K_j{=}i)$ has the property of p_i shown in Equation 13, i.e. it is only non-zero for assignments in $\mathcal{V}_{i+1}$. The marginal distribution $p(K_j{=}i)$ can be evaluated directly and is used as the weight w_i of l_i. It is easy to show (Siegrist, 2020) that such a marginal distribution of a multinomial is a binomial distribution with parameters k and p_j, i.e.

$$p(K_j{=}i) = \binom{k}{i}\, p_j^i\, (1-p_j)^{k-i}. \tag{17}$$

The conditional multinomial distribution $p(K_1, \dots, K_{j-1}, K_{j+1}, \dots, K_m \mid K_j{=}i)$ can also be represented in closed form: It is again a multinomial distribution, with parameters $k-i$ and $\frac{p_1}{1-p_j}, \dots, \frac{p_m}{1-p_j}$ (Siegrist, 2020):

$$p(K_1, \dots, K_{j-1}, K_{j+1}, \dots, K_m \mid K_j{=}i) = \frac{(k-i)!}{\prod_{n=1}^m k_n!} \prod_{n=1}^m \left(\frac{p_n}{1-p_j}\right)^{k_n} \tag{18}$$

Intuitively, this distribution represents the "remaining" draws from the urn, after knowing that value v_j has been drawn i times. This multinomial distribution is encoded in split result l_i. As this strategy is an instance of the general splitting strategy, it is correct, i.e. the weighted split results constructed this way describe the same distribution over ground states as the original lifted state l. The splitting algorithm which constructs these split results is shown in Algorithm 2.

Algorithm 3 In lifted state l with weight p, split entity e on constraint $q == v^*$, distributed according to **multivariate hypergeometric** distribution ρ.

```
function SPLIT-HYPERGEOMETRIC($l{=}(s,\gamma)$, $p$, $q$, $e$, $\rho = (v_j, n_j)_{j=1}^{m}$)
    $k \leftarrow s\#e$;  $P' \leftarrow \emptyset$;  $\mathrm{d} \leftarrow e(q)$
    for each composition $(k_1, \ldots, k_m) \in \mathcal{C}(\rho, k)$ do
        $s' \leftarrow s \uminus [\![\, k\, e \,]\!]$
        $\rho' \leftarrow \rho$ without $(k_1, \ldots, k_m)$
        $\gamma' \leftarrow \gamma \oplus \langle \mathrm{d}\colon \rho' \rangle$                ▷ Add changed representation to context
        for $j = 1, \ldots, m$ where $k_j > 0$ do      ▷ Create state with entities for each $k_j > 0$
            Create new distribution type $\mathrm{d}_{v_j}$
            $e' \leftarrow e \oplus \langle \mathrm{q}\colon \mathrm{d}_{v_j} \rangle$                 ▷ New entity with constant value $v_j$ at $q$
            $s' \leftarrow s' \uplus [\![\, k_j\, e' \,]\!]$            ▷ New structure where $k_j$ entities have value $v_j$
            $\gamma' \leftarrow \gamma' \oplus \langle \mathrm{d}_{v_j}\colon \delta_{v_j} \rangle$                  ▷ Add representation of constant
        $p' \leftarrow p_\rho(k_1, \ldots, k_m)\, p$                       ▷ Probability of $l'$, see Equation 19
        $P' \leftarrow P' \cup \{((s', \gamma'), p')\}$                          ▷ Collect all split results
    return $P'$
```

5.2.3 Hypergeometric Distribution

Next, we consider urns *without* replacement. The multivariate hypergeometric distribution with representation $\rho = (v_j, n_j)_{j=1}^{m}$ describes sampling balls *without replacement* from an urn which has n_j balls of value v_j. The probability of drawing exactly k_j balls of each value v_j (with $k = \sum_j k_j$ and $n = \sum_j n_j$) is:

$$p_\rho(k_1, \ldots, k_m) = \frac{\prod_{j=1}^{m} \binom{n_j}{k_j}}{\binom{n}{k}} \tag{19}$$

Here, we cannot apply the same strategy as for multinomial distributions. The reason is that drawing specific values $v_1, \ldots, v_k$ for e leads to changes in the remaining urn (the values $v_1, \ldots, v_k$ are removed from the urn), which also affects other entities in $l = (s, \gamma)$ that have values distributed according to ρ.

Instead, we create a split result for each possible combination of values that can be assigned to the $k = s\#e$ entities e. Note that in this case, it does not matter which specific value v^* we are interested in, was we are considering all value assignments anyway. However, the value v^* is relevant for a common special case, as outlined below. Again, we only consider *equality constraints* c that test whether a property q of an entity $e \in \mathrm{dom}(s)$ has a specific value v^*. We assume that q is distributed according to a hypergeometric distribution, i.e. $e(q) = d$, and $\gamma(d) = \rho = (v_j, n_j)_{j=1}^{m}$.

More concretely, splitting is performed as follows: Let $C(\rho, k)$ be the set of compositions of k of size m (where $\rho = (v_j, n_j)_{j=1}^{m}$), i.e. a way of writing k as the sum of m integers $k_1 + \cdots + k_m$, with the additional constraint that k_j is at most n_j. Each composition $(k_1, \ldots, k_m) \in C(\rho, k)$ corresponds to an assignment of p_ρ, where k_j balls of value v_j are drawn. The subsets $\mathcal{V}_i$ which are used to define the split results (see Section 5.2.1) each contain a single assignment, i.e. a single composition from $C(\rho, k)$. For each subset $\mathcal{V}_i$ (i.e. for each composition $(k_1, \ldots, k_m) \in C(\rho, k)$), a lifted state l_i is constructed, where the entity structures e are removed, and one entity structure is inserted for each $k_j > 0$, with multiplicity k_j, that is identical to e, except that q is distributed according to δ_{v_j}. The weight w_i of l_i is given by the probability of that assignment, i.e. $w_i = p_\rho(k_1, \dots, k_m)$ (see Equation 19). The splitting algorithm is shown in Algorithm 3.

Example 17. Consider the lifted state

$$l = (\llbracket\, 3\langle \text{N: } \mathbf{N}, \text{L: } X\rangle, 2\langle \text{N: } \mathbf{N}, \text{L: } Y\rangle \,\rrbracket, \langle \mathbf{N}\text{: } \mathcal{U}(3A, 2B, 1C)\rangle.$$

We want to split l based on the constraint $c(e) = (e(N) == A)$ for the entity structure $\langle \text{N: } \mathbf{N}, \text{L: } X\rangle$. Applying the procedure defined above results in six split results, one for each possible assignment of values to the entity structure e.

$$\begin{aligned}
l_1 &= (\llbracket\, 3\langle \text{N: } A, \text{L: } X\rangle, 2\langle \text{N: } \mathbf{N}, \text{L: } Y\rangle \,\rrbracket, && \langle \mathbf{N}\text{: } \mathcal{U}(2B, 1C)\rangle \\
l_2 &= (\llbracket\, 2\langle \text{N: } A, \text{L: } X\rangle, 1\langle \text{N: } B, \text{L: } X\rangle, 2\langle \text{N: } \mathbf{N}, \text{L: } Y\rangle \,\rrbracket, && \langle \mathbf{N}\text{: } \mathcal{U}(1A, 1B, 1C)\rangle \\
l_3 &= (\llbracket\, 2\langle \text{N: } A, \text{L: } X\rangle, 1\langle \text{N: } C, \text{L: } X\rangle, 2\langle \text{N: } \mathbf{N}, \text{L: } Y\rangle \,\rrbracket, && \langle \mathbf{N}\text{: } \mathcal{U}(1A, 2B)\rangle \\
l_4 &= (\llbracket\, 1\langle \text{N: } A, \text{L: } X\rangle, 2\langle \text{N: } B, \text{L: } X\rangle, 2\langle \text{N: } \mathbf{N}, \text{L: } Y\rangle \,\rrbracket, && \langle \mathbf{N}\text{: } \mathcal{U}(2A, 1C)\rangle \\
l_5 &= (\llbracket\, 1\langle \text{N: } A, \text{L: } X\rangle, 1\langle \text{N: } B, \text{L: } X\rangle, 1\langle \text{N: } C, \text{L: } X\rangle, 2\langle \text{N: } \mathbf{N}, \text{L: } Y\rangle \,\rrbracket, && \langle \mathbf{N}\text{: } \mathcal{U}(2A, 1B)\rangle \\
l_6 &= (\llbracket\, 2\langle \text{N: } B, \text{L: } X\rangle, 1\langle \text{N: } C, \text{L: } X\rangle, 2\langle \text{N: } \mathbf{N}, \text{L: } Y\rangle \,\rrbracket, && \langle \mathbf{N}\text{: } \mathcal{U}(3A)\rangle
\end{aligned}$$

The weight of each split component is defined by Equation 19. For example, $w_2 = \frac{\binom{3}{2}\binom{2}{1}}{\binom{6}{3}} = 0.3$. Note that the entity structures $\langle \text{N: } \mathbf{N}, \text{L: } Y\rangle$ are not manipulated by the splitting procedure (although the *distribution* of their values is of course manipulated). $\circ$

In general, this procedure quickly leads to a combinatorial explosion in the number of split components, as $|C(\rho, k)|$ can be very large. However, there are two common special cases where the number of split components is low. A very simple special case is $n = k$, i.e. all values of the distribution are assigned to the entities. In this case, only a single composition exists, and thus there is only a single split component. Another special case is $n_{v^*} = 1$, i.e. the value we are interested in exists exactly once in the urn, which is discussed in the following section.

