

Universität  
Rostock



Traditio et Innovatio

# Lifted Bayesian Filtering in Multi-Entity Systems

## Dissertation

to obtain the academic degree of  
**Doktor-Ingenieur (Dr.-Ing.)**  
of the Faculty of Computer Science and Electrical Engineering  
at the University of Rostock

submitted by

**Stefan Lüdtke**

born on August 5, 1991 in Rostock

November 23, 2020



Dieses Werk ist lizenziert unter einer  
Creative Commons Namensnennung 4.0 International  
Lizenz.

**First reviewer:** Prof. Dr.-Ing. Thomas Kirste, University of Rostock

**Second reviewer:** Prof. Dr. Heiner Stuckenschmidt, University of Mannheim

**Year of submission:** 2020

**Year of defense:** 2021



## Abstract

*Bayesian filtering* (BF) is a general probabilistic framework for estimating the state of a dynamic system that can be observed only indirectly through noisy measurements. This thesis focuses on systems that consist of multiple, interacting *entites* (e.g. agents or objects), for which the system dynamics can be specified naturally by *multiset rewriting systems* (MRSs). Unfortunately, BF in MRSs is computationally challenging due to the combinatorial explosion in the state space size.

Therefore, we investigate efficient BF algorithms for such multi-entity systems. The main insight is that the state space that is underlying an MRS exhibits a certain *symmetry*, which can be exploited to increase inference efficiency.

This thesis provides five main contributions. First, we show how distributions over multisets can be decomposed into two factors: A distribution over the structures and multiplicities of entities, and a distribution over values of the entities' properties. This representation allows to group together entities with identical structure, thus achieving a substantial reduction in representation complexity. As this representation bears some similarity to other concepts from lifted probabilistic inference, we call it a *lifted* representation.

Secondly, we introduce a BF algorithm that works directly on this lifted representation, which is able to achieve a factorial reduction in space and time complexity, compared to conventional, ground filtering. When observations or system dynamics break symmetry, the algorithm automatically adapts by *splitting*.

When a maximally parallel action execution semantics is used – when all entities can act in parallel – exact BF can become intractable due to the large number of parallel actions. To alleviate this problem, our third contribution is a Markov chain Monte Carlo algorithm that *samples* parallel actions instead of performing full enumeration.

Fourth, we address the problem that due to symmetry breaks, the algorithm must perform splitting, so that the model can become completely propositional over time and inference becomes intractable. This is done by introducing inverse *merging* operations for a number of practically relevant special cases.

Finally, we empirically evaluate the lifted BF algorithm on real-world human activity recognition domains, and show that the algorithm can be more efficient than propositional BF. To the best of our knowledge, this is the first attempt to provide BF for systems with MRS dynamics and the first attempt that allows to perform prediction and update directly on the lifted representation.



## Zusammenfassung

Bayes'sches Filtern (BF) ist ein probabilistischer Ansatz zur Zustandsschätzung dynamischer Systeme, die nur über verrauschte Sensordaten beobachtet werden können. In dieser Arbeit werden Systeme betrachtet, die aus mehreren, interagierenden Entitäten (z.B. Agenten oder Objekten) bestehen. Die Systemdynamik solcher Systeme kann auf natürliche Art durch *Multiset Rewriting Systems* (MRS) spezifiziert werden. BF in MRS ist allerdings durch die kombinatorische Explosion in der Größe des Zustandsraums solcher Systeme rechnerisch aufwendig.

Aus diesem Grund betrachten wir in dieser Arbeit effiziente Algorithmen für BF in Systemen mit MRS-Dynamik. Die wesentliche Erkenntnis ist, dass der Zustandsraum solcher Systeme *Symmetrien* aufweist, die ausgenutzt werden können, um die Effizienz der Inferenz zu erhöhen.

Diese Arbeit leistet fünf wesentliche Beiträge. Als Erstes zeigen wir, wie Verteilungen über Multimengen in zwei Faktoren aufgeteilt werden können: Eine Verteilung über die Struktur und die Multiplizität der Entitäten und eine Verteilung über die Werte der Entitäten. Diese Repräsentation der Verteilung erlaubt es, Entitäten mit identischer Struktur zu gruppieren, sodass die Repräsentationskomplexität beträchtlich reduziert wird. Da diese Repräsentation Zusammenhänge zu anderen Konzepten aus dem Forschungsbereich der *Lifted Probabilistic Inference* aufweist, bezeichnen wir diese als *geliftete* Repräsentation.

Zweitens führen wir einen BF-Algorithmus ein, der direkt auf der gelifteten Repräsentation arbeitet, wodurch sich die Zeit- und Platzkomplexität gegenüber dem grundierten Algorithmus um einen faktoriellen Faktor reduziert. Wenn die Beobachtungen oder die Systemdynamik zu einem Symmetriebruch führen, kann der Algorithmus die Repräsentation automatisch durch *Splitting* anpassen.

Wenn eine maximal-parallele Aktionsausführungs-Semantik genutzt wird, d.h. wenn alle Entitäten gleichzeitig agieren können, ist exaktes BF aufgrund der großen Anzahl paralleler Aktionen nicht mehr durchführbar. Um dieses Problem zu lösen, entwickeln wir einen *Markov Chain Monte Carlo*-Algorithmus, der parallele Aktionen sampelt anstatt diese vollständig aufzuzählen.

Viertens behandeln wir das Problem, dass der Algorithmus durch Symmetriebrüche wiederholt splitten muss, sodass das Modell im Laufe der Zeit vollständig grundiert wird, wodurch Inferenz nicht mehr durchführbar ist. Hierzu führen wir für eine Reihe von praktisch relevanten Fällen *Merging*-Algorithmen ein, die sich invers zum Splitting verhalten.

Schließlich evaluieren wir den gelifteten BF-Algorithmus empirisch mit realen Aktivitätserkennungsszenarien und zeigen, dass der Algorithmus effizienter als grundiertes BF sein kann. Der in dieser Arbeit eingeführte BF-Algorithmus erlaubt erstmals BF in Systemen mit MRS-Dynamik. Es handelt sich um den ersten Ansatz, um BF direkt auf der gelifteten Repräsentation durchzuführen.





## Acknowledgements

First, I would like to thank my supervisor, Thomas Kirste. Without his support, abundance of ideas and constant availability for feedback, this thesis would not have been possible. He continuously raised the bar and always encouraged me to strive for excellence, which certainly contributed hugely to the quality of this thesis. I would also like to thank Heiner Stuckenschmidt for agreeing to review this thesis.

I am very grateful to my colleagues at MMIS who contributed substantially to this thesis. In particular, I would like to thank Sebastian for being a great mentor; Frank for introducing me to R and the tidyverse; Kristina for our collaboration on so many smaller and larger projects; Max for introducing me to the concepts that led to this thesis in the first place, and for the inspiring cooperation on the initial LiMa papers; Albert for guiding and helping me during the first few months at MMIS when I did not know at all what it means to do research, and for sharing his vast experience in sensor data processing; Petra for always being of great help; Peter for having all the solutions to technical problems; and all other colleagues and collaborators.

Finally, special thanks to my friends and family for their tremendous support and motivation during this time.



# Contents

<b>1. Introduction</b>	<b>1</b>
1.1. Motivation . . . . .	1
1.2. Problem Description and Requirements . . . . .	3
1.3. Contributions . . . . .	4
1.4. Outline of the Thesis . . . . .	6
<b>2. Background</b>	<b>7</b>
2.1. Probabilistic Inference . . . . .	8
2.1.1. Probabilistic Graphical Models . . . . .	8
2.1.2. Probabilistic Inference . . . . .	9
2.1.3. Relational Graphical Models . . . . .	10
2.2. Bayesian Filtering . . . . .	11
2.2.1. Problem Setup . . . . .	11
2.2.2. Particle Filtering . . . . .	13
2.2.3. Rao-Blackwellized Particle Filtering . . . . .	14
2.2.4. Computational State Space Models . . . . .	15
2.2.5. Marginal Filtering . . . . .	16
2.3. Multiset Rewriting Systems . . . . .	17
2.3.1. Multiset Rewriting Systems . . . . .	17
2.3.2. Maximally Parallel MRS . . . . .	19
<b>3. Symmetry-Aware Probabilistic Inference: A Systematic Review</b>	<b>21</b>
3.1. Systematic Literature Review . . . . .	22
3.1.1. Research Question . . . . .	22
3.1.2. Search Procedure . . . . .	23
3.1.3. Paper Selection . . . . .	23
3.1.4. Analysis Procedure: Algorithm Properties . . . . .	24
3.1.5. Quantitative Results . . . . .	26
3.2. Symmetry-Aware Inference Algorithms . . . . .	27
3.2.1. Lifted Probabilistic Inference . . . . .	27
3.2.2. Inference in Continuous Domains . . . . .	31
3.2.3. Relational Bayesian Filtering . . . . .	32
3.3. Conclusion . . . . .	36

<b>4. Lifted Marginal Filtering</b>	<b>39</b>
4.1. A Probabilistic Maximally Parallel Multiset Rewriting System with Structured Entities . . . . .	41
4.1.1. Design Considerations . . . . .	41
4.1.2. MRS with Structured Entities . . . . .	42
4.1.3. Maximally Parallel MRS . . . . .	44
4.1.4. Probabilistic Maximally Parallel MRS . . . . .	45
4.1.5. An Algorithm to Enumerate AMCAs . . . . .	48
4.2. Bayesian Filtering in Multiset Rewriting Systems . . . . .	49
4.2.1. Prediction . . . . .	49
4.2.2. Update . . . . .	50
4.3. Factorizing Multiset Distributions . . . . .	52
4.3.1. Decomposing Multisets of Structured Entities . . . . .	53
4.3.2. Distributions of Value Sequences . . . . .	54
4.3.3. Lifted States . . . . .	57
4.4. Lifted Filtering . . . . .	61
4.4.1. Applying Constraints and Effects to Lifted States . . . . .	61
4.4.2. Splitting . . . . .	62
4.4.3. Disjointness of Lifted States . . . . .	69
4.4.4. The Lifted Marginal Filtering Algorithm . . . . .	71
4.5. Experimental Evaluation . . . . .	74
4.5.1. Evaluation Scenarios . . . . .	74
4.5.2. Exact Inference . . . . .	77
4.5.3. Approximate Inference . . . . .	80
4.5.4. Summary . . . . .	83
<b>5. Approximating the System Dynamics using MCMC</b>	<b>85</b>
5.1. An MCMC Algorithm for $p(K   l)$ . . . . .	86
5.2. Experimental Evaluation . . . . .	89
<b>6. Lifted Marginal Filtering in Asymmetrical Models</b>	<b>93</b>
6.1. Problem Statement . . . . .	95
6.2. Merging Similar States . . . . .	97
6.2.1. Divergence Measures for Lifted States . . . . .	98
6.2.2. Computing Merged States . . . . .	100
6.2.3. Handling Different Distribution Types . . . . .	101
6.2.4. Experimental Evaluation . . . . .	104
6.3. Merging Disjoint States . . . . .	107
6.3.1. $M$ and $l^*$ known: Testing for Mergeability . . . . .	109
6.3.2. $l^*$ Known, $M$ Unknown: Identifying Mergeable Subsets . . . . .	111
6.3.3. $M$ and $l^*$ Unknown: A Greedy Search Algorithm . . . . .	113
6.3.4. Experimental Evaluation . . . . .	116
6.4. Merging Normal Distributions . . . . .	119
6.4.1. Merging Entities by Gaussian Mixture Reduction . . . . .	120
6.4.2. Experimental Evaluation . . . . .	121
6.5. Assumed Density Merging . . . . .	124
6.5.1. Algorithm Overview . . . . .	125

6.5.2. Time Points for Merging . . . . .	126
6.5.3. Exploiting Temporal Structure . . . . .	127
6.5.4. Experimental Evaluation . . . . .	127
6.6. Conclusion & Future Work . . . . .	130
<b>7. Discussion &amp; Conclusion</b>	<b>133</b>
7.1. Summary . . . . .	134
7.2. Discussion: Why Lifting Works Here . . . . .	135
7.3. Future Work . . . . .	136
<b>A. Notation</b>	<b>161</b>
<b>B. More Related Work</b>	<b>163</b>
B.1. The Lifted (Dynamic) Junction Tree Algorithm . . . . .	163
B.2. Lumpability and Syntactic Markovian Bisimulation . . . . .	164
B.3. Probabilistic Programming Languages . . . . .	165
B.4. Knowledge Compilation and Tractable Models . . . . .	165
B.5. Statistical Relational Learning . . . . .	165
B.6. First-Order Markov Decision Processes . . . . .	166
<b>C. Assignment of Papers to Groups</b>	<b>169</b>
<b>D. AMCA Computation as Constraint Satisfaction</b>	<b>171</b>
<b>E. Expressiveness of Sequential and Maximally Parallel Multiset Rewriting</b>	<b>173</b>
<b>F. Disjointness of Lifted States</b>	<b>177</b>
F.1. Disjointness of Typed States . . . . .	177
F.1.1. Identifying Overlap . . . . .	178
F.1.2. Shattering . . . . .	179
F.2. Disjointness of Untyped States . . . . .	180
F.2.1. Untyped States . . . . .	181
F.2.2. Matchings . . . . .	181
F.2.3. Reduction to a Constraint Satisfaction Problem . . . . .	184
F.2.4. Shattering . . . . .	185
<b>G. Details of Experiments</b>	<b>187</b>
G.1. Kitchen Scenario . . . . .	187
G.2. Learning a Lifted Representation by Gaussian Mixture Fitting for the Travian Scenario . . . . .	190



# Introduction

## 1.1. Motivation

The objective of this thesis is to provide efficient recursive state estimation in *multi-entity systems*, i.e. dynamic systems consisting of multiple interacting *entities*, e.g. agents or objects. This task arises, for example, in the context of *human activity recognition* (HAR), as illustrated by the following example.

**Example 1.** 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 time step, given the sensor data.

The task of estimating the state  $x_t$  of a time-varying system that is indirectly observed through noisy measurements is called *Bayesian filtering*, or *recursive Bayesian state estimation* [188]. Specifically, Bayesian filtering assumes that the system has the *Markov property* – that the system state  $x_t$  at time  $t$  only depends on the systems state  $x_{t-1}$  at time  $t - 1$ . Thus, the system dynamics can be expressed by a probabilistic *transition model*  $p(X_t | X_{t-1})$ . Additionally, we assume that each observation  $y_t$  only depends on the current state  $x_t$ , which is expressed by an *observation model*  $p(Y_t | X_t)$ .

The transition model can simply be represented by a transition matrix, as done in *hidden Markov models* [205]). However, for complex systems with many states and transitions where prior domain knowledge is available, it is often more natural to describe the transition model by a *computational process*, i.e. by a (probabilistic) algorithm or a set of probabilistic rules that define how the system can evolve over time. For example, the tracking domain above can be described naturally by the set of probabilistic actions that each of the agents can perform.

For multi-entity situations, a natural choice for the underlying formalism to specify the system dynamics are *multiset rewriting systems* (MRS) [9]. In a MRS, each system state  $x_t$  is a *multiset* (a set where elements can occur more than once), and the system dynamics is specified by *rewriting rules*. MRSs allow the concise specification of systems that consist of multiple, (inter-)acting entities that can be grouped into “species”.

## 1. Introduction

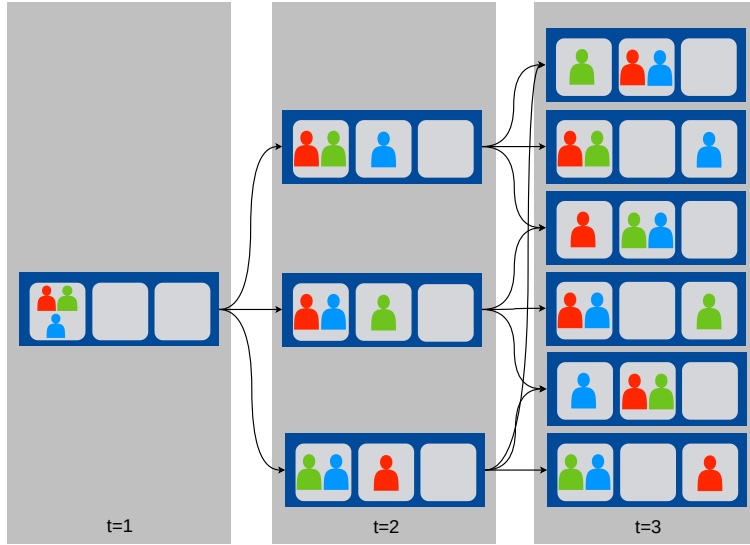


Figure 1.1.: System dynamics of the simple tracking domain: Three agents can move between locations, and initially, all agents are at the left location. At each time step, one of the agents can move one place to the right. The number of states grows quickly over time, due to the combinatorial explosion of the number of system states, an inherent property of multi-entity systems.

To see why Bayesian filtering in systems with MRS dynamics is hard, consider Figure 1.1, that shows a simplified version of the system described in Example 1 with three agents that can move between three locations and that are initially all at the left location. Even for such a simple system, the number of possible discrete system states can easily be very large: In the example, when there are  $n$  agents and  $m$  locations, the system can be in  $n^m$  different states. More generally, the number of possible system states typically grows exponentially with some of the system properties.

As over time, many of those states can simultaneously have non-zero probability, maintaining the categorical distribution over system states by complete enumeration is infeasible, and one has to resort to approximate inference methods, like particle filtering [55] or marginal filtering [164]. As the system state is categorical, existing methods to represent the distribution over system states *parametrically* (e.g. Kalman filtering [97]) can also not be used. Even worse, the system state of an MRS is a single, multiset-valued random variable, so the univariate distribution over system states does not directly contain any structure, like (conditional) independence or exchangeability, that could be exploited to arrive at a more efficient representation.

Still, the distribution often exhibits a certain regularity: For example, the three states at  $t = 2$  in Figure 1.1 are identical up to permutation of the agents' identities (represented by colors in the figure). Such regular structures arise naturally in MRSs due to the symbolic, high-level specification of system dynamics – just like relational probabilistic models induce symmetries in the distribution described by them, which opens up the possibility of using *lifted* inference algorithms in relational graphical models [158]. Intuitively, the distribution in Figure 1.1 could be described by a statement like “*two agents are at the left location and one is at the center location, and the colors red, green and blue are assigned to the agents*”



with uniform probability, but each agent has a unique color”.

In this thesis, we will devise a formalism that allows to compactly express such statements, and that still allows to apply multiset rewriting rules directly, without requiring to generate all system states first, such that Bayesian filtering can be performed efficiently. Along the way, we will also discuss the problems outlined above: What is a suitable decomposition of multisets, so that a distribution over multisets can be represented efficiently? How can multiset rewriting be applied directly to the efficient representation? How can these concepts be used for an efficient Bayesian filtering algorithm?

In summary, the goal of this thesis can be stated as follows:

**Research goal:** The goal of this thesis is to devise efficient Bayesian filtering algorithms for systems with MRS dynamics by exploiting symmetry in the distribution of system states.

## 1.2. Problem Description and Requirements

We start by defining the usage situations for which we want to provide an efficient inference algorithm, as well as requirements on the inference algorithm, in more detail. At this point, we can only state these properties and requirements in general terms – formal and algorithmic details will be provided in subsequent chapters.

**Properties of Usage Situations** This thesis is motivated by the observation that inference in multi-entity systems (e.g. for multi-agent activity recognition domains as illustrated in Example 1) is difficult, due to the combinatorial explosion in number of system states. More generally, the types or situations for which we want to provide an efficient inference algorithm can be characterized as follows:

- **Stochastic process:** We consider stochastic processes, i.e. dynamic systems that evolve probabilistically over time. More specifically, we consider time-discrete, first-order Markov processes, i.e. processes that evolves in discrete time steps, and the system state at time  $t$  depends only on the system state at time  $t - 1$ .
- **Hidden state and observations:** The state of the system is not observed directly, but through *observations* that depend on the current system state.
- **Multiple, interacting entities:** The system consists of multiple entities that simultaneously act and/or interact. These entities are not necessarily multiple agents, as in Example 1, but could for example also be multiple *objects* that are present in the environment.
- **Structured Entities:** The entities are not flat, unstructured objects, but consist of multiple *properties*. For example, the persons in the office domain have a name and location, while other entities, e.g. a coffee cup, could have properties that describe the content type and temperature. The properties can be both discrete (like the name) or continuous (like the temperature).
- **Variable entity numbers:** The number of entities can change over time. For example, in the office domain, people can enter or leave the observed environment. As

## 1. Introduction

another example, players in a multiplayer strategy game that we will consider later can build additional villages.

Examples of scenarios with these properties that we will be visiting later on include multi-agent tracking, activity and context recognition for cooking assistance, and predicting opponents' moves in an online multiplayer strategy game.

**Algorithm Requirements** As outlined above, the general idea for achieving efficient Bayesian filtering in such situations is to exploit the fact that sometimes, not all entities need to be distinguished. When this is the case, the algorithm should *group those entities together* and handle them jointly as a group. In other cases, however, it might be necessary to distinguish those entities again, because they can behave differently or are distinguished by observations. From these general considerations, we derived the following requirements on the filtering algorithm and model that the algorithm operates on.

- **Abstract model:** The algorithm can operate on a representation where entities are grouped together, as long as they do not need to be distinguished. Specifically, the algorithm needs to be able to perform inference on this representation directly, without generating the original, ground representation first. As we will see later, entities do not need to be completely identical to be grouped, but can already be grouped when their properties follow a joint *exchangeable* distribution.
- **Splitting:** When necessary due to system dynamics or observations, the algorithm can slit up the grouping and handle entities individually.
- **Merging:** The algorithm can perform an operation that is inverse to splitting: When possible, symmetries in the distribution are recognized and corresponding entities are grouped together. This operation is necessary to avoid the complete grounding of the distribution over time.

MRSs seem like a sensible choice to model these types of systems: They describe dynamic multi-entity systems in a natural way and directly allow grouping of entities due to the representation of states by multisets. Therefore, for now, we will focus on MRSs for modeling the system dynamics and investigate how Bayesian filtering can be performed in MRSs. We will revisit this choice after performing a literature review of related methods in Chapter 3.

### 1.3. Contributions

**(1) Systematic literature review** We perform a systematic literature review on MRSs and other symmetry-aware inference methods, to investigate which of these methods can be used for (or as part of) an efficient Bayesian filtering algorithm for multi-entity systems as specified above. From an initial set of more than 4,000 papers, we identify 116 relevant papers that discuss symmetry-aware inference algorithms. We find that no method can fully solve the problem domain already, but we identify a number of ideas and concepts that can be used as building blocks for the inference algorithm.

Independently of the specific goal of this thesis, the review provides a classification scheme of the identified approaches, for the first time drawing connections between different research fields, like lifted probabilistic inference, logical filtering, and multiple object tracking.

**(2) Bayesian filtering in probabilistic MRSs** One of the outcomes of the literature review is the fact that Bayesian filtering for systems with MRS dynamics has not been considered previously. So far, MRSs have only been used in the context of simulation studies. Therefore, we systematically introduce how Bayesian filtering can be performed in MRSs by direct application of the Bayesian filtering equations.

**(3) Efficient representation of distributions over multisets** Bayesian filtering in MRSs is intractable due to the combinatorial explosion in the number of possible multisets, i.e. system states. To alleviate this, the central technical contribution of this thesis is a suitable decomposition of multisets, such that the distribution over multisets can be factorized by exploiting independence. Additionally, when factors are *exchangeable* (which naturally occurs due to the fact that in the original multisets, the order of entities does not matter), concepts from lifted inference can be used to arrive at an even more efficient representation. Due to the relationship to lifted inference, we call this more efficient representation the *lifted* representation.

**(4) Bayesian filtering on the lifted representation** We show how Bayesian filtering can be performed directly on the lifted representation, without resorting to the original, much larger *ground representation*. Specifically, we discuss how the *prediction* and *update* steps can be performed on the lifted representation. When the symmetry breaks due to the system dynamics or observations, the state representation automatically adapts by *splitting* operations. We show empirically that in the best case (when the distribution is fully exchangeable), this algorithm leads to a factorial reduction in inference complexity.

**(5) Approximate filtering** This algorithm directly lends itself to an approximate version by limiting the number of explicitly represented lifted states in the posterior. We empirically show that the approximate algorithm can have a lower variance of the estimate and a lower estimation error than the ground algorithm with the same number of explicitly represented states.

**(6) MCMC-based approximations for maximally parallel multiset rewriting** Two issues remain that prevent the resulting lifted Bayesian filtering algorithm from scaling directly to large, real-world models. The first problem occurs when a maximally parallel action semantics is used – when all entities can act in parallel. In this case, the number of possible parallel actions (compound actions) can become very large, even when the lifted representation is used, such that not all compound actions can be enumerated completely. Therefore, we introduce a Markov Chain Monte Carlo (MCMC) algorithm to approximate the distribution of compound actions. The algorithm allows to accurately perform maximally parallel multiset rewriting in systems with thousands of entities, where complete enumeration of compound actions is infeasible.

**(7) Methods to retain a compact representation in the presence of asymmetries** Symmetry breaks in the transition or observation model can require repeated splitting, until the representation becomes completely ground over time. We show methods for retaining a lifted representation, that work by identifying subsets of lifted states that afford a joint representation by a *single* lifted state. As this task is intractable in general, we provide algorithms for a

## 1. Introduction

number of practically relevant special cases, where factors of the distribution over multisets are

- (7a) multinomial distributions,
- (7b) multivariate hypergeometric distributions, or
- (7c) normal distributions.
- (7d) Furthermore, we show how an even more compact representation can be obtained by exploiting the fact that some information can be safely “forgotten” at specific points in time during the filtering process.

We empirically show that these methods allow to substantially reduce the representational complexity, and thus, the lifted algorithm can be more efficient than ground inference even for real-world sensor-based human activity recognition tasks where symmetry breaks easily.

## 1.4. Outline of the Thesis

The remainder of this thesis consists of 6 chapters. While Chapters 2 and 3 introduce the required background and related work, Chapters 4 to 6 contain the main contributions.

- In **Chapter 2**, we introduce Bayesian filtering and multiset rewriting systems as the formalisms and models that are used throughout this thesis.
- **Chapter 3** contains a systematic literature review of probabilistic inference algorithms that make use of symmetries of the underlying distribution (Contribution 1).
- In **Chapter 4**, we present the main technical contribution of this thesis: An efficient Bayesian filtering algorithm for systems with MRS dynamics. The presentation of the algorithm is divided into multiple steps: First, we introduce *probabilistic maximally parallel MRSs* and argue why this type of MRS is required for Bayesian filtering. Next, we show how Bayesian filtering can be done in MRSs in a naive way (Contribution 2), then introduce a more efficient representation for distributions over multisets (Contribution 3) and finally show how multiset rewriting and thus Bayesian filtering can be performed directly on this efficient representation (Contribution 4). This section also includes an empirical evaluation of the exact and approximate versions of the algorithm (Contribution 5).
- In **Chapter 5**, we introduce and evaluate an MCMC-based algorithm for approximating the distribution over parallel actions (Contribution 6).
- In **Chapter 6**, methods for handling *symmetry breaks* are introduced (Contribution 7). Each of the methods is evaluated empirically, showing that the Bayesian filtering algorithm proposed in this thesis can be more efficient than ground inference even for real-world application domains when using suitable strategies to cope with symmetry breaks.

Finally, in Chapter 7, we summarize our work and present conclusions as well as possible directions for future research.

# Background

**CHAPTER SUMMARY** *In this chapter, we lay the foundation for this thesis, by introducing the formalisms and models that are used throughout. We start by providing a brief overview of probabilistic graphical models and inference in Section 2.1. In Section 2.2, we introduce the special case of inference in dynamic systems, known as Bayesian filtering, including Monte Carlo-based inference algorithms for continuous and discrete domains. In Section 2.3, the fundamentals of Multiset Rewriting Systems, the formalism that is used in this thesis to specify dynamic multi-entity systems, are introduced.*

*Parts of this chapter are based on:*

[134] Stefan Lüdtke, Max Schröder, Frank Krüger, Sebastian Bader, and Thomas Kirste. State-Space Abstractions for Probabilistic Inference: A Systematic Review. *Journal of Artificial Intelligence Research*, 63:789–848, 2018.

[130] Stefan Lüdtke and Thomas Kirste. Lifted Bayesian Filtering in Multiset Rewriting Systems. *Journal of Artificial Intelligence Research*, accepted, 2020.

## Contents

---

<b>2.1. Probabilistic Inference</b>	<b>8</b>
2.1.1. Probabilistic Graphical Models	8
2.1.2. Probabilistic Inference	9
2.1.3. Relational Graphical Models	10
<b>2.2. Bayesian Filtering</b>	<b>11</b>
2.2.1. Problem Setup	11
2.2.2. Particle Filtering	13
2.2.3. Rao-Blackwellized Particle Filtering	14
2.2.4. Computational State Space Models	15
2.2.5. Marginal Filtering	16
<b>2.3. Multiset Rewriting Systems</b>	<b>17</b>
2.3.1. Multiset Rewriting Systems	17
2.3.2. Maximally Parallel MRS	19

---

## 2. Background

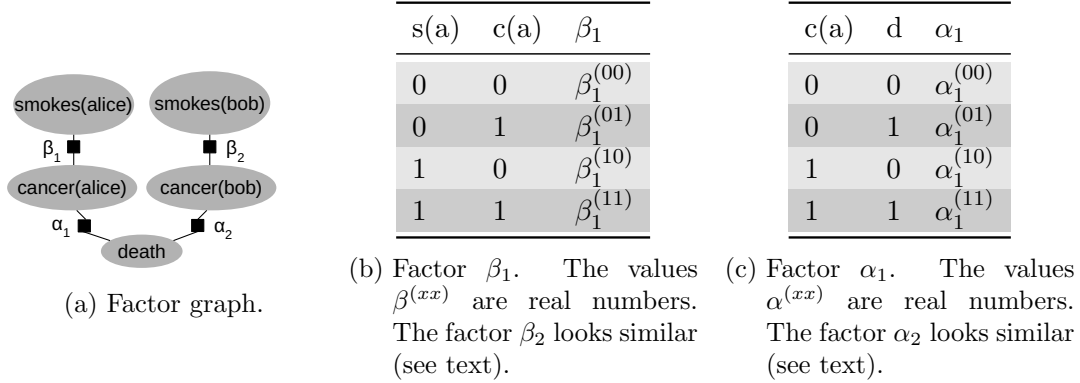


Figure 2.1.: Factor graph for Example 2, adapted from Richardson and Domingos [179].

## 2.1. Probabilistic Inference

In this section, we introduce central concepts of probabilistic graphical models and probabilistic inference. A more detailed introduction can be found, for example, in Russell [182] or Koller and Friedman [116]. As the algorithms presented in this thesis do not heavily rely on graphical models (but rather make use of MRSs as the underlying formalism), we limit the presentation here to the concepts that are necessary for discussing the related work in Chapter 3. We also briefly introduce *relational graphical models*, which are the basis for the *lifted probabilistic inference* algorithms discussed in Section 3.2.1.

### 2.1.1. Probabilistic Graphical Models

Let  $X_1, \dots, X_n$  be a set of  $n$  random variables (RVs)<sup>1</sup>. We are interested in representing (and reasoning about) probability distributions of those random variables. When all RVs  $X_1, \dots, X_n$  are *discrete*, a simple way to represent the joint distribution  $p(X_1, \dots, X_n)$  is to use a table that lists the probability of each assignment. Unfortunately, this table is very large – when all RVs are *binary*, the table has  $2^n$  rows.

Probabilistic graphical models are a data structure that allow to represent such a joint distribution more compactly when the distribution exhibits (conditional) independence. Here, we will focus on *factor graphs* as an example of graphical models. They represent a joint probability distribution over RVs  $X_1, \dots, X_n$  by decomposing the distribution  $p(X_1, \dots, X_n)$  into a set of factors  $F$ . Each factor  $\phi \in F$  maps a vector of RV assignments to non-negative real numbers, and the product of all factors describes the joint distribution (multiplied by a normalization constant  $Z^{-1}$  ensuring that the total probability sums to one):

$$p(X_1 = x_1, \dots, X_n = x_n) = Z^{-1} \prod_{\phi \in F} \phi(x_\phi), \quad (2.1)$$

where  $x_\phi$  denotes the subset of values of RVs that is necessary to compute the factor  $\phi$ . Each factor can then be represented, for example, as a table. A factor graph is a visualization of the factorization structure (see Figure 2.1 for an example).

<sup>1</sup>We use uppercase letters to denote random variables, and lowercase letters to denote realizations of random variables. Instead of  $p(X = x)$ , we will usually write  $p(x)$ .

**Example 2.** [Smokers] Each person either smokes or does not smoke. For people who smoke, the chance of getting cancer is higher than for people who do not smoke. Whether or not at least one person died last year depends on the number of people who have cancer.

For now, let us assume that only two people, Alice and Bob, exist. We can then model this scenario with the binary random variables  $smokes(alice)$ ,  $cancer(alice)$ ,  $smokes(bob)$ ,  $cancer(bob)$  and  $death$ <sup>2</sup>. The factor graph for this scenario can be seen in Figure 2.1. It describes a joint probability by multiplying all of the factors, for example:

$$\begin{aligned} & p(s(a)=1, s(b)=1, c(a)=0, c(b)=0, d=0) \\ &= Z^{-1} \beta_1(s(a)=1, c(a)=0) \beta_2(s(b)=1, c(b)=0) \alpha_1(d=0, c(a)=0) \alpha_2(d=0, c(b)=0) \quad (2.2) \\ &= \beta_1^{(10)} \beta_2^{(10)} \alpha_1^{(00)} \alpha_2^{(00)} \end{aligned}$$

### 2.1.2. Probabilistic Inference

Given a graphical model, different types of *queries* can be posed: One relevant case are *conditional probability queries*, where we want to compute a conditional probability  $p(Q | E=e)$  of some variables  $Q$ , given evidence on variables  $E$  – in the example, we might want to know the probability that Alice has cancer. The process of calculating such probabilities is called *probabilistic inference*. Inference can always be performed by computing the complete joint distribution, and summing out (marginalizing) the variables we are not interested in. However, a joint distribution over  $n$  binary RVs has size  $2^n$ , so efficient inference algorithms avoid this. For example, *variable elimination* (VE) [244] is an inference algorithm that eliminates the non-query and non-evidence variables one by one without computing the joint distribution. A variable is eliminated by multiplying all factors that contain this variable, and then marginalizing this variable.

**Example 3.** Consider the graphical model given in Figure 2.1 and the query  $p(s(a), s(b), d=1)$ . VE eliminates the non-query and non-evidence variables  $c(a)$  and  $c(b)$  one by one: The RV  $c(a)$  is eliminated by multiplying the factor  $\alpha_1$  and  $\beta_1$ , resulting in a factor  $f_0$  that has the following representation as a table (with 8 rows):

s(a)	c(a)	d	$f_0$
0	0	0	$\beta_1^{(00)} \alpha_1^{(00)}$
0	0	1	$\beta_1^{(00)} \alpha_1^{(01)}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$

The RV  $c(a)$  is summed out of  $f_0$ , resulting in a factor

$$f_1(s(a), d) = \sum_v f_0(s(a), c(a)=v, d) = \sum_v \beta_1(s(a), c(a)=v) \alpha_1(c(a)=v, d)$$

that is represented by the following table:

<sup>2</sup>For readability, we use  $c(a)$  and  $c(b)$  instead of  $cancer(a)$  and  $cancer(b)$ ,  $s(a)$  and  $s(b)$  instead of  $smokes(a)$  and  $smokes(b)$ , and  $d$  instead of  $death$ .

## 2. Background

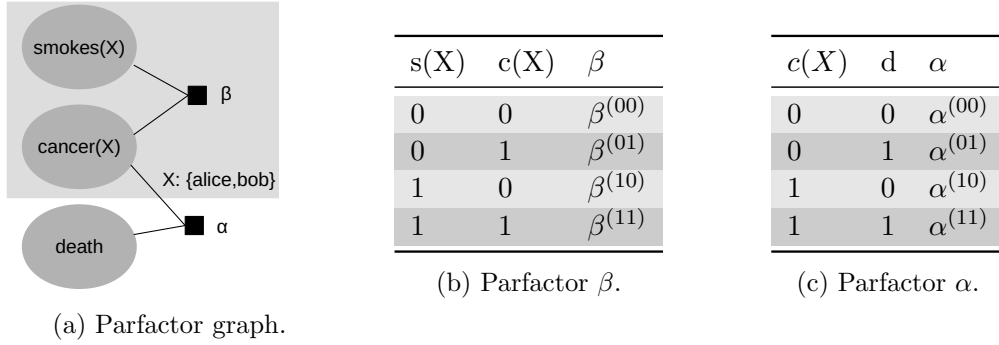


Figure 2.2.: Parfactor graph for Example 2, using par-RVs and plate notation [32].

$s(a)$	$d$	$f_1$
0	0	$\beta_1^{(00)} \alpha_1^{(00)} + \beta_1^{(01)} \alpha_1^{(10)}$
0	1	$\beta_1^{(00)} \alpha_1^{(01)} + \beta_1^{(01)} \alpha_1^{(11)}$
$\vdots$	$\vdots$	$\vdots$

Thus, the distribution  $p(s(a), s(b), c(b), d)$  can be represented by the factors  $\alpha_2$ ,  $\beta_2$  and  $f_1$  as follows:

$$p(s(a), s(b), c(b), d) = Z^{-1} f_1(s(a), d) \beta_2(s(b), c(b)) \alpha_2(c(b), d)$$

Afterwards, the same procedure is performed for  $c(b)$ :  $\alpha_2$  and  $\beta_2$  are multiplied,  $c(b)$  is marginalized, the result is multiplied with  $f_1$ . The result directly represents the distribution for the query above.

### 2.1.3. Relational Graphical Models

*Relational graphical models* use a high-level language (like first-order or relational logic) to compactly encode large graphical models. *Parametric factor graphs* (parfactor graphs) [173] are a specific instance of such relational graphical models. The idea of parfactor graphs is to represent the redundant factors (e.g. the factors  $\beta_1$  and  $\beta_2$  in Example 2) only once, which is achieved by extending factor graphs by a relational language.

A *parametric random variable* (par-RV) represents a set of random variables, one for each assignment of the parameters. The domain of each parameter is called *population* (i.e. a set of individuals). For example, if  $X$  is a parameter with the domain  $\{a, b\}$ , then  $s(X)$  is a par-RV, and the parameter assignments  $s(a)$  and  $s(b)$  both represent a random variable. We call these RVs the *groundings* of the par-RV.

A parametric factor, or *parfactor*, is a function that maps par-RV assignments to the non-negative reals, i.e. it represents a set of factors, one for each grounding of the par-RVs. For example, the parfactor  $\beta(s(X), c(X))$  represents the two factors  $\beta_1(s(a), c(a))$  and  $\beta_2(s(b), c(b))$ . A set of par-RVs and parfactors can be represented by a parfactor graph. The parfactor graph for Example 2 is shown in Figure 2.2 (using plate notation [32]). A parfactor graph defines a joint probability distribution as the normalized product of all groundings of the parfactors.



The appealing property of parfactor graphs is that the joint distribution can be calculated more directly, without complete grounding: Factors that correspond to the same parfactor and that have the same assignment of RVs need to be evaluated only once, raised to the power of the number of corresponding factors. For example, consider the joint probability  $p(s(a)=1, s(b)=1, c(a)=0, c(b)=0, d=0)$  considered in Equation 2.2. The factors  $\beta_1$  and  $\beta_2$ , as well as  $\alpha_1$  and  $\alpha_2$  correspond to the same parfactors and have the same assignment of involved RVs. Thus the joint can be calculated as:

$$\begin{aligned}
& p(s(a)=1, s(b)=1, c(a)=0, c(b)=0, d=0) \\
&= Z^{-1} \beta_1(s(a)=1, c(a)=0) \beta_2(s(b)=1, c(b)=0) \alpha_1(d=0, c(a)=0) \alpha_2(d=0, c(b)=0) \\
&= Z^{-1} \prod_{X \in \{a,b\}} \beta(s(X)=1, c(X)=0) \alpha(d=0, c(X)=0) \tag{2.3} \\
&= Z^{-1} \beta(s(X)=1, c(X)=0)^2 \alpha(d=0, c(X)=0)^2
\end{aligned}$$

More generally, the relational specification of the model leads to a certain type of symmetry in the distribution described by it.

**Definition 1.** [exchangeable decomposition [159]] Let  $\{X_1, \dots, X_n\}$  be a set of random variables, and let  $\mathbf{X} = \{\mathbf{X}_1, \dots, \mathbf{X}_k\}$  be a partition of the random variables into  $k$  subsets. We call  $\mathbf{X}$  an *exchangeable decomposition*, when

$$p(\mathbf{X}_1=\mathbf{x}_1, \dots, \mathbf{X}_k=\mathbf{x}_k) = p(\mathbf{X}_1=\mathbf{x}_{\pi(1)}, \dots, \mathbf{X}_n=\mathbf{x}_{\pi(k)}) \tag{2.4}$$

for all permutations  $\pi$  of  $1, \dots, k$ .

When the size of all subsets in  $\mathbf{X}$  is one, we call the random variables  $\{X_1, \dots, X_n\}$  *fully exchangeable*.

The example above does not have an exchangeable decomposition due to the  $\alpha$  factor that is more difficult to handle. For the submodel that consists only of the  $s(X)$  and  $c(X)$  RVs and the  $\beta$  factors, the joint distribution  $p(s(a), s(b), c(a), c(b))$  obviously has the exchangeable decomposition  $\{\{s(a), c(a)\}, \{s(b), c(b)\}\}$ .

Distributions that have an exchangeable decomposition always allow for tractable marginal and MAP inference [159]. This fact is exploited by *lifted probabilistic inference* algorithms like *first-order variable elimination* [173, 207] that are discussed in Section 3.2.1.

## 2.2. Bayesian Filtering

In this section, we introduce the problem of *Bayesian filtering* (BF), also known as sequential state estimation. This task arises when we want to estimate the state of a dynamic system that can only be observed through noisy and ambiguous measurements. An introduction to Bayesian filtering can, for example, be found in Murphy [149], and we provide a concise summary of the basic concepts here for convenience.

### 2.2.1. Problem Setup

Let  $X_1, \dots, X_T$  with  $x_t \in \mathcal{X}$  be a *sequence* of random variables (RVs). For simplicity of notation, we denote sequences of RVs  $X_1, \dots, X_T$  as  $X_{1:T}$ . We call  $X_t$  the *state* of the

## 2. Background

system at time  $t$ . Note that  $X_t$  can be vector-valued. We assume that the distribution of states at time  $t$  only depends on the state at time  $t - 1$ , i.e. that the states form a *first-order Markov chain*.

**Definition 2.** [Markov chain, stationary Markov chain] Let  $X_{1:T}$  be a sequence of random variables. The sequence is a *Markov chain* when the joint distribution factorizes as

$$p(X_{1:T}) = p(X_1) \prod_{t=2}^T p(X_t | X_{t-1}). \quad (2.5)$$

We call a Markov chain *stationary* if the distribution  $p(X_t | X_{t-1})$  is identical for all  $t$ .

Thus, the complete joint distribution of a stationary Markov chains is represented by a single conditional probability  $p(X_t | X_{t-1})$  and the probability  $p(X_1)$ . We call  $p(X_1)$  the *prior probability* and  $p(X_t | X_{t-1})$  the *transition model* of the Markov chain.

Furthermore, we assume that the state  $X_t$  cannot be observed directly, but that for each time  $t$ , an observation  $y_t$  is generated from the state  $x_t$ , based on a distribution  $p(Y_t | X_t)$ . More formally, let  $Y_{1:T}$  be a sequence of RVs (called *observations*) with the property that the distribution of  $Y_t$  only depends on  $X_t$  (this property is called the *sensor Markov assumption*). In this case, the joint distribution of states and observations factorizes as

$$p(X_{1:T}, Y_{1:T}) = p(X_{1:T}) p(Y_{1:T} | X_{1:T}) = \left( p(X_1) \prod_{t=2}^T p(X_t | X_{t-1}) \right) \left( \prod_{t=1}^T p(Y_t | X_t) \right) \quad (2.6)$$

Again, we assume that the distribution  $p(Y_t | X_t)$  is fixed, i.e. identical for each  $t$ , and call the distribution the *observation model*. The complete setup, consisting of the transition model (that describes how the system evolves over time), the observation model (that describes how observations are generated from states), and the prior distribution (our initial belief about the state of the system) is called a *state space model*.

**Definition 3.** [state space model [149]] Let  $p_0(X_1)$  be a prior distribution,  $p_X(X_t | X_{t-1})$  be a transition model, and  $p_Y(Y_t | X_t)$  be an observation model. We call a triple  $(p_0, p_X, p_Y)$  a *state space model* (SSM).

There are several typical inference tasks for such systems. *Filtering* means to compute the distribution  $p(X_t | y_{1:t})$  (the *marginal filtering density*, or *belief state*). For *smoothing*, the task is to compute the distribution  $p(X_t | y_{1:T})$  with  $t < T$ , i.e. an estimate of a *past* system state, given all observations up to the present. *Prediction* asks for the distribution  $p(X_{k+\delta} | y_{1:t})$ , i.e. an estimate of *future* states. Finally, the *maximum a posteriori* (MAP) estimate consists of the most probable state sequence, i.e.  $\arg\max_{x_{1:T}} p(x_{1:T} | y_{1:T})$ .

A system that keeps track of the current situation at time  $t$  to make decisions in an *online* fashion (i.e. that does not have access to observations  $y_i$  with  $i > t$ ) performs filtering. Thus, in the following, we focus on filtering.

In principle, the full joint distribution  $p(X_{1:T}, Y_{1:T})$  can be described by a large graphical model, and BF could be performed by any standard probabilistic inference algorithm, like variable elimination. However, the length  $T$  of the observation sequence is typically unbound, so that the size of the graphical model becomes larger and larger over time, and thus inference by *unrolling* the model is infeasible. Instead, it is straightforward to obtain a recursive

formula for the marginal filtering density  $p(X_t | y_{1:t})$ , by making use of the Markov assumption for the transition model and the sensor Markov assumption (for a derivation, see e.g. [149]). Given the prior distribution  $p(X_{t-1} | y_{1:t-1})$ , i.e. the marginal filtering density of time  $t - 1$ , we can first compute the *prediction*

$$p(X_t | y_{1:t-1}) = \int_{\mathcal{X}} p(X_t | x_{t-1}) p(x_{t-1} | y_{1:t-1}) dx_{t-1}. \quad (2.7)$$

and then the *update*

$$p(X_t | y_{1:t}) = Z^{-1} p(y_t | X_t) p(X_t | y_{1:t-1}), \quad (2.8)$$

where  $Z = p(y_t)$  is a normalization factor. Thus, inference can be performed recursively, such that inference complexity is linear in the number of time steps  $T$ .