5.2.4 Hypergeometric Distribution with Unique Values

Here, we discuss splitting of hypergeometric distributions for the special case where the value v^* we are interested in exists exactly once in the urn, i.e. $n_{v^*} = 1$. This special case arises for example when the values represent unique identifiers of the entities, like names. In principle, we can proceed as outlined above, but this results in an unnecessarily large number of split components. Ideally, would only need to generate two split components: (a) Constraint c can be satisfied exactly once by an entity structure e; (b) Constraint c cannot be satisfied by any entity structure e. Other cases (where c can be satisfied more than once) do not exist, as $n_{v^*} = 1$. In general, however, case (b) cannot be captured by a single lifted state, as the resulting distribution is not exchangeable, but we can decompose this case further: Suppose there are other entity structures e' with a property q that reference ρ. When e does not have value v^*, either one of the other entity structures e must take

Algorithm 4 In lifted state l with weight p, split entity e on constraint $q == v^*$, distributed according to **multivariate hypergeometric** distribution ρ **where $\mathbf{n_{v^*} = 1}$**.

```
function SPLIT-HYPERGEOMETRIC-UNIQUE($l$=$(s,\gamma)$, $p$, $q$, $e$, $\rho = (v_j, n_j)_{j=1}^m$)
    $k \leftarrow s\#e$; $P' \leftarrow \emptyset$; $\mathrm{d} \leftarrow e(q); n \leftarrow \sum_{j=1}^m n_j$
    Create new distribution type $\mathrm{d}_{v^*}$
    $\rho' \leftarrow \rho$ without $v^*$
    $\gamma' \leftarrow \gamma \oplus \langle \mathrm{d}\colon \rho', \mathrm{d}_{v^*}\colon \delta_{v^*} \rangle$        ▷ Add changed representation to context
    $E \leftarrow \{e \in \mathrm{dom}(s) \mid \langle \mathrm{q}\colon d \rangle \in e\}$ ▷ Entities that reference $\rho$ via d, i.e. that can take $v^*$
    for each $e' \in E$ do        ▷ Create one state for each $e'$ that can take $v^*$
        $e'' \leftarrow e' \oplus \langle \mathrm{q}\colon \mathrm{d}_{v^*} \rangle$        ▷ New entity with constant value $v^*$ at $q$
        $s' \leftarrow s \uplus [\![ 1e' ]\!] \oplus [\![ 1e'' ]\!]$        ▷ New structure where one entity has value $v^*$
        $p' \leftarrow \frac{s\#e'}{n} p$        ▷ Probability of $l'$, see Equation 20
        $P' \leftarrow P' \cup \{((s', \gamma'), p')\}$        ▷ Collect all split results
    if $n > \sum_{e' \in E} s\#e'$ then
        $p' \leftarrow 1 - \frac{\sum_{e' \in E} s\#e'}{n}$
        $P' \leftarrow P' \cup \{((s, \gamma'), p')\}$        ▷ Collect all split results
    return $P'$
```

value v^*, or no entity structure takes value v^* (if there are fewer entities than the number m of values in the urn).

Thus, the split can be performed as follows: For each entity structure $e_i \in \mathrm{dom}(s)$ that has a property q' that is distributed according to ρ, generate a split component l_i, where a single instance of e_i is removed. Then, insert an entity with multiplicity of 1 that is identical to e_i, except that q is distributed according to δ_{v^*}. Let $k_i = s\#e_i$ be the multiplicity of e_i in $l = (s, \gamma)$, and n be the total number of elements in the urn. The weight of split component l_i, i.e. the probability that v^* is taken by any of the entities e_i, is the hypergeometric distribution of choosing v^* once, and choosing $k_i - 1$ other values for e_i:

$$w_i = \frac{\binom{1}{1}\binom{n-1}{k_i-1}}{\binom{n}{k_i}} = \frac{k_i}{n} \tag{20}$$

Finally, if m is larger than the total number of all entity structures e_i in l, generate an additional split component, where v^* is removed from the urn (i.e. v^* is not taken by any entity). The weight of this split component is the remaining probability mass, i.e. $1 - \frac{\sum_{e'} s\#e'}{n}$. The algorithm is shown in Algorithm 4.

When proceeding like this, the number of split components is identical to the number of different entity structures e_i in l that have some property distributed according to ρ. We assume that in many cases, there are few such entity structures, as opposed to the large number of combinations of values of length $s\#e$ that need to be considered in the general case. Note that in the case where the values represent *unique* identifiers of the entities, $n_1 = \cdots = n_m = 1$, i.e. we can *always* use this splitting procedure instead of the general case. The following example illustrates this splitting procedure.

Algorithm 5 Lifted Marginal Filtering.

```
• Input: Actions $A$, observation model $o$, prior distribution $p(X_0)$ represented as $P_0 = \{(l_0^{(i)}, p_0^{(i)})\}_{i=1}^N$, sequence of observations $y_1, \dots, y_T$
• For $t = 1, \dots, T-1$
    1. Prediction
        – Split $P_t$ on each action precondition (see Algorithms 2, 3 and 4)
        – Calculate prediction (see Algorithm 1): $P_{t+1|t} = \textsc{predict}(P_t, A)$
    2. Update
        – Split $P_{t+1|t}$ on observation model constraints
        – Update weights of each $(l_{t+1|t}^{(i)}, p_{t+1|t}^{(i)}) \in P_{t+1|t}$: $p_{t+1}^{(i)} = p(y_t \mid l_{t+1|t}^{(i)})\, p_{t+1|t}^{(i)}$
        – Let $P_{t+1}^* = \{(l_{t+1|t}^{(i)}, p_{t+1}^{(i)})\}_{i=1}^M$
    3. Prune $P_{t+1}$, e.g. by keeping $N$ states with highest probability (or more elaborate pruning strategy, see Fearnhead and Clifford (2003))
```

Example 18. Consider the lifted state

$$l = (\llbracket\, 3\langle \text{N: } \mathbf{N}, \text{L: } X\rangle, 2\langle \text{N: } \mathbf{N}, \text{L: } Y\rangle \,\rrbracket, \langle \mathbf{N}\text{: } \mathcal{U}(3A, 2B, 1C)\rangle).$$

We want to split l, based on the constraint $c(e) = (e(N) == C)$ for the entity structure $\langle \text{N: } \mathbf{N}, \text{L: } X\rangle$. Note that $n_C = 1$. Applying the split variant outlined above leads to three split components (one for each entity structure that references $\mathbf{N}$, and one for the case where C is not taken by any of the entities):

$$\begin{aligned}
l_1 &= (\llbracket\, 1\langle \text{N: } C, \text{L: } X\rangle, 2\langle \text{N: } \mathbf{N}, \text{L: } X\rangle, 2\langle \text{N: } \mathbf{N}, \text{L: } Y\rangle \,\rrbracket, && \langle \mathbf{N}\text{: } \mathcal{U}(3A, 2B)\rangle)\\
l_2 &= (\llbracket\, 3\langle \text{N: } C, \text{L: } X\rangle, 1\langle \text{N: } C, \text{L: } Y\rangle, 1\langle \text{N: } \mathbf{N}, \text{L: } Y\rangle \,\rrbracket, && \langle \mathbf{N}\text{: } \mathcal{U}(3A, 2B)\rangle)\\
l_3 &= (\llbracket\, 3\langle \text{N: } \mathbf{N}, \text{L: } X\rangle, 2\langle \text{N: } \mathbf{N}, \text{L: } Y\rangle \,\rrbracket, && \langle \mathbf{N}\text{: } \mathcal{U}(3A, 2B)\rangle)
\end{aligned}$$

The probability of each split component is defined by Equation 20, i.e. $w_1 = 3/6$, $w_2 = 2/6$ and $w_3 = 1/6$. ○

5.3 The Lifted Marginal Filtering Algorithm

We conclude this section with summarizing the overall Lifted Marginal Filtering (LiMa) algorithm. LiMa (see Algorithm 5) performs marginal filtering (using the MRS-based transition model and constraint-based observation model, see Section 3) directly on the lifted representation, performing splitting operations when necessary. Specifically, splitting can be required for both the prediction step (to make sure that the preconditions of all actions are determinate) and the update step (for the constraints of the observation model).

The LiMa algorithm directly lends itself to an approximate version: Just as in the (ground) marginal filter, the number of states can be limited by a *pruning* operation, e.g.

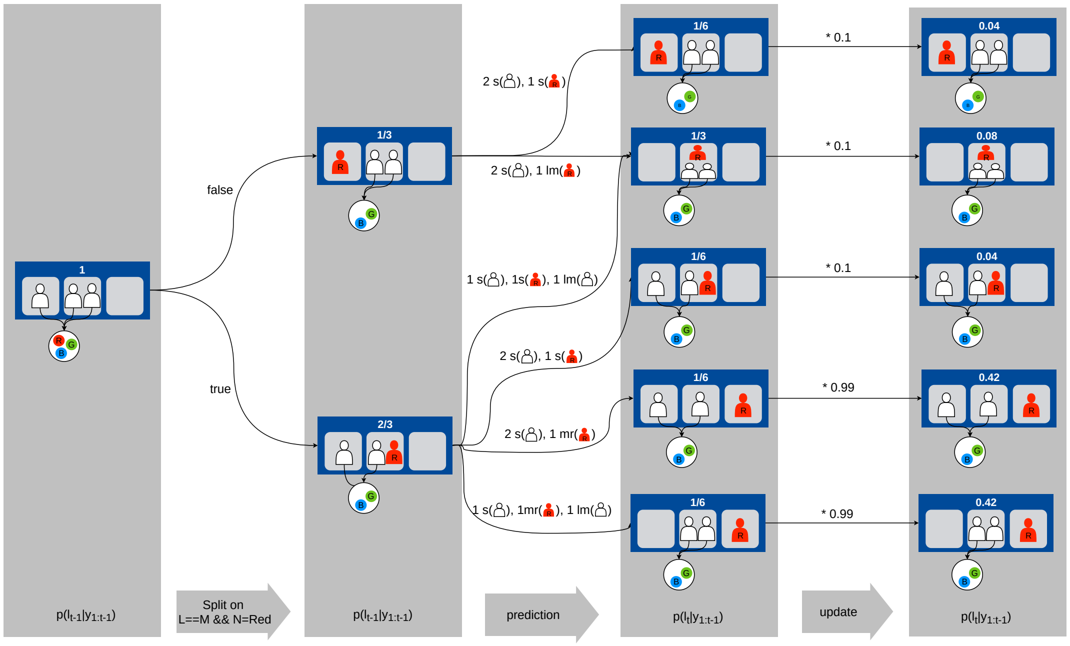


Figure 4: The split-prediction-update cycle of LiMa for the scenario described in Example 19. Here, RGB denotes an urn without replacement containing the three elements R, G and B.

by keeping only the N most likely states at each time step (see Fearnhead & Clifford, 2003, for a discussion of more elaborate pruning strategies).

Example 19. We consider a variant of the office scenario (Example 6), where agents are described by a location (L) and a name (N). Agents can be at either of three positions (L, M, R) and three actions can be performed: Staying at the current position (s), moving from the left to the middle position (lm), and moving from the middle to the right position (mr). All three actions have identical weight. The first two actions can be performed by any agent (that is at the corresponding position), while the latter action (mr) can only by performed by agent Red (only this agent is authorized to access the right location).

The right room is observed by a presence sensor that indicates whether at least one agent is at the corresponding location. The sensor has a false positive rate of 0.1 and a false negative rate of 0.01.

Suppose that the prior state distribution $p(L_{t-1} \,|\, y_{1:t-1})$ consists of only a single lifted state $l = (s, \gamma)$ (i.e. $p(L_{t-1} = l \,|\, y_{1:t-1}) = 1$) with

$$s = [\![\, 2\langle \text{N: } \mathbf{N}, \text{L: } L\rangle, 1\langle \text{N: } \mathbf{N}, \text{L: } M\rangle \,]\!], \gamma = \langle \mathbf{N}\text{: } \mathcal{U}(R, G, B)\rangle.$$

The precondition $c(e) = (e(L) == M) \wedge (e(N) == R)$ – the agent must have location L and name R – of the action mr is indeterminate for the entity $e = \langle \text{N: } \mathbf{N}, \text{L: } M\rangle$. Thus, a split of e on c is performed, resulting in two lifted states $l_1 = (s_1, \gamma_1)$ and $l_2 = (s_2, \gamma_2)$ with

$$s_1 = [\![\, 2\langle \text{X: } \mathbf{N}, \text{Y: } 1\rangle, 1\langle \text{X: } A, \text{Y: } 2\rangle \,]\!], \gamma_1 = \langle \mathbf{N}\text{: } \mathcal{U}(B, C)\rangle,$$
$$s_2 = [\![\, 1\langle \text{X: } A, \text{Y: } 1\rangle, 1\langle \text{X: } \mathbf{N}, \text{Y: } 1\rangle, 1\langle \text{X: } \mathbf{N}, \text{Y: } 2\rangle \,]\!], \gamma_2 = \langle \mathbf{N}\text{: } \mathcal{U}(B, C)\rangle$$

and weights $w_1 = 1/3$, $w_2 = 2/3$. In l_1, two compound actions are applicable, and four compound actions are applicable in l_2. Applying the compound actions leads to five states with non-zero weight. In general, it could be necessary to split on the observation model constraints at this point. However, the preconditions of the presence sensor observations do not require a split. Thus, we can directly weight each of the lifted states l_t by the observation likelihood $p(Y_t = 1 \mid l_t)$. See Figure 4 for an illustration of this example. ○

Complexity of Lifted Filtering Finally, we discuss the time and space complexity of the LiMa algorithm. First, it is easy to see that the representation complexity of the lifted representation (i.e. the number of states that need to be maintained explicitly) is never larger than the representation complexity of the original, ground representation: In the worst case, each lifted state represents exactly one ground state, so that both representations coincide.

On the other hand, the representation complexity of the lifted representation can be substantially smaller than the original, ground state representation.

Example 20. Consider a lifted state where the context contains only delta distributions, except for a single factor, which represents a uniform distribution over permutations of n values. This lifted state represents $n!$ ground states, i.e. the lifted representation is smaller than the ground representation by a factor of $n!$. As a specific example, the lifted state $l = (s, \gamma)$ with

$$s = [\![\, 1\langle \mathrm{X}\colon \mathbf{N}, \mathrm{Y}\colon 1\rangle, 1\langle \mathrm{X}\colon \mathbf{N}, \mathrm{Y}\colon 2\rangle, 1\langle \mathrm{X}\colon \mathbf{N}, \mathrm{Y}\colon 3\rangle \,]\!], \qquad \gamma_1 = \langle \mathbf{N}\colon \mathcal{U}(A, B, C)\rangle$$

represents a distribution over $3! = 6$ ground states. ○

More generally, the reduction in representation complexity that is achieved by the lifted representation is proportional to the number of ground states that is represented by each lifted state (i.e. the cardinality of the *region* of the lifted state). An upper bound on this number can be given as follows: For a factor ρ, let $|\rho|$ denote the support of ρ, i.e. the number of distinct value sequences can be drawn from the factor. The number of ground states $|\text{region}(l)|$ represented by a lifted state l can then be up to[11]

$$|\text{region}(l)| \leq \prod_{(d,\rho)\in\gamma} |\rho|$$

As the support of ρ is typically exponential in the number of RVs of ρ (for example, a multinomial distribution of n values from which we draw m times has a support of n^m, and a hypergeometric distribution over n unique values has a support of $n!$), the representation complexity of the lifted representation can be substantially smaller than the ground representation.

Runtime of the LiMa algorithm is linear in the number of states, as compound actions need to be computed individually for each state during the prediction step. Therefore, the results for representation complexity directly transfer to results for algorithm runtime: The time complexity of LiMa is smaller by a factor of $|\text{region}(l)|$ compared to ground filtering.

[11] This upper bound ignores the fact that multiple value sequences from ρ can be mapped to the same *canonical* sequence (and thus the same ground state). The bound holds exactly when each value sequences corresponds to exactly one canonical value sequence, as in Example 20.

Note that this does not mean that the complexity of lifted filtering grows only polynomially with respect to the number of entities in the state. Instead, lifted inference complexity can still grow exponentially with the number of entities, when there is at least one property that is represented explicitly (i.e. via delta distributions). This behavior can, for example, be observed in the experimental evaluation of the tracking scenario (Section 6.2): In that scenario, complexity of lifted inference is smaller by a factor of $n!$ (where n is the number of agents) than ground inference, because the distribution over the agents' names is represented efficiently, but complexity of lifted inference still grows exponentially with the number of agents, because of the exponential number of explicitly represented joint assignments of the *location* property.

Furthermore, *splitting* increases the representation complexity. In the worst case, repeated splitting (of all properties) results in the degeneration of all distribution representations to delta distributions (which have a support of 1), so that representation complexity and runtime complexity of lifted and ground filtering coincide.

In summary, space and time complexity of lifted filtering can be substantially smaller than complexity of ground filtering, by a factor that is proportional to the support of the distributions in the context (which is typically exponential for distributions other than delta distributions). However, when repeated splitting of all properties is required, the complexity of lifted filtering can degenerate to the complexity of ground filtering. Next, we investigate how these theoretical properties of LiMa manifest empirically.

6. Experimental Evaluation

In this section, we empirically investigate whether LiMa can indeed perform more efficient inference due to the lifted state representation. Specifically,the goal of the experiments was to assess the benefit of LiMa for realistic scenarios (consisting of actual activity sequences of human protagonists, and real sensor data), instead of only running simulations. The reasoning is that in the ideal case, LiMa can obviously achieve a factorial reduction in complexity and thus we investigate here whether this increased efficiency also manifests in actual, realistic applications – that might contain a substantial amount of symmetry breaks, i.e. require splitting. We will answer the following research questions:

Q1 (Representation Size) Does the lifted state representation lead to a significantly smaller cardinality of the filtering distribution (i.e. can it achieve a higher representation efficiency) than a ground state representation?

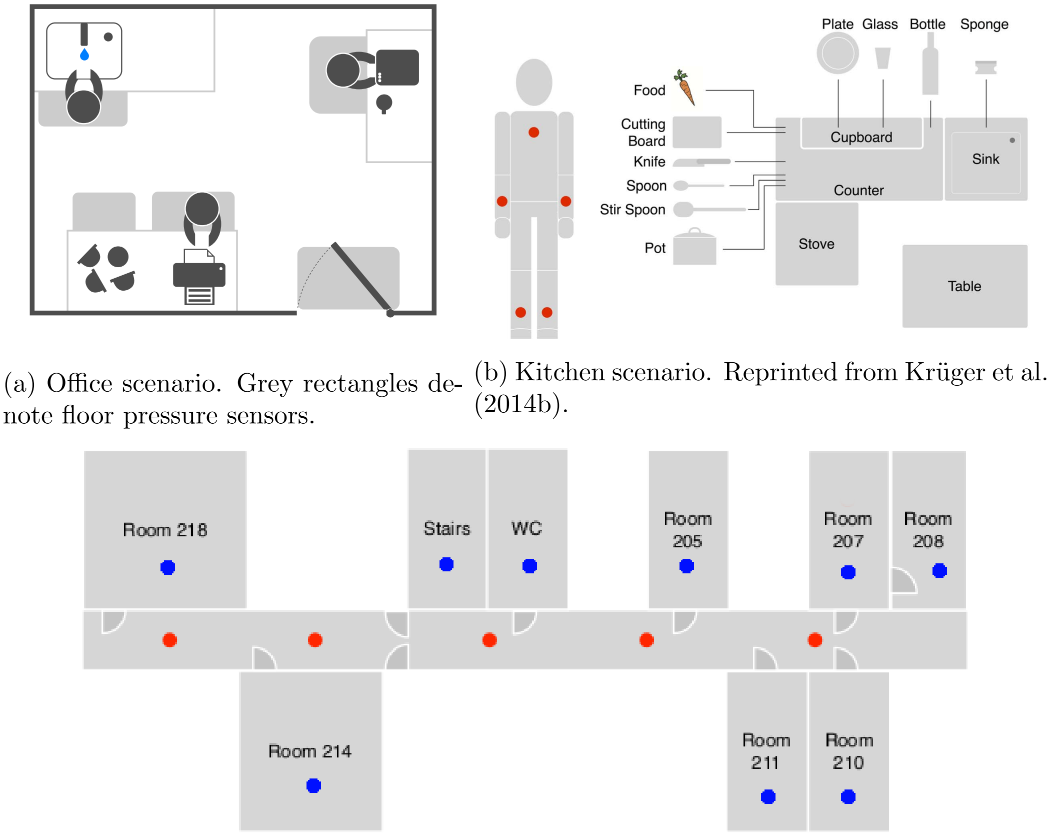
Q2 (Approximation Quality) Can LiMa achieve a more accurate state estimation, when introducing approximations by limiting the maximum number of states that represent the filtering distribution (i.e. when performing *pruning*)?