Note that in general, Equation 2.7 requires integration (when  $X_t$  is a continuous RV). For the discrete case, this integral becomes a sum. Furthermore, when the states discrete RVs, the transition model  $p(X_t | X_{t-1})$  can be represented by a matrix, and the model is known as a *hidden Markov model*. The inference algorithm that recursively computes Equations 2.7 and 2.8 for a hidden Markov model is known as *forward algorithm*. Another well-known case arises when the marginal filtering density is Gaussian, the transition model is linear-Gaussian (i.e. a linear function with Gaussian noise), and the observation model is conditional Gaussian. This case is known as *Kalman filter* model, for which the recursion can be computed in closed form [97].

### 2.2.2. Particle Filtering

In general, it is not straightforward to compute Equations 2.7 and 2.8 directly, as they both require the evaluation of complex, high-dimensional integrals: Equation 2.7 requires integrating  $x_{t-1}$  out of the joint density, and the computation of the normalization factor  $Z = p(y_t)$  in Equation 2.8 requires integrating over states  $x_t$ . Thus, different approximation strategies for these distributions have been devised in the last 50 years, for example the Gaussian sum filter [204] (that approximates the filtering density by a mixture of  $K$  normal distributions), the extended Kalman filter (that can be applied to nonlinear systems, by linearizing the transition and observation model, but still assuming normality of the filtering density and noise) and the Boyen-Koller algorithm [25] (which maintains a factorized version of the filtering density).

Another widely used class of approximate Bayesian filtering algorithms, that do not require any linearity or normality of the SSM are *sequential Monte Carlo* methods [8]. The core idea is to approximate the joint filtering distribution  $p(X_{1:t} | y_{1:t})$  by a set of weighted samples (*particles*)  $\{(w_t^{(i)}, x_{1:t}^{(i)})\}_{i=1}^N$ :

$$p(X_{1:t} | y_{1:t}) \approx \sum_{i=1}^N w_t^{(i)} \delta_{x_{1:t}^{(i)}}(X_{1:t}), \quad (2.9)$$

where  $\delta_{x_{1:t}^{(i)}}$  denotes the delta mass located at  $x_{1:t}^{(i)}$ . The bootstrap *particle filter* (PF) performs the prediction and update steps of Bayesian filtering by using this approximation. Intuitively, the PF simulates possible sequences based on the transition model, and weighs each particle according to the observation model. Specifically, the transition is performed by sampling

## 2. Background

new values for each particle from the transition model<sup>3</sup>, i.e.  $x_t^{(i)} \sim p(X_t | x_{t-1}^{(i)})$ , and set  $x_{1:t}^{(i)} = (x_{1:t-1}^{(i)}, x_t^{(i)})$ . The update step evaluates the importance weight according to the observation model, i.e.  $w_t^{(i)} = p(y_t | x_t^{(i)})$ .

Finally, a *resampling* step is required to avoid particle degeneracy: Without resampling, the particle weight would become more and more skewed, until practically, only a single particle has non-zero weight, and the algorithm thus fails to represent the distribution adequately. Resampling is performed by sampling  $N$  particles with replacement from the set  $\{x_{1:t}^{(i)}\}_{i=1}^N$  according to weights  $w_t^{(i)}$ .

Also, if we are only interested in the marginal filtering density  $p(X_t | y_{1:t})$ , we can simply ignore the previous part  $x_{1:t-1}^{(i)}$  of each particle. It can be shown that the particle-based approximation shown in Equation 2.9 converges to the true joint filtering distribution when  $N$  approaches infinity [8]. For continuous domains, particle filtering has been successfully used for a wide range of application domains, like robot localization [214], multi-target tracking [165], ecological models [28], or stock market prediction [34].

PF can be computationally expensive for high-dimensional state spaces due to a phenomenon called *sample impoverishment* [60]: Resampling duplicates particles, such that there is a loss in “diversity” in particles after resampling, i.e. the particles are utilized poorly for representing the complete state space. This problem is more prevalent in high-dimensional state spaces, so that the number of particles must grow *exponentially* with the number of dimensions to achieve a given accuracy [20]. For continuous state spaces, this problem can be approached, for example, by adding noise to the samples after resampling [81].

### 2.2.3. Rao-Blackwellized Particle Filtering

As discussed above, PF is typically computationally expensive in high-dimensional state spaces. Closed-form exact algorithms like Kalman filtering, on the other hand, can only be applied when the SSM adheres to specific conditions. Fortunately, both methods can sometimes be combined, when *some* of the state variables satisfy conditions that allow to handle them analytically, given the other variables. Particle filtering then only needs to be performed for the remaining variables, i.e. a state space with lower dimensionality, requiring fewer particles for a good approximation.

Specifically, suppose that the state variable  $X_t$  is a vector of variables  $X_{t,1:n}$  that can be partitioned into parts  $S_{t,1:s}$  and  $V_{t,1:v}$  with  $s+v=n$ . Then, we rewrite the marginal filtering distribution as

$$p(S_t, V_t | y_{1:t}) = p(S_t | y_{1:t}) p(V_t | S_t, y_{1:t}). \quad (2.10)$$

Now, assume that  $p(V_t | S_t, y_{1:t})$  can be handled analytically, while  $p(S_t | y_{1:t})$  is handled by particle filtering. That is, *partial* assignments (of the variables  $S_t$ ), combined with a closed-form representation of the distribution over variables  $V_t$  can be used as particles. More precisely, each particle is a triple  $(w_t^{(i)}, s_t^{(i)}, \rho_t^{(i)})$ , where  $s_t^{(i)}$  is an assignment of the RVs  $s_t$ , and  $\rho_t^{(i)}$  is a closed-form representation of the distribution  $p(V_t | S_t, y_{1:t})$  (e.g. a mean vector and covariance matrix). This method is known as *Rao-Blackwellized particle filtering* (RBPF) [59], due to the close relationship to the Rao-Blackwell theorem.

---

<sup>3</sup>This approach can be generalized by sampling from an arbitrary *proposal distribution*, but deriving better proposal can be challenging, so we will focus on the transition model as proposal distribution.

**Example 4.** A prominent example of RBPF arises in the context of simultaneous localization and mapping (SLAM): A robot moves around an unknown two-dimensional world, and simultaneously needs to learn the map and estimate its position in that map. The map can, for example, be represented by the locations of  $L$  two-dimensional obstacles  $v_1, \dots, v_L$  (we assume that the obstacles do not move, so we omit the time index). The robot location at time  $t$  is the two-dimensional vector  $s_t$ .

The interesting observation here is that conditional on the robot’s current location  $s_t$ , the obstacle locations  $v$  are independent, i.e.  $p(v | s_t, y_{1:t}) = \prod_{l=1}^L p(v_l | s_t, y_{1:t})$ . Thus, RBPF can be used, by sampling the robot’s location  $s_t$ , and running  $L$  independent Kalman filters inside each particle. This procedure is known as *fastSLAM* [214].

### 2.2.4. Computational State Space Models

In the following, we introduce a method for compactly representing the transition model of hidden Markov models (i.e. SSMs with categorical state space) known as *computational state space models*.

For HMMs, the transition model is typically specified as a matrix  $T$ , where columns denote the prior states  $x_{t-1}$ , rows denote the posteriors  $x_t$ , and each cell contains the corresponding conditional probability  $p(x_t | x_{t-1})$ . Such a matrix has quadratic size in the number of system states  $|\mathcal{X}|$ , which makes this representation computationally expensive when  $|\mathcal{X}|$  is large. However, a matrix-based representation is unnecessary in cases where the *computational process* that is underlying the transition model is known: In this case, this computational description can be used directly as a specification of  $p(X_t | X_{t-1})$ . This method is known as *computational state space models* (CSSMs) [123]. Specifically, the transition model of a CSSM is represented by a function  $\text{PREDICT}(x_{t-1})$ , that returns the conditional probability  $p(X_t | x_{t-1})$  as a set of tuples  $(x_t, p_t)$ . In CSSMs, the states typically need an *algebraic* structure to allow computations on states, as opposed to standard HMMs, where states are atomic and only allow to test for identity. For example, a states can be a map of variable names to values. CSSMs have certain advantages compared to explicit, matrix-based representations of the transition model:

- The transition model is typically represented much more compactly than by an explicit, matrix-based representation.
- The definition of *latent infinite* state spaces, i.e. where the state space is infinite, but only a finite number of discrete states have non-zero probability, becomes straightforward: For example, the state space becomes infinite when the state represents a natural number, and the prediction function allows increasing the number by one (with some probability).
- CSSMs allow the *knowledge-based* construction of transition models, instead of requiring extensive amounts of training data.

Recently, a number of approaches that follow this paradigm have been proposed. *Computational causal behavior models* [120] are a CSSM variant tailored towards human activity recognition (HAR). They use the planning domain definition language (PDDL) 2.1 [67] to specify the transition model. This way, a set of actions is described, each consisting of (i) a set of preconditions that need to be satisfied for the action to be applicable to a state, and

## 2. Background

(ii) an effect function that manipulates the state to compute the posterior state. Furthermore, a distribution over applicable actions is defined such that the probability of an action is proportional to its *goal-directedness*, i.e. actions that reduce the goal distance are assigned a higher probability. Other CSSM approaches include the approach by Ramírez and Geffner [178], which also uses PDDL as a computational description of the PREDICT function, and defines inference in terms of a POMDP, and the approach by Sadilek and Kautz [183], which uses a Markov logic network (MLN) as a symbolic description, but *unrolls* the MLN into a graphical model instead of performing BF.

### 2.2.5. Marginal Filtering

The state space of CSSMs is typically categorical, with a very large number of different categories (i.e. possible states). Thus exact inference could be performed by the forward algorithm, as used for HMMs. However, due to the large number of states, the complete enumeration of all states and their probabilities as required by the forward algorithm, although theoretically possible, is practically infeasible.

Instead, when the inference algorithm cannot exploit any structure of the state space (as attempted later in this thesis), we need to resort to approximate algorithms. In principle, inference could simply be done by particle filtering. However, particle filtering is not suitable for large, categorical state spaces [164]: As no metric is defined on the states  $x_t$ , each state (i.e. each category) acts as a separate, independent dimension, which makes particle filtering challenging due to the *sample impoverishment* problem: More specifically, the resampling step leads to many duplicate particles that represent the same system state. Thus, the particles represent the overall distribution poorly, only the most likely states are represented by (multiple) particles, while many of the less likely states are not represented by any particle. Unfortunately, existing methods to circumvent the sample impoverishment problem, like adding noise to states [81], cannot be applied to categorical state spaces.

As a solution, Nyolt et al. proposed the *marginal filter* [164]. The idea is to prevent sample impoverishment by avoiding the need for resampling, while still maintaining a limited number of particles. This is achieved by maintaining particles that only represent the *marginal* filtering distribution  $p(X_t | y_{1:t})$  instead of the joint filtering distribution  $p(X_{1:t} | y_{1:t})$ , and by computing *all* posterior states for each particle instead of sampling from the transition model. This is possible in categorical state spaces where each state has only a finite number of successor states (i.e. where  $p(X_t | x_{t-1})$  has finite support for each  $x_{t-1}$ ).

Specifically the marginal filter works as follows: The marginal filtering distribution  $p(X_{t-1} | y_{1:t-1})$  is represented by a set of weighted samples  $\{(w_{t-1}^{(i)}, x_{t-1}^{(i)})\}$ . Instead of sampling from the transition model, the prediction step is performed exactly, by computing  $p(x_t | x_{t-1}^{(i)})$  for all  $x_t$  and all  $i$ , and merging particles with identical state  $x_t$  by summing their weight. This leads to a new set of particles  $\{(w_t^{(i)}, x_t^{(i)})\}$  with  $w_t^{(i)} = \sum_j p(x_t^{(i)} | x_{t-1}^{(j)})$  that represent the marginal distribution  $p(X_t | y_{1:t-1})$ . Note that in this set, each state occurs at most once. The algorithm can be realized efficiently by representing the marginal filtering distribution as a *map*  $\mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$  of states to probabilities: This way, a map insertion operator  $\oplus$  can be defined that directly performs the summation of weights of identical states (see Appendix A).

The update step can then also be computed exactly, by multiplying each weight  $w_t^{(i)}$  with the observation likelihood  $p(y_t | x_t^{(i)})$ . The only approximation for each time step is then to

---

**Algorithm 1** Marginal filter.

---

1. Initialization
    - Let  $P_0 \leftarrow \langle x_0^{(i)} : w_0^{(i)} \rangle_{i=1}^N$  be a representation of the categorical distribution  $p(x_0)$
  2. Prediction
    - Let  $P_t \leftarrow \langle \rangle$  be an empty map.
    - **For** each prior state  $\langle x_{t-1}^{(i)} : w_{t-1}^{(i)} \rangle \in P_{t-1}$ :
      - Compute all successor states of  $x_{t-1}^{(i)}$ :  $\langle x_t^{(j)} : w_t^{(j)} \rangle_{j=1}^M \leftarrow \text{PREDICT}(x_{t-1}^{(i)})$
      - Multiply prior weight and marginalize:  $P_t \leftarrow P_t \oplus \langle x_t^{(j)} : w_t^{(j)} * w_{t-1}^{(i)} \rangle_{j=1}^M$
  3. Update
    - **For** each  $\langle x_t^{(i)} : w_t^{(i)} \rangle \in P_t$ :  $w_t^{(i)} \leftarrow w_t^{(i)} p(y_t | x_t^{(i)})$ .
    - Normalize  $P_t$ .
  4. Pruning
    - Keep  $N$  states with highest weight (or more elaborate pruning strategy [163])
    - Set  $t \leftarrow t + 1$  and go to step 2
- 

limit the number of particles that represent  $p(X_t | y_{1:t})$  by an operation called *pruning*. An unbiased and optimal (in the sense of least squared error) pruning strategy is presented by Nyolt and Kirste [163]. The algorithm is specified in more detail in Algorithm 1.

Note that this algorithm does not require resampling and different particles never represent identical states, thus sample impoverishment does not occur. Marginal filtering has been successfully applied to BF tasks in large CSSMs, e.g. a CSSM for human activity recognition in a cooking scenario with 146 million discrete states [164].

## 2.3. Multiset Rewriting Systems

In the following, we provide an introduction to Multiset Rewriting Systems (MRSs). The presentation is kept relatively brief and informal at this point. Later (Section 4.1), we will show formal details of a newly proposed MRS that can be used to specify the transition model of a state space model.

### 2.3.1. Multiset Rewriting Systems

Multiset Rewriting Systems (MRSs) are a formalism to model multi-entity systems. They are for instance used to model cell-biological processes [47, 64, 96], population dynamics [171] or network protocols [35]. In MRSs, states of a dynamic system are multisets, describing which “things” and how many of them exist at a specific point in time.

**Definition 4.** [multiset] Let  $\mathcal{E}$  be a set of *entities*. A multiset  $x \in \mathcal{X}$  (over  $\mathcal{E}$ ) is a map (partial function)  $x : \mathcal{E} \rightarrow \mathbb{N}$  from entities to natural numbers (called *multiplicities* in this

## 2. Background

context). In the following, the set of multisets over  $\mathcal{E}$  is denoted as  $\text{mset } \mathcal{E}$ .

For entities  $e_1, \dots, e_k \in \mathcal{E}$  and multiplicities  $n_1, \dots, n_k \in \mathbb{N}$ , we write  $\llbracket n_1 e_1, \dots, n_k e_k \rrbracket$  to denote a multiset, where the multiplicity of  $e_i$  is  $n_i$  and the multiplicities of all entities not listed is zero<sup>4</sup>. 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  $x \ominus x'$ , and multiset subset relation  $x \sqsubseteq x'$  are defined as usual [23]. 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*<sup>5</sup>. An action is a triple  $(l, r, \kappa) \in \mathcal{X} \times \mathcal{X} \times \mathbb{R}_{\geq 0}$ , where  $l$  and  $r$  are multisets called *reactants* and *products*, respectively, and  $\kappa$  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 reactants are contained in  $x$ , i.e.  $l \sqsubseteq x$ , and is *applied* to  $x$  by removing the reactants, and adding the products:  $x' = (x \ominus l) \uplus r$ . We denote actions  $a = (l, r, w)$  as  $l \xrightarrow{\kappa} r$ . 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 5.** Consider the MRS  $(\mathcal{E}, A, x_0)$  with  $\mathcal{E} = \{A, B\}$ ,  $A = \{a_1, a_2, a_3\}$ ,  $a_1 = \llbracket 1X \rrbracket \xrightarrow{\kappa_1} \llbracket \rrbracket$ ,  $a_2 = \llbracket 1X, 1Y \rrbracket \xrightarrow{\kappa_2} \llbracket 2X \rrbracket$ ,  $a_3 = \llbracket 2Y \rrbracket \xrightarrow{\kappa_3} \llbracket 3Y \rrbracket$  and  $x_0 = \llbracket 2X, 1Y \rrbracket$ , which models a simple predator-prey system where  $X$  are predators and  $Y$  are prey,  $\kappa_1$  is the rate of death of predators,  $\kappa_2$  is the consumption rate, and  $\kappa_3$  is the reproduction rate of prey. The initial state  $x_0$  has the successor states  $\llbracket 1X, 1Y \rrbracket$  (by applying  $a_1$ ) and  $\llbracket 3X \rrbracket$  (by applying  $a_2$ ). The action  $a_3$  is not compatible to  $x_0$ , as there are not enough  $Y$  entities in  $x_0$ .

As we have seen in the example, for a given state  $x$ , multiple actions can be compatible. Thus, we need a mechanism for deciding which action to apply to  $x$  to obtain a successor state. Specifically, we are interested in modeling *probabilistic* systems, where a probability distribution over possible actions is defined.

Most of the modeling formalisms in systems biology specify the probabilistic semantics via a continuous-time Markov chain (CTMC), and draw simulations by using the Gillespie algorithm [75]: The algorithm first samples the time when the next action occurs from an exponential distribution, and then samples the specific action that occurs at that time.

In this thesis, we are interested in MRSs as a model for describing the system dynamics of *state space models*, for which we want to perform Bayesian filtering. In state space models, the system dynamics is specified in terms of a discrete-time Markov chain (DTMC), as the time steps at which the system state can change are externally imposed due to the observations. Thus, the CTMC semantics is not suitable for our purposes, and instead, we continue with describing a DTMC-based semantics [10].

Suppose that  $A$  is the set of actions that is applicable to  $x$ . The unnormalized probability of  $a = (l, r, \kappa) \in A$  is computed as  $v_x(a) = \kappa \prod_{e \in \text{dom}(l)} \binom{x \# e}{l \# e}$ . The binomial coefficient models in how many different ways the action can be instantiated, i.e. the number of ways in which entities from  $x$  can be chosen as the reactants of  $a$ . The probability  $p(a | x)$  of selecting action  $a$  in state  $x$  is simply the normalized probability, i.e.  $p(a | x) = v_x(a) / \sum_{a' \in A} v_x(a')$ .

**Example 6.** Consider the MRS from Example 5, the state  $x = \llbracket 2X, 3Y \rrbracket$  and the kinetic constants  $\kappa_1 = \kappa_2 = 1$  and  $\kappa_3 = 2$ . In  $x$ , all actions  $a_1, a_2$  and  $a_3$  are compatible, and their

<sup>4</sup>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.

<sup>5</sup>In the planning community, a rewriting rule would be called *action schema*.



unnormalized probabilities are  $v_x(a_1) = \kappa_1 \binom{2}{1} = 2$ ,  $v_x(a_2) = \kappa_2 \binom{2}{1} \binom{3}{1} = 6$  and  $v_x(a_3) = \kappa_3 \binom{3}{2} = 6$ . Normalizing these probabilities leads to  $p(a_1 | x) = 1/7$ ,  $p(a_2 | x) = p(a_3 | x) = 3/7$ .

Based on this definition of action probabilities, it is easy to *sample* a trajectory (i.e. a sequence of states) as follows: Given a state, test which actions are applicable, select one of the actions according to their probabilities, compute the successor state, and repeat.

### 2.3.2. Maximally Parallel MRS

In the simple model described above, we assumed that at each time step, only a single action is executed. For many practical applications, this assumption does not hold, and it becomes necessary to model multiple actions that occur in parallel in a single transition. Specifically, when using an MRS to specify the transition model of a state space model, multiple entities might be able to act between observations, i.e. time steps.

*Maximally parallel* MRSs (MPMRSs) [9] are a principled formalism to model such systems. Maximally parallel rule application is, for example, typically used in P systems [170], motivated by the fact that in cell-biological systems, multiple reactants can interact at the same time. In MPMRSs, each state transition consists of the parallel application of a *multiset of actions*, called *compound action*. A compound action is called *applicable* to a state  $x$  when the multiset of all reactants is contained in  $x$ , and *maximal* when no action can be added to the compound action so that it is still applicable. The set of all applicable and maximal compound actions (AMCAs) define the system dynamics. An AMCA  $k$  is applied to a state  $x$  by removing all of the reactants from  $x$  and adding all the products to  $x$ , i.e.  $x' = x \setminus \left( \biguplus_{(l,r,\kappa) \in k} l \right) \uplus \left( \biguplus_{(l,r,\kappa) \in k} r \right)$ .

Similarly to single-action (sequential) semantics, a distribution over AMCAs can be defined based on the weights of individual actions and the number of ways to instantiate the reactants, as shown by Barbuti et al. [9].

MPMRSs are strictly more expressive than sequential MRSs: Sequential semantics can be modeled in an MPMRS by introducing a *mutex* entity to the state that required by each action, so that each applicable and maximal compound action has a cardinality of one. On the other hand, an MPMRS cannot simply be modeled by sequential semantics, as shown in Appendix E.

This concludes the general introduction to MRSs. In Section 4.1, we present the requirements and definitions for the specific MRS that is used in this thesis as a basis for efficient Bayesian filtering.



# Symmetry-Aware Probabilistic Inference: A Systematic Review

**CHAPTER SUMMARY** *We systematically review the existing work on probabilistic inference methods that exploit symmetries, with an emphasis on Bayesian filtering. We derive a novel classification scheme, that allows us to identify eight groups of symmetry-aware inference algorithms. For the first time, this survey draws connections between research fields like lifted inference, logical filtering and multiset rewriting, and outlines the common idea shared by these approaches.*

*Parts of this chapter are based on:*

[134] Stefan Lüdtke, Max Schröder, Frank Krüger, Sebastian Bader, and Thomas Kirste. State-Space Abstractions for Probabilistic Inference: A Systematic Review. *Journal of Artificial Intelligence Research*, 63:789–848, 2018.

## Contents

---

<b>3.1. Systematic Literature Review</b>	<b>22</b>
3.1.1. Research Question	22
3.1.2. Search Procedure	23
3.1.3. Paper Selection	23
3.1.4. Analysis Procedure: Algorithm Properties	24
3.1.5. Quantitative Results	26
<b>3.2. Symmetry-Aware Inference Algorithms</b>	<b>27</b>
3.2.1. Lifted Probabilistic Inference	27
3.2.2. Inference in Continuous Domains	31
3.2.3. Relational Bayesian Filtering	32
<b>3.3. Conclusion</b>	<b>36</b>

---

### 3. Symmetry-Aware Probabilistic Inference: A Systematic Review

As outlined in Chapter 1, the goal of this thesis is to devise an efficient Bayesian filtering algorithm for multi-entity systems. In Section 1.2, we identified multiset rewriting systems (MRSs) as an obvious choice to model multi-entity systems, that can also inherently exploit symmetry in the system by grouping identical entities together. However, there are many other concepts for making use of symmetries in probabilistic models. Most prominently, the field of *lifted probabilistic inference* [50] is concerned with efficient inference in relational probabilistic models (like Markov Logic Networks [180] or parametric factor graphs [173]), by exploiting the symmetries that arise due to the high-level specification of the probabilistic model.

In this chapter, we review these methods to investigate if and how they can handle the usage situations and requirements described in Section 1.2. Such algorithms have been devised in a number of research fields, like modeling and simulation or multiple object tracking. Although they are concerned with a similar problem, the relationship between all of these approaches has not been investigated systematically, and is not always obvious due to a different terminology. Therefore, we use the procedure of a *systematic literature review* to relate approaches from these different research fields.

## 3.1. Systematic Literature Review

In the following, we briefly describe the systematic review methodology that we applied. Systematic literature reviews are common in fields like biomedical research, for collecting evidence regarding, for example, the effect of treatments, or the accuracy of diagnostic tests. Their goal is identifying all relevant research regarding a specific research question, by following a reproducible and objective process. Unstructured reviews have a higher chance to miss out contributions, either because they have not been found or because of *narrative distortion*, i.e. the fact that the author of a review is more likely to include papers that support the argumentation structure of the review.

According to Kitchenham [115], a systematic literature review consists of the following steps: (1) Definition of the research question, (2) definition of the search procedure, (3) identification of research items (papers), (4) paper selection, (5) paper analysis. In the following, we describe how each step of this procedure was performed for this systematic review.

### 3.1.1. Research Question

As described above, the goal of this review was to give an overview over symmetry-aware inference algorithms from different research fields, and to organize the research by identifying the underlying structure. Specifically, these questions can be stated as follows:

- Q1** Which methods exist for exploiting symmetries in probabilistic models to increase inference efficiency?
- Q2** Which types of problems can different methods be applied to, and how is this reflected by the properties of the methods?
- Q3** How are these methods related to each other, i.e. are similar concepts used in multiple approaches?

First term set	Second term set
lifted	Bayesian inference
first order	probabilistic inference
higher order	probabilistic reasoning
symmetry	graphical model
permutation	Bayesian network
multiset	state space model
	recursive Bayesian estimation
	Bayesian filtering
	particle filter
	hidden Markov model
	probabilistic multiset rewriting
	multi-agent
	multi-target
	multi-object
	activity recognition
	plan recognition

Table 3.1.: Search terms used to construct search query.

### 3.1.2. Search Procedure

For the literature search, we used the publication databases ScienceDirect, IEEE Xplore, ACM digital library, and Scopus. These databases were chosen based on their relevance for computer science publications, and the possibility to perform a search only on title, abstract and keywords of a publication<sup>1</sup>. Our definition of search terms was based on 10 pilot papers [9, 51, 79, 88, 104, 124, 141, 155, 173, 200] that were the result of an initial exploration. The search terms have been iteratively refined during the search process. The resulting terms are shown in Table 3.1.

We constructed the query by connecting all terms in a set with logical OR and both sets with logical AND. This query describes all papers where at least one of the terms of the first set and at least one of the elements of the second set occurs. The search has been performed on the title, keywords and abstract of the publications.

### 3.1.3. Paper Selection

The search results have been assessed based on the following inclusion criteria.

- I1** The paper is written in English.
- I2** The paper is peer-reviewed.
- I3** The full text of the paper is available via IEEEExplore, the ACM Digital Library, SpringerLink, ScienceDirect, or other sources like the author's website.
- I4** The paper includes a novel algorithmic contribution.

<sup>1</sup>Another common publication database, SpringerLink, was not used because it only allowed full text searches as of January 2017.

### 3. Symmetry-Aware Probabilistic Inference: A Systematic Review

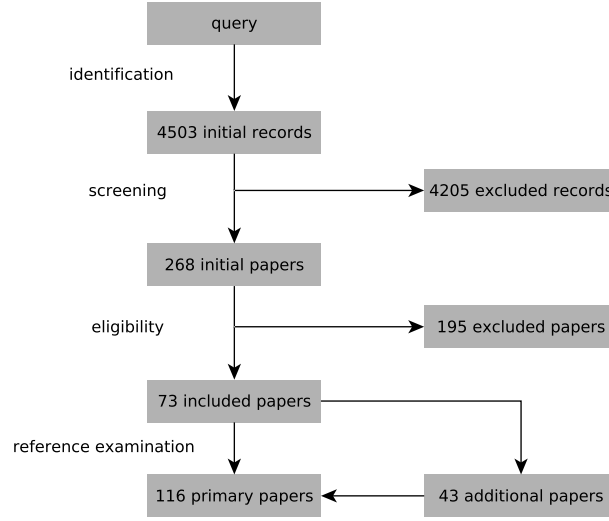


Figure 3.1.: Flow diagram of paper selection.

**I5** The paper is considering a probabilistic model.

**I6** The paper presents an inference algorithm for the probabilistic model.

**I7** The inference algorithm makes use of symmetries in the probabilistic model.

Criteria **I1-I3** make sure that the analysis of the papers is feasible for us, and criterion **I4** ensures that application and review papers are excluded. Criterion **I5** implies that only approaches that model a probability distribution have been considered. Methods in *deterministic* settings, like first-order resolution, or state space abstraction in search problems [86], were excluded by this criterion. Finally, criterion **I7** ensures that only approaches that exploit some form of symmetry were considered. Here, we interpreted the term *symmetry* liberally to mean exchangeability of random variables, graph symmetries in a probabilistic graphical model, random variables that can be handled analytically (because their distribution follows a parametric form), and the case where the assignments have an underlying algebraic structure that can be exploited.

Paper inclusion/exclusion followed a three-step process. At first, only the title, abstract, and keywords of each publication have been examined. The full-text of the remaining papers has been examined in more detail. By examining the references in the remaining papers, we identified additional relevant papers (see flow diagram in Figure 3.1).

#### 3.1.4. Analysis Procedure: Algorithm Properties

We analyzed the remaining papers in order to answer research questions **Q1 – Q3**. This was done by identifying six *properties* of inference algorithms, that follow directly from the algorithm requirements described in Section 1.2: The concepts that the algorithms use for abstraction (parametization and groping of RVs), their capabilities to modify the degree of abstraction (identification, splitting, merging) and their ability to perform inference in dynamic domains (online inference). In the following, these properties are presented briefly.

**Can the algorithm efficiently handle redundant parts of the model? (Group Variables)**

Probabilistic models can contain *redundancies*, such that parts of the model behave identical. For example, a graphical model that has been obtained by grounding a relational probabilistic model (like a parfactor graph [173] or a Markov Logic Network [180]) typically contains multiple identical factors. As a concrete example, consider a distribution over random variables  $A$ ,  $B$  and  $C$  with  $p(A, B, C) = p(C) p(B | C) p(A | C)$ , i.e.  $A$  is independent of  $B$ , given  $C$ . Now, suppose that the distribution was generated from a relational graphical model and thus obeys to some additional symmetry, say  $p(A | C) = p(B | C)$ . In this case, the joint distribution simplifies to  $p(A, B, C) = p(C) p(A | C)^2$ , i.e. we do not need to compute the value of  $P(B | C)$  at all, which can lead to vastly increased inference time. This property describes whether probabilistic inference algorithms can make use of such redundancies.

**Can the algorithm handle distributions at the parametric level? (Parameterization)**

Another method for handling a distribution more efficiently – apart from exploiting redundancies – is to *parameterize* the distribution. That is, when the distribution follows some parametric form, it is sufficient to store and manipulate the parameters, instead of enumerating all values. For example, the Rao-Blackwellized particle filter (Section 2.2.3) makes use of this concept, by handling some factors of the filtering distribution parametrically.

**Can the algorithm obtain a more specific distribution representation? (Splitting)**

The next three properties describe the capabilities of the algorithms to modify the degree of abstraction: *Merging* and *Splitting*. Splitting is the process of obtaining a more specific (propositional) representation from an abstract representation (in logic, this operation is known as *grounding*). Splitting operations are necessary for incorporating observations: In the example above, when we obtain evidence for the random variable  $A$ , but not for  $B$ , the factors  $p(B | C)$  and  $p(A | C)$  need to be treated separately again.

**Can the algorithm obtain a more abstract distribution representation? (Merging)**

Merging (or lifting) is the reverse process to splitting: Obtaining a more abstract or aggregated representation, by identifying redundancies. Merging is necessary in all domains where either the problem is given in a propositional form, or domains where the problem degenerates over time by repeated splitting operations. Splitting and merging only change the *representation* of a distribution, they do not change the distribution itself (in the approximate case, they try to change the distribution as little as possible).

**Can the algorithm handle information about individuals? (Identification)** A common question arising for abstract representations (like relational probabilistic models) is how information about single individuals is handled. To integrate such information, the algorithm either needs to perform a *split* operation, or, when the model is given in propositional form and merging operations are applied to it, consider the evidence before merging.

**Can the algorithm perform inference in dynamic domains? (Online)** This property describes the difference between (general) probabilistic inference and Bayesian filtering: Probabilistic inference answers a query for a *single* point in time, given evidence, and Bayesian filtering answers a *sequence* of queries, one for each time step. Bayesian filtering problems

### 3. Symmetry-Aware Probabilistic Inference: A Systematic Review

Crit.	#	Explanation
<b>I1</b>	3	Paper not written in English
<b>I2</b>	0	Full-text not available
<b>I3</b>	9	Paper not peer reviewed
<b>I4</b>	31	Paper does not contain a novel algorithmic contribution (e.g. application and review papers)
<b>I5</b>	11	Model is not probabilistic (e.g. inference in first-order logic)
<b>I6</b>	77	No inference algorithm for probabilistic model (e.g. because paper presents an algorithm for learning the model structure, or something completely different, like planning or model checking)
<b>I7</b>	63	Propositional model used, or the inference algorithm relies on complete grounding

Table 3.2.: Reasons for excluding 195 of the 268 papers that remained after examining title, keywords and abstract of the 4503 initial records.

cannot be solved efficiently by general-purpose probabilistic inference algorithms via *unrolling* the model, as the length of the observation sequence is typically unbound, so that the size of the resulting graphical model would grow indefinitely over time. Instead, efficient Bayesian filtering algorithms perform *online* inference, where inference complexity is linear in the number of time steps, and observation sequences of indeterminate length can be processed.

#### 3.1.5. Quantitative Results

From the 4503 initial records that have been retrieved by the database search, 4235 have been excluded by only examining their title, keywords, and abstract. The relevance of the remaining 268 papers (regarding the inclusion criteria) has been examined based on the full-text. 195 of those papers have been excluded based on the inclusion criteria, as shown in Table 3.2. The high number of papers excluded because of **I6** shows that the query terms have been chosen broadly, such that also a great number of papers that are not concerned with probabilistic inference have been retrieved. Most of the papers excluded because of **I7** are concerned with inference in relational probabilistic models by complete grounding.

The remaining 73 papers were considered relevant and included into this review. The references of these papers were examined, which lead to the identification of another 43 relevant papers. Thus, 116 papers have been included in this review in total. This corresponds to a precision of  $73/4503 = 1.6\%$  and a recall of  $73/116 = 62.9\%$  of the initial query.

The properties of the approaches presented in these 116 papers have been evaluated, as described in Section 3.1.4. We then clustered the approaches, based on these properties. That is, all approaches having the same manifestation of the properties form a cluster. With this process, we found eight distinct groups (i.e. clusters). We assigned names to the groups that seemed appropriate to us. The groups are shown in Table 3.3. The complete list of all papers per group is shown in Appendix C. We want to emphasize that the groups have not been predefined, but they are a result of the individual analysis of each paper.

As can be seen from Table 3.3, the “lifted inference” groups contain by far the most papers, showing that lifted inference is an active research area. Figure 3.2 shows the chronological



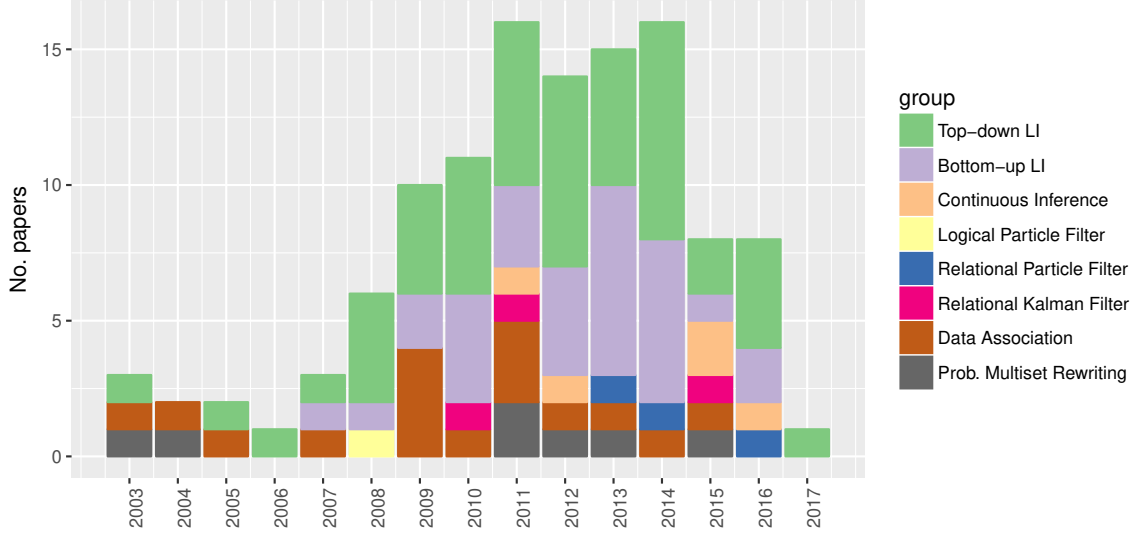


Figure 3.2.: Number of examined papers per year. The papers have been retrieved from January to February 2017. The groups are based on the analysis and clustering of approaches, as described in the text.

development of the research area. Although the first lifted inference paper was published in 2003, the majority of lifted inference papers has been published after 2008. The drop in the total number of included papers after 2014 may be due to the fact that not all papers from 2015 and 2016 are properly indexed at the used publication databases at the time of retrieving the papers (January – February 2017).

## 3.2. Symmetry-Aware Inference Algorithms

In this section, the main result of the systematic review is presented: Each of the eight groups that were identified by the systematic review are described in more detail. Additional related work – that was not included in this review because it was not identified by the search procedure described above or because it did not match the inclusion criteria, but that we still considered relevant – is presented in Appendix B.

### 3.2.1. Lifted Probabilistic Inference

*Lifted probabilistic inference* algorithms aim at exploiting the inherent symmetries and redundancies that arise in relational graphical models (see Section 2.1.3 for an example of such redundancies), i.e. they exploit *exchangeability* of the underlying distribution.

A formal definition of lifted inference is given by Van den Broeck [224]: An inference algorithm is *domain-lifted* for a relational graphical model  $\Delta$ , query  $q$  and evidence  $e$  if it computes  $p(q|e)$  in polynomial time in each  $|D_1|, \dots, |D_k|$ , where  $D_i$  is the domain of the parameters of the par-RVs appearing in  $\Delta$ ,  $q$  or  $e$ . Here, for the inclusion of algorithms, we did not use this strict definition, as many inference algorithms that exploit symmetries do not achieve fully domain-lifted inference, or the definition is not directly applicable (e.g. for

### 3. Symmetry-Aware Probabilistic Inference: A Systematic Review

Online	Identification	Group Variables	Parametrization	Splitting	Merging	No. Papers	Name
□	■	■	□	■	-	50	LI Top-down
□	■	■	□	-	■	31	LI Bottom-up
□	■	□	■	■	■	5	Continuous Inference
■	□	■	□	-	-	7	Multiset Rewriting
■	■	■	□	■	□	1	Logical Particle Filter
■	■	□	■	■	□	3	Relational Particle Filter
■	■	■	■	■	■	3	Relational Kalman Filter
■	■	□	■	-	-	16	Data Association

Table 3.3.: Groups of inference approaches, based on the properties defined in Section 3.1.4.

■: has property, □: does not have property, -: property not necessary/not meaningful.

approximate algorithms).

In the last 15 years, a large number of lifted inference approaches have been proposed. Following Kersting [104], we distinguish two classes of lifted inference algorithms (which also arise naturally due to the properties defined in Section 3.1.4): *Top-down* algorithms start with a relational model, and perform splitting operations when necessary, and *bottom-up* algorithms start with a propositional representation and identify the symmetrical structure by merging.

**Top-down Lifted Inference: First-Order Variable Elimination** Poole [173] proposed the first ideas related to lifted inference, in an algorithm known as *first-order variable elimination* (FOVE). The idea is to perform variable elimination directly on a parfactor graph, eliminating entire par-RVs in one step, instead of single RVs.

**Example 7.** Consider the graphical model of Example 2 and the query  $P(s(X), d=1)$ . Recall that inference in the propositional model (with  $X = \{a, b\}$ ) requires two elimination steps, the elimination of  $c(a)$  and  $c(b)$  as shown in Example 3. In the parfactor graph (Figure 2.2), we can in principle directly eliminate the par-RV  $c(X)$  by multiplying the parfactors  $\beta$  and  $\alpha$  and marginalizing  $c(X)$  to get a factor

$$f(s(X), d) = \sum_v \alpha(c(X)=v, d) \beta(s(X), c(X)=v)$$

which can be represented by the table

$s(X)$	$d$	$f$
0	0	$\beta^{(00)} \alpha^{(00)} + \beta^{(01)} \alpha^{(10)}$
0	1	$\beta^{(00)} \alpha^{(01)} + \beta^{(01)} \alpha^{(11)}$
$\vdots$	$\vdots$	$\vdots$

This factor directly leads to the query solution  $P(s(X), d=1) = f(s(X), d=1)$ .

The elimination step performed for eliminating  $c(X)$  in the example is called *inversion elimination*. Not all cases can be handled this way: For example, consider the case of eliminating  $d$ : In the ground factor graph, eliminating  $d$  means we need to multiply all  $\alpha_i$  factors, resulting in a factor of all  $c(X)$ , i.e. a factor that has exponential size with respect to the domain. In this case, inversion elimination cannot be applied, and FOVE as proposed by Poole [173] needs to ground  $c(X)$  and create the exponentially large factor of all groundings of  $c(X)$ .

However, the resulting factor (after eliminating  $d$ ) is *exchangeable*: That is, the probability only depends on the *number* of groundings of  $c(X)$  that are true, instead of the specific assignment. This was first realized by de Salvo Braz et al. [51], who presented an elimination operator that can handle this case. Later, Milch et al. [141] proposed an explicit representation of such factors, called *counting formulae*.

Concretely, for efficiently eliminating the RV  $d$ , the parfactor  $\alpha$  is first converted into a counting formula, that is indexed by *histograms* of the par-RV  $c(X)$ , i.e. that contains one entry for each number of groundings of  $c(X)$  that are true. Specifically, the resulting factor  $\alpha'$  has the form

$\#_X c(X)$	$d$	$\alpha'$
0	0	$\alpha^{(00)}^2$
1	0	$\alpha^{(00)} \alpha^{(10)}$
$\vdots$	$\vdots$	$\vdots$
2	1	$\alpha^{(11)}^2$

This conversion increases the size of the factor, but allows to eliminate the parfactor directly. From the resulting factor  $\alpha'$ , the RV  $d$  can then be eliminated directly.

Subsequently, additional elimination rules that make FOVE applicable to more cases without grounding have been proposed [6, 207, 210]. Using these rules, inference problems containing at most two parameters per parfactor can always be solved in polynomial time in the parameter domain size.

Using FOVE to answer multiple queries on the same relational graphical model is unnecessarily inefficient, as some intermediate results need to be computed repeatedly for each query. The lifted junction tree algorithm [26] alleviates this problem by introducing first-order junction trees, that intuitively represent the relational graphical model in preprocessed form so that marginal probability queries can be answered efficiently.

Other top-down lifted inference algorithms are based on recursive conditioning [174], transformation into a weighted model counting problem [223, 79], or rewriting rules [93]. Ideas related to lifted inference also arose independently in the probabilistic database community [206, 16].

### 3. Symmetry-Aware Probabilistic Inference: A Systematic Review

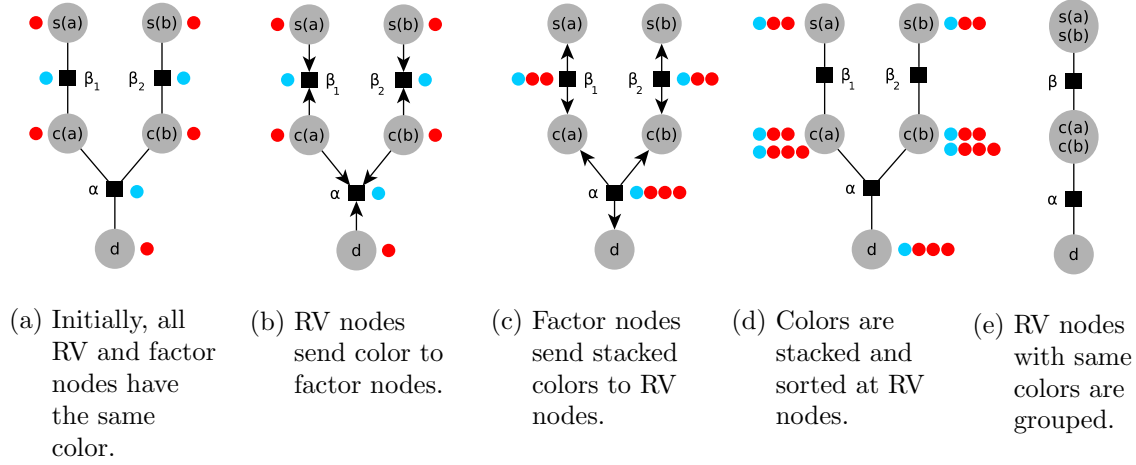


Figure 3.3.: Steps of lifted BP factor graph compression. Adapted from Kersting et al. [105].

**Bottom-up Lifted Inference: Lifted Belief Propagation** In contrast to top-down lifted inference algorithms, bottom-up algorithms start with a propositional model and perform merging operations to obtain a relational structure that can be exploited. Thus, bottom-up approaches are potentially applicable to a larger class of problems, as they do not require that the model is given in relational form (or to even contain exact symmetries, instead they can approximate the model by a symmetric one). However, the necessary merging operations pose an additional overhead: The propositional model can be very large, and merging typically requires at least linear time in the propositional model size.

A well-known bottom-up lifted inference algorithm is lifted belief propagation [105]. The idea is to perform belief propagation (BP) on a factor graph where each node represents a set of nodes that would send and receive the same messages in standard BP. This lifted factor graph is obtained by simulating BP and keeping track of which nodes send and receive the same messages. In this simulation, each node sends its color (a signature) instead of the actual message. Initially, all RV and factor nodes have the same color signature. The colors a node receives extend the current color of the node. This color signature is sent in consecutive messages. After one iteration (all nodes have sent and received a message), nodes with the same color signature are grouped for the next iteration.

**Example 8.** Figure 3.3 shows the steps of simulating BP and compressing the factor graph of Example 2. The nodes  $s(a)$  and  $s(b)$ ,  $c(a)$  and  $c(b)$  as well as  $\beta_1$  and  $\beta_2$  have the same color signature after one iteration. Thus, they are grouped together in the factor graph. Afterwards, a modified BP algorithm is performed on the compressed factor graph. This algorithm needs to consider the actual number of messages sent and received by the grouped nodes. For example, a message sent from node  $c(a), c(b)$  to  $\alpha$  actually represents two identical messages.