6.1 Evaluation Scenarios

To evaluate these research questions, we modeled three application scenarios in LiMa. Table 1 provides an overview of the scenarios. In the following, each scenario is described briefly, and intuition on the chosen modeling approach is provided.

Scenario	# Actions	# Entities	# Runs	Length	Data
Office	15	1-6	360	51.5	Simulated
Tracking	40	1-7	35	473.4	Real
Kitchen	72	16	7	92.6	Real

Table 1: Evaluation scenarios.



(a) Office scenario. Grey rectangles denote floor pressure sensors.

(b) Kitchen scenario. Reprinted from Krüger et al. (2014b).

(c) Tracking scenario. Dots denote locations of presence sensors. Reprinted from Lüdtke et al. (2017).

Figure 5: Evaluation scenarios

Office This simulated scenario (originally presented by Schröder et al., 2017, dataset available at Schröder et al., 2016) consists of one to six persons that act in an office environment consisting of six locations (see Figure 5a). The agents can move between locations, carry objects (coffee capsules, cups, water, paper) and perform certain activities, like brewing coffee or printing documents. In this scenario, only a single agent is acting per time step, i.e. the scenario does not have a compound action semantics. The activities have a causal structure, e.g. to make a coffee, the coffee machine must have been filled with a coffee capsule. Each location is equipped with a presence sensor (e.g. a floor pressure sensor), that indicates whether at least one person is present at that location. The sensor data is always correct, i.e. there are no false positives or false negatives.

The scenario is modeled in LiMa such that each agent and coffee capsule is represented by a separate entity. The entities corresponding to agents have properties describing their name, location and whether they hold an object. As the sensor data do not allow to distinguish *which* person (and coffee capsule) is at each location, the identities of the agents and capsules are modeled by an urn without replacement.

Tracking This scenario (originally presented by Krüger et al., 2014a, dataset available at Kasparick & Krüger, 2013) is also concerned with tracking the locations of agents (but no other context information, like in the previous scenarios). Here, 14 locations are available (see Figure 5c). The data for this scenario is based on real, observed motion trajectories. Trajectories for one to seven persons have been recorded, that are moving simultaneously in the environment. For each number of persons, five trajectories have been obtained, i.e. there is a total of 35 data sets.

For each recorded trajectory, sensor observations of presence sensors located at each of the five corridor locations (see Figure 5c) have been simulated. As is the office scenario, the observations are always correct, i.e. there are no false positives or false negatives. The mean length of the observation sequences is 474.3 time steps, i.e. more than 15 times the length of the previous scenarios. In this scenario, the only actions that agents can perform is moving between locations. The probability of each action (moving between two specific rooms, or not moving) has been estimated empirically from the data. The scenario is modeled such that each agent is represented by a separate entity with properties describing their identity and location. Again, we chose an urn without replacement to model the distribution of the agents' names.

Kitchen This scenario (originally presented by Krüger et al., 2014b, dataset available at Krüger et al., 2015) serves as a large, real-world evaluation of LiMa. The task here is to perform activity and context recognition in a kitchen scenario (see Figure 5b), where an agent performs the subtasks (i) preparing the kitchen, (ii) cooking, (iii) preparing the table, (iv) eating, and (v) washing the dishes.

Experiments with 7 participants have been performed. The participants performed 16 different types of actions, e.g. *take*, *move* or *fill*. The participants were instrumented with 5 inertial measurement units (IMUs), recording linear acceleration and angular velocity (3 axis each) at 120 Hz. Out of the 30 IMU signals, 180 features such as variance and energy were computed with a window size of 128 samples and 75% overlap. Afterwards, a principal component analysis was performed, and the 21 principal components with the largest eigenvalues were selected.

In the resulting dataset, each action has a distinct *duration* distribution, which is not necessarily a geometric distribution, thus requiring to model the duration distribution explicitly, by concepts similar to hidden semi-Markov models (Yu, 2010). This, however, would add additional parameters, which we wanted to avoid in the context of this evaluation. Therefore, we reduced the dataset so that each action lasts for exactly one timestep, by sampling one observation for each segment where the same action is executed.

We modeled the domain as a PMPMRS as follows: Each of the 10 objects shown in Figure 5b, as well as the agent, is modeled as an entity (with properties like location, clean/dirty, cooked, etc.). For example, a sub-multiset of a reachable state in this PMPMRS is

$$\begin{aligned} x = \llbracket\, & 1\langle \text{N: Spoon}, \text{Dirty: yes}, \text{Pos: Sink}\rangle, \\ & 1\langle \text{N: Plate}, \text{Dirty: yes}, \text{Pos: Sink}\rangle, \\ & 1\langle \text{N: Pot}, \text{Dirty: yes}, \text{Pos: Counter}\rangle, \ldots \rrbracket \end{aligned} \tag{21}$$

A lifted state representation is obtained by representing the identities of objects by an urn without replacement (i.e. a uniform distribution of permutations). The action weights were chosen according to a goal distance heuristic, where the goal is that the meal has been finished and all objects are washed. We investigated two different observation models $p(y_t \mid l_t)$: (i) *Crisp* observations of the actual action c_t, i.e. $p(y_t \mid l_t) = \mathbb{1}(a_t{=}y_t)$ where a_t is the action executed in state l_t and y_t is the actually executed action; and (ii) we used the preprocessed real sensor data as observations, and assumed $p(y_t \mid l_t)$ to be a multivariate normal distribution (conditional on the executed action), i.e. $p(y_t \mid l_t) \sim \mathcal{N}(\mu_{a_t}, \Sigma_{a_t})$, where y_t are the preprocessed sensor data, a_t is the action executed in l_t, and the parameters μ_{a_t} and Σ_{a_t} have been learned from the data.

Due to splitting, over time, the distribution $p(L_t \mid y_{1:t})$ consists of lifted states where the names of some entities follow a joint hypergeometric distribution, while names of other entities follow a delta distribution, as in the following example:

$$\begin{aligned} l_1 &= (\llbracket\, 1\langle \text{N: } \mathit{Spoon}, \text{Pos: } S\rangle, 2\langle \text{N: } \mathbf{N}, \text{Pos: } C\rangle \,\rrbracket, && \langle \mathbf{N}\text{: } \mathcal{U}(\mathit{Plate}, \mathit{Pot})\rangle) \\ l_2 &= (\llbracket\, 1\langle \text{N: } \mathbf{N}, \text{Pos: } S\rangle, 1\langle \text{N: } \mathit{Plate}, \text{Pos: } C\rangle, 1\langle \text{N: } \mathbf{N}, \text{Pos: } C\rangle \,\rrbracket, && \langle \mathbf{N}\text{: } \mathcal{U}(\mathit{Spoon}, \mathit{Pot})\rangle) \end{aligned}$$

For this scenario, we perform an approximation that projects the joint distribution of names back to a single hypergeometric distribution (i.e. an urn without replacement containing all names). This way, all lifted states that are different just in the distribution of names are projected onto the same lifted state. In that case, the probability of the new state is the sum of all states projected onto that state. In the example, both states l_1 and l_2 are projected to the lifted state $l' = (\llbracket\, 1\langle \text{N: } \mathbf{N}, \text{Pos: } S\rangle, 2\langle \text{N: } \mathbf{N}, \text{Pos: } C\rangle \,\rrbracket, \langle \mathbf{N}\text{: } \mathcal{U}(\mathit{Spoon}, \mathit{Plate}, \mathit{Pot})\rangle)$, and $p(l') = p(l_1) + p(l_2) = 1$.

Here, we manually examine the causal structure of the actions, and identify five situations where splitting (for a specific value) will not occur in the future. At those situations, the projection operation can be performed safely, without splitting directly afterwards. For example, once the cooking process is finished, it is not necessary to distinguish the pot from the other objects, so the pot's identity does not need to be represented explicitly any more.

6.2 Exact Inference

Experimental Setup For assessing **Q1**, we performed *exact* filtering (i.e. without pruning) using the lifted and the ground state representation (i.e. the conventional marginal filtering algorithm). Note that both cases represent exactly the same distribution, via Equation 11.

We did not compare LiMa to any other BF algorithm apart from marginal filtering, because no other filtering algorithm is directly applicable to these large state spaces and structured, causal system dynamics: BF algorithms that enumerate the transition model as

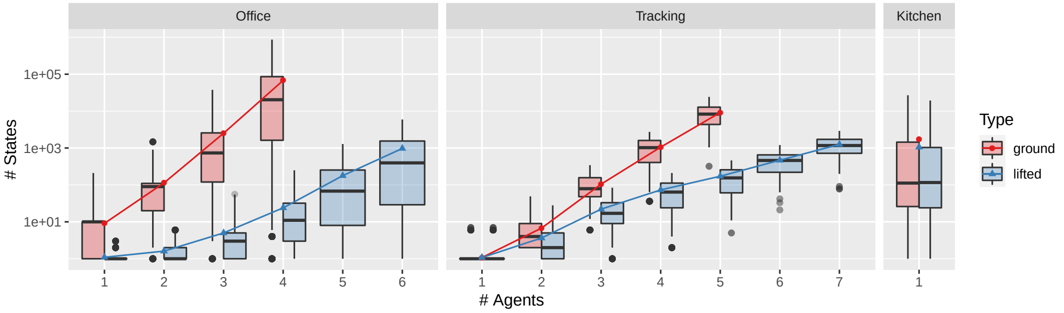


Figure 6: Exact filtering results: Number of states required for exact filtering, using lifted and ground state representations. The line shows the mean number of states for each configuration. Note the logarithmic scale of the y axis. Ground filtering has been infeasible for 5 and 6 agents (office) or for 6 and 7 agents (tracking).

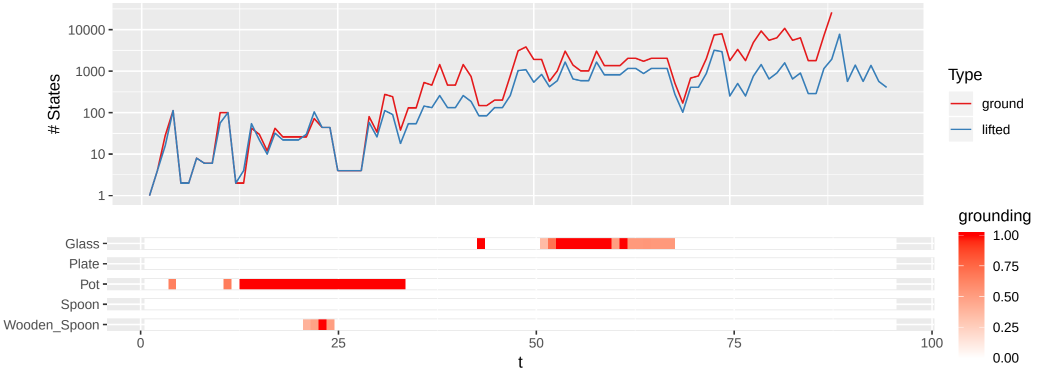


Figure 7: Top: Number of states required for filtering in the kitchen scenario over time for subject 6, using lifted and ground states. Bottom: Fraction of states in which each object identity is represented explicitly. Ground filtering is stopped at $t = 89$, because it exceeds 50,000 states.

a matrix are infeasible due to the large state space size. Approximate algorithms like particle filtering can in principle be used, but Nyolt et al. (2015) already showed that marginal filtering is superior to particle filtering in categorical state spaces and thus, it is sufficient to use (ground) marginal filtering for comparison. We compared the number of explicitly represented (ground or lifted) states that are necessary to represent the distribution as a measure of inference efficiency (runtime of the filtering algorithm is linear in the number of states).

We used all three scensarios for evaluation. For the kitchen scenario, we used the crisp observation model (observing the actual actions), to keep ground filtering feasible. For the office and tracking scenarios, we aggregated all runs that involve the same number of agents. All experiments have been performed using an implementation of LiMa in Haskell.

Results Figure 6 shows the number of states that are necessary for exact filtering in the three scenarios, with respect to their state space size (which depends on the number of agents). The figure shows that the necessary number of states is several times smaller when the lifted state representation is used. This difference becomes more pronounced with increasing number of agents: For the office scenario with 1 (2, 3, 4) agents, the ground representation needs 8.3 (70.7, 503, 2842) times the number of states than the lifted representation. For the tracking scenario with 2 (3, 4, 5) agents, this factor is 1.8 (4.8, 14.4, 53.3). For the kitchen scenario, the difference in the number of states is not as pronounced: The mean number of states for lifted filtering is 1054.7, as compared to 1709.8 states for ground filtering.

Figure 7 (top) shows the required number of states over time for the kitchen scenario. We can observe that specifically in the situations where filtering is “difficult” (i.e. where the number of possible states is high), the lifted state representation leads to a much lower number of state representatives. Figure 7 (bottom) shows the fraction of states in which each of the object identities is represented explicitly. This plot shows that at each point in time, it is sufficient to know the identities of just some of the objects. For example, during cooking, it is necessary to know that the object on the stove is the pot, but this can be safely forgotten later on.

Discussion The results show that the lifted state representation can lead to more efficient inference when the scenario permits this: The office and tracking scenarios contain many entities (agents, coffee capsules), leading to a combinatorial explosion in the support of the ground distribution. As their identities be discriminated by observations and do not need to be discriminated due to the system dynamic, the lifted representation can reduce the complexity by an exponential factor. In the real-world scenarios, this difference is not as pronounced, as there are fewer symmetries that can be exploited, required number of states (and thus inference runtime) can still be reduced considerably.

Still, even when using the lifted representation, the required number of states increases exponentially with the number of agents in the office and tracking scenarios. This relationship is due to the exponential number of possible joint assignments of the *location* properties of the entities, as discussed in Section 5.3: Even the lifted representation needs to distinguish between the situations “two agents are at location A, one agent is at location B”, and “one agent is at location A, two agents are at location B” explicitly. Therefore, even lifted exact filtering can be infeasible for very large state spaces – thus requiring approximate filtering, as discussed next.

6.3 Approximate Inference

Experimental Setup For assessing **Q2**, we performed approximate filtering for the tracking and kitchen scenarios (for the latter, the real sensor observations were used). That is, the algorithm performed *pruning* to a fixed number of states after each update step. Specifically, the unbiased and optimal pruning strategy (with respect to least squared error) proposed by Fearnhead and Clifford (2003) was used here.

We then computed an estimate from the filtering distribution, that represents a measure that is relevant for answering application specific questions: For the tracking scenario, we computed the number of persons per room, and for the kitchen scenario, we computed an estimate of the action class that was performed. The “quality” of the approximation

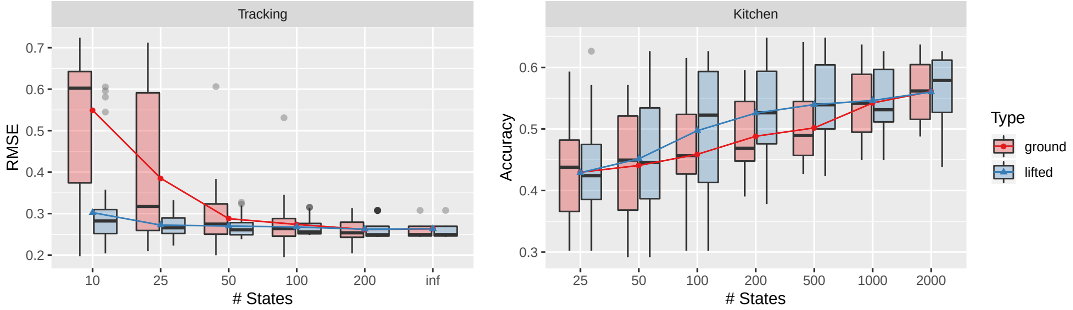


Figure 8: Approximate filtering results: RMSE/Accuracy of tracking/kitchen scenario with respect to available states. For the tracking scenario, "inf" denotes an unlimited number of states (exact filtering), i.e. a lower bound on RMSE.

is assessed on this estimate (in terms of root mean squared error or accuracy, explained below).

For the tracking scenario, we investigate the root mean squared error (RMSE) of the number of agents per room. Let n_L be the overall number of locations, let $n_{r,t}$ be the true number of agents at location r at time t, and let $\hat{n}_{r,t}$ be the point estimate of the number of agents at room r for an approximate filtering distribution $\hat{p}(l_t \mid y_{1:t})$. The RMSE is then

$$\text{RMSE} = \sqrt{\frac{\sum_{t=1}^{T} \sum_{r=1}^{n_L} (n_{r,t} - \hat{n}_{r,t})^2}{T * n_L}}. \tag{22}$$

The RMSE is 0 when exactly the right number of agents is estimated per room. Only the 5 runs where 5 agents are present simultaneously were used in this experiment, as they are the largest runs that afford exact ground inference. We performed experiments with a maximum of 10, 25, 50, 100 and 200 states (at the pruning step). Thus, $5 * 10 * 5 * 2 = 500$ experiments were performed for this scenario.

For the kitchen scenario, we assess the *accuracy* of estimating the performed action (out of 16 available actions): Let a_t be the true action class performed at time t, and let $\hat{a}_t$ be the point estimate of the performed action class (i.e. the most likely action class) for an approximate filtering distribution $\hat{p}(l_t \mid y_{1:t})$. The accuracy is then

$$\text{Accuracy} = \frac{\sum_{t=1}^{T} \mathbb{1}(a_t = \hat{a}_t)}{T} \tag{23}$$

We performed experiments with a maximum of 25, 50, 100, 200, 500, 1000 and 2000 states i.e. $7{*}10{*}7{*}2 = 980$ experiments were performed for this scenario, and thus 1480 experiments were performed overall for assessing **Q2**.

Results Figure 8 shows the RMSE or accuracy for both scenarios, and different numbers of available states. In the tracking scenario, RSME decreases when the number of states is increased. For 10, 25 and 50 states, RMSE of lifted filtering is significantly lower than ground filtering ($p < 0.05$, $n = 50$ using Wilcoxon signed rank test). When sufficiently many states are available, both algorithms reach a saturation where a further increase in the number of states has no significant effect on performance. However, lifted filtering reaches this saturation with fewer states (no significance difference of RMSEs when using 25 and 50 states) than ground filtering (no significance difference of RMSEs when using 50 and 100 states). Furthermore, for low numbers of states, the variance of the RMSE is much higher for ground filtering than for lifted filtering.