Other bottom-up algorithms find symmetries in the graphical model by examining graph automorphisms of the graphical model. These automorphisms can be used for lifted variational inference [30] and lifted sampling-based inference [155, 228].

Another interesting property of bottom-up algorithms is that they can potentially also be applied to models that are not exactly symmetric. In this case, the model can be approx-

imated by a symmetrical one, so that exact or approximate lifted inference algorithms can be applied [202, 229, 222].

### 3.2.2. Inference in Continuous Domains

A substantial amount of probabilistic inference research is concerned with discrete RVs, although many practical problems require modeling continuous variables. For inference in graphical models containing continuous RVs, algorithms for discrete models cannot be used directly, as they typically rely on enumerating all values of the RV. Instead, it is necessary to describe the functional form of the factors containing continuous RVs and manipulate them analytically. Typical operations that need to be handled are marginalization and multiplication of such continuous factors. In general, these operations can be difficult, as they involve the evaluation of complex integrals that are not analytically tractable. Therefore, recent research [184, 198, 17] has focused on *piecewise polynomial* functions for describing factors which can be manipulated efficiently, as illustrated by the following example.

**Example 9.** The position of an object is observed by a noisy sensor measurement. Both the position ( $x$ ) and the observation ( $o$ ) are continuous RVs. The sensor can either fail, or work properly (modeled as a binary RV  $b$ ). In the former case, the observation density is uniform in the interval  $[0, 10]$ . In the latter case, the conditional observation density is a quadratic function, centered at the real position and truncated at a distance of one from the true position. This continuous distribution can be represented by a case statement as follows<sup>2</sup>:

$$p(o | x, b) = \begin{cases} -(o - x)^2 + 5/6 & b = 0 \wedge x - 1 \leq o \leq x + 1 \\ 1/10 & b = 1 \wedge 0 \leq o \leq 10 \\ 0 & \text{otherwise} \end{cases}$$

In the approach by Sanner and Abbasnejad [184], inference in such models is defined in terms of variable elimination. When a variable is marginalized from a case statement factor, the necessary integration is calculated symbolically. The resulting factor can be more complex than the original factor (i.e. contain more cases), but it is again a piecewise polynomial function and thus can be represented by case statements. In the context of this review, this is a splitting operation.

We can also think of a merging operation for continuous inference algorithms: Given a case statement, a merging operation finds an equivalent case statement with fewer cases. For example, consider the case statement

$$p(a) = \begin{cases} -a & -1 \leq a \leq 0 \\ a & 0 < a \leq 1 \\ 0 & \text{otherwise,} \end{cases}$$

where the first two cases can be merged into the single case  $|a|$ , when  $-1 \leq a \leq 1$ . Such operations are implicitly performed in the approach by Sanner and Abbasnejad [184], who represent case statements as some variant of algebraic decision diagrams (ADDs).

Inference algorithms in continuous or hybrid models that rely on piecewise polynomials have also been devised in the context of belief propagation [198] and weighted model counting [17].

---

<sup>2</sup>The added constant 5/6 ensures that the density is always positive and integrates to one.

### 3. Symmetry-Aware Probabilistic Inference: A Systematic Review

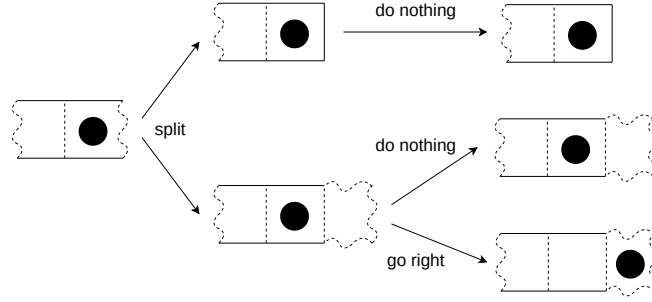
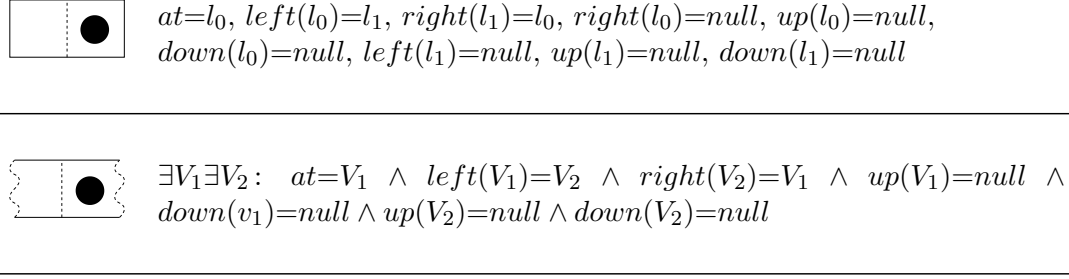


Figure 3.4.: Top: example of a state; middle: example of a hypothesis; bottom: split and prediction steps of the LPF. Figures adapted from Zettlemoyer et al. [242].

#### 3.2.3. Relational Bayesian Filtering

The remaining groups of algorithms are concerned with Bayesian filtering, i.e. inference in dynamic systems. They have been derived in a variety of research field, with different application domains and research goals: The logical particle filter [242] allows BF in worlds described by logical formulae, the relational particle filter [160, 161, 162] extends probabilistic logic programs by continuous RVs and has been used for object tracking, the relational Kalman filter [41, 39] uses continuous FOVE [38] for Kalman filtering, data association methods [194, 88] perform BF for the case where the state space is the symmetric group, and multiset rewriting systems allow to compactly describe dynamic systems where the order of elements in the state does not matter. In the following, each of those methods is described briefly.

**Logical Particle Filter** The logical particle filter (LPF) [242] is a Bayesian filtering algorithm where (ground) states are conjunctive formulae with constants denoting objects in the world, and proposition and function symbols representing the object's properties and relations. Consider a robot that moves around a maze, and needs to track its location as well as the map. An example of a state is shown in Figure 3.4 (top): The agent is at some location  $l_0$ , and there is just one other location  $l_1$  left of  $l_0$ .

The LPF does not represent states explicitly, but maintains *hypotheses*, that describe (possibly infinite) sets of states. A hypothesis is a first-order sentence, consisting of existentially quantified variables, and conjunctions of literals formed with these variables. For example, the hypothesis shown in Figure 3.4 (middle) represents all states where there is a location left of the agent's location, but no locations upwards or downwards of those two locations.

The transition model is described in terms of *rules* that have preconditions and proba-

bilistic effects. A state transition is performed as follows: First, a split operation is applied to each hypothesis, to ensure that the applicability of transition rules can be determined. For example, in the hypothesis  $h_1$  introduced above, the applicability of the rule go-right cannot be determined, as in only some of the states described by  $h_1$ , a location left of the agent's location exists. A split leads to two hypotheses: One where the rule can be applied, and one where it cannot be applied. Afterwards, the transition model is applied to each hypothesis separately. Observations can also require a split, e.g. when we observe that there *is* a location to the right with some probability.

The LPF does not work exactly, but maintains a set of weighted samples  $\{(h^{(i)}, w^{(i)})\}_{i=1}^n$  of the hypotheses. Similar to the conventional particle filter, the LPF performs *resampling* from the set of hypotheses  $\{h^{(i)}\}_{i=1}^n$ , according to their weights at each time step.

In summary, the LPF allows to maintain abstract state descriptions, where the values of some state variables is not determined. This way, all states that are only different on those indeterminate variables are grouped together. However, opposed to lifted inference algorithms, *exchangeability* of the distribution cannot be exploited in general: There is no formalism to specify that a *specific number* of state variables have a certain value (like counting formulae in lifted inference).

A problem not approached by the LPF is that variables that are instantiated once stay instantiated for this particle, i.e. merging operations for LPFs have not been devised. This can lead to a complete propositionalization of the filtering distribution over time.

**Relational Particle Filter** The relational particle filter (RPF) [160, 161, 162] is a Bayesian filtering algorithm where states, as well as the transition and observation model, are described by *distributional clauses*.

Distributional clauses are a formalism to specify conditional distributions. They have the form  $h \sim D \leftarrow B \simeq b$ , which describes the conditional distribution  $p(h \mid B=b) \sim D$ . Each of  $H$ ,  $B$  and  $D$  can have logical variables. For example, the clause

$$size(X) \sim beta(2, 3) \leftarrow material(X) \simeq metal$$

describes a conditional distribution  $p(size(X) \mid material(X)=metal)$  for each  $X$ . A *dynamic* distributional clause (DDC) furthermore allows RVs to have time indices. Thus, DDCs can be used to describe the conditional probabilities  $p(x_t \mid x_{t-1})$  and  $p(y_t \mid x_t)$  of state space models. For example, consider a dynamic system consisting of a ball and a box that can change their position over time. The transition model can be described in terms of a DDC. For example, the DDC  $pos(ID)_{t+1} \sim gaussian(\simeq pos(ID)_t, 0.1)$  represents the case where the new position of each object is distributed according to a normal distribution around the old position.

The inference algorithm performs particle filtering, using distributional clauses to specify the transition and observation model. Each particle is an assignment of values to some of the RVs, while the other RVs do not have a specific value, but its distribution is described by distributional clauses. A transition might require to know the specific value of an RV. This is achieved by sampling from the corresponding distribution – obtaining a new set of particles – and applying the transition model to each particle separately. This procedure is an instance of splitting.

The RPF can be understood as a Rao-Blackwellized particle filter (RBPF), where distributional clauses are used as a closed-form representation. However, in contrast to the

### 3. Symmetry-Aware Probabilistic Inference: A Systematic Review

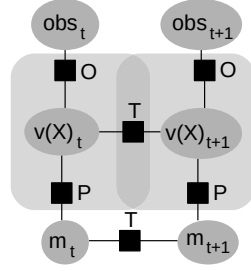


Figure 3.5.: Parfactor graph describing the relational Kalman filter for Example 10.  $P$  par-factors describe the state distribution,  $T$  parfactors correspond to the transition model,  $O$  parfactors correspond to the observation model.

standard RBPF [59], which handles some of the RVs completely in the parameter domain (i.e. updating the parameters), the RPF always needs to instantiate distributional clauses by sampling when the corresponding distributions are accessed. The RPF thus performs some form of *lazy* inference – it maintains the parametric representation until grounding is inevitable, which can lead to a complete grounding over time.

**Relational Kalman Filter** The relational Kalman filter [41] is an algorithm for Bayesian filtering that is based on continuous FOVE [38]. The standard Kalman filter assumes that the filtering distribution is a multivariate normal. Opposed to that, the system state in the relational Kalman filter is modeled as a relational pairwise model (RPM) [38], an extension of parfactor graphs where the par-RVs are continuous and the parfactors have the form

$$\phi(X, Y) \propto \exp \left( -\frac{(X - Y - \mu)^2}{\sigma^2} \right),$$

where  $X$  and  $Y$  are par-RVs. RPMs essentially represent a multivariate normal distribution with additional independence assumptions. The transition and observation model are also defined by RPMs. Based on this state representation, a Bayesian Filtering algorithm is defined in terms of continuous FOVE [38], i.e. by marginalizing out variables of the previous time step.

**Example 10.** We are interested in estimating the true value of a number of real estates over time, based on observations of sales prices and other factors like the housing market index. The value of real estate  $i$  at time  $t$  is modeled as a Gaussian RV  $v_t(i)$ , and the housing market index is modeled as a Gaussian RV  $m_t$ . At each step, several sales prices will be observed.

If we initially assume each real estate to have an identical value, the estimated  $v_t(i)$  will be the same for all unobserved  $i$ . Thus, all of these values can be represented by a single, parametric RV  $v_t(X)$ .

The dependency between the state RVs at a single time step  $t$  is represented by a parfactor (specifically, an RPM)  $P(v_t(X), m_t)$  and the observation model is an RPM  $O(v_t(X), obs_t)$ . The transition model can (for example) be described by RPMs  $T_v(v_t(X), v_{t+1}(X))$  and  $T_m(m_t, m_{t+1})$ . Figure 3.5 shows the parfactor graph describing the situation. The prediction and update steps thus have to be performed only once for each par-RV, instead of once for each RV.



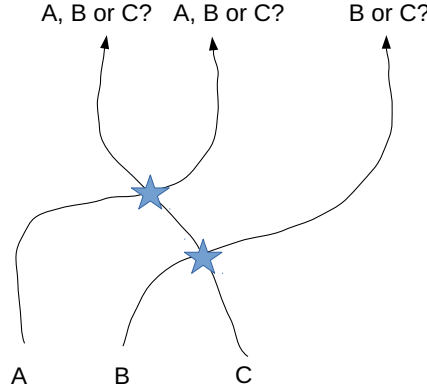


Figure 3.6.: Data association problem. Three objects A, B and C move in 2D space. The identities of the objects cannot be observed directly. When they come too close, we become uncertain about the correspondence of the objects and the tracks. Adapted from Huang et al. [88].

The key challenge of the relational Kalman filter arises when individual observations about RVs corresponding to the same par-RV are made. In this case, a split operation needs to be performed to handle each observed RV individually. Interestingly, even when individual observations are made and therefore the means of the RVs become different, the variances or RVs remain identical, which is sufficient for maintaining a relational representation. An algorithm to approximately merge variables that have become distinct due to observations has been described by Choi et al. [39].

**Data Association** Data association algorithms are concerned with the following problem: Given a number of *tracks*  $t_1, \dots, t_n$  (e.g. radar measurements, tracks of people in a video) that correspond to objects  $o_1, \dots, o_n$ , maintain the correct correspondence between tracks and objects (or, more general, a distribution of object-track associations). The problem is visualized in Figure 3.6. This problem can be viewed as performing Bayesian filtering in a state space where each state is a permutation of objects. There are  $n!$  many of these permutations, so the naive approach to maintain a distribution of those permutations explicitly is infeasible. Thus, the central task of data association algorithms is to maintain an efficient representation of distributions of permutations, and mechanisms to perform the prediction and update steps of Bayesian filtering directly on this representation.

Two conceptually different approaches for this goal have been devised. The first one, known as the Fourier-theoretic approach [87, 88, 89, 91, 117], utilizes a Fourier transformation over the symmetric group  $\mathbb{S}_n$  (the group that represents permutations of  $n$  objects). Instead of maintaining the complete distribution  $p(\sigma)$ ,  $\sigma \in \mathbb{S}_n$ , the distribution is approximated by its first few Fourier matrices, just like a function  $f(x)$ ,  $x \in \mathcal{R}$  can be approximated by its first few Fourier coefficients.

The second approach [194] maintains a compact representation of the distribution over permutations matrices by an *information matrix*  $\Omega$ . The information matrix contains unnormalized marginal probabilities  $\Omega_{ij}$  for each association of track  $i$  with identity  $j$ .

Given the information matrix  $\Omega$ , we can calculate the probability of any permutation

### 3. Symmetry-Aware Probabilistic Inference: A Systematic Review

matrix  $A$  as  $p(A) = 1/Z \exp \text{tr } A^T \Omega$ . Calculating the partition function  $Z$  is difficult, as it involves summing over all permutation matrices. However, the prediction and update steps of the Bayesian filter can be performed directly on the information matrix: The observation of an association of a track  $i$  with a specific object  $j$  leads to an increase of the corresponding value  $\Omega_{ij}$ , and the mixing of tracks  $i_1$  and  $i_2$  leads to the same values in columns  $i_1$  and  $i_2$  in the information matrix. It is also possible to maintain higher-order information matrices. For example, the *second-order information matrix* maintains scores  $\Omega_{(i_1, i_2), (j_1, j_2)}$ , where  $(i_1, i_2)$  denotes a pair of tracks and  $(j_1, j_2)$  denotes a pair of identities.

Both approaches can be seen as a projection of the true distribution to a low-dimensional subspace, but using different metrics – the L2 metric for the Fourier-theoretic approach and the KLD metric for the information-theoretic approach [95]. Both approaches have been compared by Jiang et al. [95]. They found that the Fourier-theoretic approach is better suited for scenarios with high uncertainty, while the information-theoretic approach is better suited for scenarios with low uncertainty about the data association.

**Probabilistic Multiset Rewriting Systems** Probabilistic Multiset Rewriting Systems (PMRSs) naturally emerge in this review as another category of symmetry-aware inference algorithms. Here, we only discuss them from the perspective of the algorithm properties (Section 3.1.4) and compare them with the other methods discussed in this survey – for a general introduction to PMRSs, see Section 2.3.

The appealing property of PMRSs is their intrinsic ability to aggregate redundant parts of the state and handle them as a group, due to the multiset state model. Specifically, when computing the applicable rules (and their probabilities), they only need to reason about the *number* of entities of a species in a multiset, instead of distinguishing them individually. This concept is strongly related to counting formulae in C-FOVE, where probabilities only depend on the *number* of RVs of a parfactor with a specific value, and not the specific RVs that have each value.

In contrast to relational graphical models that typically assume a fixed set of RVs, PMRSs can directly express situations where the number and/or types of entities change over time (which is common for multi-entity situations, as discussed in Section 1.2).

On the other hand, existing PMRSs do not consider observations, and thus cannot be used directly for BF. Furthermore, there is no way for existing MRS algorithms to distinguish individual entities from the other entities of that species, i.e. for splitting a species.

From a lifted inference point of view, PMRSs can be seen as constantly maintaining the filtering distribution in counting formula (i.e. lifted) form, without the possibility of observations or splitting (but with additional flexibility for expressing models with changing numbers of RVs).

### 3.3. Conclusion

The goal of this literature review was to investigate in how far existing methods can be used for efficient BF in multi-entity situations. We now summarize our findings and draw conclusions regarding a suitable method for this purpose.

- Lifted inference provides general concepts and algorithms to exploit exchangeability. However, existing algorithms were not devised for Bayesian filtering, and unrolling the

model is highly inefficient. Furthermore, they assume that the distribution is given as a graphical model, which does not easily allow for changing numbers or types of entities, which is a fundamental requirement for the multi-entity situations considered here (see Section 1.2).

- The logical particle filter (LPF) and the relational particle filter (RPF) are more closely related to the goal of this thesis. They are both Bayesian filtering methods that use a rule-based formalism to specify the transition model, and both methods exploit some form of redundancy of the distribution to achieve more efficient inference: The LPF makes use of the fact that some state variables can be uninstantiated, such that those variables can be ignored (thus reducing the number of different states), while the RPF can maintain distributions of some RVs in parametric form until they need to be accessed. However, they both cannot express the type of redundancy that arises in multi-entity situations when not all entities can be distinguished, so these entities could be grouped together. Furthermore, the LPF and RPF suffer from the fact that the representation can become completely propositionalized over time.
- The relational Kalman filter and data association methods can be very efficient in special cases, but are not applicable to the general Bayesian filtering problem in multi-entity systems, where entities can have discrete properties or properties with non-Gaussian distributions.
- Probabilistic multiset rewriting systems (PMRS) provide a general formalism for multi-entity systems where the system dynamics can be described by a set of rules, and where numbers and types of entities can change over time. Furthermore, they intrinsically handle the case where multiple entities are identical efficiently. However, Bayesian filtering algorithms – that allow the observation of individuals – have not been devised for PMRS.

In summary, PMRSs are capable of modeling the types of multi-entity systems we are interested in: The system dynamics can be described symbolically by rewriting rules, and they directly allow for changing numbers and types of entities – as required by the usage situations considered in this thesis, see Section 1.2. None of this is easily possible in (relational) graphical models – graphical models are simply not designed for this use case.

Therefore, from here on, we will focus solely on PMRSs as the modeling formalism that is underlying our BF algorithm for multi-entity systems.

Unfortunately, PMRSs have been devised for *simulations*, and thus lack a mechanism for incorporating observations. How to systematically handle observations for PMRSs, and how to perform (efficient) BF for PMRSs is the topic of the next section.



# Lifted Marginal Filtering

**CHAPTER SUMMARY** *In this chapter, we present the main technical contribution of this thesis: A lifted Bayesian filtering algorithm for systems with multiset rewriting dynamics. In Section 4.1, we start by describing the variant of MRSs on which our BF algorithm is based. Then, after showing how (ground) Bayesian filtering can be performed for this MRS (Section 4.2), we derive a more efficient representation of distributions over multisets (Section 4.3). We then present a filtering algorithm that works directly on this efficient representation (Sections 4.4). Finally, we empirically show that this algorithm leads to a factorial reduction in the representational complexity of the distribution and algorithm runtime, and in the approximate case has a lower variance of the estimate and a lower estimation error (Section 4.5).*

*Parts of this chapter are based on:*

[130] Stefan Lüdtke and Thomas Kirste. Lifted Bayesian Filtering in Multiset Rewriting Systems. *Journal of Artificial Intelligence Research*, accepted, 2020.

[132] Stefan Lüdtke, Max Schröder, Sebastian Bader, Kristian Kersting and Thomas Kirste. Lifted Filtering via Exchangeable Decomposition. *In Proceedings of the 27th International Joint Conference on Artificial Intelligence*, 2018.

[133] Stefan Lüdtke, Max Schröder, and Thomas Kirste. Approximate Probabilistic Parallel Multiset Rewriting using MCMC. *In Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, pages 73–85. Springer, 2018.

[136] Stefan Lüdtke, Kristina Yordanova, and Thomas Kirste. Human Activity and Context Recognition using Lifted Marginal Filtering. *IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE, 2019.

## Contents

---

<b>4.1. A Probabilistic Maximally Parallel Multiset Rewriting System with Structured Entities . . . . .</b>	<b>41</b>
4.1.1. Design Considerations . . . . .	41
4.1.2. MRS with Structured Entities . . . . .	42
4.1.3. Maximally Parallel MRS . . . . .	44
4.1.4. Probabilistic Maximally Parallel MRS . . . . .	45
4.1.5. An Algorithm to Enumerate AMCAs . . . . .	48
<b>4.2. Bayesian Filtering in Multiset Rewriting Systems . . . . .</b>	<b>49</b>
4.2.1. Prediction . . . . .	49
4.2.2. Update . . . . .	50
<b>4.3. Factorizing Multiset Distributions . . . . .</b>	<b>52</b>
4.3.1. Decomposing Multisets of Structured Entities . . . . .	53
4.3.2. Distributions of Value Sequences . . . . .	54
4.3.3. Lifted States . . . . .	57
<b>4.4. Lifted Filtering . . . . .</b>	<b>61</b>
4.4.1. Applying Constraints and Effects to Lifted States . . . . .	61
4.4.2. Splitting . . . . .	62
4.4.3. Disjointness of Lifted States . . . . .	69
4.4.4. The Lifted Marginal Filtering Algorithm . . . . .	71
<b>4.5. Experimental Evaluation . . . . .</b>	<b>74</b>
4.5.1. Evaluation Scenarios . . . . .	74
4.5.2. Exact Inference . . . . .	77
4.5.3. Approximate Inference . . . . .	80
4.5.4. Summary . . . . .	83

---

## 4.1. A Probabilistic Maximally Parallel Multiset Rewriting System with Structured Entities

From the literature review, we concluded that MRSs can be usefully employed for the symbolic modeling of multi-entity systems. Their appealing property is that they can directly work with groups of entities that cannot be distinguished. However, MRSs have not been used in the context of BF before, and many existing MRSs cannot be used directly for this purpose. Therefore, in this section, we present a variant of MRSs that will be the foundation of the efficient BF algorithm for multi-entity systems presented afterwards.

In Section 4.1.1, we start by discussing the specific requirements and design consideration of this MRS. Then, we present several extensions of MRS that are necessary to satisfy these requirements: *Structured entities* (which allow more flexibility and expressiveness) are introduced in Section 4.1.2, and a specific variant of probabilistic maximally parallel MRSs that accounts for the structured species is introduced in Section 4.1.3.

### 4.1.1. Design Considerations

In the following, we describe the technical requirements and considerations of the MRS that is underlying our efficient BF algorithm. The requirements directly follow from (a) the usage situations we want to model, as introduced in Section 1.2, and (b) the fact that the MRS is used in the context of BF, i.e. to specify the transition model of a state space model.

**Structured Entities** As introduced in Section 1.2, we are interested in systems where entities consist of a number of *properties*, and therefore, an MRS to model such systems requires a formalism to represent such *structured entities*. For example, consider the multi-agent activity recognition scenario in Example 1, where agents can move around multiple rooms, and pick up and manipulate objects. When the entities are flat (unstructured), each combination of properties of an agent (name, location, handled objects, current goal, ...) needs to be modeled as a separate entity. Properties can also be continuous, leading to an infinite number of different entities.

Furthermore, not all entities need to have all properties, but different entities can have different subsets of properties, e.g. in Example 1, a person is described by other properties than a coffee cup. Therefore, a vector- or sequence-based representation, where element  $i$  denotes the value of the  $i$ -th property, is not suitable. Instead, the MRS presented below represents entities as *maps*, i.e. partial functions, of property names to values.

Additionally, the structure in the entities will later allow to derive a more abstract representation, by grouping entities together that are “almost” identical, except for some properties that we did not observe, but for which we know the joint distribution.

**Flexible Specification of Actions** When using structured entities, the number of different species can easily become very large (due to the combinatorial explosion in the number of combinations of values) or even infinite (when properties have a continuous domain). In this case, the conventional specification of actions via reactants and products becomes unsuitable, as this would lead to an infinite number of actions: For example, consider the action “drinking coffee from a cup”, which can be performed when the cup is not empty. When the entity

#### 4. Lifted Marginal Filtering

representing the cup also contains the coffee temperature as a property, an infinite number of actions would be necessary, one for each possible temperature in the reactant and product.

Obviously, to encode such an infinite number of actions in a finite specification, a more flexible syntax for specifying actions is necessary. Specifically, we will use a constraint-based formalism for describing reactants, and describe products as functions of the reactants.

**Probabilities** The next two requirements arise from the fact that the MRS is used to describe the transition model of a state space model, i.e. the conditional distribution  $p(x_t | x_{t-1})$ . First of all, this means the MRS needs to specify a *distribution* over successor states, i.e. we need a *probabilistic* MRS.

**Discrete-Time Maximally Parallel Semantics** In the BF task, observations are made at *discrete* time steps. Subsequently, it is natural to use an MRS that has a discrete-time semantics, so that a single state transition occurs between observations.

However, *one state transition* does not mean that exactly a single action had occurred (i.e. a single agent acted): Instead, in the usage situations we are concerned with, multiple agents can usually act at the same time between observations. For example, in the multi-agent activity recognition scenario in Example 1, suppose that sensor observations (or preprocessed observations, e.g. aggregated features) are obtained once every 10 seconds. Between observations, each of the agents can individually perform an action. Therefore, we will use a maximally parallel MRS as a systematic formalism for such situations.

##### 4.1.2. MRS with Structured Entities

Entities with attributes and corresponding schematic actions can be found in many rule-based languages in systems biology, e.g. [47, 64, 96]. Here, we present a simple variant of these approaches, that can also naturally handle cases where the numbers of properties per entity type can change over time, which makes our approach flexible and expressive. Specifically, in our model, entities are *property-value maps*.

**Definition 5.** [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  $\mathcal{E} := \mathcal{P} \rightarrow \mathcal{V}$ , i.e. a map of property names  $\mathcal{P}$  to values  $\mathcal{V}$ .

We use  $f = \langle k_1: v_1, \dots, k_n: v_n \rangle$  to denote the partial function  $f$  where  $f(k_1) = v_1, \dots, f(k_n) = v_n$ . For example, the entity  $e$  with the keys *Name* (with value *A*) and *Loc* (with value *Table*) is denoted as  $e = \langle \text{Name: } A, \text{Loc: } \text{Table} \rangle$ . In the following, to distinguish entities from other functions, we use the notation  $e.K$  to denote the value corresponding to the key  $K$  of entity  $e$ . For example, the value for the key *Name* of entity  $e$  is denoted as  $e.\text{Name}$ .

**Example 11.** We are modeling a simplified version of the person tracking and activity recognition task introduced in Example 1: Multiple persons (agents) 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 here). 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  $\mathcal{V} = \{\text{Door}, \text{Table}, A, B, C\}$ .



#### 4.1. A Probabilistic Maximally Parallel Multiset Rewriting System with Structured Entities

A state of the system where agent A is at the table, and agents B and C are at the door, is described by the following multiset:

$$x = \llbracket 1\langle \text{Name: } A, \text{Loc: } Table \rangle, 1\langle \text{Name: } B, \text{Loc: } Door \rangle, 1\langle \text{Name: } C, \text{Loc: } Door \rangle \rrbracket \quad (4.1)$$

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. This makes the definition of actions by reactants and products, as shown in Section 2.3, infeasible, as one action definition would be necessary for each of the infinitely many entities. Instead, actions are here defined in terms of *preconditions* (that describe which constraints a structured entity must satisfy so 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 6.** [action] Let  $c \in \mathcal{C}$  be a sequence of boolean functions of entities, i.e.  $\mathcal{C} := \text{seq}(\mathcal{E} \rightarrow \{\text{true}, \text{false}\})$ , called *preconditions*, and let  $f \in \mathcal{F}$  be a function that manipulates a state, given a sequence of entities, i.e.  $\mathcal{F} := \text{seq} \mathcal{E} \times \mathcal{X} \rightarrow \mathcal{X}$ . (called *effect*). The *weight*  $\kappa$  of an action is a positive real number. An *action*  $a \in \mathcal{A}$  is a triple  $a = (c, f, \kappa) \in \mathcal{C} \times \mathcal{F} \times \mathbb{R}_{>0}$ .

Before we can give 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 a new property-value pair to an entity, and adding an entity to the state.

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

$$f(\langle \dots, e, \dots \rangle, x) = (x \uplus \llbracket 1e \rrbracket) \uplus \llbracket 1e' \rrbracket, \text{ where} \\ e' = e \odot \langle k: v \rangle$$

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.  $f(e, x) = x \uplus \llbracket 1e^* \rrbracket$ . We denote such an effect as “ $+e^*$ ”.
- The *composition* of two effects  $f_1$  and  $f_2$  is defined as  $(f_1 \circ f_2)(i_1 \odot i_2, x) = f_1(i_1, f_2(i_2, x))$ .

**Example 12.** In the office domain (Example 11), agents can move between locations. One of the actions – that describes movement between the door and the table – is the action  $\text{move-d-t} = (c, f, \kappa)$ , that is defined as follows<sup>1</sup>:

$$c(e) = (e.\text{Loc} == \text{Door}) \\ f(\langle e \rangle, x) = e.\text{Loc} \leftarrow \text{Table}$$

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*.

<sup>1</sup>Note that the expression  $e.\text{Loc} == \text{Door}$  is a Boolean expression in the constraint language (in this case, an equality constraint).

#### 4. Lifted Marginal Filtering

**Definition 7.** [action instance] An *action instance* is a pair  $(a, i) \in \mathcal{A} \times \text{seq } \mathcal{E}$  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.  $\text{items}(i) \sqsubseteq x$ .

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  $f(\langle e_1, e_2 \rangle, x) = x \cup e_2$ . 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 13.** Consider the state

$$x = \llbracket 1e_A, 1e_B, 1e_C \rrbracket,$$

where  $e_A = \langle \text{Name: } A, \text{Loc: } Table \rangle$ ,  $e_B = \langle \text{Name: } B, \text{Loc: } Door \rangle$  and  $e_C = \langle \text{Name: } C, \text{Loc: } Door \rangle$ . The action *move-d-t* from Example 12 can be applied to all entities where  $Loc == Door$ , 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

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

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

The set of all action instances of an action that are compatible with a given state can be computed by a simple backtracking algorithm that has linear runtime in the number of action instances.

##### 4.1.3. Maximally Parallel MRS

Next, we introduce the maximally parallel semantics of the MRS. The definitions are similar in spirit to conventional maximally parallel MRS as introduced in Section 2.3.2, but account for the constraint-based actions defined above. As usual in maximally parallel MRSs (MPMRSs), each state transition consists of a parallel execution of a multiset of action instances, called *compound action*.

**Definition 8.** [applicable and maximal compound action (AMCA)] A *compound action*  $k \in \mathcal{K}$  is a multiset of action instances, i.e.  $\mathcal{K} := \text{mset}(\mathcal{A} \times \text{seq } \mathcal{E})$ . 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

#### 4.1. A Probabilistic Maximally Parallel Multiset Rewriting System with Structured Entities

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 AMCAs, which define the transition model.

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 9.** [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 14.** Consider the state

$$x = \llbracket 2e_A, 1e_B \rrbracket$$

where  $e_A = \langle \text{Name: } A, \text{Loc: } Table \rangle$  and  $e_B = \langle \text{Name: } B, \text{Loc: } Door \rangle$ . Suppose that there are two actions *move-d-t* ( $a_m$ ), as defined in Example 13) and *stay* ( $a_s$ ), which has a precondition that is always true, and its effect is the identity.

There is just a single 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 – either both agents at the table move, just one of them moves or both stay where they are:

$$\begin{aligned} k_1 &= \llbracket 2(a_s, e_A), 1(a_s, e_B) \rrbracket \\ k_2 &= \llbracket 1(a_m, e_A), 1(a_s, e_A), 1(a_s, e_B) \rrbracket \\ k_3 &= \llbracket 2(a_m, e_A), 1(a_s, e_B) \rrbracket \end{aligned}$$

The situation is shown in Figure 4.1.

Single-action (sequential) application can be modeled in a maximally parallel MRS by introducing an additional *mutex* entity to the state that is a precondition to each action. On the other hand, MPMRS dynamics cannot be replicated in a sequential MRS in general, as discussed in Appendix E.

##### 4.1.4. Probabilistic Maximally Parallel MRS

Next, we introduce a probabilistic semantics for the MPMRS and show how the resulting formalism specifies a transition model  $p(x_t | x_{t-1})$ .

#### 4. Lifted Marginal Filtering

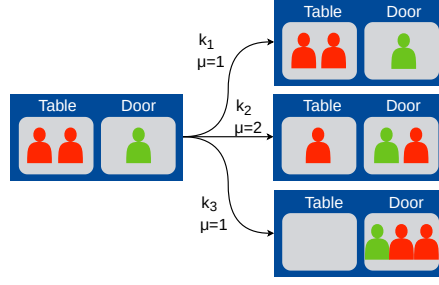


Figure 4.1.: AMCAs and successor states for Example 14: Either both red entities (entities with name A) stay at their location, one of them moves, or both move. The AMCA  $k_2$ , where one entity moves, has multiplicity 2, as either of the two entities could have moved.

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. [9] – 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*  $\alpha$  as the number of ways the bindings of  $\alpha$  can be chosen from the entities of a state  $x$ .

**Definition 10.** [multiplicity of an action instance] Let  $\alpha$  be an action instance, let  $i = \langle i_1, \dots, i_n \rangle$  be the entities bound in  $\alpha$ , and let  $x$  be a state. The multiplicity  $\mu$  of  $i$  in  $x$  is

$$\mu(i = \langle i_1, \dots, i_n \rangle, x) = \begin{cases} (x \# i_1) m(\langle i_2, \dots, i_n \rangle, x \cup \llbracket 1i_1 \rrbracket) & \text{if } |i| > 0 \\ 1 & \text{otherwise} \end{cases}$$

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 an AMCA  $k$  in state  $x$  is in principle just the product of the component action instances’ multiplicities regarding the corresponding remaining state. However, when defining the multiplicity this way, the multiplicity of the compound action is overestimated, as the action instances in  $k$  do not have an order, so not each permutation of action instances should not be counted separately.

For example, consider the state  $x = \llbracket 2y \rrbracket$  and the AMCA  $k_1 = \llbracket 2(a_1, \langle y \rangle) \rrbracket$ . Although the multiplicity  $\mu_x(a_1, y)$  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. On the other hand, the AMCA  $k_2 = \llbracket 1(a_2, \langle y, y \rangle) \rrbracket$  should have a multiplicity of 2, as the bound entities have a sequential order, so there are two

#### 4.1. A Probabilistic Maximally Parallel Multiset Rewriting System with Structured Entities

distinct ways to assign entities from  $x$  to the sequence. To obtain these desired multiplicities, we therefore need to divide by the number of permutations of identical action instances.

**Definition 11.** [multiplicity of an AMCA] Let  $k$  be an AMCA of the state  $x$ . The uncorrected multiplicity  $\mu'$  of  $k$  in  $x$  is

$$\mu'(k, x) = \begin{cases} \mu_x(\alpha) \mu'(k \cup \llbracket 1\alpha \rrbracket, x \cup \text{items}(i)), & \text{where } \alpha = (a, i) \in \text{dom}(k) \text{ if } k \neq \llbracket \rrbracket \\ 1 & \text{otherwise} \end{cases} \quad (4.2)$$

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

$$\mu_x(k) = \frac{\mu'(k, x)}{z(k)}, \text{ where } z(k) = \prod_{\alpha \in \text{dom}(k)} k\#\alpha! \quad (4.3)$$

This definition accounts for the positional preconditions, by only dividing by the number of permutations of action instances, but not by the number of permutations of bound entities in each action instance (as done by Barbuti et al. [9], where action preconditions are *multisets* and thus, the position of bound entities is irrelevant).

Finally, we define probabilities of AMCAs as their normalized multiplicity, multiplied by the product of the individual actions' weights.

**Definition 12.** [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

$$v_x(k_i) = \mu_x(k_i) \prod_{\alpha \in \text{dom}(k_i)} \kappa_a^{k_i\#\alpha}. \quad (4.4)$$

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

$$p(k | x) = \frac{v_x(k)}{Z}, \text{ where } Z = \sum_{k_i \in K} v_x(k_i) \quad (4.5)$$

**Example 15.** Consider the AMCAs

$$\begin{aligned} k_1 &= \llbracket 2(a_s, e_A), 1(a_s, e_B) \rrbracket \\ k_2 &= \llbracket 1(a_m, e_A), 1(a_s, e_A), 1(a_s, e_B) \rrbracket \\ k_3 &= \llbracket 2(a_m, e_A), 1(a_s, e_B) \rrbracket \end{aligned}$$

of the state  $x = \llbracket 2e_a, 1e_b \rrbracket$  that were shown in Example 14. Their multiplicities (Equation 4.3) are  $\mu_x(k_1) = \mu_x(k_3) = 2 * 1 * 1 / (2 * 1) = 1$  and  $\mu_x(k_2) = 2 * 1 * 1 / (1 * 1 * 1) = 2$ . Thus, assuming that the actions  $a_m$  and  $a_s$  have equal weight, the probabilities of the AMCAs are  $p(k_1 | x) = p(k_3 | x) = 0.25$  and  $p(k_2 | x) = 0.5$  (see Figure 4.1).

Computing the weight of an AMCA is closely related to the weighted model counting problem (WMC) [36]: In WMC, we are given a propositional theory  $\Delta$ , 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  $\Delta$ . Here, an AMCA corresponds to a propositional theory, and each assignment of specific entities from a state  $x$  to the AMCA corresponds to a model of that theory. In our case, the number of models can be computed directly via Equation 4.4.

**Algorithm 2** Enumerate AMCAs.

---

```

1: function ENUM-CA( $x, AI, k$ )
2:   if  $k$  maximal in  $x$  with respect to  $AI$  then
3:     return  $\{k\}$  ▷  $k$  is AMCA
4:    $K \leftarrow \emptyset$  ▷ The set where compound actions are collected
5:   for  $\{(a, i) \in AI \mid i \sqsubseteq x\}$  do ▷ Action instances applicable to  $x$ 
6:      $x' \leftarrow x \cup i$  ▷ Remaining state
7:      $k' \leftarrow k \uplus \llbracket 1(a, i) \rrbracket$  ▷ Add action instance to compound action
8:      $AI' \leftarrow \{(a', i') \in AI \mid (a', i') \geq (a, i)\}$  ▷ Allow only  $\geq$  instances in recursive call
9:      $K' \leftarrow \text{ENUM-CA}(x', AI', k')$  ▷ Recursive call
10:     $K \leftarrow K \cup K'$ 
11:  return  $K$ 

```

---

**4.1.5. An Algorithm to Enumerate AMCAs**

For simulation studies, the goal is to draw sample trajectories: Given a state  $x$ , draw a sample from  $p(k \mid x)$  (Equation 4.5), apply that AMCA to  $x$  and repeat the process. To be able to sample from  $p(k \mid x)$ , all AMCAs of  $x$  need to be enumerated, to obtain the normalization factor  $Z$ . Furthermore, enumeration of all AMCAs will also be necessary later when using MRSs for represent the transition model of a Bayesian filter, as for Bayesian filtering, we are not only interested in a sample from the posterior, but in the complete posterior distribution. In WMC terminology, we first need to *generate* all theories that have a non-zero model count, and then perform WMC for each theory.

An algorithm for enumerating all AMCAs for a given state  $x$  and a set of action instances is shown in Algorithm 2. Naively, this can be done by a simple 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  $x$ , 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 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 algorithm.

Alternatively, AMCA enumeration can be formalized as a Constraint Satisfaction Problem (CSP), so that each solution of the CSP corresponds to an AMCA (see Appendix D for details). An approximate, MCMC-based algorithm that samples AMCAs instead of performing complete enumeration is presented in Chapter 5.

## 4.2. Bayesian Filtering in Multiset Rewriting Systems

As described in Section 2.2, a state space model (for which we want to perform BF) consists of three components: A *transition model*  $p(X_t | X_{t-1})$  that describes the system behavior, an *observation model*  $p(Y_t | X_t)$  that describes how observations are related to states, and a prior distribution  $p(X_1)$ . In this section, we describe how the transition model can be derived from a given PMPMRS, and introduce a formalism for specifying observation models for multiset states. This allows to perform exact BF in PMPMRSs (instead of *sampling* state trajectories, for which MRSs are typically used).

This naive approach turns out to be infeasible for most practical cases, but forms the basis for the *lifted* filtering algorithm presented later.

### 4.2.1. Prediction

We first show how the transition model  $p(X_t | X_{t-1})$  can be derived from a given set of multiset rewriting rules. For each state  $x_t$ , a PMPMRS defines a distribution  $p(K | x_t)$  of AMCAs (Equation 4.5), which naturally leads to a transition model: Intuitively, the probability of a posterior state  $x_t$  for a given state  $x_{t-1}$  is the summed probability of all AMCAs that lead to  $x_{t+1}$ . More formally, the distribution is computed by marginalizing over the AMCAs  $K_{x_{t-1}}$  of  $x_{t-1}$  as follows<sup>2</sup>:

$$\begin{aligned}
 p(x_t | x_{t-1}) &= \sum_{k \in K_{x_{t-1}}} p(x_t, k | x_{t-1}) \\
 &= \sum_{k \in K_{x_{t-1}}} p(x_t | x_{t-1}, k) p(k | x_{t-1}) \\
 &= \sum_{k \in K_{x_{t-1}}} \mathbb{1}(f_k(x_{t-1})=x_t) p(k | x_{t-1}).
 \end{aligned} \tag{4.6}$$

The filtering distribution  $p(X_t | y_{1:t})$  is simply a *categorical* distribution in this case, i.e. we maintain a set of pairs  $(x, w)$ , where  $x$  is a possible state, and  $w$  is its probability (we only need to store the states which have non-zero probability). Technically, this can be realized as a map from states to probabilities, as in the marginal filter.

Algorithm 3 shows how a prediction step of BF (Equation 2.7) is performed in this approach, given a set of actions  $A$  and a prior state distribution  $p(X_t | y_{1:t})$  as a map  $\langle x_t^{(i)} : w_t^{(i)} \rangle_{i=1}^N$ . For each state  $x_t^{(i)}$ , we enumerate all action instances and AMCAs, compute their probabilities (Equation 4.5), compute successor states and their probabilities (Equation 4.6), and finally multiply with the prior and marginalize over  $X_t$ .

**Example 16.** Consider the following variation of the office scenario: There are three rooms (L, M, R), and two agents (A and B). 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

<sup>2</sup>We use the fact that action effects are assumed to be deterministic, i.e.  $p(x_t | x_{t-1}, k) = \mathbb{1}(f_k(x_{t-1})=x_t)$  – actions with nondeterministic effects can be modeled by introducing one action for each possible outcome of the nondeterministic effect, so that nondeterministic effects are replaced by nondeterministic choice of deterministic effects.

#### 4. Lifted Marginal Filtering

---

**Algorithm 3** Prediction.

---

- Input: Actions  $A$ , categorical distribution  $p(X_t | y_{1:t})$ , represented as  $\langle x_t^{(i)} : w_t^{(i)} \rangle_{i=1}^N$
  - Let  $P_{t+1} \leftarrow \langle \rangle$  be an empty map
  - **For** each prior state  $\langle x_t^{(i)}, w_t^{(i)} \rangle \in P_t$ :
    - Let  $AI_x = \text{ENUM-AI}(x, A)$  be the action instances of  $x_t^{(i)}$
    - Let  $K_x = \text{ENUM-CA}(x, AI_x)$  be the AMCAs of  $x_t^{(i)}$  (see Algorithm 2)
    - **For** each  $k \in K_x$ :
      - Compute successor states, multiply prior weight and marginalize:
      - $P_{t+1} \leftarrow P_{t+1} \oplus \langle f_k(x_t^{(i)}) : w_t^{(i)} * p(k | x_t^{(i)}) \rangle$
  - Return  $P_{t+1}$
- 

point during BF, the filtering distribution looks as follows:

$$\begin{aligned}
 p(x_1) &= 1/3, \quad p(x_2) = 2/3, \\
 x_1 &= \llbracket 1\langle \text{N: } \textcolor{red}{A}, \text{L: L} \rangle, 1\langle \text{N: } \textcolor{green}{B}, \text{L: M} \rangle \rrbracket \\
 x_2 &= \llbracket 1\langle \text{N: } \textcolor{red}{A}, \text{L: L} \rangle, 1\langle \text{N: } \textcolor{green}{B}, \text{L: M} \rangle \rrbracket
 \end{aligned}$$

Each state has four AMCAs: Either no agent moves, agent A moves, agent B 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 4.2 (center column).

#### 4.2.2. Update

As we saw above, PMPMRs directly induce a transition model  $p(x_t | x_{t-1})$ . 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  $p(Y_t | X_t)$  is not straightforward, as the domain of  $X_t$  will typically be very large (and can even be infinite), 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



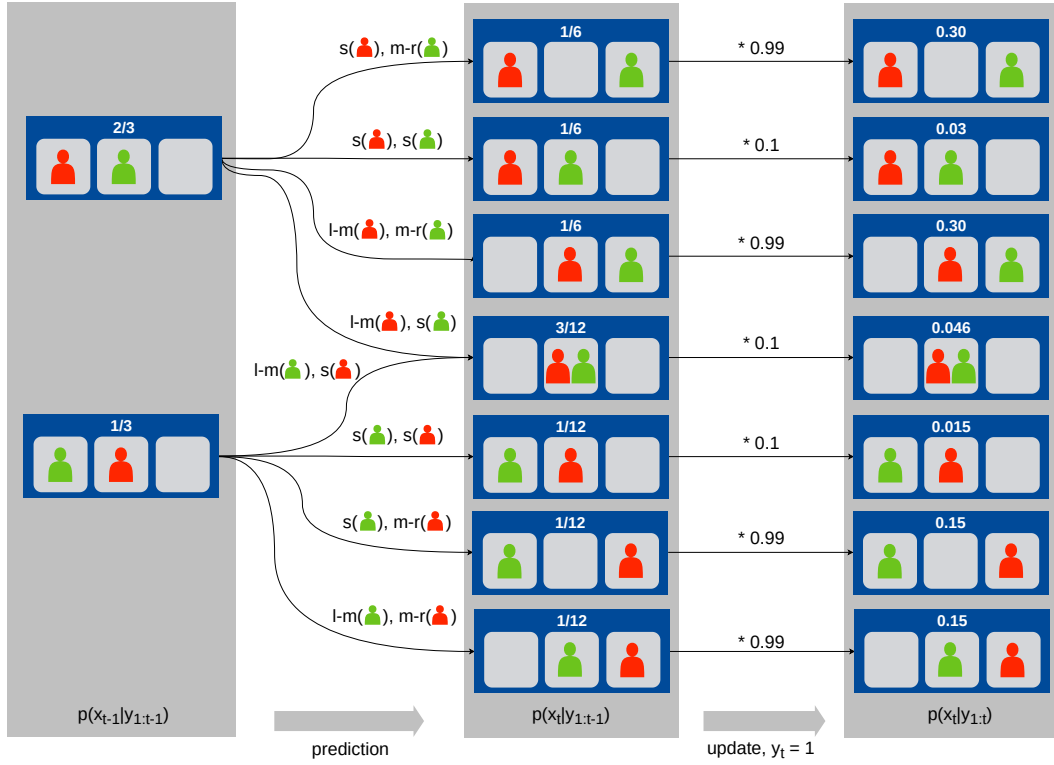


Figure 4.2.: Example of prediction and update for the office scenario (Examples 16 and 17).

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  $p(y|x)$  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 13.** [observation model] Let  $c \in \mathcal{CN} = \mathcal{X} \rightarrow \{true, false\}$  be a counting constraint, and let  $y \in \mathcal{Y}$  be an *observation*. We call  $o \in \mathcal{Y} \times \mathcal{CN} \rightarrow \mathbb{R}$  *observation model*, when for any state  $x$  and observation  $y$ , exactly a single constraint  $c$  with  $(y, c) \in \text{dom}(o)$  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 17.** Suppose that the office environment is equipped with presence sensors that are active when at least one agent is at a specific location. The sensors have a false positive rate of 0.1 and a false negative rate of 0.01. Specifically, we assume that only the right

#### 4. Lifted Marginal Filtering

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  $c(e) = e.Loc == \text{Right}$  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 4.2 (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 weighed with 0.1. Afterwards, the distribution is re-normalized.

### 4.3. Factorizing Multiset Distributions

The naive exact filtering algorithm outlined above needs to enumerate all states with non-zero probability explicitly, which can be computationally expensive, as MRS typically allow for a large number of states. This is necessary because the algorithm treats the states (i.e. multisets) as atomic terms, without considering their internal structure. In this section, we show how 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.

Technically, we present 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. 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 [66] 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  $\phi$  (that will be introduced in Section 4.3.1), 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 4.3.2, we can assume that the distribution  $p(V | S)$  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 4.4).

### 4.3.1. 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 14.** [typed entity, typed state] A *typed entity*  $e_d \in \mathcal{E}_d$  is a partial function  $e_d : \mathcal{P} \rightarrow (\mathcal{D} \times \mathcal{V})$ , 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

$$x = \llbracket 1\langle N: A_N, L: 1_{L_1} \rangle, 1\langle N: B_N, L: 2_{L_2} \rangle \rrbracket$$

consists of two entities with a property  $N$  that is drawn from a distribution with type  $N$ , one of those has a property  $L$  that is drawn from  $L_1$ , and the other one has a property  $L$  drawn from  $L_2$ . 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.

Our goal is to find a mapping between multisets (i.e. abstract objects) to syntactic structures (*terms*), such that (i) a distribution over terms can be represented more efficiently, 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 existing methods for efficiently representing distributions over tuples, like graphical models, could then 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.

Deriving a suitable bijection is not difficult, but involves some tedious technical details that are outlined in the remainder of this section. Specifically, we proceed in two steps: First, we define the canonical (ordered) *serializations*  $\sigma$  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  $\sigma$  into a pair  $(s, \mathbf{v})$  of *multiset structure*  $s$  (where multiset rewriting can be applied) and *value sequence*  $\mathbf{v}$  (such that distributions over  $\mathbf{v}$  can be represented more efficiently).

**Definition 15.** [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  $\varsigma(e) = \langle k, e.k \rangle_{k \in \text{dom}(e)}$ , 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  $\sigma$  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  $\sigma$  of  $x$  has the form

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

#### 4. Lifted Marginal Filtering

where  $\{e_1, \dots, e_n\} = \text{dom}(x)$ .

**Example 18.** The state

$$x = \llbracket 1\langle N: A_N, L: 1_{L_1} \rangle, 1\langle N: B_N, L: 2_{L_2} \rangle \rrbracket$$

has the canonical serialization

$$\sigma = \langle \langle \langle N, \langle N, A \rangle \rangle, \langle L, \langle L_1, 1 \rangle \rangle \rangle, \langle \langle N, \langle N, B \rangle \rangle, \langle L, \langle L_2, 2 \rangle \rangle \rangle \rangle$$

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

**Definition 16.** [decomposition function] Let  $x$  be a typed state, and let  $\sigma$  be the canonical serialization of  $x$ .

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

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

The association between entities in  $x$  and values is maintained by the order of the elements in the value sequence. 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 18 is

$$\begin{aligned} \phi(x) &= (s, \mathbf{v}), \text{ where} \\ s &= \llbracket 1\langle N: \mathbf{N}, L: \mathbf{L}_1 \rangle, 1\langle N: \mathbf{N}, L: \mathbf{L}_2 \rangle \rrbracket \\ \mathbf{v} &= \langle A, 1, B, 2 \rangle. \end{aligned}$$

#### 4.3.2. 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  $\phi$ .

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

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

**Independence** We assume that  $p(\mathbf{v} \mid s)$  factorizes into independent factors according to the *distribution types* (i.e. random variables 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  $x$ ). For example, the type sequence of  $s = \llbracket 1\langle \mathbf{N}: \mathbf{N}, \mathbf{L}: \mathbf{L}_1 \rangle, 2\langle \mathbf{N}: \mathbf{N}, \mathbf{L}: \mathbf{L}_2 \rangle \rrbracket$  is  $\langle \mathbf{N}, \mathbf{L}_1, \mathbf{N}, \mathbf{L}_2, \mathbf{N}, \mathbf{L}_2 \rangle$ .

In the following, we write  $\mathbf{v}^{(d)}$  to denote the sub-sequence of values with distribution type  $d$ . Based on this, the distribution  $p(\mathbf{v} \mid s)$  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  $p(\mathbf{v} \mid s)$  via the following sampling semantics: Given the factors  $p(\mathbf{v}^{(d)} \mid s)$  of a distribution  $p(\mathbf{v} \mid s)$  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  $p(\mathbf{V}^{(d)} \mid s)$  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 distribution over  $s$  and a (factorized) distribution  $p(\mathbf{v} \mid s)$ .

**Example 19.** For the office domain (Example 11), suppose we need to represent the situation “one of the three agents with names  $\mathbf{A}$ ,  $\mathbf{B}$  and  $\mathbf{C}$  is at room 1, and the other two agents are 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:

$$\begin{aligned} p(x_1) &= p(x_2) = p(x_3) = 1/3, \text{ where} \\ x_1 &= \llbracket 1\langle \mathbf{N}: \mathbf{A}_{\mathbf{N}}, \mathbf{L}: 1_{\mathbf{L}_1} \rangle, 1\langle \mathbf{N}: \mathbf{B}_{\mathbf{N}}, \mathbf{L}: 2_{\mathbf{L}_2} \rangle, 1\langle \mathbf{N}: \mathbf{C}_{\mathbf{N}}, \mathbf{L}: 2_{\mathbf{L}_2} \rangle \rrbracket \\ x_2 &= \llbracket 1\langle \mathbf{N}: \mathbf{B}_{\mathbf{N}}, \mathbf{L}: 1_{\mathbf{L}_1} \rangle, 1\langle \mathbf{N}: \mathbf{A}_{\mathbf{N}}, \mathbf{L}: 2_{\mathbf{L}_2} \rangle, 1\langle \mathbf{N}: \mathbf{C}_{\mathbf{N}}, \mathbf{L}: 2_{\mathbf{L}_2} \rangle \rrbracket \\ x_3 &= \llbracket 1\langle \mathbf{N}: \mathbf{C}_{\mathbf{N}}, \mathbf{L}: 1_{\mathbf{L}_1} \rangle, 1\langle \mathbf{N}: \mathbf{A}_{\mathbf{N}}, \mathbf{L}: 2_{\mathbf{L}_2} \rangle, 1\langle \mathbf{N}: \mathbf{B}_{\mathbf{N}}, \mathbf{L}: 2_{\mathbf{L}_2} \rangle \rrbracket. \end{aligned}$$

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

$$s = \llbracket 1\langle \mathbf{N}: \mathbf{N}, \mathbf{L}: \mathbf{L}_1 \rangle, 2\langle \mathbf{N}: \mathbf{N}, \mathbf{L}: \mathbf{L}_2 \rangle \rrbracket$$

and value sequences

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

Recall that the sequence of distribution types corresponding to these values is  $\langle \mathbf{N}, \mathbf{L}_1, \mathbf{N}, \mathbf{L}_1, \mathbf{N}, \mathbf{L}_2 \rangle$ . Therefore, the value sequences decompose into sub-sequences for each distribution type as follows:

	$\mathbf{N}$	$\mathbf{L}_1$	$\mathbf{L}_2$
$\mathbf{v}_1$	$\langle \mathbf{A}, \mathbf{B}, \mathbf{C} \rangle$	$\langle 1 \rangle$	$\langle 2, 2 \rangle$
$\mathbf{v}_2$	$\langle \mathbf{B}, \mathbf{A}, \mathbf{C} \rangle$	$\langle 1 \rangle$	$\langle 2, 2 \rangle$
$\mathbf{v}_3$	$\langle \mathbf{C}, \mathbf{A}, \mathbf{B} \rangle$	$\langle 1 \rangle$	$\langle 2, 2 \rangle$

#### 4. Lifted Marginal Filtering

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  $p(\mathbf{v}^{(\mathbf{L}_1)} | s) \sim \delta_1$ ,  $p(\mathbf{v}^{(\mathbf{L}_2)} | s) \sim \delta_{\langle 2,2 \rangle}$  and

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

**Exchangeability** As seen in Example 19, we did not achieve a more efficient representation so far: Representing  $p(x)$  directly required to store three states explicitly. By using the decomposition  $\phi$ , we still need to store three sequences  $\mathbf{v}^{(\mathbf{N})}$  explicitly for representing  $p(\mathbf{v}^{(\mathbf{N})} | s)$ , and in addition need to represent  $p(s)$ ,  $p(\mathbf{v}^{(\mathbf{L}_1)})$  and  $p(\mathbf{v}^{(\mathbf{L}_2)})$ . However, the factors  $p(\mathbf{v}^{(d)} | s)$  can be represented more efficiently than by complete enumeration by exploiting *exchangeability*, as shown next.

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  $\mathcal{V}_s^{(d)}$ . All non-canonical sequences, e.g.  $\langle A, C, B \rangle$ , must have probability of 0 in  $p(\mathbf{v}^{(d)} | s)$ , as they cannot arise from the the serialization process outlined above.

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.

$$p(\mathbf{v}_*^{(d)} | s) = \sum_{\{\mathbf{v}^{(d)} | \pi_s(\mathbf{v}^{(d)}) = \mathbf{v}_*^{(d)}\}} \tilde{p}(\mathbf{v}^{(d)} | s), \quad (4.7)$$

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

**Example 20.** Consider the distribution  $\tilde{p}(\mathbf{v}^{(d)} | s)$  with  $\tilde{p}(A, B, C | s) = \tilde{p}(A, C, B | s) = \tilde{p}(B, C, A | s) = 1/3$  (and  $\tilde{p}(\mathbf{v}^{(d)} | s) = 0$  for all other sequences  $\mathbf{v}^{(d)}$ ). Note that  $\pi_s(A, B, C) = \pi_s(A, C, B) = \langle A, B, C \rangle$ , and  $\pi_s(B, C, A) = \langle B, A, C \rangle$ . Thus, the corresponding distribution  $p(\mathbf{v}_*^{(d)} | s)$  over canonical sequences is given by  $p(A, B, C | s) = 2/3$  and  $p(B, A, C | s) = 1/3$ .

In general, this does not need to lead to a more efficient representation: The distribution  $\tilde{p}(\mathbf{v}^{(d)} | s)$  can even have a larger support than  $p(\mathbf{v}^{(d)} | s)$ , as illustrated in the example above. However, we have gained some flexibility in defining  $\tilde{p}$ , as multiple definitions of  $\tilde{p}$  can result in the same  $p$ .

Specifically, when all RVs in  $\tilde{p}(\mathbf{v}^{(d)} | s)$  are *exchangeable*, the factor can be represented much more efficiently than by complete enumeration [57]: For instance, an exchangeable distribution of  $n$  binary random variables can be represented by  $n + 1$  parameters rather than requiring  $2^n$  parameters as in the naive representation.

Furthermore, Equation 4.7 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.

$$p(\mathbf{v}_*^{(d)} | s) = \alpha(\mathbf{v}_*^{(d)}) \tilde{p}(\mathbf{v}_*^{(d)} | s), \quad (4.8)$$

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

Finally, the factor  $\alpha(\mathbf{v}^{(d)})$  is simply the product of the number of permutations of the sub-sequences of  $\mathbf{v}^{(d)}$  corresponding to each entity structure. For example, when all values in each of the sub-sequences of  $\mathbf{v}^{(d)}$  are unique, and the sub-sequences for each entity have length  $k_1, k_2, \dots, k_n$  (i.e. the entity structures in  $s$  with values of type  $d$  have multiplicities  $k_1, k_2, \dots, k_n$ ), then  $\alpha(\mathbf{v}^{(d)}) = \prod_{i=1}^n k_i!$ . Either way, Equation 4.8 can be calculated without explicitly enumerating all sequences  $\mathbf{v}^{(d)}$ .

**Example 21.** Consider  $\tilde{p}(\mathbf{v}^{(N)} | s) \sim \mathcal{U}(A, B, C)$ , where  $\mathcal{U}(A, B, C)$  represents a uniform distribution of the six permutations of A, B, C (which is obviously exchangeable). Via Equation 4.8, this distribution represents the distribution  $p(\mathbf{v}^{(N)} | s)$  shown in Example 19. For example, the probability  $p(B, A, C | s)$  is computed as

$$p(B, A, C | s) = \alpha(B, A, C) \tilde{p}(B, A, C | s) = 1! * 2! * 1/6 = 1/3.$$

**Mixtures** We made strong assumptions to the parametric form of  $p(\mathbf{v} | s)$ , instead of allowing arbitrary distributions. Fortunately, distributions where these assumptions do not hold can be represented as a *mixture* of such distributions: Each distribution can be represented as a mixture of products of exchangeable distributions – in the worst case, each mixture component describes exactly a single variable assignment (see Figure 4.3 for an example). Furthermore, exchangeability of the factors  $\tilde{p}$  is often a reasonable assumption: For example, when a property cannot be observed directly, exchangeability of that property arises naturally, as illustrated in Example 19.

Note that we do not attempt to *find* such a decomposition into exchangeable factors for a given distribution  $p(x)$ <sup>3</sup>. Instead, the idea is that an abstract (*lifted*) representation of the probabilistic model – in the form of  $p(\mathbf{v} | s)$  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<sup>4</sup>.

### 4.3.3. Lifted States

In the previous section, we showed that by making independence and exchangeability assumptions for  $p(\mathbf{v} | s)$ , only a set of exchangeable factors for each  $s$  needs to be maintained to represent that distribution, instead of completely enumerating all value sequences  $\mathbf{v}$ . In

<sup>3</sup>This task is pursued in *bottom-up lifted inference*, e.g. lifted belief propagation [105].

<sup>4</sup>Similar to what is done in *top-down lifted inference*, e.g. first-order variable elimination [173].

#### 4. Lifted Marginal Filtering

$v_1$	$v_2$	$v_3$	$v_4$	$v_5$	$v_6$	$p$
A	B	C	1	1	2	0.1
A	C	B	1	1	2	0.1
B	A	C	1	1	2	0.1
B	C	A	1	1	2	0.1
C	A	B	1	1	2	0.1
C	B	A	1	1	2	0.1
A	B	C	1	2	2	0.2
A	C	B	1	2	2	0.2

$v_1$	$v_2$	$v_3$	$v_4$	$v_5$	$v_6$	$p$
$\mathcal{U}(A, B, C)$			$\delta_1$	$\delta_1$	$\delta_2$	0.6
$\delta_A$	$\mathcal{U}(B, C)$		$\delta_1$	$\delta_2$	$\delta_2$	0.4

Figure 4.3.: Decomposition of a distribution  $p(v) = p(v_1, \dots, v_6)$  (left) into a mixture of products of exchangeable distributions (right).

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 4.4).

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 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  $\rho$ , we write  $p_\rho$  for the distribution that is represented by  $\rho$ .

The idea to represent  $p(\mathbf{v} | s)$  is to maintain a representation  $\rho$  for each factor  $p(\mathbf{v}^{(d)} | s)$ . As we have seen above, we can instead represent the *exchangeable* factors  $\tilde{p}(\mathbf{v}^{(d)} | s)$ , from which the factors  $p(\mathbf{v} | s)$  can be directly computed via Equation 4.8. Technically, this is realized as a map from distribution types  $d$  to representations  $\rho$  of exchangeable factors. We call this representation the *context* of  $s$ .

**Definition 17.** [context] A *context*  $\gamma \in \Gamma$  is a map from distribution types to representations of exchangeable distributions, i.e.  $\Gamma = \mathcal{D} \rightarrow \mathcal{R}$ . The distribution of canonical value sequences that is induced by a context  $\gamma$  and a structure  $s$  is

$$p(\mathbf{v} | s, \gamma) = \prod_{\langle d, \rho \rangle \in \gamma} p_\rho(\mathbf{v}^{(d)} | s). \quad (4.9)$$

Thus, the distribution  $p(\mathbf{v} | s)$  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 [59]. Intuitively, Rao-Blackwellization leads to more accurate estimates of the filtering distribution, because each particle represents a complete region of 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 4.4.2).

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*  $\gamma$  that represents  $p(V | s)$ . This is equivalent to directly maintaining a categorical distribution over pairs  $(s, \gamma)$ .



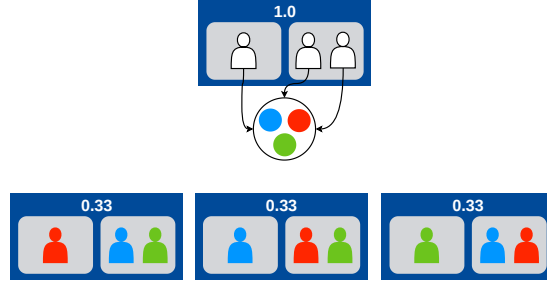



Figure 4.4.: Visualization of the lifted state from Example 22 (top) and its corresponding ground states (bottom). Colors indicate the different *names* of the agents. Here,  denotes an urn without replacement containing the three elements **A**, **B** and **C**. Instead of distribution types, arrows are used here to specify the association between entities and corresponding distribution of values.

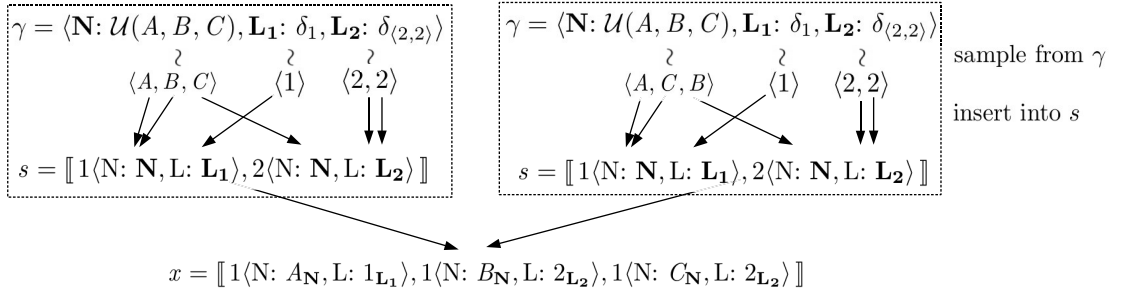


Figure 4.5.: Example illustrating the sampling semantics of lifted states: Here, two samples are drawn from  $l = (s, \gamma)$ , each by sampling a value from each factor in  $\gamma$ , and inserting into the places specified by the distribution types. In this case, both samples lead to the same ground state  $x$ .

**Definition 18.** [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  $\gamma$ . That is, the distribution of ground states, given a lifted state  $l = (s, \gamma)$  is simply

$$p(x \mid s, \gamma) = \mathbb{1}(s_x = s) p(\mathbf{v} \mid s, \gamma), \quad (4.10)$$

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  $\gamma$ , and insert the values into  $s$  at the positions indicated by the distribution types.

**Example 22.** Consider the lifted state  $l = (s, \gamma)$  with

$$s = [1\langle \mathbf{N}: \mathbf{N}, \mathbf{L}: \mathbf{L}_1 \rangle, 2\langle \mathbf{N}: \mathbf{N}, \mathbf{L}: \mathbf{L}_2 \rangle] \text{ and } \gamma = \langle \mathbf{N}: \mathcal{U}(A, B, C), \mathbf{L}_1: \delta_1, \mathbf{L}_2: \delta_{\langle 2, 2 \rangle} \rangle.$$

This lifted state (and its corresponding ground states) is visualized in Figure 4.4. It represents exactly the distribution of ground states shown in Example 19, which can be easily seen via

#### 4. Lifted Marginal Filtering

the sampling semantics (see Figure 4.5). The probability of each ground state can also be computed in closed form via Equation 4.10. For example, the probability of the ground state

$$x = \llbracket 1\langle \mathbf{N}: A_{\mathbf{N}}, \mathbf{L}: 1_{\mathbf{L}_1} \rangle, 1\langle \mathbf{N}: B_{\mathbf{N}}, \mathbf{L}: 2_{\mathbf{L}_2} \rangle, 1\langle \mathbf{N}: C_{\mathbf{N}}, \mathbf{L}: 2_{\mathbf{L}_2} \rangle \rrbracket$$

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

$$\begin{aligned} s &= \llbracket 1\langle \mathbf{N}: \mathbf{N}, \mathbf{L}: \mathbf{L}_1 \rangle, 2\langle \mathbf{N}: \mathbf{N}, \mathbf{L}: \mathbf{L}_2 \rangle \rrbracket \\ \mathbf{v}^{(\mathbf{L}_1)} &= 1, \quad \mathbf{v}^{(\mathbf{L}_2)} = \langle 2, 2 \rangle, \quad \mathbf{v}^{(\mathbf{N})} = \langle A, B, C \rangle. \end{aligned}$$

The context  $\gamma$  represents the distributions  $\tilde{p}(\mathbf{v}^{(\mathbf{L}_1)} | s) \sim \delta_1$ ,  $\tilde{p}(\mathbf{v}^{(\mathbf{L}_2)} | s) \sim \delta_{\langle 2, 2 \rangle}$ ,  $\tilde{p}(\mathbf{v}^{(\mathbf{N})} | s) \sim \mathcal{U}(A, B, C)$ . Furthermore,  $\alpha(1) = 1$ ,  $\alpha(2, 2) = 1$  and  $\alpha(A, B, C) = 2$ . Thus, according to Equation 4.10, the probability of  $x$  in  $l$  is computed as

$$p(x | s, \gamma) = \alpha(1) \tilde{p}(1) \alpha(2, 2) \tilde{p}(2, 2) \alpha(A, B, C) \tilde{p}(A, B, C) = 1 * 1 * 1 * 1 * 2 * 1/6 = 1/3.$$

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 22 above can be written as

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

Later, we will need the concept of the *region* of an entity structure (or a lifted state), which is the set of all entities (or ground states) that are assigned a non-zero probability.

**Definition 19.** [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  $l$ .

Note that regions can be infinite, if the context  $\gamma$  of  $l$  contains representations of continuous distributions. Finally, a *distribution* of lifted states  $p(L)$  defines a distribution of ground states via

$$p(x) = \sum_{l=(s, \gamma)} p(l) p(x | s, \gamma) \quad (4.11)$$

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.

As a final remark, note that lifted states directly allow to represent cases where  $p(\mathbf{v} | s)$  is *continuous*, by maintaining representations of continuous distributions in  $\gamma$ . For example, in Section 6.4, we will consider a situation where properties follow a bivariate normal distribution. This was not possible in the original representation of  $p(x)$  that worked by enumerating all states  $x$  and their probabilities, as the infinite number of states – that arise when at least one of the values follow a continuous distribution – could not be enumerated exhaustively.

To summarize the results so far, we showed how to efficiently represent distributions over structured multisets, by exploiting independence and 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 Bayesian filtering, by applying multiset rewriting directly to the structure part of the lifted states, and manipulating the value distributions only when necessary.

## 4.4. Lifted Filtering

In this section, we present *Lifted Marginal Filtering* (LiMa), a Bayesian filtering algorithm that works directly on the lifted state representation. The system dynamics is defined in terms of a PMPMRS, as introduced in Section 4.2. The key insight of this section is that the prediction and update steps can be performed directly on lifted states, so that they are equivalent to performing the same transformations to all ground states that are represented by the lifted state.

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 4.4.2) 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.

### 4.4.1. 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 4.2.2) 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  $l = (s, \gamma)$ .

We say that an entity structure  $e$  in a state  $l$  satisfies a constraint  $c$  when all groundings of  $e$  with respect to  $l$  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 4.4.2).

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 23.** Consider the lifted state  $l = (s, \gamma)$  where

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

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

Thus, given a set of actions  $A$  and a lifted state  $l$ , we can compute *lifted* action instances: A lifted action instance is a pair  $(a, e)$ , where  $a \in A$  is an action, and  $e = \langle e_1, \dots, e_n \rangle$  is a

#### 4. Lifted Marginal Filtering

sequence of entity structures, such that each  $e_i$  satisfy the corresponding constraint  $c_i$  of  $a$ . Now, we can go through all previous definitions, replacing ground states with lifted states: Lifted AMCAs and weights of lifted AMCAs can be defined analogously to Definitions 8 and 12, and thus we can define a distribution  $p(k|l)$  of AMCAs  $k$  of a given lifted state  $l$ .

In the same way as AMCA computation is related to weighted model counting (see Section 4.1.4), computation of AMCA weights for lifted states is closely related to lifted weighted model counting (LWMC) [79]: Each AMCA corresponds to a theory, and each assignment of specific entities from a state to the AMCA corresponds to a model of that theory. In contrast to LWMC, we perform splits up front during theory generation (i.e. generation of AMCAs), so that no splits need to be performed during model counting (see Section 4.4.2).

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  $l$ .

For the simple effect functions introduced in Section 4.1.2 (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. In principle, effects could also directly operate on the parameters of the factors in  $\gamma$  (e.g. performing Kalman filtering on a Gaussian factor). However, the simple constraints and effects discussed here already allow modeling of many practically relevant situations (see Section 4.5), and thus, we leave more complex constraints and effects as a topic for future work.

**Example 24.** Consider the effect  $f = e.Y \leftarrow 2$ , and suppose that the effect is applied to the entity structure  $e = \langle X: \mathbf{N}, Y: 1 \rangle$  in the lifted state  $l = (s, \gamma)$  where

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

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

$$s' = \llbracket 1\langle X: \mathbf{N}, Y: 1 \rangle, 2\langle X: \mathbf{N}, Y: 2 \rangle \rrbracket.$$

Thus, similar to Equation 4.6, the distribution  $p(k|l)$  induces a transition model  $p(l_t|l_{t-1})$  on lifted states. As constraints can be tested directly on lifted states, the constraint-based observation model (4.2.2) can also be applied directly to lifted states. Overall, Bayesian filtering can be performed on the lifted representation, without generating the ground states first. The resulting *Lifted Marginal Filtering* (LiMa) algorithm is presented in more detail in Section 4.4.4.

##### 4.4.2. 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 23). 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 partitions by a separate lifted state, because a *different* set of AMCAs is applicable for each partition

(and to compute the lifted successor states of  $l$ , it is necessary that a fixed set of AMCAs is applicable to all groundings of  $l$ ).

We want to compute lifted states that describe the partitions without requiring a complete enumeration of all ground states first. This is done by manipulating the lifted states 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.

$$\sum_i w_i p(x \mid s_i, \gamma_i) = p(x \mid s, \gamma), \quad (4.12)$$

and (ii) for each  $l_i \in L$ , all ground states  $x \in \text{region}(l_i)$  lie in the same partition regarding  $c^5$  (i.e.  $\forall l_i \forall x \in \text{region}(l_i) : x \models \#_c^i$ ).

**Example 25.** Consider the lifted state  $l = (s, \gamma)$  with

$$s = \llbracket 2\langle X: \mathbf{N}, Y: 1 \rangle, 2\langle X: \mathbf{N}, Y: 2 \rangle \rrbracket, \quad \gamma = \langle \mathbf{N}: \mathcal{U}(3A, 2B) \rangle$$

and the constraint  $c(e) = (e.N == A) \wedge (e.Y == 1)$ . This constraint is indeterminate in  $l$ , as it is satisfied zero times, once, or twice for different groundings of  $l$ . The splitting 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. The split results are identical to  $l$ , except that instead of entity structures  $\langle X: \mathbf{N}, Y: 1 \rangle$ , they contain groundings of that entity structure (i.e. where  $e(X) = A$  or  $e(X) = B$ ):

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

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 4.12). 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.

**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<sup>6</sup>. 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  $\rho$ . To keep the notation

<sup>5</sup>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, and instead, partitions for each  $n$  need to be created anyways.

<sup>6</sup>Conjunctions of those constraints can be handled consecutively by multiple splits.

#### 4. Lifted Marginal Filtering

**Algorithm 4** In lifted state  $l$  with weight  $p$ , split entity  $e$  on constraint  $q == v^*$ , distributed according to **multinomial** distribution  $\rho$ .

---

```

1: function SPLIT-MULTINOMIAL( $l=(s, \gamma), p, q, v^*, e, \rho$ )
2:    $k \leftarrow s \# e$ ;  $P' \leftarrow \emptyset$ ;  $d \leftarrow e.q$ 
3:   Create new distribution type  $d_{v^*}$ 
4:    $\rho' \leftarrow \rho$  without  $v^*$   $\triangleright$  Multinomial with parameters  $\frac{p_1}{1-p_{v^*}}, \dots, \frac{p_m}{1-p_{v^*}}$ , see Eq. 4.18
5:    $\gamma' \leftarrow \gamma \oplus \langle d: \rho', d_{v^*}: \delta_{v^*} \rangle$   $\triangleright$  Add changed representations to context
6:    $e' \leftarrow e \oplus \langle q: d_{v^*} \rangle$   $\triangleright$  New entity with constant value  $v^*$  at  $q$ 
7:   for  $i = 0, \dots, k$  do
8:      $s' \leftarrow s \cup \llbracket (k-i) e \rrbracket \uplus \llbracket i e' \rrbracket$   $\triangleright$  New structure where  $i$  entities have value  $v^*$ 
9:      $p' \leftarrow \binom{k}{i} p_{v^*}^i (1-p_{v^*})^{k-i} p$   $\triangleright$  Probability of  $l'$ , see Equation 4.17
10:     $P' \leftarrow P' \cup \{(s', \gamma'), p'\}$   $\triangleright$  Collect all split results
11:  return  $P'$ 

```

---

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  $\mathcal{V}_1, \dots, \mathcal{V}_n$ . 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  $p(V=\mathbf{v})$ :

$$p_i(\mathbf{v}) = \begin{cases} \frac{p(\mathbf{v})}{\sum_{\mathbf{v}' \in \mathcal{V}_i} p(\mathbf{v}')} & \text{if } \mathbf{v} \in \mathcal{V}_i \\ 0 & \text{otherwise} \end{cases} \quad (4.13)$$

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

$$p(V) = \sum_i w_i p_i(V). \quad (4.14)$$

For each factor  $p_i(V)$ , the splitting algorithm creates a split result  $l_i$ , which is identical to  $l$  except that the representation  $\rho$  of  $p(V)$  is replaced by a representation of  $p_i(V)$ . Due to Equation 4.14 and the fact that all other factors of  $l$  and all split results  $l_i$  are identical, this splitting procedure satisfies Equation 4.12 above, i.e. correctness requirement (i). To satisfy correctness requirement (ii) – the constraint  $c$  must be satisfied for a fixed number 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 4.5), 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).

**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:

$$p_\rho(k_1, \dots, k_m) = \frac{k!}{\prod_{n=1}^m k_n!} \prod_{n=1}^m p_n^{k_n}. \quad (4.15)$$

Intuitively, the splitting algorithm creates one split result for each possible value of  $K_j$ . That is, split result  $l_i$  represents the situation where  $K_j = i - 1$ , i.e. where the value  $v_j$  was sampled  $i - 1$  times from the urn. Before describing the splitting algorithm more formally, the concept is illustrated by the following example.

**Example 26.** Consider the lifted state<sup>7</sup>

$$l = (\llbracket 3\langle N: N, L: X \rangle, 2\langle N: N, L: Y \rangle \rrbracket, \quad \langle N: \mathcal{M}(1/2 A, 1/3 B, 1/6 C) \rangle).$$

We want to split  $l$  based on the constraint  $c(e) = (e.N == A)$  for the entity structure  $\langle N: N, L: X \rangle$ . The splitting algorithm generates four split results, where split result  $l_i$  contains  $i - 1$  times the entity structure  $\langle N: A, L: X \rangle$  (which has taken value  $A$ ), and  $k - i - 1$  times the entity structure  $\langle N: N_1, L: X \rangle$  (which has not taken the value  $A$ ):

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

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 4.17 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. A split of a lifted state  $l = (s, \gamma)$  on an equality constraint of the form  $e(q) == v_j$  (where  $e(q)$  follows a multinomial distribution and  $k = s\#e$  is the multiplicity of  $e$ ) results in  $k + 1$  split results. The set  $\mathcal{V}_{i+1}$  that is used to define the split result  $i + 1$  contains all assignments where the value  $v_j$  has been sampled  $i$  times, 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

$$p(K_1, \dots, K_{j=i}, \dots, K_n) = p(K_{j=i}) p(K_1, \dots, K_{j-1}, K_{j+1}, \dots, K_n \mid K_{j=i}) \quad (4.16)$$

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 4.13, 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 [199] that such a marginal distribution of a multinomial is a binomial distribution with parameters  $k$  and  $p_j$ , i.e.

$$p(K_{j=i}) = \binom{k}{i} p_j^i (1 - p_j)^{k-i}. \quad (4.17)$$

<sup>7</sup>We use  $\mathcal{M}$  to indicate a multinomial distribution, i.e. an urn with replacement.

#### 4. Lifted Marginal Filtering

**Algorithm 5** In lifted state  $l$  with weight  $p$ , split entity  $e$  on constraint  $q == v^*$ , distributed according to **multivariate hypergeometric** distribution  $\rho$ .

---

```

1: function SPLIT-HYPERGEOMETRIC( $l=(s, \gamma)$ ,  $p$ ,  $q$ ,  $e$ ,  $\rho = (v_j, n_j)_{j=1}^m$ )
2:    $k \leftarrow s \# e$ ;  $P' \leftarrow \emptyset$ ;  $d \leftarrow e.q$ 
3:   for each composition  $(k_1, \dots, k_m) \in \mathcal{C}(\rho, k)$  do
4:      $s' \leftarrow s \cup \llbracket k e \rrbracket$ 
5:      $\rho' \leftarrow \rho$  without  $(k_1, \dots, k_m)$ 
6:      $\gamma' \leftarrow \gamma \oplus \langle d: \rho' \rangle$  ▷ Add changed representation to context
7:     for  $j = 1, \dots, m$  where  $k_j > 0$  do ▷ Create state with entities for each  $k_j > 0$ 
8:       Create new distribution type  $d_{v_j}$ 
9:        $e' \leftarrow e \oplus \langle q: d_{v_j} \rangle$  ▷ New entity with constant value  $v_j$  at  $q$ 
10:       $s' \leftarrow s' \uplus \llbracket k_j e' \rrbracket$  ▷ New structure where  $k_j$  entities have value  $v_j$ 
11:       $\gamma' \leftarrow \gamma' \oplus \langle d_{v_j}: \delta_{v_j} \rangle$  ▷ Add representation of constant
12:       $p' \leftarrow p_\rho(k_1, \dots, k_m) p$  ▷ Probability of  $l'$ , see Equation 4.19
13:       $P' \leftarrow P' \cup \{(s', \gamma'), p'\}$  ▷ Collect all split results
14:   return  $P'$ 

```

---

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  $\frac{p_1}{1-p_j}, \dots, \frac{p_m}{1-p_j}$  [199]:

$$p(K_1, \dots, K_{j-1}, K_{j+1}, \dots, K_m \mid K_j=i) = \frac{(k-i)!}{\prod_{n=1}^m k_n!} \prod_{n=1}^m \left( \frac{p_n}{1-p_j} \right)^{k_n} \quad (4.18)$$

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  $l$ . This splitting algorithm is shown in Algorithm 4.

**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:

$$p_\rho(k_1, \dots, k_m) = \frac{\prod_{j=1}^m \binom{n_j}{k_j}}{\binom{n}{k}} \quad (4.19)$$

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

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, as 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  $e \in \text{dom}(s)$  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  $k_1 + \dots + k_m$ , with the additional constraint that  $k_j$  is at most  $n_j$ . Each composition  $(k_1, \dots, k_m) \in C(\rho, k)$  corresponds to an assignment of  $p_\rho$ , where  $k_j$  balls of value  $v_j$  are drawn. The subsets  $\mathcal{V}_i$  which are used to define the split results 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  $(k_1, \dots, k_m) \in C(\rho, k)$ ), 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  $\delta_{v_j}$ . 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 4.19). This splitting algorithm is shown in Algorithm 5.

**Example 27.** Consider the lifted state

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

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

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

The weight of each split component is defined by Equation 4.19. For example,  $w_2 = \frac{\binom{3}{2}\binom{2}{1}}{\binom{6}{3}} = 0.3$ . Note that the entity structures  $\langle \mathbf{N}: \mathbf{N}, \mathbf{L}: Y \rangle$  are not manipulated by the splitting procedure (although the *distribution* of their values is of course manipulated).

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.

**Hypergeometric Distribution with Unique Values** Finally, 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

#### 4. Lifted Marginal Filtering

**Algorithm 6** In lifted state  $l$  with weight  $p$ , split entity  $e$  on constraint  $q == v^*$ , distributed according to **multivariate hypergeometric** distribution  $\rho$  where  $\mathbf{n}_{v^*} = \mathbf{1}$ .

---

```

1: function SPLIT-HYPERGEOMETRIC-UNIQUE( $l=(s, \gamma)$ ,  $p$ ,  $q$ ,  $e$ ,  $\rho = (v_j, n_j)_{j=1}^m$ )
2:    $k \leftarrow s \# e$ ;  $P' \leftarrow \emptyset$ ;  $d \leftarrow e.q$ ;  $n \leftarrow \sum_{j=1}^m n_j$ 
3:   Create new distribution type  $d_{v^*}$ 
4:    $\rho' \leftarrow \rho$  without  $v^*$ 
5:    $\gamma' \leftarrow \gamma \oplus \langle d: \rho', d_{v^*}: \delta_{v^*} \rangle$  ▷ Add changed representation to context
6:    $E \leftarrow \{e \in \text{dom}(s) \mid \langle q: d \rangle \in e\}$  ▷ Entities that reference  $\rho$  via  $d$ , i.e. that can take  $v^*$ 
7:   for each  $e' \in E$  do ▷ Create one state for each  $e'$  that can take  $v^*$ 
8:      $e'' \leftarrow e' \oplus \langle q: d_{v^*} \rangle$  ▷ New entity with constant value  $v^*$  at  $q$ 
9:      $s' \leftarrow s \cup \llbracket 1e' \rrbracket \oplus \llbracket 1e'' \rrbracket$  ▷ New structure where one entity has value  $v^*$ 
10:     $p' \leftarrow \frac{s \# e'}{n} p$  ▷ Probability of  $l'$ , see Equation 4.20
11:     $P' \leftarrow P' \cup \{((s', \gamma'), p')\}$  ▷ Collect all split results
12:   if  $n > \sum_{e' \in E} s \# e'$  then
13:      $p' \leftarrow 1 - \frac{\sum_{e' \in E} s \# e'}{n}$ 
14:      $P' \leftarrow P' \cup \{((s, \gamma'), p')\}$  ▷ Collect all split results
15:   return  $P'$ 

```

---

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  $\rho$ . When  $e$  does not have value  $v^*$ , either one of the other entity structures  $e$  must take 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  $e_i \in \text{dom}(s)$  that has a property  $q'$  that is distributed according to  $\rho$ , 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  $\delta_{v^*}$ . 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  $e_i$ :

$$w_i = \frac{\binom{1}{1} \binom{n-1}{k_i-1}}{\binom{n}{k_i}} = \frac{k_i}{n} \quad (4.20)$$

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 6.

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  $\rho$ . 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,  $n_1 = \dots = n_m = 1$ , i.e. we can *always* use this splitting procedure instead of the general case. The following example illustrates this splitting procedure.

**Example 28.** Consider the lifted state

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

We want to split  $l$ , based on the constraint  $c(e) = (e.N == C)$  for the entity structure  $\langle \mathbf{N}: \mathbf{N}, \mathbf{L}: X \rangle$ . 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):

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

The probability of each split component is defined by Equation 4.20, i.e.  $w_1 = 3/6$ ,  $w_2 = 2/6$  and  $w_3 = 1/6$ .

#### 4.4.3. Disjointness of Lifted States

When naively performing Bayesian filtering in the lifted representation, situations can arise where regions of lifted states are not disjoint – see Figure 4.6 for an example. On the one hand, this can sometimes be desirable, because it can allow for a compact representation of distributions that can be decomposed into mixtures of “low-frequency” components (e.g. a uniform distribution over permutations) and “high-frequency components” (e.g. a distinct permutation with higher probability). On the other hand, overlapping states can lead to situations where a distribution over  $n$  ground states is represented by up to  $2^n$  lifted states in the worst case (when each lifted state represents a different subset of the ground states).

Thus, although the distribution  $p(x)$  is still well-defined even when lifted states have overlapping regions, it is sometimes preferable to prevent such an overlap, to avoid an unnecessarily complex representation. Specifically, we require an operation that identifies overlapping states on the structural level, without complete grounding, and splits them accordingly so that all split results are either identical (so that they can be merged trivially) or disjoint.

**Example 29.** Consider the states  $l_1 = (s_1, \gamma_1)$  and  $l_2 = (s_2, \gamma_2)$  with overlapping regions shown in Figure 4.6, where

$$\begin{aligned} s_1 &= \llbracket 2\langle X: \mathbf{N}, Y: 1 \rangle, 1\langle X: \mathbf{N}, Y: 2 \rangle \rrbracket, & \gamma_1 &= \langle \mathbf{N}: \mathcal{U}(A, B, C) \rangle \\ s_2 &= \llbracket 2\langle X: \mathbf{N}, Y: 1 \rangle, 1\langle X: A, Y: 2 \rangle \rrbracket, & \gamma_2 &= \langle \mathbf{N}: \mathcal{U}(B, C) \rangle. \end{aligned}$$

For this pair of lifted states, a ground state  $x \in \text{region}(l_1) \cap \text{region}(l_2)$  exists:

$$x = \llbracket 1\langle X: B, Y: 1 \rangle, 1\langle X: C, Y: 1 \rangle, 1\langle X: A, Y: 2 \rangle \rrbracket.$$

#### 4. Lifted Marginal Filtering

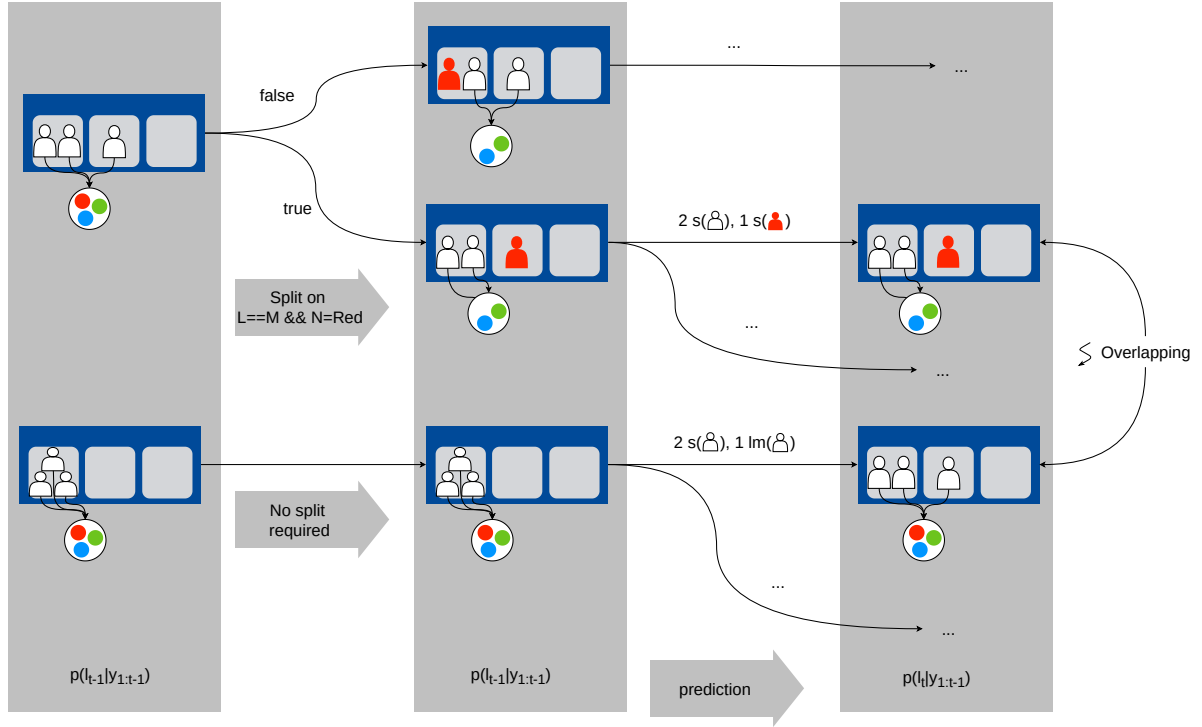


Figure 4.6.: Example of overlapping regions, that can occur due to splitting and effect application. Here,  $\textcircled{RGB}$  denotes an urn without replacement containing the three elements R, G and B.