The kitchen scenario shows a similar behavior: Increasing the number of states increases accuracy. For 100 and 200 states, accuracy of lifted filtering is significantly higher than ground filtering ($p < 0.05$, $n = 70$ using Wilcoxon signed rank test). For fewer states (25 and 50), there is no significant difference, as the number of states is insufficient for both algorithms to achieve reasonable results. For > 200 states, there is also no significance difference, as both algorithms eventually reach a saturation state.

Discussion The results show that in some cases, LiMa exhibits a significantly smaller estimation error with lower variance than ground marginal filtering, and is never significantly worse (using $\alpha = 0.05$). The reason for this is that intuitively, LiMa can represent the filtering distribution more accurately with a given number of states: Given a fixed number n of states, the support of the ground filtering distribution is exactly n. Instead, LiMa can maintain a filtering distribution with support $> n$, as each lifted state can represent a distribution over multiple ground states.

6.4 Summary

The empirical evidence shows that LiMa can indeed achieve a lower representational complexity (or lower error in the approximate case) for the realistic applications investigated here. More specifically, the following conclusions can be drawn:

- As long as any exchangeability is present in the filtering distribution, LiMa needs fewer states to represent the exact filtering distribution. In the worst case, when no exchangeability can be exploited, LiMa coincides exactly with ground marginal filtering.

- When limiting the number of available states to a fixed number, estimates calculated with LiMa can have a lower variance and lower error than ground marginal filtering. This behavior is not present for very low numbers of states (where both methods cannot follow the causally correct sequence) or for very high numbers (as both algorithms are saturated).

7. Related Work

The goal of LiMa is to perform efficient BF in MRSs (or, more generally speaking, systems where the system dynamics is given by an algorithmic description). Technically, this is achieved by performing Rao-Blackwellization on distributions of multiset states. Thus, there are a number of relevant, related lines of research: *Probabilistic Programming Languages* also describe distributions algorithmically; *Lifted probabilistic inference* algorithms are concerned with exploiting exchangeability for probabilistic inference (e.g. in graphical models); *Relational Filtering* algorithms perform BF in systems where states can be described by relational logic; and *Data Association* methods allow efficient BF for distributions of permutations. In the following, we discuss each of the approaches in more detail.

Probabilistic Programming Languages Probabilistic Programming Languages (PPLs) describe, similar to LiMa, a complex distribution by the algorithmic process that generates the distribution. There are two branches of PPLs: *Probabilistic Logic Programs* (like ProbLog, Fierens et al., 2015 or Prism, Sato & Kameya, 2008) add probabilistic annotations to facts of a logic program. Imperative and functional PPLs (e.g. Pyro, Bingham et al., 2019 or Church, Goodman et al., 2008) are based on general-purpose programming languages, and allow arbitrary constructs like loops and branches.

Thus, the underlying distributions can be complex, and inference is often computationally expensive. Inference methods either perform exact or approximate (sampling-based) path enumeration (Wood et al., 2014; Gehr et al., 2016), but these methods cannot exploit the structure of the underlying distribution for more efficient inference. Alternatively, inference algorithms can compile the program into a symbolic representation like a probabilistic graphical model (McCallum et al., 2009; Pfeffer, 2009) or a Binary Decision Diagram, and then perform inference on this representation. This way, conditional and context-specific independence of the distribution can be exploited. However, we are not aware of any inference algorithms for PPLs that exploit *exchangeability* that arises due to the algorithmic description, which was the main motivation for developing LiMa.

Lifted Probabilistic Inference The goal of *Lifted Probabilistic Inference* is to exploit *symmetries* in graphical models for efficient inference. For symmetrical graphical models – containing identical sub-graphs – it is sufficient to perform inference on one representative instance of the repeated sub-graphs, as inference computations are identical on all instances. Furthermore, the number of identical instances needs to be counted, to correctly consider the influence they have on the overall inference result. In the last 15 years, a large number of lifted inference approaches have been devised, e.g. lifted versions of Variable Elimination (Poole, 2003; Milch et al., 2008; Taghipour et al., 2014), Belief Propagation (Singla & Domingos, 2008; Kersting et al., 2009), Recursive Conditioning (Poole et al., 2011), and Weighted Model Counting (Gogate & Domingos, 2011; Van Den Broeck et al., 2011).

Symmetries in graphical models correspond to (partial) exchangeability of the distribution they represent. Thus, alternatively, lifted inference can be characterized as performing the following steps, as outlined by Niepert and Van den Broeck (2014): Decompose a distribution into exchangeable components, construct a sufficient statistics of these components, generate all values of the statistics and count the number of instances for each value of the statistic. The lifted state representation in LiMa follows exactly this scheme. The multiset structures s correspond to sufficient statistics of the ground states x. Similarly, compound action computation is also an instance of this scheme. A compound action is basically a sufficient statistic of its corresponding ground compound actions, and its multiplicity corresponds to the number of its instances (i.e. the number of corresponding ground compound actions).

In several special cases, the lifted state representation of LiMa can be encoded as a lifted graphical model, e.g. as a parfactor graph with generalized counting formulae (Taghipour et al., 2014). Basically, each multiset structure s corresponds to a row in a parfactor in generalized counting form, and uniform distribution of identities can be encoded as the logical variables of the parfactor. However, the system dynamics cannot be compactly expressed as a lifted graphical model. Expressing the hard constraints arising in compound action computation (each entity must perform exactly one action and can only perform actions whose preconditions it satisfies) as a graphical model results in very large graphical models even for simple scenarios. Instead, LiMa works directly with the parallel MRS formulation, allowing to express the transition model compactly via a *computational* description.

In summary, LiMa is based heavily on concepts devised for lifted inference algorithms. However, existing lifted inference algorithms cannot be used directly for the application domains we are considering in this work. Specifically, (lifted) graphical model cannot express the rules-based system dynamics directly.

Relational Filtering There are a number of approaches for inference in dynamic systems (i.e. BF) that are, similar to LiMa, concerned with maintaining compact distribution representations.

The Relational Kalman filter (Choi et al., 2011, 2015) is based on lifted inference, more specifically continuous First-Order Variable Elimination (continuous FOVE) (Choi et al., 2010). The filtering distribution is modeled as a relational pairwise model (RPM), an extension of parfactor graphs where the parfactors are products of normal distributions. A RPM essentially represents a multivariate normal distribution with additional independence assumptions. Prediction and update operations of BF are defined by using continuous FOVE whenever marginalization is required. The approach is limited to Gaussian filtering distributions and a linear transition model (just like the standard Kalman filter).

Stochastic Relational Processes (SRPs) (Thon et al., 2011) describe states by logical interpretations, and use causal probabilistic time logic (CPT-logic) to describe state dynamics. A theory in CPT-logic (i.e. a representation of the transition model) consists of a set of *rules*. Each rule consists of a *body*, describing what must hold in a state for the rule to be applied, and *head* elements, each one describing a possible effect of the rule. This formalism is closely related to the rule-based transition model used in LiMa. Like MRS rules, CPT-logic rules describe the transition model computationally – by the *function* that generates the posterior distribution. For a discussion of the relationship to CPT-logic and PDDL, see Thon et al. (2009). Furthermore, CPT-logic allows multiple rules to be applied in a single prediction step, similar to *parallel* MRSs (although the semantics is different in detail, e.g. a single fact of an interpretation can be used to satisfy multiple CPT-logic rule bodies). In contrast to the lifted states used in LiMa, SRPs use a ground state representation – although a part of the transition model can be calculated in a lifted way (for calculating the successor states, not all ground rules need to be generated). Furthermore, CPT-logic is not concerned with *updating* a distribution on observed values.

The Relational Particle Filter (RPF) (Nitti et al., 2016) is another BF algorithm that uses compact descriptions of a distribution. Specifically, it uses *distributional clauses* to represent states, the transition model and the observation model. A distributional clause represents a distribution, similar to the lifted states of LiMa, where each state also describes a distribution of ground states. Thus, the RPF can be seen as an instance of a Rao-Blackwellized Particle Filter – except that it does not rely on a fixed segmentation of sampled and exactly maintained variables, as the transition model might require to sample from the parametric distributions. The RPF maintains univariate parametric distributions, and is not concerned with efficiently handling exchangeable distributions, as done in LiMa.

Data Association These approaches are concerned with efficiently representing distributions of permutations, and BF with such distributions. Specifically, they handle the following problem: Given a number of *tracks* $t_1, \ldots, t_n$ (e.g. tracks of people in a video) that correspond to objects $o_1, \ldots, o_n$, they maintain the distribution of object-track associations. This is a BF problem in a state space of permutations. Two conceptually different approaches for this goal have been devised. The *Fourier-theoretic* approach (Huang et al., 2009; Kondor et al., 2007) uses a Fourier transformation over the symmetric group $\mathbb{S}_n$, the group that represents permutations of n objects. Instead of maintaining a categorical distribution $p(\sigma), \sigma \in \mathbb{S}$ of cardinality $n!$ directly, the distribution is approximated by its first few Fourier matrices, just like a function $f(x), x \in \mathbb{R}$ can be approximated by its first few Fourier coefficients. The *information-theoretic* approach maintains a compact representation of the distribution over permutations by an information matrix Ω. The information matrix contains unnormalized marginal probabilities Ω_{ij} for each association of track i with object j. For both approaches, prediction and update steps can be defined directly for the compact representation, without requiring the original, much larger distribution.

Similar to LiMa, these approaches aim at a compact representation of the filtering distribution. More specifically, exchangeability that arises due to handling distributions of permutations has been one of the main motivations for developing LiMa. Uniform distributions of permutations (or, more generally, exchangeable distributions) are represented very efficiently in LiMa. The more asymmetrically the distribution becomes, the higher the more states need to be maintained by LiMa (as the representation becomes more and more ground). Instead, Data Association methods can maintain an efficient representation even in those cases. However, they achieve this efficiency due to the approximations that are performed, whereas LiMa is exact. Data Association methods can only handle state spaces of permutations, and specific system dynamics that model *mixing* of tracks, whereas LiMa can model more general state spaces and transition models. Still, combining Data Association methods with LiMa to obtain efficiency in less symmetrical cases is an interesting topic for future research.

8. Conclusion and Future Work

In this work, we presented an efficient Bayesian filtering algorithm for systems whose dynamics is represented by a probabilistic Multiset Rewriting System (MRS). Technically, we devised a suitable decomposition of multisets into a *multiset structure* and a *value sequence*. This allows to represent the distribution more efficiently, by making use of independence and exchangeability in the distribution over value sequences that naturally arise in MRSs. Multiset rewriting can be performed directly on that representation, without generating the original multisets first. We empirically showed that this approach can lead to a factorial reduction in representational complexity of the exact filtering distribution. In the approximate case, the approach can lead to a lower variance and lower error of estimates of the filtering distribution.

The modeling formalism we presented here is quite general: In principle, the approach allows to perform Bayesian filtering in all dynamic systems where the system dynamics can be formalized as an MRS, or any other symbolic or rule-based approach that can be translated into an MRS. However, not all systems that can be represented this way also show the symmetries that allow efficient inference: Symmetries can break due to the system dynamics (when action preconditions depend on specific ground states, instead of being decidable for complete lifted states), or due to the observation model (when the observation likelihood is different for all ground states). In the worst case, the algorithm resorts to the ground marginal filtering algorithm (Nyolt et al., 2015). For lifted inference, methods to cope with this problem have been proposed, that work by approximating the true distribution by a symmetric distribution, such that lifted inference is possible (Singla et al., 2014; Venugopal & Gogate, 2014). In a sense, the projection method that was used for the kitchen scenario can be seen as a first step in that direction. A future research goal is to devise more general methods for identifying and projecting onto *sufficiently similar* symmetric distributions. Furthermore, methods for detecting situations where such over-symmetric approximations are sensible (where the symmetric distribution does not need to be split immediately) can be devised, e.g. by examining the causal structure of the actions.

As repeated splitting can also be induced by the observation model $p(y_t \mid x_t)$, our future work focuses on methods to learn observation models that preserve the symmetries in the model, e.g. by constraining the distribution $p(y_t \mid x_t)$ that is learned, given the symmetries in the MRS model. Additional directions for future work are methods for learning the weights of actions, or even the actions themselves (e.g. by employing ideas for learning Markov Logic Networks, Van Haaren et al., 2016), and using other representation formalisms for $p(\mathbf{v} \mid s, \gamma)$, e.g. Sum-Product Networks (Poon & Domingos, 2011) or Exchangeable Variable Models (Niepert & Domingos, 2014).

Acknowledgments

We would like to thank Max Schröder, Sebastian Bader and Kristian Kersting for their collaboration on an earlier, much shorter version of this work that appeared in IJCAI 2018 (Lüdtke et al., 2018a).

Appendix A. Notation

The concepts in this work rely heavily on maps, multisets, and lists of variable length. Thus, we need a suitable notation for working with such objects.

Maps (partial functions) are denoted by $\nrightarrow$, e.g. the type of a map taking elements from X and mapping to elements from Y is denoted as $X \nrightarrow Y$. Furthermore, dom denotes the domain of a map (the set of all elements $x \in X$ such that $f(x)$ is defined), ran denotes the range (codomain) of the map (i.e. Y in the previous example), and img denotes its image (i.e. all $y \in Y$ for which there exists an $x \in X$ with $f(x) = y$). A map m with $m(x_1) = y_1, \ldots, m(x_i) = y_i$ is denoted as $m = \langle x_1 : y_1, \ldots, x_i : y_i \rangle$. Two maps can be combined by the operator $\oplus$, which is defined as:

$$(m_1 \oplus m_2)(k) = \begin{cases} m_2(k) & \text{if } k \in \operatorname{dom}(m_2) \text{ or } (k \in \operatorname{dom}(m_1) \land k \in \operatorname{dom}(m_2)) \\ m_1(k) & \text{if } k \in \operatorname{dom}(m_1) \land k \notin \operatorname{dom}(m_2) \\ \text{undefined} & \text{otherwise} \end{cases}$$

Algorithm 6 Enumerate compound actions.

```
function ENUM-CA($x$,$AI$,$k$)
    if $k$ maximal in $x$ with respect to $AI$ then
        return $\{k\}$                                              ▷ $k$ is AMCA
    $K \leftarrow \emptyset$                        ▷ The set where compound actions are collected
    for $\{(a,i) \in AI | i \sqsubseteq x\}$ do                  ▷ Action instances applicable to $x$
        $x' \leftarrow x \uminus i$                                         ▷ Remaining state
        $k' \leftarrow k \uplus \llbracket 1(a,i) \rrbracket$               ▷ Add action instance to compound action
        $AI' \leftarrow \{(a',i') \in AI | (a',i') \geq (a,i)\}$   ▷ Allow only $\geq$ instances in recursive call
        $K' \leftarrow$ ENUM-CA($x'$,$AI'$,$k'$)                        ▷ Recursive call
        $K \leftarrow K \cup K'$
    return $K$
```

Sequences are maps of indices to elements. We use $\langle a_1, \ldots, a_n \rangle$ as shorthand notation for $\{1 : a_1, \ldots, n : a_n\}$. The concatenation of two sequences s_1 and s_2 is denoted by $s_1 \oplus s_2$, i.e. $\langle a_1, \ldots, a_i \rangle \oplus \langle a_{i+1}, \ldots, a_j \rangle = \langle a_1, \ldots, a_j \rangle$. The length of a sequence s is denoted as $|s|$. The i-th element of the sequence s is denoted as s_i, i.e. $\langle a_1, \ldots, a_i, \ldots, a_n \rangle_i = a_i$.

Multisets are maps from elements to natural numbers (multiplicities). We write $\llbracket n_1 \, a_1, \ldots, n_i \, a_i \rrbracket$ to denote the multiset $\langle a_1 : n_1, \ldots, a_i : n_i \rangle$. The expression $x\#a$ denotes the multiplicity of element a in x, i.e. $x\#a = x(a)$. The function items(s) returns the multiset of elements in the sequence s, in which each element a appears exactly as often as a appears in s, i.e. items$(s) = \biguplus_{a \in s} a$.

Appendix B. An Algorithm for Enumerating Compound Actions

Computing the set of all AMCAs for a given state x and a set AI of action instances can also be performed by backtracking search: Start with an empty multiset k. For each action instance that still "fits in" x (such that the resulting compound action is still compatible with x), add it to k and do a recursive call with the new k and the remaining l, until the compound action is maximal.

Directly proceeding like this would produce many repetitions of the same AMCA, that are just different in the *order* in which the action instance have been inserted. As multiset insertion is commutative, the insertion order is not relevant for the resulting AMCA. Thus, we can improve the efficiency of the algorithm by defining an arbitrary order on the action instances, and allow only insertion of action instances that are not "smaller" than the action instance inserted last. This way, each AMCA is generated exactly once, and subtrees that would correspond to other insertion orders are not expanded. This procedure is shown in Algorithm 6.

This algorithm has linear time complexity in the number of AMCAs. However, this number can easily become very large, as it does not only depend on the number of distinct entities, but also on the overall number of entities in the state: It is at most the multiset coefficient $\binom{m+n-1}{n} = \frac{(m+n-1)!}{n!\,(m-1)!}$, where n is the total number of entities in the state and m is the total number of action instances. Thus, AMCA enumeration is one of the main computational challenges of the BF algorithm. An approximate algorithm that samples AMCAs instead of enumerating all AMCAs has been presented in (Lüdtke et al., 2018b).

References

Arnaud, D., de Freitas, N., & Gordon, N. (2001). *Sequential Monte Carlo Methods in Practice*. Springer-Verlag New York.

Barbuti, R., Levi, F., Milazzo, P., & Scatena, G. (2011). Maximally Parallel Probabilistic Semantics for Multiset Rewriting. *Fundamenta Informaticae*, *112*(1), 1–17.

Bingham, E., Chen, J. P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P., Horsfall, P., & Goodman, N. D. (2019). Pyro: Deep universal probabilistic programming. *The Journal of Machine Learning Research*, *20*(1), 973–978.

Blizard, W. D., et al. (1988). Multiset theory. *Notre Dame Journal of formal logic*, *30*(1), 36–66.

Bulling, A., Blanke, U., & Schiele, B. (2014). A tutorial on human activity recognition using body-worn inertial sensors. *ACM Computing Surveys (CSUR)*, *46*(3), 33.

Chavira, M., & Darwiche, A. (2008). On probabilistic inference by weighted model counting. *Artificial Intelligence*, *172*(6-7), 772–799.

Chen, L., Hoey, J., Nugent, C. D., Cook, D. J., & Yu, Z. (2012). Sensor-based activity recognition. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *42*(6), 790–808.

Choi, J., Amir, E., & Hill, D. (2010). Lifted Inference for Relational Continuous Models. In *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, UAI'10, pp. 126–134, Catalina Island, CA. AUAI Press.

Choi, J., Amir, E., Xu, T., & Valocchi, A. (2015). Learning Relational Kalman Filtering.. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pp. 2539–2546.

Choi, J., Guzman-Rivera, A., & Amir, E. (2011). Lifted Relational Kalman Filtering.. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, pp. 2092–2099.