Note that the lifted states  $l_1$  and  $l_2$  above only share the grounding  $x$  when ignoring the distribution types of groundings. When instead taking the distribution types into account,  $l_1$  and  $l_2$  do not share any groundings when, as all groundings of  $l_1$  and  $l_2$  have different distribution types.

The distribution types might be relevant in some cases, e.g. for informing the merging strategy about the distribution a value belongs to (see Chapter 6). However, in other cases, they can be irrelevant, so that we can “ignore” the distribution type for defining overlap of states, to be able to obtain a more efficient representation.

Therefore, we consider operations for identifying and eliminating states with overlapping regions for two cases: The case where ground states are only considered identical when their types are also identical, and the case where identity does *not* depend on the distribution type (i.e. considering *untyped* ground states). In the former case, identifying overlapping states is simple: States overlap when they have identical structure  $s$ , and for each type  $d$ , the factors of type  $d$  in  $l_1$  and  $l_2$  have overlapping support. In the latter case, identifying overlap is slightly more complex, as we do not know a priori how the entities from  $l_1$  and  $l_2$  are associated to the entities in  $x$  – in contrast to the previous case, where the distribution types directly provide this information. The following example shows how the overlapping states  $l_1$  and  $l_2$  from above can be split so that all split results are disjoint.

**Example 30.** We split  $l_1$  on  $e = \langle X: \mathbf{N}, Y: 2 \rangle$  with the constraint  $e.N == A$ , resulting in

---

**Algorithm 7** Lifted Marginal Filtering.

---

- Input: Actions  $A$ , observation model  $o$ , prior distribution  $p(L_0)$  represented as  $P_0 = \langle l_0^{(i)} : p_0^{(i)} \rangle_{i=1}^N$ , 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 4, 5 and 6)
      - Calculate prediction (see Algorithm 3):  $P_{t+1|t} = \text{PREDICT}(P_t, A)$
    2. Update
      - Split  $P_{t+1|t}$  on observation model constraints
      - Update weights of each  $\langle l_{t+1|t}^{(i)} : p_{t+1|t}^{(i)} \rangle \in P_{t+1|t}$ :  $p_{t+1}^{(i)} = p(y_t | l_{t+1|t}^{(i)}) p_{t+1|t}^{(i)}$
      - Let  $P_{t+1}^* = \langle l_{t+1|t}^{(i)} : p_{t+1}^{(i)} \rangle_{i=1}^M$
    3. Optional: Prune  $P_{t+1}$ , e.g. by keeping  $N$  states with highest probability (or more elaborate pruning strategy, see [163])
- 

two states  $l'_1 = (s'_1, \gamma'_1)$  and  $l''_1 = (s''_1, \gamma''_1)$  where

$$\begin{aligned} s'_1 &= \llbracket 1\langle X: A, Y: 1 \rangle, 1\langle X: N, Y: 1 \rangle, 1\langle X: N, Y: 2 \rangle \rrbracket, & \gamma'_1 &= \langle N: \mathcal{U}(B, C) \rangle \\ s''_1 &= \llbracket 2\langle X: N, Y: 1 \rangle, 1\langle X: A, Y: 2 \rangle \rrbracket, & \gamma''_1 &= \langle N: \mathcal{U}(B, C) \rangle. \end{aligned}$$

Now,  $l_2 = l'_1$ , so they can be trivially merged by summing their probabilities, and  $\text{region}(l_2) \cap \text{region}(l''_1) = \emptyset$ .

Note that such split are only applied to states with overlapping regions, but other states can be kept in fully lifted form: For example, the state  $l_3 = (s_3, \gamma_3)$  with

$$s_3 = \llbracket 3\langle X: N, Y: 1 \rangle \rrbracket, \quad \gamma_3 = \langle N: \mathcal{U}(A, B, C) \rangle$$

does not overlap with  $l_1$  nor  $l_2$  and thus does not need to be split.

The algorithms for identifying and eliminating overlap are described in full detail in Appendix F. We call this process *shattering* (named after shattering in FOVE [51], although in contrast to FOVE, shattering is not required here to guarantee the correctness of the algorithm, but merely increases efficiency).

Whether shattering should be employed depends on the specific application scenario and should be weighed against potential benefit of overlapping states. For example, in Section 6.2, we will consider situations where overlapping states (that describe almost the same ground distribution) can be approximated by a single lifted state – which is not possible when the distribution has been shattered before.

#### 4.4.4. The Lifted Marginal Filtering Algorithm

We conclude this section by summarizing the overall Lifted Marginal Filtering (LiMa) algorithm. LiMa (see Algorithm 7) performs marginal filtering (using the MRS-based transition

#### 4. Lifted Marginal Filtering

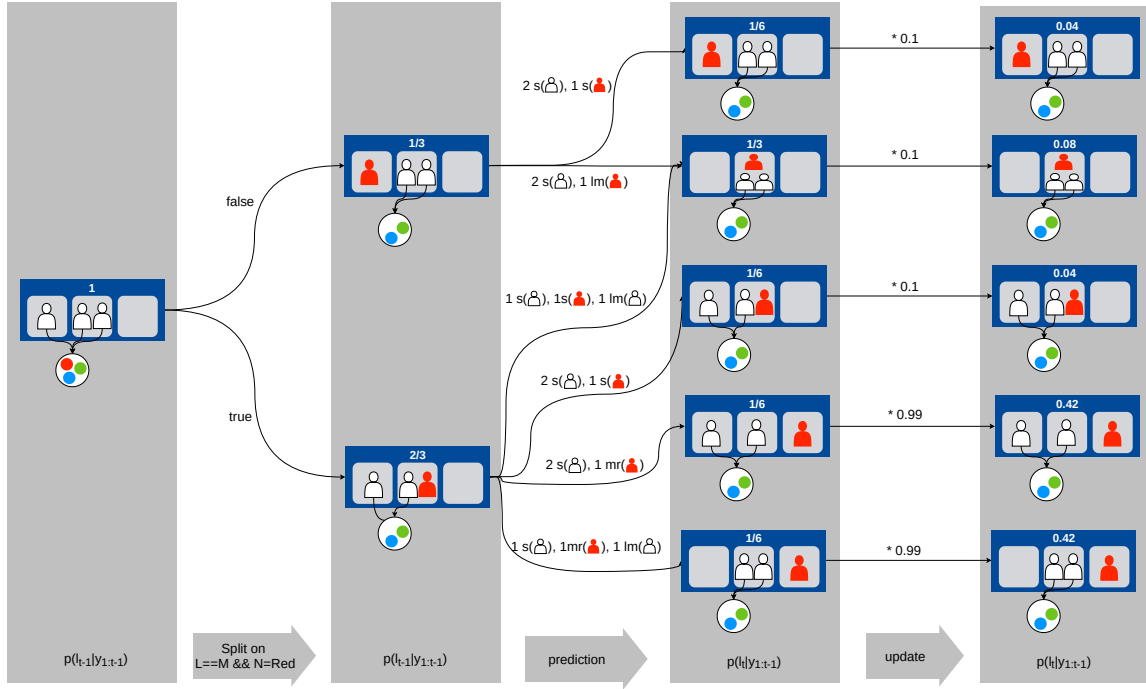


Figure 4.7.: Split-prediction-update cycle of LiMa for the scenario described in Example 31.

model and constraint-based observation model) 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. by keeping only the  $N$  most likely states at each time step (see [163], for a discussion of more elaborate pruning strategies).

**Example 31.** We consider a variant of the office scenario (Example 16), 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 be 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

$$s = \llbracket 2\langle N: \mathbf{N}, L: L \rangle, 1\langle N: \mathbf{N}, L: M \rangle \rrbracket, \quad \gamma = \langle N: \mathcal{U}(R, G, B) \rangle.$$

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  $e = \langle N: \mathbf{N}, L: M \rangle$ . 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

$$\begin{aligned} s_1 &= \llbracket 2\langle X: \mathbf{N}, Y: 1 \rangle, 1\langle X: A, Y: 2 \rangle \rrbracket, & \gamma_1 &= \langle \mathbf{N}: \mathcal{U}(B, C) \rangle, \\ s_2 &= \llbracket 1\langle X: A, Y: 1 \rangle, 1\langle X: \mathbf{N}, Y: 1 \rangle, 1\langle X: \mathbf{N}, Y: 2 \rangle \rrbracket, & \gamma_2 &= \langle \mathbf{N}: \mathcal{U}(B, C) \rangle \end{aligned}$$

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  $p(Y_t = 1 | l_t)$ . This example is illustrated in Figure 4.7.

**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 32.** 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

$$s = \llbracket 1\langle X: \mathbf{N}, Y: 1 \rangle, 1\langle X: \mathbf{N}, Y: 2 \rangle, 1\langle X: \mathbf{N}, Y: 3 \rangle \rrbracket, \quad \gamma_1 = \langle \mathbf{N}: \mathcal{U}(A, B, C) \rangle$$

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  $\rho$ , let  $|\rho|$  denote the support of  $\rho$ , 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<sup>8</sup>

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

As the support of  $\rho$  is typically exponential in the number of RVs of  $\rho$  (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.

---

<sup>8</sup>This upper bound ignores the fact that multiple value sequences from  $\rho$  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 32.

#### 4. Lifted Marginal Filtering

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.

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 4.5.2): 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. *Merging* operations that *reduce* the representation complexity – by identifying sets of lifted states that can be represented by a single lifted state – are discussed in Chapter 6.

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.

### 4.5. Experimental Evaluation

In this section, we empirically investigate whether LiMa indeed allows for more efficient inference due to the lifted state representation. Specifically, we performed lifted inference (i.e. LiMa) and ground inference (i.e. the conventional marginal filtering algorithm) for three application scenarios, to answer the following research questions:

- Q1 (Representation Size)** Does the lifted state representation lead to a significantly smaller cardinality of the exact filtering distribution (i.e. can it achieve a higher representational 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*)?

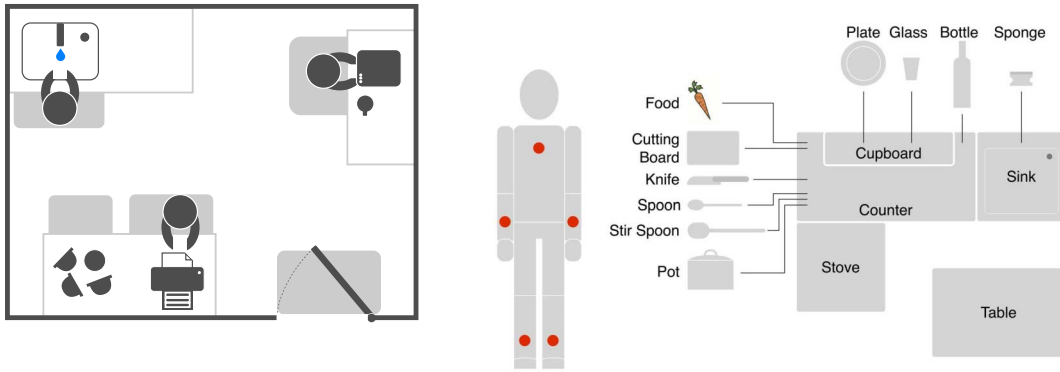
#### 4.5.1. Evaluation Scenarios

To evaluate these research questions, we modeled three application scenarios in LiMa. Table 4.1 provides an overview of the scenarios. Of the three scenarios, one (the *office* scenario) is purely simulated, allowing to evaluate the algorithm on a large number of simulated dataset.



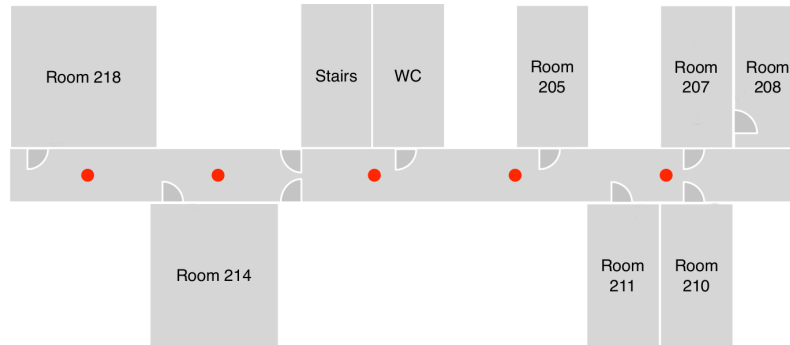
Scenario	# Actions	# Entities	# Datasets	Length	Truth	Obs.
(a) Office	15	1-6	6*120	51.5	Sim.	Sim.
(b) Tracking	40	1-7	7*5	473.4	Real	Sim.
(c) Kitchen	72	16	7	92.6	Real	Real

Table 4.1.: Evaluation scenarios. The column *Truth* describes how the ground truth was obtained (either by simulation, or by observing real human protagonists), and *Obs.* describes how the observation sequences were obtained (by simulation or from wearable sensors).



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

(c) Kitchen scenario. Reprinted from Krüger et al. [123].



(b) Tracking scenario. Dots denote locations of presence sensors. Reprinted from Lüdtkke et al. [135].

Figure 4.8.: Evaluation scenarios.

The other two scenarios consist of actual activity sequences of human protagonists and thus allow to evaluate the performance of the algorithms in realistic usage situations – that might contain a substantial amount of symmetry breaks, i.e. require splitting. In the following, each scenario is described briefly, and the intuition on the chosen modeling approach is provided.

**Office** This simulated scenario (originally presented by Schröder et al. [192], dataset available at [191]) consists of one to six persons (agents) that act in an office environment consisting of six locations (see Figure 4.8 (a)). The agents can move between locations, carry

#### 4. Lifted Marginal Filtering

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 agent is present at that location. The sensor data is always correct, i.e. there are no false positives or false negatives.

For each number of agents that act in the environment (one to six), 120 state and observation sequences have been sampled. The mean length of the observation sequences is 51.5 time steps.

The scenario has been modeled in LiMa by representing each agent and coffee capsule 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* agent (and coffee capsule) is at each location, the identities of the agents and capsules have been modeled by an urn without replacement.

**Tracking** This scenario (originally presented by Krüger et al. [122], dataset available at [100]) is also concerned with tracking the locations of agents, but no other context information, like in the previous scenario. Here, 14 locations are available (see Figure 4.8 (b)). The observation sequences are based on real, observed human motion trajectories. Trajectories for one to seven persons have been recorded, that are moving simultaneously in the environment. For each number of agents, five trajectories have been obtained, i.e. there is a total of 35 datasets.

For each recorded trajectory, sensor observations of presence sensors located at each of the five corridor locations (see Figure 4.8) have been simulated. The other rooms are not observed. 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 9 times the length of the office scenarios.

In this scenario, the only actions that agents perform is moving between locations. The probability of each action (moving between two specific rooms, or not moving) has been obtained by maximum-likelihood estimation from the observed trajectories. The scenario has been modeled by representing each agent as 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. [123], dataset available at [121]) serves as a large, real-world evaluation of LiMa. The task here is to perform activity and context recognition in a kitchen scenario, 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 action classes, e.g. *take*, *move* or *fill*. They were instrumented with 5 inertial measurement units (IMUs), recording linear acceleration and angular velocity (3 axis each) at 120 Hz (see Figure 4.8 (c) for the sensor placement). 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

Scenario	Factor	Levels	Description
Office	Agents	1–6	Number of agents (i.e. state space size)
	Dataset	1–120	Simulated dataset used
Tracking	Agents	1–6	Number of agents (i.e. state space size)
	Dataset	1–5	Dataset used
Kitchen	Run	7	

Table 4.2.: Factors and levels of experimental design for exact inference.

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 [240]. This, however, would add another layer of complexity and additional parameters, which we wanted to avoid for 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 4.8, as well as the agent, has been modeled as a separate entity with properties like location, clean/dirty, cooked, etc. For example, a sub-multiset of a reachable state in this PMPMRS is

$$\begin{aligned}
 x = \llbracket & 1\langle N: \text{Spoon}, \text{Dirty: yes}, \text{Pos: Sink} \rangle, \\
 & 1\langle N: \text{Plate}, \text{Dirty: yes}, \text{Pos: Sink} \rangle, \\
 & 1\langle N: \text{Pot}, \text{Dirty: yes}, \text{Pos: Counter} \rangle, \dots \rrbracket.
 \end{aligned}
 \tag{4.21}$$

A lifted state representation has been obtained by representing the identities of objects by an urn without replacement. The action weights have been chosen according to a goal distance heuristic, where the goal is that the meal has been finished and all objects are washed, as described by Krüger et al. [123]. The modeling approach is described in more detail in Appendix G.1.

We investigated two different observation models  $p(y_t | l_t)$ : (i) *Crisp* observations of the actual action  $c_t$ , i.e.  $p(y_t | l_t) = \mathbb{1}(a_t = y_t)$  where  $a_t$  is the action executed in state  $l_t$  and  $y_t$  is the actual (observed) action at time  $t$ ; and (ii) we used the preprocessed sensor data as observations, and assumed  $p(y_t | l_t)$  to be a multivariate normal distribution conditional on the executed action, i.e.  $p(y_t | l_t) \sim \mathcal{N}(\mu_{a_t}, \Sigma_{a_t})$ , where  $y_t$  are the preprocessed sensor data (i.e. the 21 principal components with largest eigenvalue),  $a_t$  is the action executed in  $l_t$ , and the parameters  $\mu_{a_t}$  and  $\Sigma_{a_t}$  have been estimated from the data by standard maximum-likelihood estimation.

#### 4.5.2. 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 4.11.

We did not compare LiMa to any other Bayesian filtering (BF) algorithm apart from marginal filtering, because no other filtering algorithm is directly applicable to these large

#### 4. Lifted Marginal Filtering

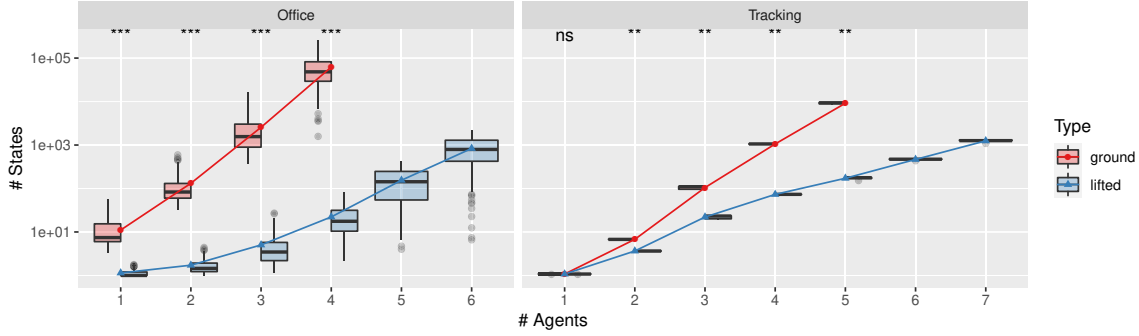


Figure 4.9.: Exact inference results for the office and tracking scenarios: Number of states required for exact filtering, using lifted and ground state representations. The line shows the mean number of states for each configuration. Note the logarithmic scale of the  $y$  axis. Ground filtering has been infeasible for 5 and 6 agents (office) or for 6 and 7 agents (tracking). Stars indicate significant differences in the required number of states between ground and lifted filtering, using Wilcoxon signed-rank test. \*:  $p < 0.05$ , \*\*:  $p < 0.01$ , \*\*\*:  $p < 0.001$ .

number of agents		1	2	3	4	5	6	7
Office	ground	9.1	113.7	2516.5	67988.7	—	—	—
	lifted	1.1	1.6	5.0	23.9	178.1	972.3	—
Tracking	ground	1.1	6.6	104.9	1056.9	9161.7	—	—
	lifted	1.1	3.7	22.0	73.2	172.0	466.3	1252.9

Table 4.3.: Exact inference results for the office and tracking scenarios (as shown in Figure 4.9): Mean number of states required for exact filtering, using lifted and ground state representations.

state spaces and structured, causal system dynamics: BF algorithms that enumerate the transition model as a matrix are infeasible due to the large state space size. Approximate algorithms like particle filtering can in principle be used, but it has already been shown that marginal filtering is superior to particle filtering in categorical state spaces [164] and thus, it is sufficient to use (ground) marginal filtering for comparison.

For both algorithms, we assessed the mean number of explicitly maintained states (ground or lifted) that are necessary to represent the distribution as a measure of inference efficiency. Specifically, for each observation sequence of all three evaluation scenarios, we computed the *mean number of states* required for ground and lifted inference. For the kitchen scenario, we used the crisp observation model (observing the actual actions), to keep ground filtering feasible. Table 4.2 summarizes the factors of the experimental design. All experiments have been performed using an implementation of LiMa in Haskell.

**Results** Figure 4.9 and Table 4.3 show the mean number of states that are necessary for exact filtering in the office and tracking scenarios, with respect to their state space size (which depends on the number of agents). It can be observed that the necessary number of states is substantially smaller when the lifted state representation is used.

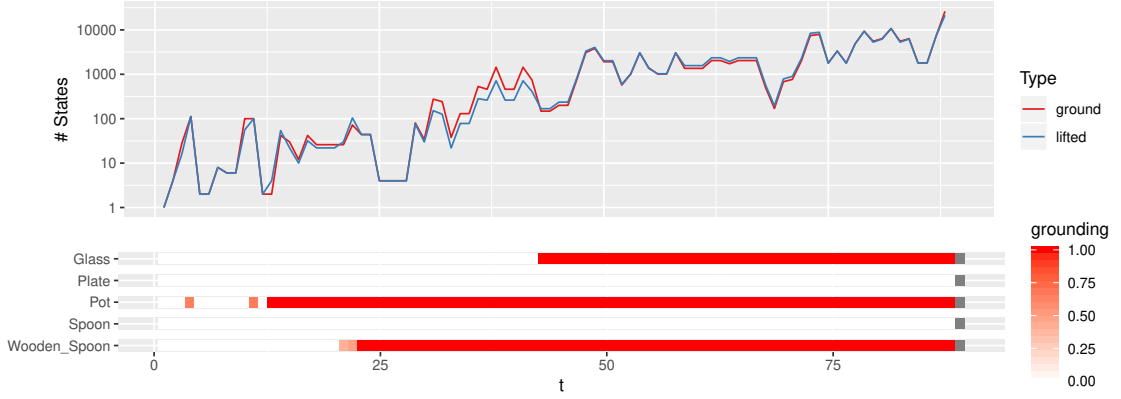


Figure 4.10.: Exact inference results for the kitchen scenario. Top: Number of states required for filtering in the kitchen scenario over time for subject 6, using lifted and ground states. Bottom: Fraction of states in which each object identity is represented explicitly. Filtering is stopped at  $t = 89$ , because it exceeds the threshold of 50,000 states.

For the office scenario, this difference in the required number of states is statistically significant for each number of agents ( $p < 10^{-16}$  using Wilcoxon signed rank test,  $n = 120$ ), and becomes more pronounced with increasing number of agents: For the Office scenario with 1 (2, 3, 4) agents, the ground representation needs 9.7 (76.6, 508, 2800) times the number of states than the lifted representation.

Similarly, for the tracking scenario, the difference in the number of required states is significant for 2 to 5 agents ( $p < 0.05$ , Wilcoxon signed rank test,  $n = 5$ ). For one agent, the ground and lifted state representations are equivalent. Again, the difference becomes more pronounced with increasing number of agents: For the tracking scenario with 2 (3, 4, 5) agents, the ground representation needs 1.8 (4.8, 14.4, 53.3) times the number of states than the lifted representation.

Figure 4.10 (top) shows the required number of states over time for the kitchen scenario and Figure 4.10 (bottom) shows the fraction of states in which each of the object identities is represented explicitly (for one of the subjects, the other subjects show a similar result). Over time, the lifted state representation becomes more and more ground, due to the fact that some of the actions require a split on some of the objects. For example, the *drink* requires that the person holds the glass (and not some other objects), and thus requires a split on the constraint  $c(e) = e.N == Glass$ . Therefore, the lifted representation needs the same number of states as the ground representation for large segments of the filtering process. Segments where the lifted representation is more efficient than the ground representation can be observed rarely, e.g. between timestep 38 and 43 for the subject shown in the figure. Overall, for the kitchen scenario, the difference in the required number of states between lifted and ground state representation is not significant.

**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 they cannot be discriminated by observations and do not need to be

#### 4. Lifted Marginal Filtering

Scenario	Factor	Levels
Tracking	Dataset	1–5
	States	10, 25, 50, 100, 200
	Repetition	1–10
Kitchen	Dataset	1–7
	States	25, 50, 100, 200, 500, 1000, 2000
	Repetition	1–10

Table 4.4.: Factors and levels of experimental design for approximate inference.

discriminated due to the system dynamic, the lifted representation can reduce the required number of states (and thus runtime) drastically.

For the kitchen scenario, the number of states required for the lifted and ground representation are not significantly different. The problem is that the actions require repeated splitting, so that the lifted representation becomes increasingly ground over time. This problem becomes specifically prevalent due to the crisp observation model employed here. To understand this, suppose that the action *drink* was observed. This action can only be performed when the agent holds the glass (and not some other object), requiring a split of that property. The crisp observation model assigns only states where this action was performed a non-zero probability. Thus, after *drink* was observed, the value *glass* is represented explicitly in *all* of the posterior states.

Furthermore, when an object identity has been split off once, it remains separate indefinitely (as shown in Figure 4.10, bottom). Intuitively, this is not always necessary. 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. Methods for retaining a lifted representation for these cases are presented in Chapter 6. As we will see later, such methods can lead to increased inference efficiency even for “asymmetrical” cases like the kitchen scenario.

#### 4.5.3. Approximate Inference

**Experimental Setup** For assessing **Q2**, we performed approximate filtering for the tracking and kitchen scenarios (for the latter scenario, the real sensor measurements were used as observations). 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 Nyolt and Kirste [163] was used here. The factors of the experimental design are shown in Table 4.4.

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 estimated number of persons per room, and for the kitchen scenario, we computed the most likely action class that was performed. The quality of the approximation was assessed on this estimate (in terms of root mean squared error or accuracy, explained below).

For the tracking scenario, we investigated 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

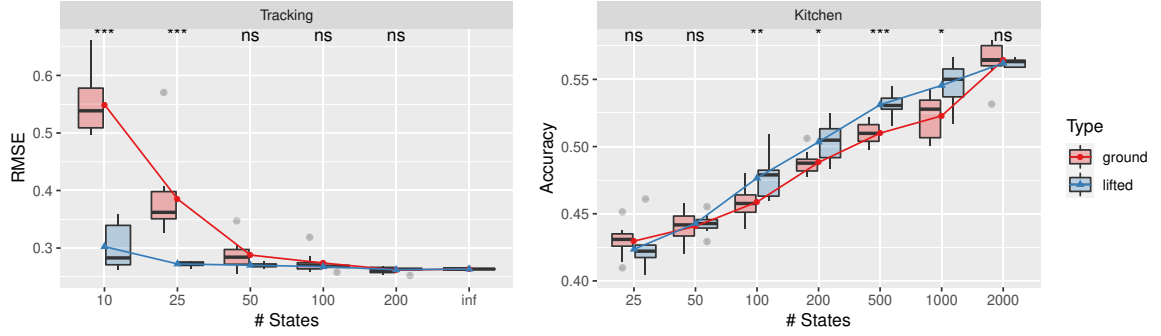


Figure 4.11.: Approximate inference results: RMSE/Accuracy of tracking/kitchen scenario with respect to available states. For the tracking scenario, “inf” denotes an unlimited number of states (exact filtering), i.e. a lower bound on RMSE.

# states	10	25	50	100	200	$\infty$ (exact)
ground	0.549	0.385	0.288	0.274	0.262	0.263
lifted	0.302	0.272	0.270	0.268	0.262	0.263

Table 4.5.: Approximate inference results (as shown in Figure 4.11): Mean RMSE of *tracking* scenario with respect to available states.

agents at room  $r$  for an approximate filtering distribution  $\hat{p}(l_t | y_{1:t})$ . The RMSE is then

$$\text{RMSE} = \sqrt{\frac{\sum_{t=1}^T \sum_{r=1}^{n_L} (n_{r,t} - \hat{n}_{r,t})^2}{T * n_L}}. \quad (4.22)$$

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

For the kitchen scenario, we assessed the *accuracy* of the estimated 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  $\hat{p}(l_t | y_{1:t})$ . The accuracy is then

$$\text{Accuracy} = \frac{\sum_{t=1}^T \mathbb{1}(a_t = \hat{a}_t)}{T}. \quad (4.23)$$

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

**Results** Figure 4.11 shows the RMSE (or accuracy) for both scenarios and different numbers of available states. In the tracking scenario, RMSE of lifted filtering is never higher

#### 4. Lifted Marginal Filtering

# states	25	50	100	200	500	1000	2000
ground	0.430	0.441	0.459	0.488	0.498	0.542	0.561
lifted	0.424	0.443	0.477	0.503	0.518	0.542	0.563

Table 4.6.: Approximate inference results (as shown in Figure 4.11): Mean accuracy of *kitchen* scenario with respect to available states.

than RMSE of ground filtering. For 10 and 25 states, RMSE of lifted filtering is significantly lower than ground filtering ( $p < 0.001$ ,  $n = 10$  using Wilcoxon signed rank test). When increasing the number of available states further, the difference in RMSE between lifted and ground filtering is no longer significant (for the 10 repetitions of the inference procedure evaluated here). Finally, for an unlimited number of available states (i.e. when performing exact filtering), the RMSEs are identical, as in the exact case, the lifted and the ground representation describe the same distribution.

The kitchen scenario shows a similar behavior: Increasing the number of states increases accuracy, and the accuracy of lifted filtering is never lower than ground filtering for a fixed number of states. Accuracy of ground filtering is significantly lower than accuracy of lifted filtering for 100, 200, 500 and 1000 states ( $p < 0.05$ ,  $n = 10$  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 2000 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. The reason 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. This effect does not manifest when the allowed number of states is very high, such that the number of states is saturated for both algorithms.

Interestingly, the lifted representation achieved a higher accuracy in the kitchen scenario, even though it did not lead to a smaller number of required states for the case of crisp observations above, where the lifted representation quickly degenerated to the ground representation. This difference can be explained by the different behavior of the crisp observation model and the probabilistic, sensor-based observation model. As discussed above, when an action that requires a split (e.g. the action *drink*, which requires a split on *glass*) is observed in the crisp observation model, that property is represented explicitly in *all* posterior states. In contrast, in the probabilistic observation model, states where an action that does not require splitting was performed can also have non-zero probability, so that a value is represented explicitly only in some of the posterior states. Overall, the probabilistic observation model thus does not have such a strong tendency to ground the model, so that the lifted state representation can be more useful, which is exactly what we observed empirically here.



### 4.5.4. Summary

The empirical evidence shows that LiMa can indeed achieve a lower representational complexity (or lower error in the approximate case) for the application scenarios 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, due to the more efficient utilization of available states.



# Approximating the System Dynamics using MCMC

**CHAPTER SUMMARY** *In this chapter, we show how to solve one of the two remaining scalability issues of Lifted Marginal Filtering: Efficiently computing the distribution  $p(K|l)$  of applicable maximal compound actions (AMCAs). This is done via a Markov Chain Monte Carlo (MCMC) algorithm that approximates the partition function of  $p(K|l)$ . The proposal works by backtracking in the search tree of the exact algorithm, and then sampling a completion of the remaining, non-maximal compound action. This approach allows to apply Lifted Marginal Filtering to systems with a large number of AMCAs, where the exact algorithm is infeasible.*

*Parts of this chapter are based on:*

*[133] Stefan Lüdtke, Max Schröder, and Thomas Kirste. Approximate Probabilistic Parallel Multiset Rewriting using MCMC. In Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz), pages 73–85. Springer, 2018.*

## Contents

---

5.1. An MCMC Algorithm for $p(K l)$ . . . . .	86
5.2. Experimental Evaluation . . . . .	89

---

## 5. Approximating the System Dynamics using MCMC

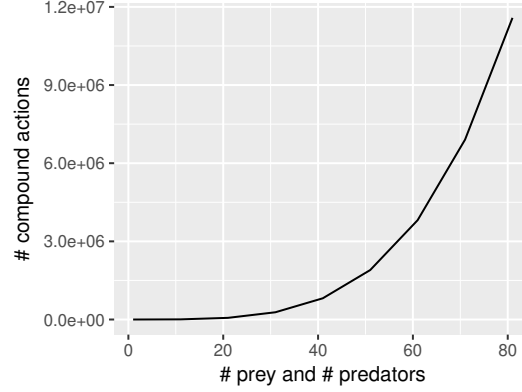


Figure 5.1.: Number of AMCAs for the predator-prey scenario, in relation to the number of predator and prey entities.

To understand the problem, consider the following example. In a simple population model, two types of entities exist: *Prey*  $x = \langle \text{Type: } X \rangle$  and predators  $y = \langle \text{Type: } Y \rangle$ . Prey can reproduce, a predator can eat a prey animal (leading to a decrease in prey population size by one and increase of predator population by one), and predators can die. We assume a *maximally parallel* semantics as introduced in Section 4.1, i.e. all animals can participate in one of the actions simultaneously. The fact that animals can also choose to *not* participate in any action is modeled by an additional *noop* action.

The number of AMCAs of a single state  $l$  in relation to the number of entities in  $l$  is shown in Figure 5.1. We see that the number of AMCAs increases dramatically, for 80 predator and 80 prey entities, there are already more than 10 million AMCAs. In general, when  $n$  is the number of entity structures in  $l$  and  $m$  is the number of action instances, there can be up to  $\binom{m+n-1}{n} = \frac{(m+n-1)!}{n!(m-1)!}$  AMCAs.

Unfortunately, when exactly computing the prediction as outlined in Section 4.2.1, we need to enumerate all AMCAs, as each AMCA can lead to a different successor state. An algorithm for enumerating AMCAs that has linear complexity in the number of AMCAs is shown in Algorithm 2. Obviously, this algorithm quickly becomes infeasible when the number of entities (and thus the number of AMCAs) increases.

However, in many cases, the number of posterior states will be much smaller than the number of AMCAs, as many AMCAs can result in the same posterior state. Therefore, even if we do not enumerate all AMCAs, but sample from  $p(K | l)$ , the chance to “miss out” highly probable posterior states is low, and we might still be able to approximate the posterior state distribution accurately. Based on this intuition, we propose a Markov chain Monte Carlo (MCMC) algorithm for approximating the distribution  $p(K | l)$ .

### 5.1. An MCMC Algorithm for $p(K | l)$

**The Metropolis-Hastings Algorithm** Markov chain Monte Carlo (MCMC) algorithms like the Metropolis-Hastings algorithm provide an efficient sampling mechanism for case where we can directly calculate a value  $v(k)$  that is proportional to the probability of  $k$ , but obtaining the normalization factor (the partition function) is difficult. The Metropolis-Hastings algorithm works by constructing a Markov chain of samples  $\mathcal{M} = k_0, k_1, \dots$  that

---

**Algorithm 8** Proposal function: Draw a sample  $k'$  from  $g(k' | k)$ .

---

```

1: function G( $k, l, n$ )
2:    $R \leftarrow \{r \mid r \subseteq k, |r| = n\}$  ▷ All sub-multisets of  $k$  of size  $n$ 
3:    $r \leftarrow \text{uniform}(R)$  ▷ Choose one of the sub-multisets uniformly
4:    $k^- \leftarrow k \setminus r$  ▷ Remaining compound action
5:    $l^- \leftarrow l \setminus \text{bound}^1(k^-)$  ▷ Remaining state of entities not bound in  $k^-$ 
6:    $K' \leftarrow \text{ENUM-CA}(l^-, AI, k^-)$  ▷ All AMCAs  $k'$  of  $l$  with  $k^- \subseteq k'$ 
7:    $k' \leftarrow \text{uniform}(K')$ 
8:   return  $k'$ 

```

---

has  $p(K)$  as its stationary distribution. The samples are produced iteratively by employing a *proposal distribution*  $g(k' | k)$  that proposes a move to the next sample  $k'$ , given the current sample  $k$ . The proposed sample is either *accepted* and used as the current sample for the next iteration, or rejected and the previous sample is kept. The *acceptance probability* is calculated as

$$A(k, k') = \min \left\{ 1, \frac{v(k') g(k | k')}{v(k) g(k' | k)} \right\}.$$

It can be shown that the Markov chain constructed this way does indeed have the target distribution  $p(K)$  (Equation 4.5) as its stationary distribution [128]. The Metropolis-Hastings algorithm thus is a random walk in the sample space (in our case, the space of AMCAs) with the property that each sample is visited with a frequency relative to its probability.

Although the Markov chain eventually converges for all proposal distributions  $g$ , the convergence speed heavily depends on the specific choice of  $g$ . If  $g$  makes only small, local changes, it might explore the sample space slowly and thus converge slowly. On the other hand, if  $g$  makes large leaps in the sample space, the acceptance probability can become low, because the next sample is more likely to be in a region of low probability, and the algorithm will again converge slowly.

**Proposal Function** In the following, we present a proposal function for compound actions. The idea is to perform *local* moves in the space of the compound actions as follows: The proposal function  $g(k' | k)$  proposes  $k'$  by randomly selecting a small number  $n$  of action instances (for example, we use  $n = 2$  in the experiments below) to delete from  $k$ . For the remaining, non-maximal compound action  $k^-$  and the remaining state  $l^-$  (consisting of the entities not bound by  $k^-$ ) it then computes the set of all possible completions of  $k^-$ , using the exact algorithm ENUM-CA (Algorithm 2). This is much easier than enumerating all AMCAs from scratch: The compound action  $k^-$  is “almost” maximal, and the exact algorithm only needs to perform the last few steps of the search. This means the proposal makes small changes to  $k$  for proposing  $k'$ , while ensuring that  $k'$  is applicable and maximal. The proposal function is shown in Algorithm 8.

**Example 33.** In a simplified population model, two types of entities exist: Prey  $x = \langle \text{Type} = X \rangle$  and predators  $y = \langle \text{Type} = Y \rangle$ . Predators can eat other animals (prey or other predators<sup>2</sup>, action  $e$ ), and all animals can reproduce (action  $r$ ). For the state  $l = \llbracket 1x, 3y \rrbracket$ ,

---

<sup>1</sup>The function  $\text{bounds}(k)$  returns the multiset of all entities bound in  $k$ .

<sup>2</sup>Predators eating other predators is uncommon for predator-prey models, but is included in this example for the sake of illustrating the algorithm.

## 5. Approximating the System Dynamics using MCMC

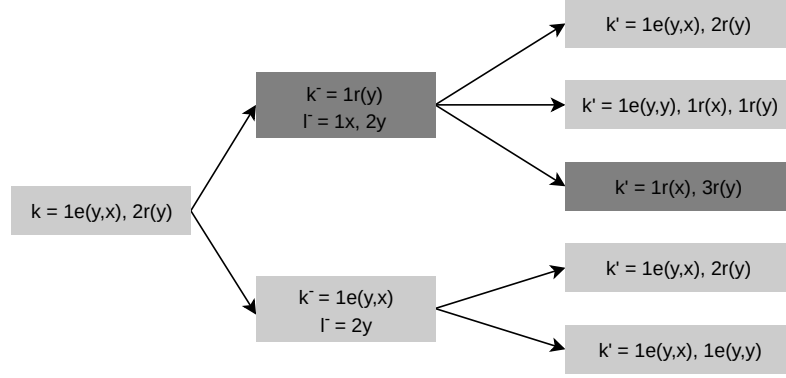


Figure 5.2.: Illustration of the proposal function, using the action instances from Example 33 and the state  $l = \llbracket 1x, 3y \rrbracket$ .

---

**Algorithm 9** Probability of the proposal  $g(k' | k)$  for given  $k'$  and  $k$ .

---

```

1: function GPROB( $k', k, l, n$ )
2:    $rem \leftarrow k \cup k'$  ▷ Action instances in  $k$ , but not in  $k'$ .
3:    $R \leftarrow \{r \mid rem \subseteq r \subseteq k, |r| = n\}$  ▷ Sub-multisets of  $k$  that could have been removed from  $k$  to get to  $k'$ .
4:   for  $r \in R$  do
5:      $k_r^- \leftarrow k \cup r$  ▷ Remaining compound action after removing  $r$ .
6:      $l_r^- \leftarrow l \cup \text{bound}(k_r^-)$  ▷ Remaining state of entities not bound in  $k_r^-$ .
7:      $n_r \leftarrow |\text{ENUM-CA}(l_r^-, AI, k_r^-)|$  ▷ Number of possible AMCAs from  $k_r^-$ .
8:    $t \leftarrow |\{r \mid r \subseteq k, |r| = n\}|$  ▷ Total number of action instance combinations that could have been removed from  $k$ .
9:    $p \leftarrow 1/t * \sum_{r \in R} 1/n_r$ 
10:  return  $p$ 

```

---

the following applicable action instances exist:  $(r, \langle y \rangle)$ ,  $(r, \langle x \rangle)$ ,  $(e, \langle y, x \rangle)$ ,  $(e, \langle y, y \rangle)$ .

Suppose the current AMCA is  $k = \llbracket 1(e, \langle y, x \rangle), 2(r, \langle y \rangle) \rrbracket$ , and in the proposal, we remove  $n = 2$  action instances. This results in two possible remaining compound actions  $k_1^- = \llbracket 1(r, \langle y \rangle) \rrbracket$  and  $k_2^- = \llbracket 1(e, \langle y, x \rangle) \rrbracket$ . The compound action  $k_1^-$  has three possible completions, and  $k_2^-$  has two possible completions, as shown in Figure 5.2.

**Probability of Proposal Step** We do not only need to sample a value from  $g$ , given  $k$  (as implemented in Algorithm 8), but for the acceptance probability, we also need to calculate the probability of  $g(k' | k)$ , given  $k'$  and  $k$ . This is implemented by Algorithm 9.

The general idea is to follow all possible choices of  $g$ , and count how many of the choices lead to  $k'$ . In  $g$ , two random choices are performed: (i) Deciding which action instances  $r$  to remove from  $k$  (line 3 in Algorithm 8), and (ii) choosing one of the compound action completions  $k'$  (line 6). Although any distribution could have been chosen (convergence is guaranteed in either case [128]), a uniform distribution was used on purpose in both cases, to simplify the computation: For a uniform distribution, it is sufficient to know the *number* of elements to compute the probability of each element (but we do not need to enumerate all elements). Specifically, for the first random choice in  $g$ , it is sufficient to know the cardinality

of  $R$  (the number of action instance combinations that could be removed from  $k$ ). For the second choice, we only need to count the number of AMCAs for those cases where  $k'$  can actually be reached. These considerations are exploited by Algorithm 9, leading to increased efficiency.

Figure 5.2 illustrates the algorithm. Suppose the dark grey path has been chosen by the proposal function. The function  $\text{GPROB}(k', k, l, 2)$  then only has to call  $\text{ENUM-CA}$  once, for  $k^- = \llbracket 1(r, \langle y \rangle) \rrbracket$  and  $l^- = \llbracket 1x, 2y \rrbracket$ , as  $k'$  can only be reached on this path. The probability is calculated as  $\text{GPROB}(k', k, l, 2) = 1/2 * 1/3 = 1/6$ .

Finally, the Markov chain is initialized with an AMCA that is identified by a depth-first-search procedure similar to  $\text{ENUM-CA}$ , except that not all branches are explored, but the search stops when the first AMCA is found.

## 5.2. Experimental Evaluation

In the following, we empirically evaluate the approximation quality and convergence of the MCMC algorithm.

**Evaluation Scenario** We used a Lotka-Volterra (predator-prey) system for evaluation. The system was modeled as a PMRMRS with two types of entities (predator and prey) and six actions (consumption, reproduction and no-op of each predator and prey). We did not provide any observation model, i.e. the update step of Bayesian filtering was not performed. In this scenario, no lifting was performed, i.e. ground states are maintained – however, the results are equally valid for lifted states, where AMCA computation works identically.

We want to emphasize that the purpose of this model was not to provide a truthful model of predator-prey dynamics, but to allow evaluation of the MCMC algorithm on an easily scalable model, that is subject to the combinatorial explosion in the number of AMCAs.

The dynamics of the system is illustrated in Figure 5.5 (left). As in other stochastic predator-prey models, each trajectory describes an oscillation of both the number of predators and prey. Fluctuations due to the stochasticity of the system lead an increased amplitude of the oscillations, until eventually, one or both of the species becomes extinct [169].

**Experimental Design** We computed the compound action distribution (using the MCMC-based algorithm, MCMC-CA and the exact algorithm,  $\text{ENUM-CA}$ ) and the posterior state distribution for a state  $x$  of the predator-prey scenario. We varied the number of predator and prey in  $x$  (2, 3, 5, 7, 15, 20, 25, 30, 40, 50, 60, 70) and the number of samples drawn by MCMC-CA (1, ..., 30000). A burn-in period of 1000 samples was used.

As usual for the evaluation of Markov chains, convergence was assessed using the *total variation distance* (TVD). Let  $p$  be the true distribution, and let  $q_n$  be the distribution of MCMC-CA after drawing  $n$  samples. The TVD is then

$$\Delta(n) = 1/2 \sum_x |p(x) - q_n(x)| \quad (5.1)$$

The *mixing time*  $\tau(\epsilon)$  measures how many samples need to be drawn until the TVD falls below a threshold  $\epsilon$ :

$$\tau(\epsilon) = \min\{t | \Delta(n) \leq \epsilon \text{ for all } n \geq t\} \quad (5.2)$$

## 5. Approximating the System Dynamics using MCMC

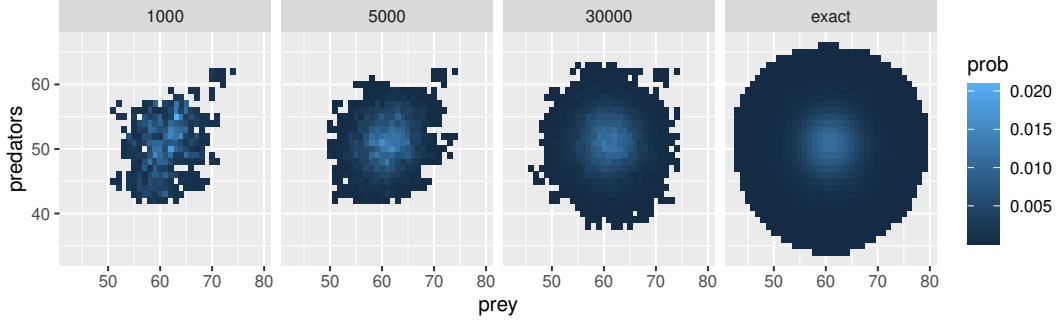


Figure 5.3.: True posterior state distribution (computed by ENUM-CA) and approximate posterior distribution (computed by MCMC-CA with 1000, 5000 or 30000 MCMC steps) of a state with 70 prey and 70 predator entities. When increasing the number of MCMC steps, the approximate posterior converges to the true posterior.

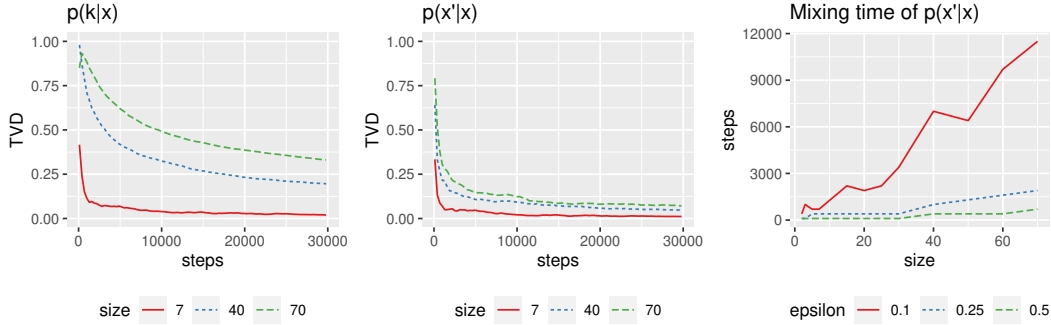


Figure 5.4.: TVD of  $p(K | x_t)$  (left) and  $p(X_{t+1} | x_t)$  (middle) for different number of samples, and states with different numbers of entities. Right: Empirical mixing time of  $p(X_{t+1} | x_t)$ , indicating that a linear increase in MCMC steps (and thus, runtime) of MCMC-CA is sufficient to achieve a given approximation quality. Note that the mixing time was estimated from a single run for each size, and is thus noisy.

We assessed the TVD and mixing time of (i) the compound action distribution, and (ii) of the posterior state distribution. The rationale here is that ultimately, only the posterior state distribution is relevant, but assessing the TVD and mixing time of the compound action distribution allows further insight into the algorithm.

**Results** Figure 5.3 shows a qualitative comparison of the posterior state distribution  $p(X_{t+1} | x_t)$  computed by both algorithms and different numbers of MCMC steps, for a state  $x_t$  with 70 predator and 70 prey entities. When increasing the number of MCMC steps, the approximate distribution converges to the true distribution. Note that the compound action distribution for this situation (not shown) has a much larger cardinality (see Figure 5.1).

A quantitative comparison of both algorithms in terms of TVD is shown in Figure 5.4. When more samples are drawn by the approximate algorithm, the TVD converges to zero, as expected (implying that the approximate algorithm works correctly). Naturally, the TVD converges slower for states with more entities (due to the much larger number of compound



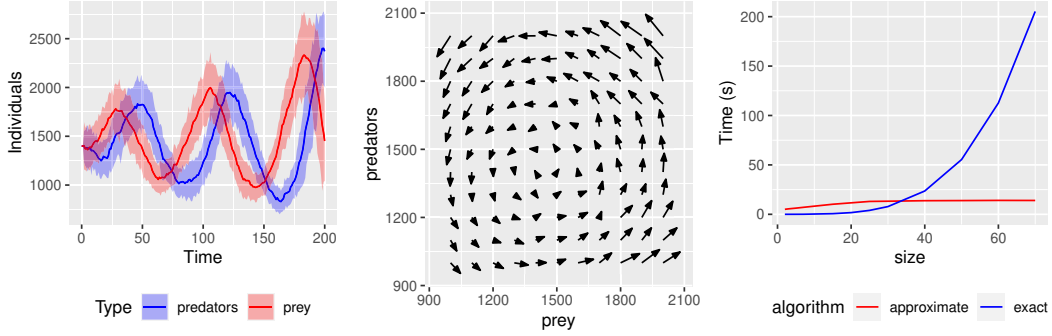


Figure 5.5.: Left: Filtering distribution for a large predator-prey model, obtained by MCMC-CA with 10,000 samples and pruning to  $n = 20$  after each prediction (the line shows the expectation and the ribbons show the standard deviation of the distribution). Middle: Expected posterior states for the predator-prey model (also using MCMC-CA with 10,000 samples). Each arrow points from an initial state to its expected successor state. Right: Runtime of the algorithms, using constant number of 10,000 samples for MCMC-CA.

actions). Furthermore, the posterior state distribution  $p(X_{t+1} | x_t)$  can be approximated more accurately than  $p(K | x_t)$ : For a state with 70 predator and prey entities, the approximate transition model is reasonably accurate (successor state TVD  $< 0.1$ ) after drawing 10,000 samples.

The runtime of both algorithms (using a fixed number of 10,000 samples for MCMC-CA) for different numbers of entities is presented in Figure 5.5 (right), showing the factorial runtime of ENUM-CA in comparison to the constant runtime of MCMC-CA (when using a constant number of samples).

Naturally, to achieve a given approximation quality, MCMC-CA requires more samples when the number of entities is increased. This relationship is represented by the empirical mixing time of  $p(X_{t+1} | x_t)$ , shown in Figure 5.4 (right). The mixing time grows approximately linear in the number of entities in the state. This suggests that to achieve the same accuracy of the approximation, the runtime of the approximate algorithm only has to grow linearly – as compared to the exact algorithm, which has a factorial runtime.

Finally, Figure 5.5 (left) visualizes the filtering distribution of the predator-prey scenario with a large number of entities, using MCMC-CA with 10,000 MCMC steps. For this large-scale model, exact compound action computation is infeasible, so a quantitative comparison with the true distribution is not possible. Still, the distribution shows the expected oscillating behavior that was also observed in smaller models, indicating that the approximate distribution is sufficiently similar to the true distribution to generate the same quantitative behavior. The corresponding transition model is visualized in Figure 5.5 (middle), by showing the expected posterior states for different prior states.

**Discussion** The results show that approximating the compound action distribution by MCMC-CA is a viable alternative to the exact algorithm. The results for the empirical mixing time suggest that to achieve the same approximation quality, the runtime of MCMC-CA only has to grow linearly – as compared to the exact algorithm, which has a factorial runtime. Thus, using MCMC-CA, it is possible to accurately calculate the posterior state distribution

## 5. *Approximating the System Dynamics using MCMC*

for situations with a large number of entities, even when the exact algorithm is infeasible.

Of course, these findings only apply to the predator-prey scenario discussed here. As Figure 5.3 shows, the posterior state distribution is unimodal, which is a specifically simple case for MCMC-based algorithms. More complex scenarios, involving multimodal posteriors, can be more challenging for MCMC-CA, although the algorithm is still guaranteed to converge eventually [128]. In Section 6.4, a much more complex scenario is shown where applying the MCMC-based algorithm still leads to accurate results.

# Lifted Marginal Filtering in Asymmetrical Models

**CHAPTER SUMMARY** *In this chapter, we approach the second scalability issue of LiMa: The fact that the system dynamics or observations can lead to symmetry breaks, so that repeated splitting must be performed until the representation degenerates to the ground form. This is done by identifying subsets of lifted states that afford to be represented by a single lifted state. Finding such subsets is difficult in the general case, but we can still show merging procedures for some relevant special cases: Multinomial distributions, multivariate hypergeometric distributions, and normal distributions. These methods allow to substantially reduce the representational complexity, and thus, LiMa can be more efficient than ground inference even for models where the symmetry breaks easily.*

*Parts of this chapter are based on:*

[129] Stefan Lüdtke, Marcel Gehrke, Tanya Braun, Ralf Möller, and Thomas Kirste. Lifted Marginal Filtering for Asymmetric Models by Clustering-based Merging. *In Proceedings of the 24th European Conference on Artificial Intelligence*. IOS Press, 2020.

[131] Stefan Lüdtke, Alejandro Molina, Kristian Kersting, and Thomas Kirste. Gaussian Lifted Marginal Filtering. *In Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, pages 230–243. Springer, 2019.

[136] Stefan Lüdtke, Kristina Yordanova, and Thomas Kirste. Human Activity and Context Recognition using Lifted Marginal Filtering. *IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE, 2019.

## Contents

---

<b>6.1. Problem Statement</b>	<b>95</b>
<b>6.2. Merging Similar States</b>	<b>97</b>
6.2.1. Divergence Measures for Lifted States	98
6.2.2. Computing Merged States	100
6.2.3. Handling Different Distribution Types	101
6.2.4. Experimental Evaluation	104
<b>6.3. Merging Disjoint States</b>	<b>107</b>
6.3.1. $M$ and $l^*$ known: Testing for Mergeability	109
6.3.2. $l^*$ Known, $M$ Unknown: Identifying Mergeable Subsets	111
6.3.3. $M$ and $l^*$ Unknown: A Greedy Search Algorithm	113
6.3.4. Experimental Evaluation	116
<b>6.4. Merging Normal Distributions</b>	<b>119</b>
6.4.1. Merging Entities by Gaussian Mixture Reduction	120
6.4.2. Experimental Evaluation	121
<b>6.5. Assumed Density Merging</b>	<b>124</b>
6.5.1. Algorithm Overview	125
6.5.2. Time Points for Merging	126
6.5.3. Exploiting Temporal Structure	127
6.5.4. Experimental Evaluation	127
<b>6.6. Conclusion &amp; Future Work</b>	<b>130</b>

---

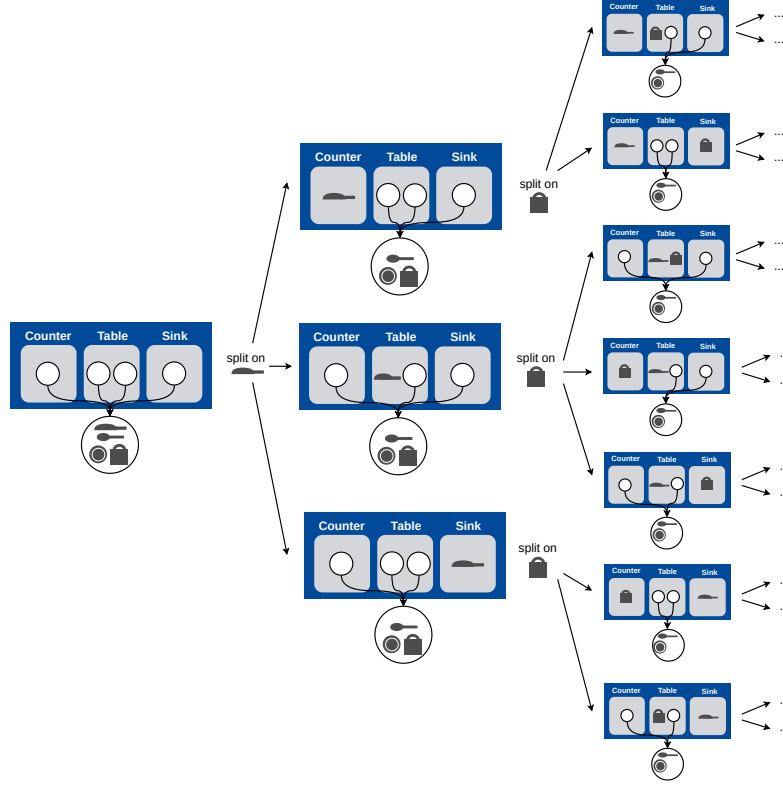


Figure 6.1.: Example of the grounding problem in the kitchen scenario: Repeatedly applying splitting operations leads to a complete grounding over time, so that the efficiency that should be provided by the lifted representation vanishes.

## 6.1. Problem Statement

We start by showing an example of the *grounding problem*, where the representational complexity of the distribution grows over time.

**Example 34.** Consider the kitchen scenario discussed in Section 4.5.1. One of the precondition of the action “put the pot on the stove” is that the person holds the pot in their hand (and not some other object). Thus, when the other preconditions of that action are satisfied, the action requires a split on the constraint  $e.N == \text{Pot}$ . Similarly, the *drink* action requires a split on  $e.N == \text{Glass}$ , filling the cooked dish on the plate with the spoon requires a split on  $e.N == \text{Spoon}$ , and so on. Thus, over time, the lifted representation degenerates to the ground representation.

The same behavior can arise due to observations: For example, we can reasonably assume that the wearable sensor data  $y_t$  depends on the type of object that is handled by the subject at time  $t$  (as different objects might be associated with distinct movements). Thus, the observation likelihood  $p(y_t | l_t)$  requires a split of  $l_t$  to determine which specific object is handled in  $l_t$ .

Figure 6.1 illustrates the problem: Each time a split is performed, the number of states increases, until finally, the states are completely ground and thus, inference in the lifted

## 6. Lifted Marginal Filtering in Asymmetrical Models

representation becomes as inefficient as ground inference. This behavior could be seen empirically in Section 4.5, where in the Kitchen scenario, the number of required lifted states coincided with the number of required ground states. Such asymmetries have been identified as the major challenge for lifted inference methods in general, and methods to handle them efficiently is regarded as vitally important for the usefulness of lifted inference [104].

Therefore, in this section, we are concerned with the following question: How can we (approximately) retain a lifted representation, i.e. how can we find a representation of the distribution with fewer lifted states, that is approximately (or even exactly) identical to the original distribution?

More precisely, given a set of lifted states  $L$  and a categorical distribution  $p(l)$  over the states in  $L$ , find a new set  $L'$  of lifted states with distribution  $p(l')$ , such that  $|L'| < |L|$  and

$$\sum_{l \in L} p(l) p(X | l) \simeq \sum_{l' \in L'} p(l') p(X | l'). \quad (6.1)$$

We call this process *merging*. Depending on the merging strategy, the equality holds either exactly or approximately.

**Related Work** In the context of lifted probabilistic inference, methods for retaining a compact representation have been devised [233, 203]. They work by identifying (approximate) symmetries (graph automorphisms) in the graphical model, e.g. by using color passing algorithms [108], or low-rank matrix approximations [225].

For Bayesian filtering, only few merging methods have been devised. For the relational Kalman filter, a method has been proposed that regroups Gaussian potentials that have been split by averaging their covariance matrices [39]. For the LDJT algorithm, a merging approach has been proposed that restores a lifted representation by identifying similar factors by density-based clustering and cosine distance function (making use of the intuition that factors that are scaled differently can still be similar) [72].

These methods work on a (relational) graphical model representation of the distribution, e.g. a parfactor graph. As discussed in Section 3.3, systems with MRS dynamics cannot easily be modeled as a graphical model. Specifically, graphical models cannot directly handle changing numbers of entities, due to the resulting latent infinite state space. Additionally, existing merging algorithms are concerned with merging *parfactors* of the joint distribution. In LiMa, however, the distribution is a *mixture*, i.e. a sum of the component distributions described by each lifted state (see Equation 4.11). The goal here is not to merge *factors*, but *mixture components* (states). Thus, the assumptions made by existing methods do not hold for LiMa, and therefore they cannot be used in a straightforward way here. Still, the high-level strategy of the merging algorithms presented here (identifying similar states by clustering, and then replacing states by cluster centers) is motivated by existing merging algorithms.

Instead, the merging methods discussed here are more closely related to mixture simplification. Most approaches in this area rely on the fact that mixture components are continuous distributions, e.g. Gaussian mixtures. For example, Runnalls [181] proposes a method that iteratively merges pairs of Gaussian components into a single component with minimal Kullback-Leibler divergence to the original components. Other approaches work by fitting a smooth distributions that best approximate a set of mixture components [243], or by using variational methods to obtain a reduced mixture [239]. As we are concerned with a



Figure 6.2.: Intuition of the different approaches for merging. Each rectangle denotes a lifted state, so that the area of the rectangle corresponds to the region of that lifted state. Left: The lifted states have approximately identical region, so that they can be approximated by a single lifted state. Right: The lifted states have disjoint regions, but complement each other in a way that the overall distribution can be represented by a single lifted state.

discrete distribution here (the distribution over ground states), these methods can also not be applied directly.

**Outline** In this chapter, we present a number of different merging approaches. We start with two complementary approaches that both work by identifying subsets  $G \subseteq L$  that can be represented by a single lifted state  $l_G$ . The intuition on both approaches is shown in Figure 6.2. The first approach, MERGE-SIMILAR (Section 6.2), identifies sets  $G$  where all  $l \in G$  represent (approximately) the same distribution over ground states  $p(X | l)$ , such that they can be approximated by a single lifted state  $l_G$ . The other approach, MERGE-DISJOINT (Section 6.3), identifies sets  $G$  where the regions of all states are pairwise disjoint, but they complement each other in such a way that they can be represented by a single lifted state.

Afterwards, we consider the merging problem from a different angle: The idea is to look at each individual state  $l = (s, \gamma)$ , and reduce the number of *different* entity structures in  $s$ . By reducing the number of entity structures, the number of action instances, thus the number of AMCAs, and thus the number of posterior states is reduced. Furthermore, this way, multiple lifted states can be mapped to the *same* lifted state, such that they can be merged trivially. Specifically, we discuss two cases where merging of entities is possible and beneficial: (i) When each entity encodes a conditional normal distribution (such that Gaussian mixture model reduction methods can be employed), and (ii) when the temporal structure of the actions allows to discard information about individuals.

## 6.2. Merging Similar States

First, we consider the case of lifted states  $l$  that can be merged because they describe approximately the same distribution over ground states  $p(X | l)$ .

Intuitively, we can think of this procedure as analogous to Gaussian mixture reduction [181], which can be done by identifying pairs of mixture components that describe “sufficiently similar” distributions and thus can be approximated by a single component. Here, the lifted state take the role of the mixture components, and the goal is to identify groups of states which describe approximately the same distribution over ground states, and thus can be replaced by a single state. We propose to identify these groups by clustering, so that the overall merging algorithm consists of the following steps:

- (i) Compute pairwise distances of the distributions  $p(X | l)$  for all  $l \in L$ ,
- (ii) Identify groups  $G$  of lifted states that afford a merge (by clustering), and

## 6. Lifted Marginal Filtering in Asymmetrical Models

(iii) Compute a single representative state  $l_G$  for each group  $G$ .

In the following, we assume that all factors in  $\gamma$  are *multinomial* distributions, (the distribution that describes drawing multiple values from an urn with replacement), because two properties of multinomial distributions will be required later: A mixture of multinomial distributions is again a multinomial distributions, and multiple consecutive draws from an urn with replacement are independent.

The remainder of this section is structured as follows. We start by showing how the distances between distributions  $p(X | l)$ , i.e. step (i) above, can be computed efficiently, without complete grounding (Section 6.2.1), and how the representative  $l_G$  for a group  $G$  of lifted states, i.e. step (iii), can be computed (Section 6.2.2). Afterwards, in Section 6.2.3, we generalize this method to cases where the states in  $G$  do not agree in their distribution types: This way, the methods cannot only *recover* a previously existing symmetry structure (that is encoded in the distribution types), but can *discover* new symmetry structures that arise from the system dynamics. Finally, empirical results are presented in Section 6.2.4, that show that this merging procedure can lead to a substantially smaller error than *pruning* (as typically done in the marginal filter) to the same number of states.

### 6.2.1. Divergence Measures for Lifted States

We start by discussing how the “distance” (more precisely, the *divergence*) between distributions  $p(X | l_1)$  and  $p(X | l_2)$  can be computed efficiently.

**Kullback-Leibler divergence** The natural choice for a divergence between distributions is the Kullback-Leibler divergence (KLD)

$$D_{\text{KLD}}(p, q) = - \sum_x p(x) \log_2 \frac{q(x)}{p(x)}. \quad (6.2)$$

Naively, we can directly compute the ground distributions  $p(X | l_1)$  and  $p(X | l_2)$  (via Equation 4.10 or by repeated splitting), and then compute the KLD of these distributions, as illustrated by the following example.

**Example 35.** Consider the states  $l_1 = (s_1, \gamma_1)$  and  $l_2 = (s_2, \gamma_2)$  with

$$\begin{aligned} s_1 &= \llbracket 1 \langle \mathbf{N}: \mathbf{N}_1 \rangle, 1 \langle \mathbf{N}: \mathbf{N}_2 \rangle \rrbracket, & \gamma_1 &= \langle \mathbf{N}_1: \mathcal{M}(0.8\textcolor{red}{A}, 0.2\textcolor{green}{B}), \mathbf{N}_2: \mathcal{M}(0.1\textcolor{red}{A}, 0.9\textcolor{green}{B}) \rangle \\ s_2 &= \llbracket 1 \langle \mathbf{N}: \mathbf{N}_1 \rangle, 1 \langle \mathbf{N}: \mathbf{N}_2 \rangle \rrbracket, & \gamma_2 &= \langle \mathbf{N}_1: \mathcal{M}(0.7\textcolor{red}{A}, 0.3\textcolor{green}{B}), \mathbf{N}_2: \mathcal{M}(0.5\textcolor{red}{A}, 0.5\textcolor{green}{B}) \rangle \end{aligned}$$

The states describe the following distribution over ground states (which can be seen by repeated splitting, or by Equation 4.10):

$x$	$p(x   l_1)$	$p(x   l_2)$
$\llbracket 1 \langle \mathbf{N}_{\mathbf{N}_1}: \textcolor{red}{A} \rangle, 1 \langle \mathbf{N}_{\mathbf{N}_2}: \textcolor{red}{A} \rangle \rrbracket$	$0.8 * 0.1$	$0.7 * 0.5$
$\llbracket 1 \langle \mathbf{N}_{\mathbf{N}_1}: \textcolor{red}{A} \rangle, 1 \langle \mathbf{N}_{\mathbf{N}_2}: \textcolor{green}{B} \rangle \rrbracket$	$0.8 * 0.9$	$0.7 * 0.5$
$\llbracket 1 \langle \mathbf{N}_{\mathbf{N}_1}: \textcolor{green}{B} \rangle, 1 \langle \mathbf{N}_{\mathbf{N}_2}: \textcolor{red}{A} \rangle \rrbracket$	$0.2 * 0.1$	$0.3 * 0.5$
$\llbracket 1 \langle \mathbf{N}_{\mathbf{N}_1}: \textcolor{green}{B} \rangle, 1 \langle \mathbf{N}_{\mathbf{N}_2}: \textcolor{green}{B} \rangle \rrbracket$	$0.2 * 0.9$	$0.3 * 0.5$

Using Equation 6.2, the KLD between  $p(x | l_1)$  and  $p(x | l_2)$  is  $D_{\text{KLD}} = 0.568$ .



Computing the divergence directly in this way is highly inefficient, as it requires to ground both states completely.

Fortunately, the KLD is additive for distributions that factorize into independent factors. Specifically, when  $p$  factorizes into  $p(x_1, x_2) = p_1(x_1) p_2(x_2)$  and  $q$  factorizes into  $q(x_1, x_2) = q_1(x_1) q_2(x_2)$ , the KLD between  $p$  and  $q$  is

$$D_{\text{KLD}}(p, q) = D_{\text{KLD}}(p_1, q_1) + D_{\text{KLD}}(p_2, q_2). \quad (6.3)$$

Furthermore, recall from Equations 4.9 and 4.10 that the distribution  $p(x | l)$  is a product over the representations in  $\gamma$  (times an indicator function for the structure):

$$p(x | s, \gamma) = \mathbf{1}(s_x = x) \prod_{\langle d: \rho \rangle \in \gamma} p_\rho(\mathbf{v}^{(d)} | s), \quad (6.4)$$

where  $\phi(x) = (s_x, \mathbf{v})$ .

This fact can be used to decompose the KLD between lifted states. First, if  $l_1$  and  $l_2$  have different structures  $s_1$  and  $s_2$ , the regions (the sets of groundings) of  $l_1$  and  $l_2$  are disjoint, and thus, the KLD is infinite<sup>1</sup>. When the structures are identical, the additivity of the KLD for factorized distributions can be exploited, so that the KLD between states can be computed as follows:

$$D_{\text{KLD}}(l_1, l_2) = \begin{cases} \sum_{d \in \text{dom}(\gamma_1)} D_{\text{KLD}}(p_{\gamma_1(d)}, p_{\gamma_2(d)}) & \text{if } s_1 = s_2 \\ \infty & \text{otherwise} \end{cases} \quad (6.5)$$

Note that this expression does not require complete grounding, but is linear in the number of distribution representations in  $l_1$  and  $l_2$  (assuming that the KLD between factors can be computed in constant time).

**Jensen-Shannon divergence** Ideally, the divergence between states should be symmetric (i.e.  $D(l_1, l_2) = D(l_2, l_1)$ ), because there is no preference for either  $l_1$  or  $l_2$  – none of them is the “true” distribution. A symmetrical divergence derived from the KLD is the Jensen-Shannon divergence (JSD)

$$D_{\text{JSD}}(p, q) = \frac{1}{2} D_{\text{KLD}}(p, m) + \frac{1}{2} D_{\text{KLD}}(q, m), \quad (6.6)$$

where  $m = \frac{1}{2}(p + q)$ . Unfortunately, additivity for independent factors does not hold for the JSD – in general, even when  $p$  and  $q$  factorize into independent components, this does not need to be the case for  $m$ . Here, we simply assume additivity of the JSD<sup>2</sup>. This will induce an error, but is empirically still a useful divergence measure (as shown in the experimental evaluation in Section 6.2.4). Under this assumption, the JSD of lifted states  $l_1$  and  $l_2$  is

$$D_{\text{JSD}}(l_1, l_2) = \begin{cases} \sum_{d \in \text{dom}(\gamma_1)} D_{\text{JSD}}(p_{\gamma_1(d)}, p_{\gamma_2(d)}) & \text{if } s_1 = s_2 \\ 1 & \text{otherwise} \end{cases} \quad (6.7)$$

Again, the divergence is maximal when  $s_1 \neq s_2$ , because  $l_1$  and  $l_2$  do not share any groundings in this case.

<sup>1</sup>Later, in Section 6.2.3, we will relax this condition, allowing merging of states that do not have identical structure.

<sup>2</sup>Another option would be to define  $m = \frac{1}{4}(p_1 + q_1)(p_2 + q_2)$  [241].

## 6. Lifted Marginal Filtering in Asymmetrical Models

**Example 36.** Consider the states  $l_1$  and  $l_2$  shown in Example 35. The true JSD of the ground distributions, according to Equation 6.6, is  $D_{\text{JSD}}(p(X | l_1), p(X | l_2)) = 0.1547$ . Under the assumption that the JSD is additive for independent factors – using Equation 6.7 – the JSD becomes  $D_{\text{JSD}}(l_1, l_2) = D_{\text{JSD}}(p_{\gamma_1(\mathbf{N}_1)}, p_{\gamma_2(\mathbf{N}_1)}) + D_{\text{JSD}}(p_{\gamma_1(\mathbf{N}_2)}, p_{\gamma_2(\mathbf{N}_2)}) = 0.1565$ .

### 6.2.2. Computing Merged States

In the previous section, we showed how divergences between lifted states can be computed efficiently. The next goal is to identify “sufficiently similar” states regarding this divergence, by using a clustering algorithm. Specifically, we propose to use DBSCAN [63], a density-based clustering algorithm. In principle, any clustering algorithm could be used for this step. We choose a density-based clustering approach here, as this way, we do not need to specify the number of clusters in advance and do not need to make a-priori assumptions about the shape of the clusters.

For each cluster  $G$  of lifted states, a single representative  $l_G$  needs to be computed, as described next. The underlying intuition is that each state  $l \in G$  is a mixture component of the ground distribution  $p(x | G)$ , and the goal is to simplify the mixture, by reducing it to a single component  $l_G$ . More specifically, according to Equations 4.9 and 4.10, the distribution  $p(x | G)$  is given by

$$p(x | G) = \sum_{l=(s,\gamma) \in G} \frac{p(l)}{p(G)} \mathbb{1}(s_x = s_G) \prod_{\langle d:\rho \rangle \in \gamma} p_\rho(\mathbf{v}^{(d)} | s), \quad (6.8)$$

where  $\phi(x) = (s_x, \mathbf{v})$  and  $p(G) = \sum_{l \in G} p(l)$ .

To see how  $l_G$  can be computed, assume that the states  $l \in G$  are *exactly* identical (i.e. (i) they have the same structure  $s_G$ , and (ii) distribution representations in  $\gamma$  are identical). This allows us to rewrite the expression for  $p(x | G)$  above so that we can read off an expression for  $l_G$  directly, which we will then also use for the case where the state  $l \in G$  are not exactly identical.

Due to assumption (i), the factor  $\mathbb{1}(s_x = s_G)$  can be moved in front of the sum in Equation 6.8. Furthermore, due to assumption (ii), the factors  $p_\rho(\mathbf{v}^{(d)} | s)$  do not depend on the structure  $s$  and have representations drawn from  $\gamma$ , but only on the mutual structure  $s_G$ , with representations drawn from a mutual context  $\gamma_G$ . Thus, the RVs  $\mathbf{v}^{(d)}$  are all conditionally independent (given  $s_G$ ), i.e. the sum and the products can be switched:

$$p(x | G) = \mathbb{1}(s_x = s_G) \prod_{\langle d:\rho \rangle \in \gamma_G} \sum_{l=(s,\gamma) \in G} \frac{p(l)}{p(G)} p_\rho(\mathbf{v}^{(d)} | s_G) \quad (6.9)$$

When the factors in  $\gamma$  are slightly different (instead of exactly identical), this independence holds only approximately.

The interesting observation here is that the expression in Equation 6.9 exactly describes the distribution induced by a single lifted state (see Equation 6.4), where each factor has the form

$$\sum_{l=(s,\gamma) \in G} \frac{p(l)}{p(G)} p_\rho(\mathbf{v}^{(d)} | s_G). \quad (6.10)$$

This means the distribution  $p(x | G)$  can be represented by a single lifted state,  $l_G = (s_G, \gamma_G)$ , where  $s_G$  is the structure of any of the states in  $G$  and the context contains representations of the factors shown in Equation 6.10.

The remaining question is how the mixture for each factor of  $l_G$  shown in Equation 6.10 can be represented by a single parametric representation. In general, this will require approximations, e.g. when attempting to represent a mixture of normal distributions by a single normal. For the multinomial distributions considered here, however, this is possible exactly, because a mixture of multinomials is again a multinomial. The overall procedure is illustrated in the following example.

**Example 37.** Consider the states  $l_1 = (s_1, \gamma_1)$  and  $l_2 = (s_2, \gamma_2)$  from Example 35 with

$$\begin{aligned} s_1 &= \llbracket 1 \langle \mathbf{N}: \mathbf{N}_1 \rangle, 1 \langle \mathbf{N}: \mathbf{N}_2 \rangle \rrbracket, & \gamma_1 &= \langle \mathbf{N}_1: \mathcal{M}(0.8A, 0.2B), \mathbf{N}_2: \mathcal{M}(0.1A, 0.9B) \rangle \\ s_2 &= \llbracket 1 \langle \mathbf{N}: \mathbf{N}_1 \rangle, 1 \langle \mathbf{N}: \mathbf{N}_2 \rangle \rrbracket, & \gamma_2 &= \langle \mathbf{N}_1: \mathcal{M}(0.7A, 0.3B), \mathbf{N}_2: \mathcal{M}(0.5A, 0.5B) \rangle \end{aligned}$$

and  $p(l_1) = 0.1$ ,  $p(l_2) = 0.3$ . Merging these states as outlined above leads to the new state  $l_G = (s_G, \gamma_G)$  with

$$s_G = \llbracket 1 \langle \mathbf{N}: \mathbf{N}_1 \rangle, 1 \langle \mathbf{N}: \mathbf{N}_2 \rangle \rrbracket, \quad \gamma_G = \langle \mathbf{N}_1: \mathcal{M}(0.725A, 0.275B), \mathbf{N}_2: \mathcal{M}(0.4A, 0.6B) \rangle,$$

For example, the factor  $\gamma_G(\mathbf{N}_1)$  is a mixture of the factors  $\gamma_1(\mathbf{N}_1)$  and  $\gamma_2(\mathbf{N}_1)$  with mixture weights  $0.1/0.4 = 0.25$  and  $0.3/0.4 = 0.75$ . The probability of  $l_G$  is  $p(l_G) = 0.1 + 0.3 = 0.4$ .

To summarize, computing a representative for a group  $G$  is simple, when the structures of the states in  $G$  are identical: For each distribution type, we simply approximate the mixture of the corresponding factors by a single factor. This amounts to making an additional independence assumption: Instead of just assuming that the factors in  $\gamma$  are independent given the state  $l$ , we assume that the factors of all states  $l \in G$  are independent (given  $G$ ).

### 6.2.3. Handling Different Distribution Types

Now, we draw our attention to the case where the structures of the states in  $G$  are *not* identical. Specifically, we consider the case where the structures are only identical *up to renaming of distribution types*, as illustrated by the following example.

**Example 38.** Consider the states  $l_1 = (s_1, \gamma_1)$  and  $l_2 = (s_2, \gamma_2)$  with

$$\begin{aligned} s_1 &= \llbracket 1 \langle \mathbf{N}: \mathbf{N}_1 \rangle, 1 \langle \mathbf{N}: \mathbf{N}_2 \rangle \rrbracket, & \gamma_1 &= \langle \mathbf{N}_1: \mathcal{M}(0.8A, 0.2B), \mathbf{N}_2: \mathcal{M}(0.1A, 0.9B) \rangle \\ s_2 &= \llbracket 1 \langle \mathbf{N}: \mathbf{N}_X \rangle, 1 \langle \mathbf{N}: \mathbf{N}_Y \rangle \rrbracket, & \gamma_2 &= \langle \mathbf{N}_X: \mathcal{M}(0.7A, 0.3B), \mathbf{N}_Y: \mathcal{M}(0.5A, 0.5B) \rangle \end{aligned}$$

The structures  $s_1$  and  $s_2$  are not identical. Therefore,  $l_1$  and  $l_2$  do not share any groundings,  $D_{\text{JSD}}(l_1, l_2) = 1$ , and the states cannot be merged by the procedure outlined above.

The underlying intuition is that the distribution types carry information about the *original factor structure* – when two factors in different states have the same type, they originated from a split. Thus, by merging states with identical structures, we *recover* symmetries that were previously present: States that originated from the same state are merged back together.

In this section, we go beyond this viewpoint: Even when factors have different types, but are highly similar (i.e. have a low divergence), it might make sense to merge the corresponding states. This way, we do not only recover symmetries, but *discover* symmetries that were not originally present in the distribution, but that arose due to the system dynamics.

## 6. Lifted Marginal Filtering in Asymmetrical Models

**Example 39.** Consider the states  $l_1$  and  $l_2$  from Example 38. When renaming the distribution types in  $l_2$  ( $\mathbf{N}_X$  as  $\mathbf{N}_1$  and  $\mathbf{N}_Y$  as  $\mathbf{N}_2$ ), the structures of  $l_1$  and  $l_2$  become identical, and the states can be merged by the method outlined above.

More generally, given two states  $l_1$  and  $l_2$  that have identical structures up to renaming of distribution types, the divergence of  $l_1$  and  $l_2$  can be defined via a *matching* that associates the types from  $l_1$  with the types from  $l_2$ .

The following concepts are described easiest when we assume that the representations in  $\gamma$  all represent *univariate* distributions, as in Example 38 – this way, it is guaranteed that factors that are later compared or merged always have the same arity. For the multinomial distributions considered here, this is not an actual restriction, as multiple consecutive draws from an urn with replacement (as described by a multinomial distribution) are independent anyways, so that multiple draws from a multinomial can be rewritten as single draws from multiple multinomials. For example, the state  $l_3 = (s_3, \gamma_3)$  with

$$s_3 = \llbracket 2 \langle \mathbf{N}: \mathbf{N}_1 \rangle, 1 \langle \mathbf{N}: \mathbf{N}_2 \rangle, \rrbracket, \quad \gamma_3 = \langle \mathbf{N}_1: \mathcal{M}(0.8A, 0.2B), \mathbf{N}_2: \mathcal{M}(0.1A, 0.9B) \rangle,$$

can be rewritten as  $l'_3 = (s'_3, \gamma'_3)$  where

$$\begin{aligned} s'_3 &= \llbracket 1 \langle \mathbf{N}: \mathbf{N}_1 \rangle, 1 \langle \mathbf{N}: \mathbf{N}'_1 \rangle, 1 \langle \mathbf{N}: \mathbf{N}_2 \rangle \rrbracket \\ \gamma'_3 &= \langle \mathbf{N}_1: \mathcal{M}(0.8A, 0.2B), \mathbf{N}'_1: \mathcal{M}(0.8A, 0.2B), \mathbf{N}_3: \mathcal{M}(0.1A, 0.9B) \rangle. \end{aligned}$$

Therefore, in the following, we assume that all factors in  $\gamma$  are univariate. Next, we can introduce *matchings* of states  $l_1$  and  $l_2$ , that associate each distribution type of  $l_1$  with exactly one distribution type of  $l_2$ , and vice versa.

**Definition 20.** [matching] Let  $T_1$  and  $T_2$  be two sets with  $|T_1| = |T_2|$ . We call a set of pairs  $m \subseteq T_1 \times T_2$  a *matching* of  $T_1$  and  $T_2$ , when

- all elements from  $T_1$  are used in  $m$ , i.e.  $\bigcup_{(t_1, t_2) \in m} t_1 = T_1$ ,
- all elements from  $T_2$  are used in  $m$ , i.e.  $\bigcup_{(t_1, t_2) \in m} t_2 = T_2$ , and
- no element from  $T_1$  or  $T_2$  is used more than once, i.e.  $|m| = |T_1| = |T_2|$ .

The set of all matchings of  $T_1$  and  $T_2$  is denoted as  $\mathcal{M}(T_1, T_2)$ .

For example, the distribution types of the states  $l_1$  and  $l_2$  shown in Example 38 have two possible matchings:

$$\begin{aligned} m_1 &= \{(\mathbf{N}_1, \mathbf{N}_X), (\mathbf{N}_2, \mathbf{N}_Y)\} \\ m_2 &= \{(\mathbf{N}_1, \mathbf{N}_Y), (\mathbf{N}_X, \mathbf{N}_Z)\} \end{aligned}$$

It is straightforward to generalize the JSD between states, given a matching of their distribution types:

$$D_{\text{JSD}}(l_1, l_2 | m) = \sum_{(d_1, d_2) \in m} D_{\text{JSD}}(p_{\gamma_1(d_1)}, p_{\gamma_2(d_2)}) \quad (6.11)$$

For example, the JSD between the states  $l_1$  and  $l_2$  shown in Example 38 for both possible matchings are  $D_{\text{JSD}}(l_1, l_2 | m_1) = 0.156$  and  $D_{\text{JSD}}(l_1, l_2 | m_2) = 0.369$ .

A priori, it is not clear which matching  $m$  of  $l_1$  and  $l_2$  should be used for defining the divergence of  $l_1$  and  $l_2$ . The intuition here is that the *optimal* matching of the factors

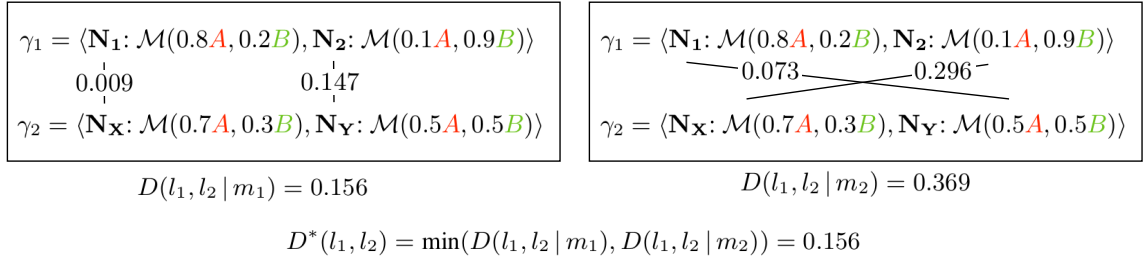


Figure 6.3.: Example of the Procrustes divergence. Edges represent the matching of distribution types; the marginal distance of the corresponding factors is shown next to the edges. In this example, two matchings  $m_1$  and  $m_2$  are possible, leading to different divergence values. The Procrustes divergence is the minimum of those values.

that has minimal divergence should be used. In statistical shape analysis, this intuition is formalized by the *Procrustes distance* [61]. It measures the distance between two sets of points and is defined as the *minimal* distance between the sets of points, for all possible Euclidean transformations (reflections, rotations and translations) of one of the sets. A closely related concept is used in the multi-target tracking literature when comparing true and estimated targets, known as optimal sub-pattern assignment distance [193]. For computing the divergence between two lifted states, we can employ the same idea, by using the *minimal* distance over all possible matchings:

$$D^*(l_1, l_2) = \min_{m \in \mathcal{M}(l_1, l_2)} D_{\text{JSD}}(l_1, l_2 | m) \quad (6.12)$$

We call the distance  $D^*$  the *Procrustes divergence* of  $l_1$  and  $l_2$  (see Figure 6.3 for an example).

Note that when there are  $n$  distribution types in  $l_1$  and  $l_2$ , there are  $n!$  possible matchings  $m$ . Still, an optimal matching can be obtained in  $\mathcal{O}(n^3)$  using the Hungarian algorithm [167]. Furthermore, the divergence  $D^*$  can be approximated in *constant* time by drawing a constant number of *samples* from the set of all matchings  $\mathcal{M}(l_1, l_2)$  and taking the minimal divergence of all samples.

Similarly, the computation of the merged state (see Equation 6.9 above) can also be generalized in this sense. Suppose that for a set  $G$  of states, matchings  $m_{l_*}^{l_i}$  between all states  $l_i \in G$  and an arbitrary reference state  $l_* \in G$  are available. Each factor  $p(\mathbf{v}^{(d)} | s_G)$  of  $l_G$  can then be computed as

$$p(\mathbf{v}^{(d)} | s_G) = \sum_{i=1}^n \frac{p(l_i)}{p(G)} p_{\gamma_i(d_i)}(\mathbf{v}^{(d_i)} | s_i), \quad (6.13)$$

where  $d_i = m_{l_*}^{l_i}(d)$  is the distribution type in  $l_i$  that is matched with the type  $d$  of  $l_*$ .

For the matchings  $m_{l_*}^{l_i}$ , we use the *optimal* matchings (having smallest Procrustes divergence), that have already been computed when calculating the Procrustes distance before. The overall merging procedure is outlined in Algorithm 10.

In summary, in this section, we showed a method for merging states that describe almost the same distribution over ground states. The algorithm does not only recover the original symmetry that was present initially (and that is still encoded in the distribution types), but can discover new symmetries that arise during inference. In the following, we evaluate this approach empirically.

## 6. Lifted Marginal Filtering in Asymmetrical Models

---

### Algorithm 10 Merge similar states.

---

```

1: function MERGE-SIMILAR(  $P = \langle l_i : p_i \rangle_{i=1}^N, \epsilon$  )
2:    $M \leftarrow \emptyset$  ▷ Set for collecting matchings
3:   for each pair  $l_i, l_j \in P$  do
4:      $D_{ij}, m_{l_i}^{l_j} \leftarrow D^*(l_i, l_j)$  ▷ Procrustes divergence and optimal matching, Eq. 6.12
5:      $M \leftarrow M \cup \{m_{l_i}^{l_j}\}$ 
6:    $\{G_k\}_{k=1}^K \leftarrow \text{DBSCAN}(D, \epsilon)$  ▷ Density-based clustering [63]
7:   for each  $G_k$  do
8:      $l_{G_k} \leftarrow \text{MERGE-GROUP}(G_k, P, M)$ 
9:      $p_{G_k} \leftarrow \sum_{l \in G_k} P(l)$ 
10:  return  $\langle l_{G_k} : p_{G_k} \rangle_{k=1}^K$ 
11: function MERGE-GROUP( $G = \{l_i\}_{i=1}^K, P = \langle l_i : p_i \rangle_{i=1}^N, M$ )
12:   $l^* \leftarrow$  arbitrary element from  $G$  ▷ State used as reference for matchings
13:   $s_G \leftarrow s^*$ 
14:   $\gamma_G \leftarrow \langle \rangle$ 
15:   $p_G \leftarrow \sum_{l \in G} P(l)$ 
16:  for each  $d \in \text{dom}(\gamma^*)$  do ▷ For each distribution type
17:     $\forall l_i \in G : d_i \leftarrow m_{l_i^*}^{l_i}(d)$  ▷ Collect distribution types to merge
18:     $\rho \leftarrow$  representation of  $\sum_{l_i=(s_i, \gamma_i) \in G} \frac{P(l_i)}{p_G} p_{\gamma_i}(d_i)$  ▷ New factor, Equation 6.10
19:     $\gamma_G \leftarrow \gamma_G \odot \langle d : \rho \rangle$ 
20:  return  $(s_G, \gamma_G)$ 

```

---

### 6.2.4. Experimental Evaluation

**Experimental Design** The goal of the experimental evaluation was to (i) investigate the error (in terms of JSD) that is induced by merging, in relation to the decrease in representational complexity (i.e. number of maintained lifted states), and (ii) the runtime of the merging algorithm.

Specifically, three different merging algorithms that differ only in the employed divergence measure were compared: The (exact) JSD of the ground distribution (GROUND), the Procrustes divergence using the optimal matching (PROCRUSTES-ALL), and the Procrustes divergence using only a single, random matching (PROCRUSTES-SINGLE). Additionally, merging was compared to *pruning* as an alternative method to limit the number of state representatives.

We used a simple multi-agent activity recognition scenario for evaluation, where  $n \in \{1, \dots, 5\}$  entities move in 4 rooms (the number of ground states in this scenario is exponential in the number of entities). Entities can either stay at their current location (action *noop*) or move to the room to the left (action *left*), when they are not already at the leftmost room. We set the lifted state  $l = (s, \gamma)$  with  $s = \llbracket n \langle \mathbf{L} : \mathbf{L}_1 \rangle \rrbracket$  and  $\gamma = \langle \mathbf{L} : \mathcal{M}(0.4A, 0.3B, 0.2C, 0.1D) \rangle$  as initial state, and performed  $t = 10$  prediction operations. The number of states after 10 prediction operations was always greater than 1 (for example, 56 states for  $n = 3$  entities). In this scenario, no observations were used, i.e. no update step was performed.

The factors of the experimental design are shown in Table 6.1. For the states resulting from

Factor	Levels	Description
Entities	2, 3, 4, 5	Entities in the state
Algorithm	GROUND, MARGINAL-ALL, MARGINAL-SINGLE, PRUNING	Merging algorithm
$\epsilon$	0.1, 0.25, 0.5, 0.75, 1, 1.5, 2	Parameter of DBSCAN. For PRUNING, the resulting number of posterior states is used
Iteration	1, ..., 10	

Table 6.1.: Factors and levels of experimental design for evaluating the MERGE-SIMILAR algorithm.