Danos, V., Feret, J., Fontana, W., Harmer, R., & Krivine, J. (2007). Rule-based modelling of cellular signalling. In *International conference on concurrency theory*, pp. 17–41. Springer.

Diaconis, P., & Freedman, D. (1980). De finetti's generalizations of exchangeability. *Studies in inductive logic and probability*, *2*, 233–249.

Doucet, A., De Freitas, N., Murphy, K., & Russell, S. (2000). Rao-Blackwellised particle filtering for dynamic Bayesian networks. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pp. 176–183. Morgan Kaufmann Publishers Inc.

Faeder, J. R., Blinov, M. L., & Hlavacek, W. S. (2009). Rule-based modeling of biochemical systems with bionetgen. In *Systems biology*, pp. 113–167. Springer.

Fearnhead, P., & Clifford, P. (2003). On-line inference for hidden markov models via particle filters. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *65*(4), 887–899.

Fierens, D., Van den Broeck, G., Renkens, J., Shterionov, D., Gutmann, B., Thon, I., Janssens, G., & De Raedt, L. (2015). Inference and learning in probabilistic logic programs using weighted boolean formulas. *Theory and Practice of Logic Programming*, *15*(3), 358–401.

Flach, P. A., & Lachiche, N. (2000). Decomposing probability distributions on structured individuals.. In *ILP Work-in-progress reports*.

Gehr, T., Misailovic, S., & Vechev, M. (2016). Psi: Exact symbolic inference for probabilistic programs. In *International Conference on Computer Aided Verification*, pp. 62–83. Springer.

Gillespie, D. T. (1977). Exact stochastic simulation of coupled chemical reactions. *The journal of physical chemistry*, *81*(25), 2340–2361.

Gogate, V., & Domingos, P. (2011). Probabilistic Theorem Proving. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence*, pp. 256–265. AUAI Press.

Goodman, N., Mansinghka, V., Roy, D. M., Bonawitz, K., & Tenenbaum, J. (2008). Church: a language for generative models with non-parametric memoization and approximate inference. In *Uncertainty in Artificial Intelligence*.

Huang, J., Guestrin, C., & Guibas, L. (2009). Fourier Theoretic Probabilistic Inference over Permutations. *Journal of Machine Learning Research*, *10*, 997–1070.

John, M., Lhoussaine, C., Niehren, J., & Versari, C. (2011). Biochemical reaction rules with constraints. In *European symposium on programming*, pp. 338–357. Springer.

Kasparick, M., & Krüger, F. (2013). Probabilistic action selection - tracking multiple persons in indoor environments. `http://dx.doi.org/10.18453/rosdok_id00000114`.

Kersting, K., Ahmadi, B., & Natarajan, S. (2009). Counting belief propagation. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence*, pp. 277–284.

Kondor, R., Howard, A., & Jebara, T. (2007). Multi-object tracking with representations of the symmetric group. *Journal of Machine Learning Research*, *2*, 211–218.

Krüger, F., Hein, A., Yordanova, K., & Kirste, T. (2015). Recognising the actions during cooking task (Cooking task dataset)..

Krüger, F., Kasparick, M., Mundt, T., & Kirste, T. (2014a). Where are my colleagues and why? Tracking multiple persons in indoor environments. In *10th International Conference on Intelligent Environments (IE), 2014*, Shanghai, China.

Krüger, F., Nyolt, M., Yordanova, K., Hein, A., & Kirste, T. (2014b). Computational State Space Models for Activity and Intention Recognition. A Feasibility Study. *PLOS ONE*, *9*(11), e109381.

Lüdtke, S., Schröder, M., Bader, S., Kersting, K., & Kirste, T. (2018a). Lifted Filtering via Exchangeable Decomposition. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*.

Lüdtke, S., Schröder, M., & Kirste, T. (2018b). Approximate probabilistic parallel multiset rewriting using MCMC. In *KI 2018: Advances in Artificial Intelligence*, pp. 73–85. Springer.

Lüdtke, S., Schröder, M., Krüger, F., & Kirste, T. (2017). Where are my colleagues? Tracking and Counting Multiple Persons using Lifted Marginal Filtering.. In *Procedings of the 4th International Workshop on Sensor-Based Activity Recognition and Interaction.*

McCallum, A., Schultz, K., & Singh, S. (2009). Factorie: Probabilistic programming via imperatively defined factor graphs. In *Advances in Neural Information Processing Systems*, pp. 1249–1257.

Milch, B., Zettlemoyer, L., Kersting, K., Haimes, M., & Kaelbling, L. (2008). Lifted probabilistic inference with counting formulas. In *Proceedings of the National Conference on Artificial Intelligence*, Vol. 2, pp. 1062–1068.

Montemerlo, M., Thrun, S., Koller, D., Wegbreit, B., et al. (2002). Fastslam: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the AAAI*, pp. 593–598.

Niepert, M., & Domingos, P. (2014). Exchangeable variable models. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 271–279.

Niepert, M., & Van den Broeck, G. (2014). Tractability through exchangeability: A new perspective on efficient probabilistic inference. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, pp. 2467–2475.

Nitti, D., De Laet, T., & De Raedt, L. (2016). Probabilistic logic programming for hybrid relational domains. *Machine Learning*, *103*(3), 1–43.

Nyolt, M., & Kirste, T. (2015). On Resampling for Bayesian Filters in Discrete State Spaces. In *Proceedings of the 27th International Conference on Tools with Artificial Intelligence*, pp. 526–533, Vietri sul Mare, Italy. IEEE Computer Society.

Nyolt, M., Krüger, F., Yordanova, K., Hein, A., & Kirste, T. (2015). Marginal filtering in large state spaces. *International Journal of Approximate Reasoning*, *61*, 16–32.

Paun, G. (2012). *Membrane Computing: An Introduction.* Springer Science & Business Media.

Pescini, D., Besozzi, D., Mauri, G., & Zandron, C. (2006). Dynamical probabilistic P systems. *International Journal of Foundations of Computer Science*, *17*(01), 183–204.

Pfeffer, A. (2009). Figaro: An object-oriented probabilistic programming language. *Charles River Analytics Technical Report*, *137*, 96.

Plötz, T., & Fink, G. A. (2009). Markov models for offline handwriting recognition: a survey. *International Journal on Document Analysis and Recognition (IJDAR)*, *12*(4), 269.

Poole, D. (2003). First-order probabilistic inference. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pp. 985–991.

Poole, D., Bacchus, F., & Kisynski, J. (2011). Towards completely lifted search-based probabilistic inference. *arXiv preprint*, *arXiv:1107.4035.*

Poon, H., & Domingos, P. (2011). Sum-product networks: A new deep architecture. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference On*, pp. 689–690. IEEE.

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

Sato, T., & Kameya, Y. (2008). New advances in logic-based probabilistic modeling by prism. In *Probabilistic inductive logic programming*, pp. 118–155. Springer.

Schröder, M., Lüdtke, S., Bader, S., Krüger, F., & Kirste, T. (2016). An Office Scenario Dataset for Benchmarking Observation-equivalent Entities..

Schröder, M., Lüdtke, S., Bader, S., Krüger, F., & Kirste, T. (2017). Abstracting from Observation-equivalent Entities in Human Behavior Modeling. In *AAAI Workshop: Plan, Activity, and Intent Recognition*.

Siegrist, K. (2020). Random: Probability, mathematical statistics, stochastic processes.. Accessed: 2020-10-01.

Singla, P., & Domingos, P. (2008). Lifted first-order belief propagation. In *Proceedings of the National Conference on Artificial Intelligence*, Vol. 2, pp. 1094–1099.

Singla, P., Nath, A., & Domingos, P. M. (2014). Approximate lifting techniques for belief propagation. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*.

Stanke, M., & Waack, S. (2003). Gene prediction with a hidden markov model and a new intron submodel. *Bioinformatics*, *19*(suppl_2), ii215–ii225.

Taghipour, N., Davis, J., & Blockeel, H. (2014). Generalized counting for lifted variable elimination. In *23rd International Conference on Inductive Logic Programming*, Vol. 8812, pp. 107–122.

Thon, I., Landwehr, N., & De Raedt, L. (2011). Stochastic relational processes: Efficient inference and applications. *Machine Learning*, *82*(2), 239–272.

Thon, I., Gutmann, B., Van Otterlo, M., Landwehr, N., & De Raedt, L. (2009). From non-deterministic to probabilistic planning with the help of statistical relational learning. In *ICAPS 2009-Proceedings of the Workshop on Planning and Learning*, pp. 23–30.

Van Den Broeck, G., Taghipour, N., Meert, W., Davis, J., & De Raedt, L. (2011). Lifted probabilistic inference by first-order knowledge compilation. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, pp. 2178–2185.

Van Haaren, J., Van den Broeck, G., Meert, W., & Davis, J. (2016). Lifted generative learning of markov logic networks. *Machine Learning*, *103*(1), 27–55.

Venugopal, D., & Gogate, V. (2014). Evidence-based clustering for scalable inference in markov logic. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 258–273. Springer.

Wang, J., Chen, Y., Hao, S., Peng, X., & Hu, L. (2019). Deep learning for sensor-based activity recognition: A survey. *Pattern Recognition Letters*, *119*, 3 – 11. Deep Learning for Pattern Recognition.

Wilson, D. H., & Atkeson, C. (2005). Simultaneous tracking and activity recognition (star) using many anonymous, binary sensors. In *International Conference on Pervasive Computing*, pp. 62–79. Springer.

Wood, F., Meent, J. W., & Mansinghka, V. (2014). A new approach to probabilistic programming inference. In *Artificial Intelligence and Statistics*, pp. 1024–1032.

Yu, S.-Z. (2010). Hidden semi-markov models. *Artificial intelligence*, *174*(2), 215–243.