$n = 2, 3, 4$  or 5 entities, the different merging (and pruning) algorithms (with different values of  $\epsilon$ , which affects the numbers of maintained states) were applied, and the JSD between the true and approximated distribution and runtime were assessed. For each factor configuration, 10 repetitions were performed. Afterwards, we evaluated the behavior of repeated merging over time. This was done by performing  $t = 20$  prediction operations and applying merging (or pruning) at each time step.

Note that the merging procedure does not require to set the number of posterior states a priori, whereas the pruning algorithm needs to be provided with that number. Thus, to allow a fair comparison between merging and pruning, we proceeded as follows: For a given state distribution  $p(X_t | y_{1:t})$  and  $\epsilon$  value of DBSCAN, merging was performed, resulting in a number  $n$  of posterior states. Then, pruning was performed, where the number of posterior states was set to  $n$ .

**Results** Figure 6.4 (right) shows the runtime of the different merging algorithms. The computationally most expensive operation in all cases is computing all pairwise state divergences. We see that using the GROUND and PROCRUSTES-ALL divergences quickly become infeasible – the runtime of GROUND is exponential in the number of entities, and the runtime of PROCRUSTES-ALL is cubic in the number of entities. The runtime of PROCRUSTES-SINGLE, on the other hand, is constant.

Next, we investigate the size of the error that is induced by merging in comparison to the size of the error induced by pruning. Figure 6.4 (left) shows the ground JSD between a merged (or pruned) distribution and the original (true) distribution, in relation to the number of allowed posterior states, for a situation with 56 prior states (resulting from 10 prediction steps from an initial state with 3 entities). In general, allowing fewer posterior states naturally leads to a larger error, and allowing 56 states (i.e. the number of states to represent the posterior exactly) leads to a JSD of 0. For a given number of allowed posterior states, merging achieves a lower JSD than pruning (note the logarithmic scale of the plot). Also note that the pruning algorithm is non-deterministic (as the retained states are sampled), whereas the merging algorithm is deterministic (given the same lifted states and  $\epsilon$ , the merge result is fixed).

We can also see that the JSD of the true and the merged distribution does not depend on the chosen divergence measures (PROCRUSTES-ALL or PROCRUSTES-SINGLE). Thus, it is sufficient to compute the pairwise state divergences via PROCRUSTES-SINGLE (which is

## 6. Lifted Marginal Filtering in Asymmetrical Models

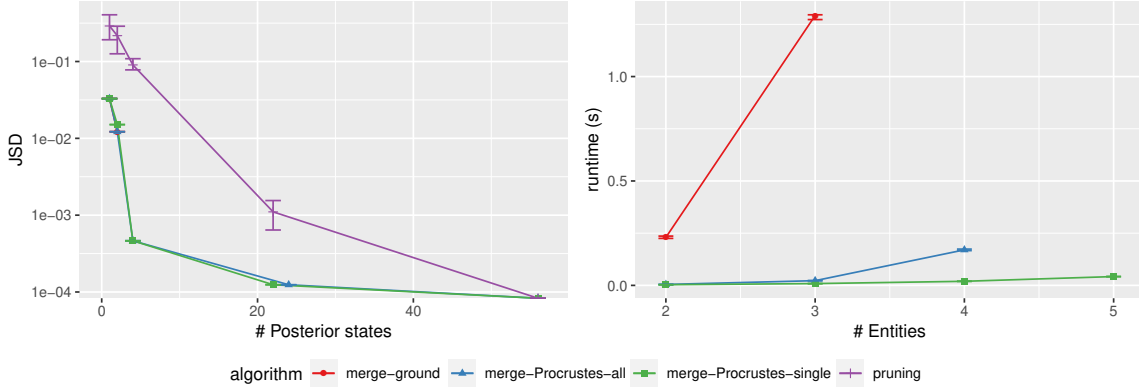


Figure 6.4.: Evaluation of the MERGE-SIMILAR algorithm. Left: Tradeoff between error (in terms of JSD) and representational complexity of the distribution (i.e. the number of lifted states) for a distribution of 56 states (with  $n = 3$  entities). All merging algorithms provide the same approximation quality, and pruning leads to a higher error for a given representational complexity. Right: Overall runtime of the different merging algorithms with respect to number of entities in the state (which has an exponential effect on the number of states with nonzero probability).

fastest), and still achieve a better approximation quality than by pruning.

Figure 6.5 shows the development of the JSD between the original and the approximated distribution over time when either performing pruning (right) or merging based on the PROCRUSTES-SINGLE divergence. The overall error does not grow indefinitely, which is a direct consequence of the contraction theorem of Boyen and Koller [25], which states that propagating two distributions  $p$  and  $q$  through a transition model leads to a constant-factor decrease in their KLD. Thus, the error between the true distribution  $p$  and the approximated distribution  $q$  cannot grow indefinitely.

The plot shows that even when merging or pruning are applied repeatedly during filtering, merging achieves a lower JSD than pruning for a given representational complexity.

**Discussion & Summary** In this experiment, limiting the number of states by merging lead to a lower approximation error than limiting the number of states by pruning. Intuitively, this is not surprising: Merging is a smooth approximation of the distribution (by averaging similar states), whereas pruning is a hard approximation, that works by setting some of the probabilities to zero. Thus, pruning can actually be usefully applied and can be performed in a computationally feasible way.

Here, we only provided a merging procedure for the case of multinomial distributions, because a mixture of multinomials can be trivially collapsed into a single multinomial. For other distributions (e.g. normal distributions), this is not as straightforward, and approximations would need to be performed.

Furthermore, we only discussed the case of merging states that describe (approximately) identical ground distributions. However, the only assumption that was necessary to rewrite the distribution as shown in Equation 6.9 (and thus to allow merging) is that the factors are conditionally independent, given the group  $G$ . Here, this independence holds because



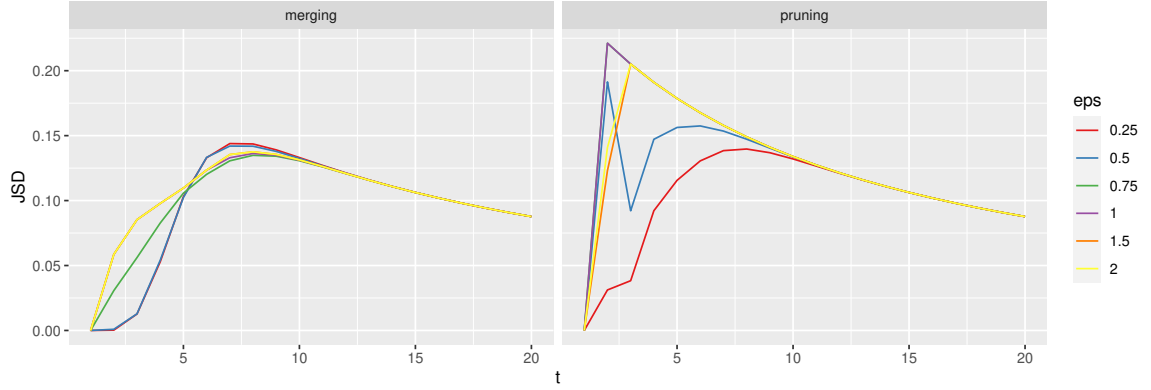


Figure 6.5.: Evaluation of the MERGE-SIMILAR algorithm: Development of JSD between original distribution and approximated distribution over time, when repeatedly performing approximation operations (merging or pruning) at each time step during inference. Merging achieves a lower error than pruning (in terms of JSD).

the distributions  $p(x|l)$  for each state  $l$  are (approximately) identical, and for each state, the factors are conditionally independent (given the state) by assumption. However, such independence could also arise due to other cases. For example, the states can be “complementary” to each other, such that they can overall be represented by a single lifted state. In the following section, we present an algorithm for exactly this case.

### 6.3. Merging Disjoint States

In the previous section, we considered states that could be merged because they describe (approximately) the same ground distribution. Now, we consider an orthogonal case: A group of states that all have pairwise disjoint regions, but can be merged because they complement each other in such a way that they can be described by a single lifted state (see Figure 6.2 for the intuition). Such situations arise naturally during filtering, when a state is split, but the split asymmetry vanishes over time due to the system dynamics.

Here, we only consider the special case where all distribution representations  $\rho \in \gamma$  are hypergeometric distributions with unique values. This special case is of high practical relevance, as it is underlying all of the scenarios shown in Section 4.5.

**Example 40.** Consider the three lifted states  $l_1 = (s_1, \gamma_1)$ ,  $l_2 = (s_2, \gamma_2)$  and  $l_3 = (s_3, \gamma_3)$  with

$$\begin{aligned} s_1 &= \llbracket 1\langle \mathbf{N}: A, L: 1 \rangle, 1\langle \mathbf{N}: \mathbf{N}, L: 2 \rangle, 1\langle \mathbf{N}: \mathbf{N}, L: 3 \rangle \rrbracket, & \gamma_1 &= \langle \mathbf{N}: \mathcal{U}(B, C) \rangle \\ s_2 &= \llbracket 1\langle \mathbf{N}: B, L: 1 \rangle, 1\langle \mathbf{N}: \mathbf{N}, L: 2 \rangle, 1\langle \mathbf{N}: \mathbf{N}, L: 3 \rangle \rrbracket, & \gamma_2 &= \langle \mathbf{N}: \mathcal{U}(A, C) \rangle \\ s_3 &= \llbracket 1\langle \mathbf{N}: C, L: 1 \rangle, 1\langle \mathbf{N}: \mathbf{N}, L: 2 \rangle, 1\langle \mathbf{N}: \mathbf{N}, L: 3 \rangle \rrbracket, & \gamma_3 &= \langle \mathbf{N}: \mathcal{U}(A, B) \rangle \end{aligned}$$

and  $p(l_1) = p(l_2) = p(l_3) = 1/3$ , shown in Figure 6.6. The same distribution over ground states is also described by the single lifted state  $l^* = (s^*, \gamma^*)$  with

$$s^* = \llbracket 1\langle \mathbf{N}: \mathbf{N}, L: 1 \rangle, 1\langle \mathbf{N}: \mathbf{N}, L: 2 \rangle, 1\langle \mathbf{N}: \mathbf{N}, L: 3 \rangle \rrbracket, \quad \gamma^* = \langle \mathbf{N}: \mathcal{U}(A, B, C) \rangle$$

and  $p(l^*) = 1$ . Thus, the uniform distribution of states  $l_1$ ,  $l_2$  and  $l_3$  can be replaced by the distribution  $p(l^*) = 1$ .

## 6. Lifted Marginal Filtering in Asymmetrical Models

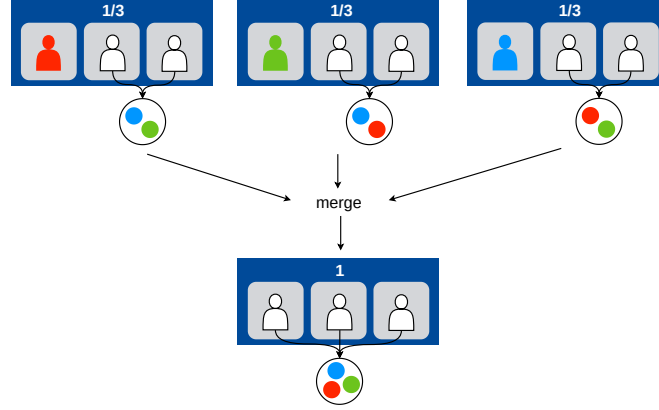


Figure 6.6.: Three states that can be merged exactly into a single lifted state, as discussed in Example 40.

**A Note on Distribution Types** Note that in the example above, the state  $l^*$  only describes the same distribution as  $l_1$ ,  $l_2$  and  $l_3$  when ignoring the distribution types of ground states. When instead taking the distribution types into account,  $l^*$  does not share any groundings with  $l_1$ ,  $l_2$  or  $l_3$ . For example, the ground state

$$x_1 = \llbracket 1\langle N: A_A, L: 1_1 \rangle, 1\langle N: B_N, L: 2_2 \rangle, 1\langle N: C_N, L: 3_3 \rangle \rrbracket$$

of  $l_1$  and the ground state

$$x^* = \llbracket 1\langle N: A_N, L: 1_1 \rangle, 1\langle N: B_N, L: 2_2 \rangle, 1\langle N: C_N, L: 3_3 \rangle \rrbracket$$

of  $l^*$  are different, because they have different distribution types. From this viewpoint, merging  $l_1$ ,  $l_2$  and  $l_3$  into  $l^*$  is not sensible, as this changes the ground distribution (more specifically, the distribution types of ground states) completely.

Still, we would like to be able to merge  $l_1$ ,  $l_2$  and  $l_3$ : If splitting  $l^*$  resulted in a uniform distribution over  $l_1$ ,  $l_2$  and  $l_3$ , merging (conceptually the inverse operation to splitting) should be able to recover  $l^*$  from the uniform distribution.

The solution here is to use the same strategy as in Section 6.2.3, by defining equality of ground states so that the distribution types are ignored: Ground states are considered equal when they are identical under an arbitrary renaming of distribution types. This convention allows to replace a uniform distribution over  $l_1$ ,  $l_2$  and  $l_3$  by  $l^*$  without affecting the ground distribution.

**Structure of this Section** We consider three different cases of increasing generality.

In Section 6.2, we start with the simple case where we have a set  $M$  of states and a state  $l^*$  are given, and the goal is to test whether the current distribution over the states in  $M$  is exactly mergeable into  $l^*$ . This case serves as the basis for the following more general cases.

In Section 6.3.2, we consider the case where the set of states  $M$  that we want to merge is not given a priori, but needs to be identified from a larger set  $P$ , i.e. only a subset of states  $M \subseteq P$  can be merged. We show an algorithm that iteratively builds a *merge candidate set*  $M$ , until  $M$  can be merged exactly into  $l^*$  (or until all states in  $P$  have been examined and  $M$  cannot be merged).

Finally, in Section 6.3.3, we generalize the merging algorithm further, by considering the case where the merge result  $l^*$  is also not given in advance. The final outcome of this section is a greedy merge algorithm which identifies subsets  $M$  and merge results  $l^*$  into which  $M$  can be merged. The algorithm is exact (in the sense that the merge results describe exactly the same ground distribution as the states before merging), but is not guaranteed to find the smallest possible representation. Using this algorithm, we are not limited to recovering known symmetries (i.e. known states  $l^*$ ), but can *discover* new symmetries that allow for a more compact representation.

### 6.3.1. $M$ and $l^*$ known: Testing for Mergeability

**Problem Statement** We start with a simple case: Given a map  $M$  that maps states  $l$  to their probability  $p(l)$  and a state  $l^*$  with probability  $p^*$ , decide whether  $M$  can be merged exactly into  $l^*$ , i.e. whether the distribution described by the states in  $M$  is identical to the distribution described by  $l^*$ . More formally, the task is to decide whether for all  $x$ ,

$$p^* p(x | l^*) = \sum_{l \in \text{dom}(M)} p(l) p(x | l). \quad (6.14)$$

If this is the case, all states in  $M$  can be replaced by  $l^*$ . A naive algorithm to test this property would test the equality individually for all states  $x$ , which is clearly infeasible.

In this section, we show how this property can be tested on the structural level (i.e. without iterating over all ground states). To do so, we need to make two restrictions on the states in  $M$  and the state  $l^*$ :

- (i) For all  $l \in \text{dom}(M) \cup l^*$ , we assume that the context of  $l$  only contains representations of hypergeometric distributions with unique values<sup>3</sup>. Although this assumption seems to be a strong restriction, it is satisfied for many scenarios of practical relevance, e.g. all of the scenarios presented in Section 4.5. This implies that the distribution over ground states described by  $l$  is *uniform*, so that  $p(x | l) = |\text{region}(l)|^{-1}$ .
- (ii) Furthermore, we only consider the case where all states  $l \in \text{dom}(M)$  have pairwise disjoint regions. This requirement can simply be satisfied by the algorithm described in Section 4.4.3.

**Simplifying the Mergeability Conditions** Under these assumptions, testing whether Equation 6.14 holds can be done more efficiently, based on a simple observation: Suppose we have  $n$  sets  $S_1, \dots, S_n$  and a set  $S$ . When the sets  $S_1, \dots, S_n$  are pairwise disjoint, they are all subsets of  $S$  (i.e.  $\forall S_i : S_i \subseteq S$ ), and have the same overall number of elements as  $S$  (i.e.  $\sum |S_i| = |S|$ ), then the sets  $S_i$  are partitions of  $S$  and thus  $\biguplus_i S_i = S$ .

This fact can be used to efficiently test for mergeability, as formalized in the following theorem.

**Theorem 1.** Let  $M = \langle l_i : p_i \rangle_{i=1}^n$  be a map of lifted states to probabilities, and let  $l^*$  be a lifted state with probability  $p^*$ . Let the regions of all  $l \in \text{dom}(M)$  be pairwise disjoint, and for all  $l \in \text{dom}(M)$ , let  $p(x | l)$  – as well as  $p(x | l^*)$  – be uniform. If

<sup>3</sup>Note that this also includes singleton distributions, i.e. constants, that are a special case of hypergeometric distributions where  $n = N = 1$ .

## 6. Lifted Marginal Filtering in Asymmetrical Models

- (a)  $\forall l \in \text{dom}(M) : \text{region}(l) \subseteq \text{region}(l^*)$ ,
- (b)  $\sum_{l \in \text{dom}(M)} |\text{region}(l)| = |\text{region}(l^*)|$ , and
- (c)  $\forall l \in \text{dom}(M) : p^* |\text{region}(l^*)|^{-1} = M(l) |\text{region}(l)|^{-1}$ ,

then

$$p^* p(x | l^*) = \sum_{l \in \text{dom}(M)} M(l) p(x | l). \quad (6.15)$$

*Proof.* Select an arbitrary  $x \in \text{region}(l^*)$ . There can be at most one  $l \in \text{dom}(M)$  with  $x \in \text{region}(l)$ , as all  $l \in \text{dom}(M)$  have pairwise disjoint regions. From the disjointness of regions of all  $l \in \text{dom}(M)$ , as well as conditions (a) and (b), it follows that  $\text{region}(l^*) = \biguplus_{l \in \text{dom}(M)} \text{region}(l)$ . Thus, there must be at least one  $l \in \text{dom}(M)$  with  $x \in \text{region}(l)$ . Therefore, there is exactly one  $l \in \text{dom}(M)$  with  $x \in \text{region}(l)$ .

Let  $\hat{l} \in \text{dom}(M)$  be the state with  $x \in \text{region}(\hat{l})$ . Thus, to show that Equation 6.15 is satisfied, it is sufficient to show that

$$p^* p(x | l^*) = M(\hat{l}) p(x | \hat{l}).$$

As  $p(x | \hat{l})$  and  $p(x | l^*)$  are uniform,  $p(x | \hat{l}) = |\text{region}(\hat{l})|^{-1}$  and  $p(x | l^*) = |\text{region}(l^*)|^{-1}$ . Thus, using condition (c),

$$p^* p(x | l^*) = p^* |\text{region}(l^*)|^{-1} = M(\hat{l}) |\text{region}(\hat{l})|^{-1} = M(\hat{l}) p(x | \hat{l}). \quad \square$$

**Example 41.** Consider the three states  $l_1$ ,  $l_2$  and  $l_3$  (that each have a probability of  $1/3$ ) and the state  $l^*$  with probability 1 from Example 40. One can easily check that the conditions (a), (b) and (c) are satisfied for  $M = \langle l_1 : 1/3, l_2 : 1/3, l_3 : 1/3 \rangle$  and  $l^*$ : The state  $l^*$  represents six ground states that each have a probability of  $1/6$ , and each of the lifted states  $l_1$ ,  $l_2$  and  $l_3$  represents exactly two of those ground states. Thus, the states  $l_1$ ,  $l_2$  and  $l_3$  can be merged exactly into  $l^*$  (as already shown above in Example 40).

Of course, we are interested in a method to check conditions (a), (b) and (c) without examining all ground states, which requires efficient algorithms for the following two operations: Computing the cardinality of the region of a lifted state, and testing for subset relations among regions.

The latter task can be solved by adapting the algorithm IS-DISJOINT described in Appendix F: The algorithm tests whether  $\text{region}(l_1) \cap \text{region}(l_2) \neq \emptyset$ , by constructing a *matching*  $m$  of  $l_1$  and  $l_2$ , such that  $\forall (e_1, e_2) \in m : \text{region}_{l_1}(e_1) \cap \text{region}_{l_2}(e_2) \neq \emptyset$ . By adapting this condition to  $\forall (e_1, e_2) \in m : \text{region}_{l_1}(e_1) \subseteq \text{region}_{l_2}(e_2)$ , the algorithm tests whether  $\text{region}(l_1) \subseteq \text{region}(l_2)$ .

**Cardinality of a Region** The remaining task for which we need to provide an efficient algorithm is computing the cardinality of the region of a given lifted state  $l = (s, \gamma)$ . Here, we again make use of the assumption that the context  $\gamma$  only contains hypergeometric distributions where all values are unique. In other words, for each distribution type  $d$ , the representation  $\rho = \gamma(d)$  is a uniform distribution over permutations of values  $v_1^{(d)}, \dots, v_k^{(d)}$ . In this case, computing the cardinality of a region is a simple combinatorial problem.

Here, we make use of the notion of canonical value sequences introduced in Section 4.3.2: The number of ground states with non-zero probability in  $l$  is the product of the number

---

**Algorithm 11** Test whether states in  $M$  can be merged exactly into  $l^*$ .

---

```

1: function MERGEABLE( $M = \langle l_i : p_i \rangle_{i=1}^n, l^*$ )
2:    $p^* \leftarrow \sum_{l \in M} M(l)$ 
3:   if  $\sum_{l \in \text{dom}(M)} |\text{region}(l)| \neq |\text{region}(l^*)|$ : return false
4:   for each  $l \in \text{dom}(M)$  do
5:     if  $\text{region}(l) \not\subseteq \text{region}(l^*)$ : return false
6:     if  $M(l) |\text{region}(l)|^{-1} \neq p^* |\text{region}(l^*)|^{-1}$  return false
7:   return true

```

---

of canonical value sequences of all distributions  $\rho \in \text{ran}(\gamma)$  (each combination of canonical value sequences corresponds to exactly one ground state of  $l$ ). Thus, we only need to count the number of canonical value sequences for each distribution type  $d$ , which was already derived for the general case in Section 4.3.2.

For the case considered here, computing this number is straightforward: A uniform distribution over permutations of  $v_1^{(d)}, \dots, v_k^{(d)}$  has a support of  $k!$ , i.e.  $k!$  distinct value sequences can be drawn from the distribution. The number of those sequences that are mapped to the same canonical sequence (i.e. that result in the same ground state when inserting into the entities in  $s$ ) is exactly the correction factor  $\alpha^{(d)}$  introduced in Section 4.3.2: When  $k_1, \dots, k_n$  are the multiplicities of entity structures in  $s$  referencing  $d$ , the correction factor (for hypergeometric distributions with unique values) is  $\alpha^{(d)} = \prod_{i=1}^n k_i$ .

Thus, the number of canonical value sequences of the distribution with type  $d$  is

$$m_s(d) = k! / \alpha^{(d)} = k! / \prod_{i=1}^n k_i = \binom{k}{k_1 \dots k_n}.$$

Overall, the number of ground states with non-zero probability in  $l$  is

$$|\text{region}(l)| = \prod_{d \in \text{dom}(\gamma)} m_s(d).$$

**Example 42.** Consider the lifted state  $l = (s, \gamma)$  with

$$s = \llbracket 2\langle \mathbf{N}: \mathbf{N}_1, \text{L}: 1 \rangle, 1\langle \mathbf{N}: \mathbf{N}_1, \text{L}: 2 \rangle, 2\langle \mathbf{N}: \mathbf{N}_2, \text{L}: 3 \rangle \rrbracket, \quad \gamma = \langle \mathbf{N}_1: \mathcal{U}(A, B, C), \mathbf{N}_2: \mathcal{U}(D, E) \rangle.$$

Here,  $m_s(\mathbf{N}_1) = 3! / (2! * 1!) = 3$ ,  $m_s(\mathbf{N}_2) = 2! / 2! = 1$  and thus  $|\text{region}(l)| = 3$ . The correctness can be easily checked by enumerating the three ground states in the region of  $l$ .

**Summary** In summary, we showed that we can efficiently check whether a given map  $M$  (of states to probabilities) describes the same distribution over ground states than a given state  $l^*$  with probability  $p^*$ . This was done by only considering uniform distributions over ground states, and checking whether the regions of all  $l \in \text{dom}(l)$  partition  $\text{region}(l^*)$ . In this case, we call  $M$  *mergeable* into  $l^*$ . The algorithm for this task is shown in Algorithm 11.

### 6.3.2. $l^*$ Known, $M$ Unknown: Identifying Mergeable Subsets

Next, we consider the case where the states  $M$  that we want to merge are not given a priori, but need to be identified from a larger map  $P$ . For now, we assume that the merge result

## 6. Lifted Marginal Filtering in Asymmetrical Models

**Algorithm 12** For a merge candidate  $M$  (w.r.t.  $l^*$ ) and a new state  $l$  with probability  $p$ , test whether  $M \oplus \langle l : p \rangle$  is still a merge candidate w.r.t.  $l^*$ .

---

```

1: function IS-CANDIDATE( $M = \langle l_i : p_i \rangle_{i=1}^n, l, p, l^*$ )
2:   if  $\text{region}(l) \not\subseteq \text{region}(l^*)$ : return false
3:   Select arbitrary  $l_i \in \text{dom}(M)$ 
4:   if  $p |\text{region}(l)|^{-1} \neq M(l_i) |\text{region}(l_i)|^{-1}$  return false
5:   return true

```

---

$l^*$  is still given. The goal is then to identify a map  $M$  with  $\text{dom}(M) \subseteq \text{dom}(P)$  that is mergeable into  $l^*$ .

In principle, this is straightforward: We simply collect in  $M$  all states  $l$  that satisfy conditions (a) and (c) for mergeability (Theorem 1), i.e. where  $\text{region}(l) \subseteq \text{region}(l^*)$  and where

$$p^* |\text{region}(l^*)|^{-1} = P(l) |\text{region}(l)|^{-1}.$$

When the states furthermore satisfy condition (b), i.e. when

$$\sum_{l \in \text{dom}(M)} |\text{region}(l)| = |\text{region}(l^*)|,$$

then  $M$  is mergeable into  $l^*$ .

**Merge Candidates** More formally, we call a map  $M$  a *merge candidate* of  $l^*$ , when conditions (a) and (c) from Theorem 1 are satisfied. When  $M$  is a merge candidate, there is a completion  $C$  so that  $M \oplus C$  is mergeable into  $l^*$ . This is easy to see: The completion  $C$  needs to satisfy conditions (a) and (c) of Theorem 1, and have groundings  $\text{region}(l^*) \setminus \bigcup_{l \in \text{dom}(M)} \text{region}(l)$ . This way, it provides the remaining groundings so that condition (b) from Theorem 1 is also satisfied. Such states always exist (in the worst case, each state in  $C$  describes exactly one grounding, but are not necessarily present in  $P$ ).

**Example 43.** Consider the states  $l_1 = (s_1, \gamma_1)$ ,  $l_2 = (s_2, \gamma_2)$  and  $l_3 = (s_3, \gamma_3)$  from Example 40, where:

$$\begin{aligned}
s_1 &= \llbracket 1\langle \mathbf{N}: A, \mathbf{L}: 1 \rangle, 1\langle \mathbf{N}: \mathbf{N}, \mathbf{L}: 2 \rangle, 1\langle \mathbf{N}: \mathbf{N}, \mathbf{L}: 3 \rangle \rrbracket, & \gamma_1 &= \langle \mathbf{N}: \mathcal{U}(B, C) \rangle \\
s_2 &= \llbracket 1\langle \mathbf{N}: B, \mathbf{L}: 1 \rangle, 1\langle \mathbf{N}: \mathbf{N}, \mathbf{L}: 2 \rangle, 1\langle \mathbf{N}: \mathbf{N}, \mathbf{L}: 3 \rangle \rrbracket, & \gamma_2 &= \langle \mathbf{N}: \mathcal{U}(A, C) \rangle \\
s_3 &= \llbracket 1\langle \mathbf{N}: C, \mathbf{L}: 1 \rangle, 1\langle \mathbf{N}: \mathbf{N}, \mathbf{L}: 2 \rangle, 1\langle \mathbf{N}: \mathbf{N}, \mathbf{L}: 3 \rangle \rrbracket, & \gamma_3 &= \langle \mathbf{N}: \mathcal{U}(A, B) \rangle
\end{aligned}$$

The map  $M \langle l_1 : 1/3, l_2 : 1/3 \rangle$  is a merge candidate, which can be seen by the fact that conditions (a) and (c) of Theorem 1 are satisfied, or by selecting  $C = \langle l_3 : 1/3 \rangle$  and noting that  $M \oplus C$  is mergeable. On the other hand, the map  $M = \langle l_1 : 1/3, l_2 : 1/4 \rangle$  is not a merge candidate, as condition (c) is not satisfied.

In the context of the overall merge algorithm presented below, it is sufficient to test whether a given merge candidate  $M$  stays a merge candidate when a new state  $l$  with probability  $p$  is inserted into  $M$ . A procedure for testing this is shown in Algorithm 12.

---

**Algorithm 13** Identify states  $M$  that are mergeable into given state  $l^*$  from a given map  $P$ , and perform the merge.

---

```

1: function MERGE-WITH-KNOWN-RESULT( $P = \langle l_i : p_i \rangle_{i=1}^n, l^*$ )
2:    $M \leftarrow \langle \rangle$ 
3:   for  $i$  in  $1, \dots, n$  do
4:     if MERGE-CANDIDATE( $M, l_i, p_i, l^*$ ) then
5:        $M \leftarrow M \oplus \langle l_i, p_i \rangle$ 
6:       if MERGEABLE( $M, l^*$ ) then
7:          $p^* \leftarrow \sum_{l \in M} M(l)$ 
8:         return  $P \ominus M \oplus \langle l^* : p^* \rangle$ 
9:   return  $P$ 

```

---

**A Search Algorithm for  $M$**  Based on the notion of merge candidates, the algorithm to identify a map  $M$  that is mergeable w.r.t.  $l^*$  from a larger map  $P$  is straightforward (see Algorithm 13). The algorithm simply iterates over all states in  $P$ , and maintains a merge candidate  $M$ . Each state  $l \in \text{dom}(P)$  that can be inserted into  $M$  so that  $M$  is still a merge candidate is inserted, until either  $M$  can be merged (in which case the merge is directly performed by returning  $P \ominus M \oplus \langle l^* : p^* \rangle$ ), or until all states in  $P$  have been iterated and  $M$  cannot be merged.

### 6.3.3. $M$ and $l^*$ Unknown: A Greedy Search Algorithm

In this section, we consider the case where the merge result  $l^*$  is also unknown a priori. That is, the task can be stated as follows: Given a map  $P$  of states and probabilities, identify a sub-map  $M$  of  $P$  and a state  $l^*$ , so that the states in  $M$  are mergeable into  $l^*$ . When such a map  $M$  and a state  $l^*$  have been identified, the merge is performed (by setting  $P' = P \ominus M \oplus \langle l^* : p^* \rangle$ ) and the process is iterated for the new map  $P'$ . Overall, the goal is to reduce the number of posterior states as much as possible.

**Exhaustive Algorithm** We first describe an *optimal* algorithm (in the sense of the smallest number of posterior states). We assume that the distribution is given as a map  $P = \langle l_i : p_i \rangle_{i=1}^n$  of lifted states to probabilities. The merging algorithm performs two steps:

- (i) Test each subset  $M$  of states in  $\text{dom}(P)$  for mergeability
- (ii) Select the merges that are actually performed to obtain the minimal number of posterior states.

To understand why the second step is necessary, note that each state can be involved in multiple different merges, so that not all possible merges can be performed simultaneously. For example, suppose that the states  $\{l_1, \dots, l_4\}$  exist, and the subsets  $M_1 = \{l_1, l_2\}$ ,  $M_2 = \{l_1, l_3\}$  and  $M_3 = \{l_3, l_4\}$  can be merged. When  $M_2$  is merged,  $M_1$  and  $M_3$  cannot be merged any more, as  $l_1$  (or  $l_3$ ) is no longer available. In this example, the minimal number of posterior states is obtained by merging  $M_1$  as well as  $M_3$ . In general, identifying the optimal merges is an instance of the set cover problem, a classical NP-hard problem [99]. Thus, overall, the optimal algorithm requires exponential time, both for testing mergeability for each of the exponentially many subsets, and for solving the NP-hard problem of identifying the optimal merges.

## 6. Lifted Marginal Filtering in Asymmetrical Models

---

### Algorithm 14 Greedy algorithm for merging disjoint states.

---

```

1: function MERGE-DISJOINT( $P = \langle l_i : p_i \rangle_{i=1}^n$ )
2:   if  $P = \langle \rangle$ : return  $\langle \rangle$  ▷ Occurs due to recursive call when we are done
3:    $M \leftarrow \langle l_1 : p_1 \rangle$  ▷ First (or arbitrary) state in  $P$ 
4:    $R \leftarrow \text{MERGE-RESULTS}(l_1)$ 
5:   for  $i$  in  $2, \dots, n$  do
6:      $R' \leftarrow \{l^* \mid \text{IS-CANDIDATE}(M, l_i, p_i, l^*), l^* \in R\}$  ▷ Update merge results
7:     if  $R' \neq \emptyset$  then
8:        $R \leftarrow R'; M \leftarrow M \oplus \langle l_i : p_i \rangle$ 
9:       if  $\text{MERGEABLE}(M, l^*)$  for any  $l^* \in R$  then
10:         $p^* \leftarrow \sum_{l \in M} M(l)$ 
11:         $P \leftarrow P \ominus M \oplus \langle l^* : p^* \rangle$ 
12:        return  $\text{MERGE-DISJOINT}(P)$  ▷ Look for further merges
   return  $\langle l_1 : p_1 \rangle \oplus \text{MERGE-DISJOINT}(P \ominus \langle l_1 : p_1 \rangle)$  ▷ No merge was performed for  $l_1$ , try with rest

```

---

**Greedy Algorithm** To alleviate this hardness, we propose a greedy merge algorithm, shown in Algorithm 14.

Conceptually, the algorithm is an extension of Algorithm 13: It performs a greedy search procedure, iteratively building a merge candidate  $M$ , while additionally maintaining a set  $R$  of possible merge results of  $M$  (how to find the states in  $R$  is discussed below). At step  $i$ , the algorithm tests whether the state  $l_i \in \text{dom}(P)$  can be added to  $M$ . Specifically, it tests which of the possible merge results of  $M$  are also merge results of  $M \oplus \langle l_i : p_i \rangle$  (line 6), and whether there is at least one remaining merge result. When this is the case, it inserts  $l_i$  into  $M$  (line 8). Whenever  $M$  can be merged exactly into any state  $l^* \in R$ , the merge is performed directly (line 11), and the algorithm is recursively applied to the resulting states.

Note that this is a greedy algorithm that directly performs all merges it finds, instead of identifying optimal merges. Furthermore, the algorithm does not test all subsets of states in  $P$  for mergeability, but only tests all contiguous sub-sequences of the ordered sequence  $l_1, \dots, l_n$ . Thus, not all possible merges are identified this way.

The algorithm has quadratic runtime with respect to the number of states (as a sequence of  $n$  unique elements has  $\frac{n(n+1)}{2}$  contiguous sub-sequences), in contrast to the exponential runtime of the optimal algorithm. An obvious way to improve the greedy algorithm – to identify more merges – is to repeat the algorithm multiple times, using a random shuffle of the result of the previous run as input.

**Possible Merge Results** The remaining question is how line 4 of algorithm 14 can be realized: How can we construct the set of possible merge results for a given lifted state? Unfortunately, without additional restrictions, the size of  $R$  can be infinite: For example, consider the state  $l_1 = (s_1, \gamma_1)$  with

$$s_1 = \llbracket 1\langle N: A, L: 1 \rangle, 1\langle N: B, L: 2 \rangle, 1\langle N: C, L: 3 \rangle \rrbracket, \quad \gamma_1 = \langle \rangle$$



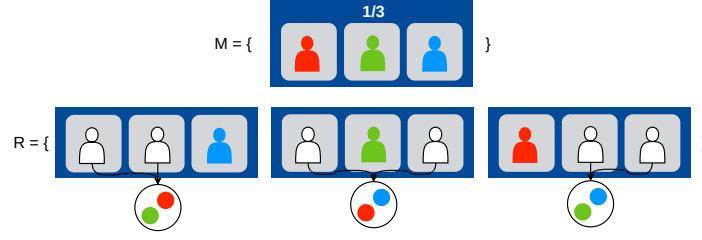


Figure 6.7.: Possible merge results for the state  $l$  shown on the upper part of the figure. Each possible merge result is identical to  $l$ , except that two of the distributions in the context have been combined into one.

---

**Algorithm 15** Create possible merge results for a state  $l$ .

---

```

1: function MERGE-RESULTS( $l = (s, \gamma)$ )
2:    $R \leftarrow \emptyset$ 
3:   for each pair  $d_1, d_2 \in \text{dom}(\gamma)$  do                                 $\triangleright$  Including singleton distributions
4:      $\rho_1 \leftarrow \gamma(d_1), \rho_2 \leftarrow \gamma(d_2)$ 
5:     Let  $d_r$  be a new distribution type
6:     Let  $\rho_r$  be the uniform distribution over permutations of all values from  $\rho_1$  and  $\rho_2$ 
7:      $\gamma_r \leftarrow \gamma \setminus \langle d_1 : \rho_1, d_2 : \rho_2 \rangle \odot \langle d_r : \rho_r \rangle$ 
8:     Let  $s_r$  be identical to  $s$ , except all occurrences of  $d_1$  and  $d_2$  are replaced by  $d_r$ 
9:      $R \leftarrow R \cup \{(s_r, \gamma_r)\}$ 
10:  return  $R$ 

```

---

and the map  $M = \langle l : p \rangle$ . There are infinitely many states  $l^*$  for which  $M$  is a merge candidate, e.g. the states  $l_1^*$  and  $l_2^*$  with

$$\begin{aligned}
 s_1^* &= \llbracket 1\langle \mathbf{N} : C, L : 1 \rangle, 1\langle \mathbf{N} : \mathbf{N}, L : 2 \rangle, 1\langle \mathbf{N} : \mathbf{N}, L : 3 \rangle \rrbracket, & \gamma_1^* &= \langle \mathbf{N} : \mathcal{U}(A, B) \rangle \\
 s_2^* &= \llbracket 1\langle \mathbf{N} : C, L : 1 \rangle, 1\langle \mathbf{N} : \mathbf{N}, L : 2 \rangle, 1\langle \mathbf{N} : \mathbf{N}, L : 3 \rangle \rrbracket, & \gamma_2^* &= \langle \mathbf{N} : \mathcal{U}(A, B, C, D) \rangle
 \end{aligned}$$

Therefore, we only consider a constrained set of possible merge results. For a given state  $l$ , we define the possible merge results as the states that are identical to  $l$ , except that two of the uniform distributions over permutations  $\rho_1$  and  $\rho_2$  in  $\gamma$  are combined into one distribution  $\rho_r$ , that is a uniform distribution of permutations of values from both  $\rho_1$  and  $\rho_2$ . For example, for the state  $l_1$  above, we only consider

$$\begin{aligned}
 s_1^* &= \llbracket 1\langle \mathbf{N} : \mathbf{N}, L : 1 \rangle, 1\langle \mathbf{N} : \mathbf{N}, L : 2 \rangle, 1\langle \mathbf{N} : C, L : 3 \rangle \rrbracket, & \gamma_1^* &= \langle \mathbf{N} : \mathcal{U}(A, B) \rangle \\
 s_2^* &= \llbracket 1\langle \mathbf{N} : \mathbf{N}, L : 1 \rangle, 1\langle \mathbf{N} : B, L : 2 \rangle, 1\langle \mathbf{N} : \mathbf{N}, L : 3 \rangle \rrbracket, & \gamma_2^* &= \langle \mathbf{N} : \mathcal{U}(A, C) \rangle \\
 s_3^* &= \llbracket 1\langle \mathbf{N} : A, L : 1 \rangle, 1\langle \mathbf{N} : \mathbf{N}, L : 2 \rangle, 1\langle \mathbf{N} : \mathbf{N}, L : 3 \rangle \rrbracket, & \gamma_3^* &= \langle \mathbf{N} : \mathcal{U}(B, C) \rangle
 \end{aligned}$$

as merge results (see Figure 6.7). The algorithm that constructs this set  $R$  for a given state  $l$  is shown in Algorithm 15.

**Summary** In summary, we provided a greedy merge algorithm that allows to identify subsets of states that can be merged, as well as the corresponding merge results. All steps of the algorithms can be performed without complete enumeration of ground states, i.e. the algorithm can efficiently identify and perform merges. Note that the algorithm is exact (it

does not change the ground distribution), but not optimal (it does not identify all possible merges).

### 6.3.4. Experimental Evaluation

The goal of the experiments was to evaluate the runtime of the greedy merge algorithm – in comparison to the exhaustive algorithm – and its ability to reduce the number of states that are required to represent the distribution. Specifically, we assessed a version of the algorithm where the greedy algorithm (Algorithm 14) is repeated  $m$  times, using a random shuffle of the results of the previous run as input. As the algorithm is exact, an error between original and merged distribution does not need to be assessed.

**Experimental Design** We performed experiments with two different models: First, a simple benchmark model was used to compare the exhaustive and the greedy versions of the MERGE-DISJOINT algorithm regarding runtime and number of posterior states. Afterwards, the algorithm was evaluated with a version of the tracking scenario described in Section 4.5.1.

**Benchmark Scenario:** For the synthetic benchmark scenario, a state distribution was obtained as follows: For a state  $l = (s, \gamma)$  with

$$s = \llbracket 3\langle \mathbf{N}: \mathbf{N}, \mathbf{L}: X \rangle, 3\langle \mathbf{N}: \mathbf{N}, \mathbf{L}: Y \rangle, 3\langle \mathbf{N}: \mathbf{N}, \mathbf{L}: Z \rangle \rrbracket, \quad \gamma = \langle \mathbf{N}: A, B, C, D, E, F, G, H, I \rangle,$$

and probability  $p(l) = 1$ , we iteratively performed  $k \in \{2, 3, 4, 5, 6\}$  splits (e.g. for  $k = 3$ , splits on the constraints  $e.N == A$ ,  $e.N == B$  and  $e.N == C$  were performed). This resulted in sets of states of size  $n \in \{9, 27, 77, 200, 450\}$ . Each of the resulting sets can always be merged into the original state  $l$ . For each set, we ran the greedy algorithm for  $m \in \{1, 2, 5, 10\}$  iterations. The exhaustive algorithm was run only for  $n = 9$  and  $n = 27$ , as it exceeded a threshold of 600 s for larger  $n$ . For each configuration, 10 repetitions were performed, and means of runtime and number of posterior states are reported for each configuration. N

**Tracking Scenario:** Additionally, we assessed the effect of the merging algorithm on a dynamic system that is a variant of the tracking scenario introduced in Section 4.5.1. Specifically, we assessed the number of states that are necessary when either merging is performed after each prediction operation, or not performed. In this model, 3 agents move around 5 rooms. The rooms follow a sequential layout, e.g. agents can move between rooms 1 and 2, but not between 1 and 3. At each timestep, each agent can either change to an adjacent room, or stay at the current room (with a certain probability).

Each observation  $y_t$  is a vector that indicates the location of each agent. Locations can also be unobserved, denoted as “NA”. For example, when  $y_t = (\text{“NA”}, 4, \text{“NA”})^T$ , the agents  $A$  and  $C$  have not been observed, and agent  $B$  has been observed at location 4. We denote the location of agent  $a$  as  $y_t^{(a)}$ . We assume that when the location is observed, it is correct in 90% of the cases, and when the location is not observed, each state is weight with a constant normalization factor. Thus, the observation model is  $p(y | x) = \prod_a p(y_t^{(a)} | x_t)$  with

$$p(y_t^{(a)} | x_t) = \begin{cases} 0.9 & \text{if } y_t^{(a)} \neq \text{“NA” and agent } a \text{ is at location } y_t^{(a)} \text{ in } x_t \\ 0.1/4 & \text{if } y_t^{(a)} \neq \text{“NA” and agent } a \text{ is not at location } y_t^{(a)} \text{ in } x_t \\ 1/5 & \text{if } y_t^{(a)} = \text{“NA”} \end{cases} \quad (6.16)$$

Scenario	Factor	Levels	Description
Benchmark	$n$	9,27,77,200,450	Prior states
	$i$	1,2,5,10	Iterations of merging algorithm
Tracking	$f$	2,5,10	Frequency of identifying an agent
	$i$	0,1,5,10	Iterations of merging algorithm

Table 6.2.: Factors and levels of experimental design for evaluating the MERGE-DISJOINT algorithm.

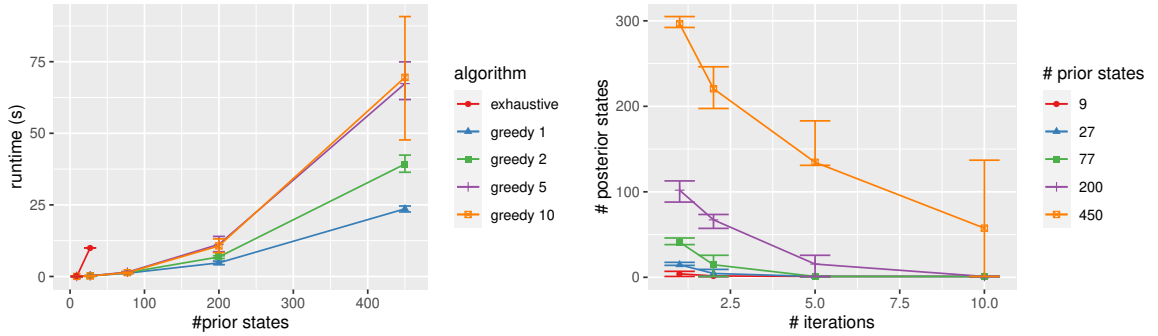


Figure 6.8.: Evaluation of the MERGE-DISJOINT algorithm. Left: Runtime of the merging algorithm with respect to number of states before merging. The runtime grows approximately linear with the number of states. Right: Number of states after merging, with respect to number of iterations of the algorithm. In principle, merging into a single state is possible in all cases. The number of posterior states decreases quickly with the number of iterations.

Observing the location of an agent leads to a *split* of the distribution: The observation likelihood  $p(y_t | x_t)$  depends on the location of a specific agent  $a$ , i.e. each state must be split into the situations where agent  $a$  is at location  $y_t^{(a)}$ , and where it is *not* at location  $y_t^{(a)}$ .

We varied the frequency  $f$  of identifying one of the agents (each 10th, 5th or 2nd time step) and the number of iterations performed by the merging algorithm (0, 1, 5 or 10). The more frequently the agents are identified, the more splits need to be performed and the larger is the representational complexity of the distribution. Note that such a relatively low number of splits (every 5th or 10th timestep) is realistic for real-world domains: In the kitchen scenario investigated above, about 4 splits need to be performed during the approximately 100 timesteps. The factors and levels of the experimental design are shown in Table 6.2.

**Results** Figure 6.8 shows runtime and number of posterior states for the benchmark scenario. The exhaustive algorithm quickly becomes infeasible: For  $n = 27$  states, it has a mean runtime of more than 10 s, and for  $n = 77$ , it exceeds the threshold of 600 s. The runtime of the greedy algorithm grows approximately linear with the number of states, and sub-linear with the number of iterations (later iterations can be faster, as the number of states is already smaller and thus the number of possible merges is smaller). From 6.8 (right), we can see that the number of posterior states decreases quickly with the number of iterations of the algorithm, such that the greedy algorithm can efficiently reduce the representational

## 6. Lifted Marginal Filtering in Asymmetrical Models

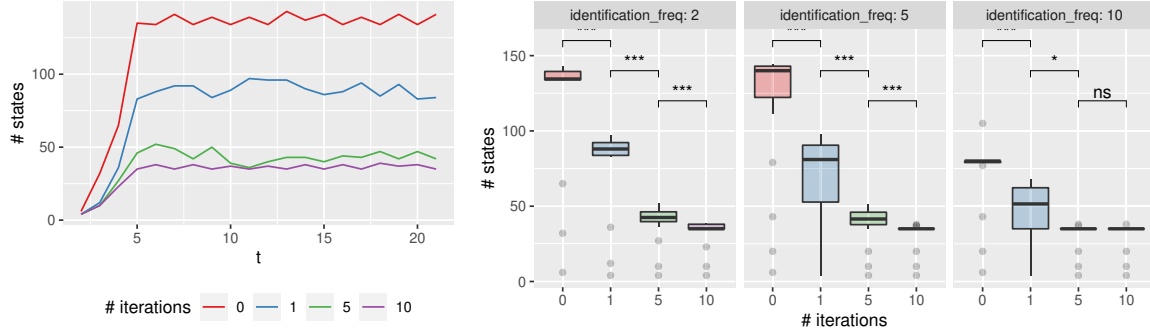


Figure 6.9.: Evaluation of the MERGE-DISJOINT algorithm. Left: Example run of the tracking scenario with identifying observations every 2nd timestep. Right: Overall results for the tracking scenario, showing mean number of states for different frequencies  $f$  of identifying observation (2, 5 or 10) and number of iterations of the greedy merging algorithm (0, 1, 5 or 10). Stars indicate significant differences in the number of states, using Wilcoxon signed-rank test. \*:  $p < 0.05$ , \*\*:  $p < 0.01$ , \*\*\*:  $p < 0.001$ .

complexity.

For the tracking scenario, Figure 6.9 (left) shows the number of states required to represent the filtering distribution  $p(x_t | y_{1:t})$  over time  $t$ , when identifying observations are made every 2nd timestep. When no merging is employed (i.e. # iterations = 0), the number of necessary states grows quickly, as each split increases the representational complexity (until the representation is effectively completely ground after only a few time steps). When the greedy merging algorithm is applied, the mean (as well as the maximum) number of states that needs to be maintained is substantially smaller.

Figure 6.9 shows the complete results for the tracking scenario. In all cases, merging can effectively reduce the number of states that need to be maintained. Even when just one iteration of the merging algorithm is performed, the required number of states is reduced significantly ( $p < 0.001$ , Wilcoxon signed rank test,  $n = 20$ ). Increasing the number of merging iterations from 1 to 5 leads to a further reduction in the number of states in all three cases ( $p < 0.05$ , Wilcoxon signed rank test,  $n = 20$ ). When increasing the number of iterations further from 5 to 10, the decrease in required states is only significant for some of the scenarios (when identifying observations occur every 2nd or 5th timestep), but not significant for others (when identifying observations occur every 10th timestep). This observation indicates a saturation effect, i.e. most merges have already been identified with 5 iterations so that increasing the number of iterations further has only a small effect.

**Summary & Future Work** The results show that the merging algorithm described here can be usefully employed. The number of state representatives is reduced substantially, and thus, the algorithm allows more efficient inference – without introducing any approximation.

Here, a group of states is only merged when it can be replaced *exactly* by a state  $l^*$ . However, the algorithm directly lends itself to approximations, such that a merge is also performed when Equation 6.14 holds only approximately. Specifically, approximations can be easily introduced into the mergeability conditions: Condition (c) of Theorem 1 can be relaxed, such that instead of requiring that the probability of all ground states of all  $l \in \text{dom}(M)$  are iden-

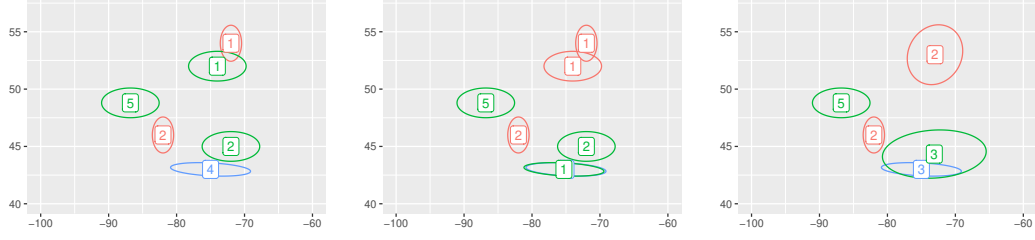


Figure 6.10.: Example of the approach for merging entities that consist of normal distributions, for the online multiplayer game described in Example 44. Each ellipse denotes the location distribution of an entity (i.e. a village), the colors denote the different owners, and the numbers denote the multiplicity of that entity. Left: Prior game state. Middle: Game state after several conquer actions occurred. Right: New game state, after merging some of the entities by Gaussian Mixture reduction methods.

tical, we only require that their difference is not larger than some value  $\epsilon$ . Similarly, condition (b) can be relaxed, so that instead of requiring that  $\sum_{l \in \text{dom}(M)} |\text{region}(l)| = |\text{region}(l^*)|$ , we only require  $\sum_{l \in \text{dom}(M)} |\text{region}(l)| - |\text{region}(l^*)| < \epsilon$ . Evaluating such approximate methods is a topic for future work.

Finally, another topic for future work is to generalize this merging algorithm to make it applicable to more cases, instead of requiring that all distributions in  $\gamma$  are uniform distributions over permutations.

## 6.4. Merging Normal Distributions

In this section, we consider another special case: All properties of an entity follow a singleton distribution except for one property, that follows a (multivariate) normal distribution. An example of a scenario for which this special case is relevant is the multiplayer online game Travian<sup>4</sup>, which was already investigated previously as an evaluation scenario for a (ground) Bayesian filtering algorithm [213].

**Example 44.** Travian is an online multiplayer strategy game consisting of a grid map. Players own one or multiple villages located on the map. We focus on the high-level aspects of the game, namely village ownership and attacks. Specifically, the entities of a game state represent villages, containing owner and location information. We assume that there is no exact location information, but the location of each village is represented by a normal distribution<sup>5</sup>. The situation depicted in Figure 6.10 (left) is represented by the lifted state  $l = (s, \gamma)$  with

$$\begin{aligned} s &= \llbracket 4 \langle \text{P: Blue, L: } \mathbf{L}_1 \rangle, 2 \langle \text{P: Green, L: } \mathbf{L}_2 \rangle, 5 \langle \text{P: Green, L: } \mathbf{L}_3 \rangle, \dots \rrbracket, \\ \gamma &= \langle \mathbf{L}_1: \mathcal{N}(\mu_1, \Sigma_1), \mathbf{L}_2: \mathcal{N}(\mu_2, \Sigma_2), \mathbf{L}_3: \mathcal{N}(\mu_3, \Sigma_3), \dots \rangle. \end{aligned} \quad (6.17)$$

<sup>4</sup>[www.travian.com](http://www.travian.com)

<sup>5</sup>For now, we assume that the location distribution is given a priori. The distribution can, for example, be estimated by fitting a Gaussian mixture to the locations of all villages of each player, as shown in Appendix G.2.

## 6. Lifted Marginal Filtering in Asymmetrical Models

Over time, the number of distinct entity structures (*species*) grows due to the system dynamics. For example, when one of the four entities  $\langle P: \text{Blue}, \text{Loc}: \mathbf{L}_1 \rangle$  is conquered by the green player, the posterior state contains three instances of  $\langle P: \text{Blue}, \text{Loc}: \mathbf{L}_1 \rangle$  and one instance of  $\langle P: \text{Green}, \text{Loc}: \mathbf{L}_1 \rangle$ . This situation is visualized in Figure 6.10 (middle).

Recall that the number of compound actions (and thus the number of posterior states) depends on the number of species in  $s$ . Thus, the goal here is to reduce the number of species in  $s$ . More specifically, given a state  $l$ , the goal is to compute a new state  $l'$  with fewer species, i.e.  $|\text{dom}(s')| < |\text{dom}(s)|$ . Intuitively, this is done by identifying species that are “sufficiently close” and merging them into a single species.

### 6.4.1. Merging Entities by Gaussian Mixture Reduction

In this section, we show how Gaussian mixture reduction methods [181] can be used for merging. This is done by deriving an expression for the the distribution over value sequences  $p(\mathbf{v} | s, \gamma)$  that can be viewed as a Gaussian mixture.

Here, we only consider the case where Gaussian factors that are referenced multiple times are products of univariate, i.i.d. Gaussian factors. For example, in the state  $l$  in Equation 6.17, the factor with type  $\mathbf{L}_1$  is referenced four time, thus  $\mathcal{N}(\mu_1, \Sigma_1)$  denotes the distribution that consists of four i.i.d. normal distributions that all have parameters  $\mu_1$  and  $\Sigma_1$ .

Furthermore, for now, we limit the discussion to multiset structures  $s = \llbracket n_1 e_1, \dots, n_m e_m \rrbracket$ , where all  $e_i$  are identical, except that they reference a different representation of a normal distribution  $\rho_i$  in  $\gamma$ . For example, in the multiplayer game described above, we consider sub-multisets where all entities have the same value for the *owner* property.

For a sub-state  $l = (s, \gamma)$  of this form, the sampling process for obtaining a ground state  $x \in \text{region}(l)$  can be described as follows:

- (i) Sample from each normal  $\rho_i$  exactly  $n_i$  times,
- (ii) insert the values into the corresponding entities  $e_i$ , and
- (iii) collect all entities in a multiset  $x$ .

Due to step (i) of the sampling process, the distribution of values  $\mathbf{v}^{(d)}$  that belong to distribution type  $d$  factorizes completely, i.e.

$$p(\mathbf{v}^{(d)} | s, \gamma) = \prod_{j=1}^n p_{\gamma(d)}(v_j^{(d)}), \quad (6.18)$$

where  $n$  is the multiplicity of the entity structure that references  $d$  and  $v_j^{(d)}$  is the  $j$ -th value of the sequence  $\mathbf{v}^{(d)}$ . Subsequently, the distribution of the complete value sequence  $\mathbf{v}$  is given by

$$p(\mathbf{v} | s, \gamma) = \prod_{(d, \rho) \in \gamma} \prod_{j=1}^n p_{\rho}(v_j^{(d)}). \quad (6.19)$$

Recall that the goal of this section is to rewrite this distribution as a Gaussian mixture. We see that the distribution  $p(\mathbf{v} | s, \gamma)$  is (almost) a Gaussian mixture, except that we sample *exactly*  $n$  times from each factor. By dropping this constraint, and assuming that each value  $v_j$  can be sampled from *any* of the normal distributions, and that the probability of selecting

the distribution  $d$  is proportional to the number of references to  $d$ , the distribution becomes

$$p(\mathbf{v} | s, \gamma) = \prod_{j=1}^N \sum_{(d, \rho) \in \gamma} \frac{n}{N} p_{\rho}(v_j), \quad (6.20)$$

where  $N = \sum_{n \in \text{range}(s)} n$  is the total multiplicity of  $s$ . This distribution describes a sequence of i.i.d. samples from a mixture, where  $p_{\rho}$  are the mixture components and  $n/N$  is the weight of each mixture. It is easy to see that the expected number of draws from each normal in this distribution is the same as in the original distribution in Equation 6.19.

For Gaussian mixtures, the problem of reducing the number of mixture components is well-understood: Given a Gaussian mixture, a mixture with fewer components that has a minimal distance to the original mixture can be computed. For our approach, we employ the same concepts: After each transition, we apply a merging procedure that reduces the number of different entity structures (i.e. mixture components). Specifically, we employ the procedure described by Runnalls [181]: Let  $s_i$  and  $s_j$  be two entities in a state  $s$ , with multiplicities  $n_i$  and  $n_j$ . Assume that they have associated value distributions  $p_i \propto \mathcal{N}(\mu_i, \Sigma_i)$  and  $p_j \propto \mathcal{N}(\mu_j, \Sigma_j)$ . The Gaussian mixture of  $p_i$  and  $p_j$  is  $p_m = w_i p_i + w_j p_j$ , where  $w_i = n_i/n$  and  $w_j = n_j/N$ . The parameters of the normal  $p_{ij} \propto \mathcal{N}(\mu_{ij}, \Sigma_{ij})$  with minimal Kullback-Leibler divergence to  $p_m$  are calculated as follows:

$$\begin{aligned} w_{ij} &= w_i + w_j, & w_{i|ij} &= \frac{w_i}{w_i + w_j}, & w_{j|ij} &= \frac{w_j}{w_i + w_j} \\ \mu_{ij} &= w_{i|ij} \mu_i + w_{j|ij} \mu_j \\ \Sigma_{ij} &= w_{i|ij} \Sigma_i + w_{j|ij} \Sigma_j + w_{i|ij} w_{j|ij} (\mu_i - \mu_j)(\mu_i - \mu_j)^T \end{aligned}$$

The weight of the new component  $p_{ij}$  is  $w_{ij}$ , and the multiplicity of the new entity is  $n_i + n_j$ . Given a mixture, it is possible to find the *optimal* components to merge, i.e. those producing the smallest change in Kullback-Leibler divergence of the mixtures before and after merging. This is done by calculating the bound  $B(i, j) = \frac{1}{2} [w_{ij} \log|\Sigma_{ij}| - w_i \log|\Sigma_i| - w_j \log|\Sigma_j|]$  for all combinations of components  $p_i$  and  $p_j$  and selecting the two components for which  $B(i, j)$  is minimal. This operation is repeated, until either the minimal bound  $B(i, j)$  exceeds a threshold or the desired number of mixture components is reached.

This merging procedure is applied separately to all sub-states that are formed by assignments of the other properties that do not follow normal distributions (in the Travian scenario, merging is done separately for each player). An example for the Travian scenario is shown in Figure 6.10 (right) – where the villages of the red player in the north-east, as well as the villages of the green player in the south-east are merged.

### 6.4.2. Experimental Evaluation

The goal of the experiments was to evaluate the effect of this merging approach on runtime and accuracy (in terms of AUC), in relationship to the domain size (i.e. number of entities) and extent of the approximation (i.e. maximum number of mixture components after merging).

**Data and Modeling Approach** We evaluated the proposed approach on a prediction task in Travian, as introduced in Example 44. The same application scenario has also been

## 6. Lifted Marginal Filtering in Asymmetrical Models

Factor	Levels	Description
Entity structures per player	1, 3, 5, $\infty$ (ground representation)	Maximum number of Entity structures per player, i.e. mixture components
Timesteps	2, ..., 5	Prediction horizon
Players	3, ..., 10	Number of players per state
Merging	yes, no	Gaussian mixture reduction
Dataset	1, ..., 10	Random subset of complete data

Table 6.3.: Factors and levels of experimental design for evaluating the Gaussian mixture merging algorithm.

investigated by Thon et al. [213] using a ground state representation. We used a similar experimental design regarding data recording, performed experiments and evaluation metrics, whenever possible.

We logged the state of a game server over 5 days and recorded high-level data (position and ownership of villages). The data was collected once every 24 hours. The data contains more than 6400 villages and 1400 players. We evaluated our approach on a subset of this data, that has been extracted in the same way as done by Thon et al. [213]: From the complete data, sequences of *local* game world states were sampled. Each sequence contains the game state for a small set of players (3 to 10 players). The sets were chosen such that there are no interactions with players not in the set, but a high number of interactions between players in the set. For each number of players, 10 such sets have been sampled. The largest sets (with 10 players) contain 311.6 villages on average.

The scenario was modeled in LiMa as illustrated in Example 44: The state contains high-level information, namely village location and ownership. In this simple model, only two actions are possible: A village stays with the same owner, or a village is conquered. The weight of each conquering action depends on the distance of the conquering player’s villages to the conquered village (following Thon et al. [213]). Specifically, the weight of the action “village  $e_j$  is conquered by an attack starting from village  $e_i$ ”, is the Bhattacharyya distance [68]  $D(\rho_i, \rho_j)$  of the corresponding location distributions  $\rho_i$  of  $e_i$  and  $\rho_j$  of  $e_j$ :

$$D(\rho_i, \rho_j) = \frac{1}{8} (\mu_i - \mu_j)^T \Sigma^{-1} (\mu_i - \mu_j) + \frac{1}{2} \ln \left( \frac{\det \Sigma}{\sqrt{\det \Sigma_i \det \Sigma_j}} \right),$$

where  $\rho_i = \mathcal{N}(\mu_i, \Sigma_i)$ ,  $\rho_j = \mathcal{N}(\mu_j, \Sigma_j)$  and  $\Sigma = \frac{\Sigma_i + \Sigma_j}{2}$ .

The initial state was obtained by fitting, for each player, a Gaussian mixture to all village locations of that player. Details are provided in Appendix G.2.

**Experimental Design** We used a factorial design, with factors shown in Table 6.3. We varied the number of players in a subset (from 3 to 10), which defines the domain size, and thus inference complexity and runtime. The number of entity structures per player in the initial state was set to either 1, 3 or 5. Additionally, experiments with the ground state representation were performed (i.e. the initial state contains one entity structure for each village). Multiple prediction steps (up to 5) were performed, without incorporating observations. After each prediction step, either Gaussian mixture merging was performed to



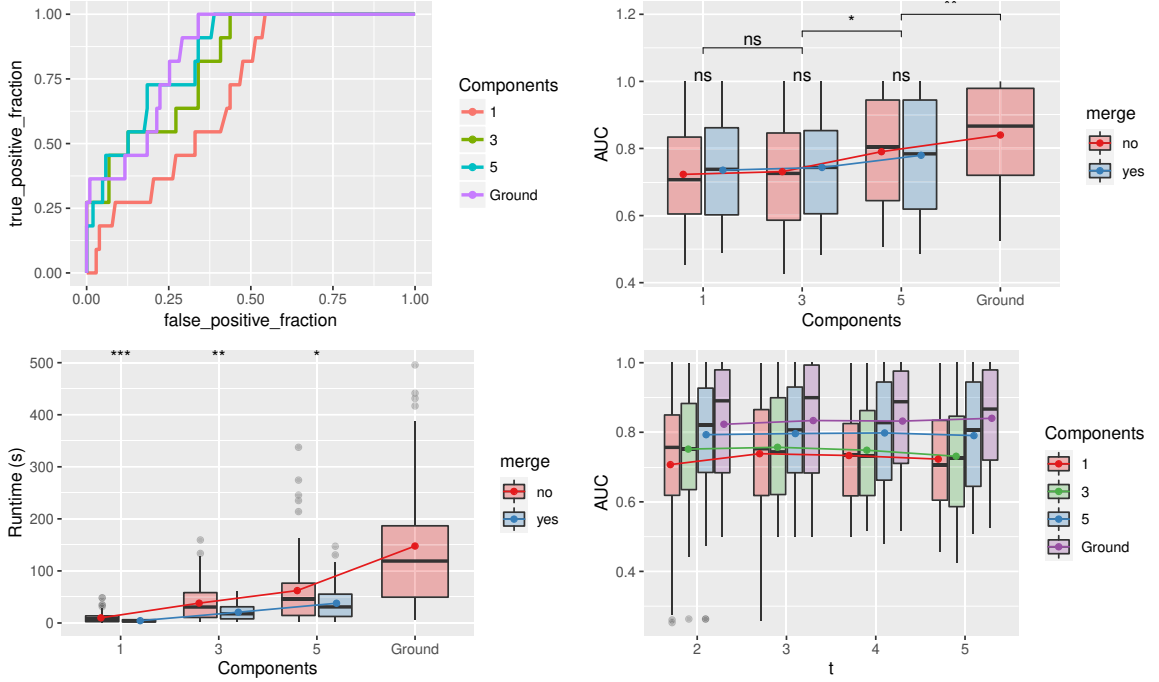


Figure 6.11.: Evaluation of Gaussian mixture merging (Travian scenario). Top left: Example of ROC curves for 3 players. Top right: AUC for different numbers of mixture components per player. Bottom left: Runtime for different number of mixture components per player. Bottom right: AUC for different prediction horizons. Stars indicate significant differences, using Wilcoxon signed-rank test. \*:  $p < 0.05$ , \*\*:  $p < 0.01$ , \*\*\*:  $p < 0.001$ .

reduce the number of mixture components (i.e. number of entity structures per player) to the maximally allowed number (1, 3 or 5), or no merging was performed.

The prediction accuracy was assessed by computing receiver operating characteristic (ROC) curves, regarding whether each conquer event did or did not occur in the transition, and by calculating the area under the curve (AUC). In all experiments, the MCMC-based algorithm to compute AMCAs as described in Chapter 5 was used with 100 samples, and pruning to a maximum number of 100 states has been performed. For each factor configuration, 10 runs have been performed. We used a prototypical implementation of our approach in R, and used the R package `mclust` [195] for fitting Gaussian mixtures by expectation maximization.

**Results** Results of the evaluation are shown in Figure 6.11. The upper left plot shows exemplary ROC curves for the subsets containing 3 players. AUC is substantially greater than 0.5, indicating that all models capture some of the characteristics of the true system dynamics. As expected, the AUC is lowest when only a single entity structure is allowed per player, as this is the most extensive approximation of the true locations. The more components are allowed, the more information about the true locations is preserved, resulting in a more accurate prediction.

Figure 6.11 (top right) shows the AUC for different numbers of allowed mixture components (i.e. species) per player, and a prediction horizon of 2. The obtained AUC values for

## 6. Lifted Marginal Filtering in Asymmetrical Models

the ground version of our approach are comparable to results reported by Thon et al. [213] (approximately 0.8 in both cases).

Allowing more components in general leads to a higher AUC, as discussed above. Here, the differences in AUC between 3 and 5 components, and between 5 components and ground inference are statistically significant ( $p < 0.05$ , Wilcoxon signed rank test,  $n = 160$ ). On the other hand, for each number of components, there is no significant difference in AUC between the runs where Gaussian mixture merging was performed or not performed (Wilcoxon signed rank test,  $n = 80$ ).

Figure 6.11 (bottom left) shows the average runtime of computing the prediction. In contrast to the AUC, the runtime increases significantly when the number of components is increased. Most notably, even the lifted models with 5 components and no merging have significantly lower runtime than the ground models ( $p < 0.05$  using Wilcoxon signed rank test,  $n = 80$ ). Furthermore, Gaussian mixture merging also leads to a significantly lower runtime, compared to the lifted models without merging ( $p < 0.05$  using Wilcoxon signed rank test,  $n = 80$ ).

Figure 6.11 (bottom right) shows the AUC for different prediction horizons (of models where mixture merging was performed). Again, the AUC is significantly higher when more components are used, and highest for ground filtering.

**Discussion** In summary, the lifted state representation allowed for a significantly lower inference runtime, but also lead to a significantly lower prediction performance in terms of AUC. Merging, on the other hand, further reduced the runtime of the lifted models significantly, without affecting AUC. Thus, the results show that merging is actually beneficial in this case.

We suspect that merging did not affect AUC in this scenario because of the smoothness of the transition model: Instead of hard constraints, the action weights depend on the values of the involved entities via a smooth function. However, when there are hard constraints on the values that are approximated by merging, merging could lead to a vastly different posterior distribution.

Overall, for the multiplayer game investigated here, the lifted representation, together with a suitable merging procedure, allowed us to perform inference for substantially larger models than what was possible by previous approaches for the same scenario [213]: For 10 players, our models contained over 300 villages on average, whereas Thon et al. [213] report results for at most 30 – 40 villages.

### 6.5. Assumed Density Merging

Finally, we present a merging concept that is based on the observation that it is sometimes possible to safely “forget” information about concrete values (that were obtained by splitting), because that information is not required later in the inference process (when no action in the future depends on that value). Specifically, it is then possible to introduce additional exchangeability assumptions for distributions involving that value.

As in Section 6.3, we only consider the case where the factors of the value distributions are uniform distributions over permutations of values  $v_1, \dots, v_k$ . Additionally, we require that there is at most one factor where  $k > 1$ , i.e. all other factors are singleton distributions (constants). This case is underlying all of the scenarios evaluated earlier in Section 4.5.

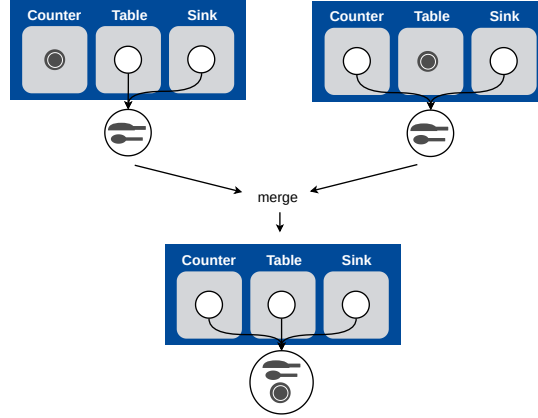


Figure 6.12.: Example of assumed density merging of two lifted states. For both states, the joint distribution of object identities is projected to an exchangeable distribution. This way, both states are mapped to the same state. Note that they cannot be merged exactly by MERGE-DISJOINT, see Example 45 for more details.

### 6.5.1. Algorithm Overview

We start by providing a concrete example of this merging approach.

**Example 45.** Consider a simplified version of the Kitchen domain (see Figure 6.12), where three objects (plate, knife and spoon, denoted as A, B, C) and three object locations (counter, table and sink, denoted as 1, 2, 3) exist, and there are two states  $l_1 = (s_1, \gamma_1)$  and  $l_2 = (s_2, \gamma_2)$  with

$$\begin{aligned} s_1 &= \llbracket 1\langle N: A, L: 1 \rangle, 1\langle N: B, L: 2 \rangle, 1\langle N: C, L: 3 \rangle \rrbracket, & \gamma_1 &= \langle N: \mathcal{U}(B, C) \rangle, \\ s_2 &= \llbracket 1\langle N: B, L: 1 \rangle, 1\langle N: A, L: 2 \rangle, 1\langle N: C, L: 3 \rangle \rrbracket, & \gamma_2 &= \langle N: \mathcal{U}(B, C) \rangle \end{aligned}$$

and  $p(l_1) = 0.5$ ,  $p(l_2) = 0.5$ . The states  $l_1$  and  $l_2$  cannot be merged exactly by MERGE-DISJOINT.

However, suppose that for all states  $l'$  that are reachable from  $l_1$  or  $l_2$ , the value of property  $N$  is never relevant for the action precondition (because whenever a constraint on  $N$  occurs in a precondition  $p$ , the precondition cannot be satisfied in  $l'$  due to some other constraint in  $p$ ). In this case, we can safely “forget” the additional information about  $N$  that is provided by the states  $l_1$  and  $l_2$ . That is, the distributions of all object names can be projected onto a uniform distribution of permutations (i.e. an exchangeable distribution).

Even when we later need to evaluate constraints on specific values of  $N$ , we can still perform this merging operation and accept that it induced an error (for example, the probability that B is at location 3 was 1/2 before merging, but 1/3 after merging).

Technically, the distribution representations<sup>6</sup>  $\rho_1 = \delta_A$  and  $\rho_2 = \mathcal{U}(B, C)$  are combined into a single representation  $\rho_r = \mathcal{U}(A, B, C)$  that represents the uniform distribution over all

<sup>6</sup>Recall that we use a shorthand notation for singleton distributions here, by writing the corresponding value directly into the structure. For example, the context  $\gamma_1$  actually contains two distribution representations,  $\mathcal{U}(B, C)$  and  $\delta_A$ .

**Algorithm 16** Assumed density merging.

---

```

1: function ADM( $l = (s, \gamma)$ )
2:   Let  $\rho_u$  be the only uniform distribution over permutations in  $\gamma$ ,
3:   and  $d_u$  be the corresponding distribution type
4:   for each singleton distribution  $\delta_v$  in  $\gamma$  with distribution type  $d_v$  do
5:     if none of the states reachable from  $l$  via AMCAs need to be split on  $v$  then
6:       Let  $\rho^*$  be the uniform distribution of permutations of  $v$  and values from  $\rho_u$ 
7:       Let  $d^*$  be a new distribution type
8:       Let  $s^*$  be identical to  $s$ , except all occurrences of  $d_v$  and  $d_u$  are replaced by  $d^*$ 
9:        $\gamma^* \leftarrow \gamma \setminus \langle d_v : \delta_v, d_u : \rho_u \rangle \odot \langle d^* : \rho^* \rangle$ 
10:      return ADM( $(s^*, \gamma^*)$ ) ▷ Recursive call, try merging other values
11:   return  $l$ 

```

---

permutations of values from  $\rho_1$  and  $\rho_2$ . Applying this operation to both  $l_1$  and  $l_2$  in both cases results in the state  $l^* = (s^*, \gamma^*)$  with

$$s^* = \llbracket 1\langle N: \mathbf{N}, L: 1 \rangle, 1\langle N: \mathbf{N}, L: 2 \rangle, 1\langle N: \mathbf{N}, L: 3 \rangle \rrbracket, \quad \gamma^* = \langle \mathbf{N}: \mathcal{U}(A, B, C) \rangle.$$

Opposed to MERGE-DISJOINT, this is an approximate operation, making the support of the distribution wider. For example, ground states that contain the entity  $\langle N: A, L: 3 \rangle$  are in the region of  $l^*$ , but not in the region of  $l_1$  or  $l_2$ . Thus, when a constraint on a value of  $N$  needs to be evaluated later on (i.e. when a split of the distribution of  $N$  is performed), this merging operation induces an error. On the other hand, the approximate representation has lower representational complexity, as seen in the example above, where multiple different states were mapped to the same state  $l^*$ . If the number of available lifted states is limited (when pruning is performed), this means that memory units become available to cover other parts of the distribution, that would otherwise not be represented at all. As we will see in Section 6.5.4, this tradeoff can indeed lead to more accurate results.

### 6.5.2. Time Points for Merging

The remaining question is at which time points this merging procedure should be applied, and which values should be merged. As outlined before, merging increases the support of the distribution. Therefore, when a constraint  $e.N == v$  needs to be evaluated after merging (and thus a split is required), the number of resulting states can be even higher than the number of states before merging. For example, splitting the state  $l^*$  from Example 45 on the constraint  $e.N == A$  results in three states ( $A$  can be at either of the three locations).

Clearly, this case needs to be avoided – the goal of merging is to reduce the number of required states, not increase it. Therefore, a value  $v$  should only be merged when a split on  $v$  is not required in the future. When this is the case, we call  $v$  *mergeable*.

**Definition 21.** [mergeability] Let  $l = (s, \gamma)$  be a lifted state. Let  $\delta_v$  be a singleton distribution in  $\gamma$ , and let  $A$  be a set of actions. When for all states  $l'$  that are reachable from  $l$  (via AMCAs of  $A$ ), no constraint of the form  $e.N == v$  needs to be evaluated to test the preconditions of actions in  $A$ , we call  $v$  *mergeable in  $l$* .

Algorithm 16 describes this merging procedure more formally. For each state  $l = (s, \gamma)$ ,

the algorithm checks whether a value  $v$  is mergeable, and if there is such a value, the merge is performed directly.

This concept is closely related to assumed density filtering, where certain assumptions are made on the form of the filtering distribution, and the true distribution is projected to a distribution that adheres to these assumptions at each time step. For example, the Boyen-Koller algorithm for inference in dynamic Bayesian networks [25] projects the true filtering distribution onto a factorized distribution at each time step. Here, instead of making independence assumptions, we assume *exchangeability* of the distribution. Additionally, we do not project to the exchangeable distribution at each time step, but only when we are certain that the exchangeability is not eliminated afterwards by splitting.

Finally, this merging concept can be seen as a generalization of the MERGE-DISJOINT algorithm described in Section 6.3: In that algorithm, the idea was to replace two representations  $\rho_1, \rho_2 \in \gamma$  by a joint, exchangeable representation  $\rho_r$ , when a set of states can be identified so that this is possible exactly. Instead, here we directly combine  $\rho_1$  and  $\rho_2$ , even when this induces an error on the underlying ground distribution.

### 6.5.3. Exploiting Temporal Structure

Finally, we discuss how mergeable values (Definition 21) can be identified. Suppose there is an action  $a$  that has a precondition that involves  $v$ , i.e. that is of the form  $(X == v) \wedge c_1 \wedge \dots \wedge c_n$ , where  $X$  is a property name and  $c_1, \dots, c_n$  are further constraints. A split on  $v$  is necessary if and only if all constraints  $c_1, \dots, c_n$  are satisfied – when one of the constraints is not satisfied, the complete precondition of  $a$  is not satisfied and the split on  $v$  does not need to be performed. Thus, we actually have to check (for each value  $v$  and each action  $a$  that involves  $v$ ) whether  $c_1 \wedge \dots \wedge c_n$  is satisfied eventually, for any state that is reachable from  $l$ . This question could be formalized in, for example, computation tree logic (CTL), so that model checking methods [42] could be applied.

Such an automatic analysis of the causal structure is a topic for future work. Here, we only performed a manual analysis of the actions of the kitchen scenario: We identified situations (described by constraints on lifted states) where specific *object names*  $v$  are mergeable (see Table 6.4). More details are provided in the description of the experimental evaluation below.

### 6.5.4. Experimental Evaluation

The main motivation for devising this merging approach was the fact that the lifted representation did not lead to a reduced representational complexity in the kitchen scenario (see Section 4.5), due to repeated splitting. Thus, in this section, we investigate the Kitchen scenario again, and evaluate whether applying merging finally allows us to achieve a substantially reduced representational complexity

**Experimental Design** Situations where merging could be applied for the kitchen scenario were identified manually, by examining the causal structure of the domain. The following three situations have been identified:

- After cooking has been finished, the pot and the wooden spoon are mergeable.
- Once the glass becomes empty (i.e. the agent drank from the glass for the last time), the glass is mergeable.
- When the agent has finished eating, the spoon and the plate are mergeable.

## 6. Lifted Marginal Filtering in Asymmetrical Models

Constraint	Mergeable values
$e.LastAction == \text{cook}$	Woden Spoon, Pot
$e_1.LastAction == \text{drink} \wedge e_2.Name == \text{Glass} \wedge e_2.Fill == \text{empty}$	Glass
$e_1.LastAction == \text{eat}$	Spoon, Plate

Table 6.4.: Situations where assumed density merging was applied for the Kitchen scenario. The table shows for each merge instant, the constraint that must be satisfied to perform this merge, and the value (of the *name* property) that is merged with the other object names.

These situations are described via constraints on the state, as shown in Table 6.4: For each state where one of the constraints is satisfied, the corresponding values of the *name* property are merged.

Similar to the evaluation in Section 4.5, we performed two experiments: Exact filtering (using the crisp action observations to keep exact filtering feasible), and approximate filtering (using the real sensor-based observations) by limiting the number of allowed states via pruning. Exact filtering was performed to assess the number of states that are required to represent the exact filtering distribution, and for approximate filtering, the activity recognition accuracy when using a fixed number of states to represent the distribution was assessed. For both cases, we compared the performance of filtering *with* merging and *without* merging, to evaluate the effect of the merging operation.

**Results** For exact filtering, Figure 6.13 (top) shows the required number of states over time for one of the subjects. When no merging is performed, there is no significant difference in the number of states that is required by the lifted and the ground representation: This fact was already observed in Section 4.5, and was the main motivation for devising this merging approach.

When merging *is* applied, the lifted representation is sometimes substantially smaller than the ground representation. The reason for this efficiency gain can be seen in Figure 6.13 (bottom), which shows the fraction of states in which each object was represented explicitly. When no merging is performed, the lifted representation degenerates over time due to repeated splitting. The merging procedure can prevent this degeneration: Whenever a specific object identity is not needed any more, it is discarded and thus, the lifted representation stays comparably compact.

Merging tends to reduce the number of required states specifically in those phases where the overall number of states is large. For example, in Figure 6.13, the difference between ground and lifted representation (with merging) is most pronounced when setting the table (timesteps 38 to 43) and when washing the dishes (from timestep 75 to 83) – which are exactly the phases where the overall number of states is large, as many permutations of object identities are possible.

Figure 6.14 shows the activity recognition accuracy, when performing approximate filtering. Recall that assumed density merging is an approximate procedure that manipulates the underlying ground distribution. Therefore, one could suspect that applying merging would have a negative effect on the activity recognition accuracy. However, merging allows to pre-

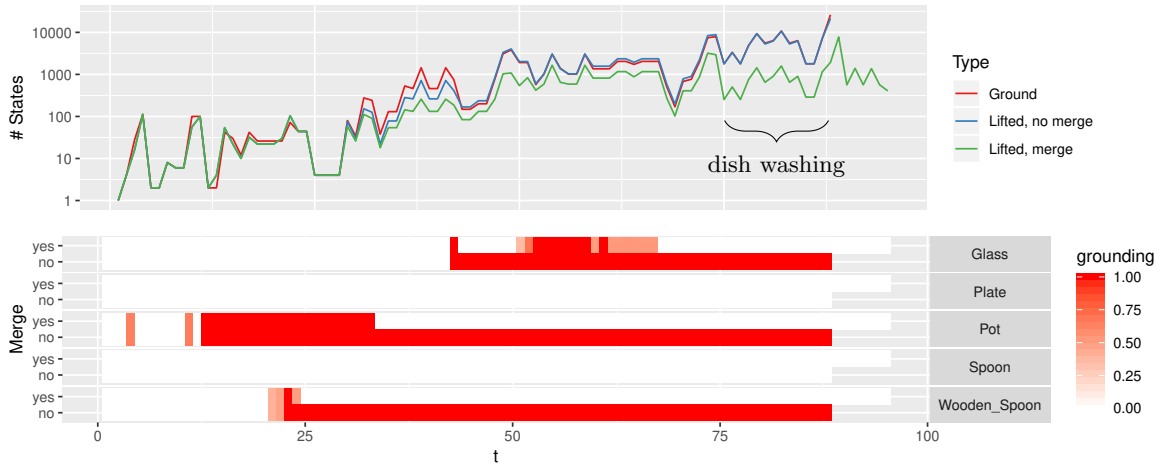


Figure 6.13.: Assumed density merging evaluation. Top: Number of states required for filtering in the kitchen scenario over time for subject 6, using ground and lifted states (with merging, and without merging). Bottom: Fraction of states in which each object identity is represented explicitly, when merging is applied, or not applied. When merging is applied, the lifted representation is sometimes substantially smaller than the ground representation, specifically between timestep 75 and 83, which corresponds to the phase where the dishes are washed and many permutations of objects are possible.

serve the lifted state representation, where intuitively, fewer states are necessary to represent the filtering distribution with a given accuracy. Thus, when the maximum number of states is fixed, merging can lead to a more accurate representation of the filtering distribution, and to a higher activity recognition accuracy, as shown in Figure 6.14.

Specifically, when 200 states are available, the accuracy of lifted filtering *with merging* is significantly higher than the accuracy of lifted filtering *without merging* ( $p < 0.05$ ,  $n = 70$  using Wilcoxon signed rank test). Additionally, lifted filtering (both with and without merging) leads to a significantly higher accuracy than ground filtering for 100, 200, 500 and 1000 available states.

**Conclusion & Future Work** In summary, the merging approach used here could effectively limit the representational complexity of the filtering distribution – even when using the crisp observation model, that amplifies the tendency of the model to become ground. Additionally, for a fixed number of available states, merging could increase the activity recognition accuracy, compared to the lifted model without merging. Thus, we could show that LiMa, together with a suitable merging strategy, can be usefully employed even for large models, with observations consisting of real sensor data, like the kitchen scenario presented here.

Here, we used handcrafted rules to identify appropriate time points for merging, to demonstrate the effect of merging when the situations for applying merging are chosen optimally. Of course, methods for automatically identifying these situations are required to make this merging approach more generally applicable without manual intervention, e.g. by employing model checking methods.

## 6. Lifted Marginal Filtering in Asymmetrical Models

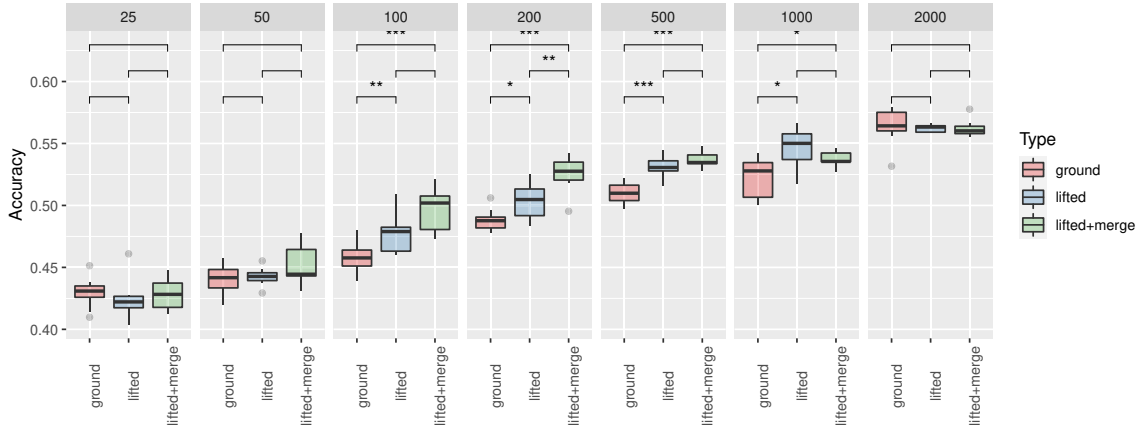


Figure 6.14.: Assumed density merging evaluation. Activity recognition accuracy for approximate filtering in the kitchen scenario. When between 100 or 1000 states are available, the accuracy of lifted filtering is significantly higher than the accuracy of ground filtering. The difference in accuracy between lifted filtering with and without merging is only significant for 200 states.

Algorithm	Merged dists.	Approx/exact	Assumptions
MERGE-SIMILAR	multinomial	approximate	all factors are multinomial
MERGE-DISJOINT	hypergeometric	exact	disjoint states, all factors are hypergeometric
MERGE-NORMALS	normal	approximate	one normal per entity, other factors are singletons
ADM	hypergeometric	approximate	other factors are singletons

Table 6.5.: Overview of the different merging algorithms presented in Chapter 6.

## 6.6. Conclusion & Future Work

In this chapter, we presented a number of different ideas for retaining a compact, lifted representation of the filtering distribution, to avoid complete grounding over time. Such methods are vitally important for efficiently applying lifted inference methods (like LiMa) to systems that are *not* perfectly symmetric – as most systems of realistic complexity, especially when the involve observations that stem from the physical world, like sensor data.

Table 6.5 summarizes the merging approaches discussed in this chapter. Each of the algorithms requires a specific parametric form of the factors that are manipulated and needs to make additional assumptions. These algorithms are only a few samples from the space of possible merging algorithms, showcasing considerations and challenges such algorithms are facing. Future work will need to focus on generalizing and unifying the existing methods, making them applicable to more cases. Important open questions and directions for future work are:

- Evaluating the merging algorithms described here with other models that are of realistic size and use real sensor data to obtain additional insight into the usefulness of each of



the algorithms under different conditions.

- Generalizing the methods: Each of the merging approaches only works under specific assumptions. While some of the assumptions might be crucial, others could be relaxed. For example, the MERGE-SIMILAR algorithm could be generalized such that the set  $G$  of merged states do not have to describe (approximately) the *same* ground distribution, but only have to factorize into (approximately) independent factors, conditional on  $G$ .
- Combining methods, to make them more generally applicable. For example, combining the MERGE-DISJOINT and MERGE-SIMILAR algorithms into a general merging approach for partially overlapping states would be of high practical relevance. Unfortunately, this is highly non-trivial, due to the interactions that arise when the context is allowed to contain distribution representations with different parametric forms: For example, the value distributions considered by MERGE-SIMILAR are generally non-uniform, but this means that the MERGE-DISJOINT algorithm cannot be applied directly any more.
- Making use of *variational* methods to derive merging algorithms: Variational methods construct an analytic approximation of the true posterior by assuming that the distribution factorizes into a number of independent factors. Such methods could be used in the context of merging, to find a lifted state (a representation of a distribution with a certain factorization structure) that approximates the true distribution over ground states.
- Devising merging algorithms that are based on methods for approximating arbitrary distributions over permutations by using non-commutative Fourier theory [87], described in more detail in Chapter 7.

At some point, all merging methods will have to resort to heuristics or approximation, as optimal merging for the general case is NP hard (as described in Section 6.3.3).



# Discussion & Conclusion

CHAPTER SUMMARY *Finally, in this chapter, we summarize the results achieved so far, and discuss directions for future research.*

*Possible future research directions include theoretical statements about lifted filtering, symmetry-preserving observation models, using more elaborate methods to represent distributions in the context, and employing concepts from noncommutative Fourier theory to allow low-frequency approximations of distributions over multisets.*

## Contents

---

7.1. Summary . . . . .	134
7.2. Discussion: Why Lifting Works Here . . . . .	135
7.3. Future Work . . . . .	136

---

### 7.1. Summary

In this thesis, we devised a lifted Bayesian filtering algorithm for multi-entity systems, i.e. systems with MRS dynamics, that exploits symmetries that naturally arise in MRSs. In the following, we summarize our main contributions.

**MRSs and the Need for Efficient Inference** This thesis was motivated by inference in dynamic *multi-entity systems*, a task that arises, for example, in the context of multi-agent human activity recognition. Multiset rewriting systems (MRSs) allow to specify such multi-entity systems in a natural way, and for this reason are used ubiquitously in the modeling and simulation community. However, the advantages of MRSs come at a price: Like other symbolic, declarative AI methods [118], MRSs typically generate very large, discrete state spaces (that have a high number of parameters), where inference quickly becomes infeasible.

Fortunately, however, the state space of MRSs naturally exhibits a certain *symmetry*: When computing posterior states, they only need to reason about the number of entities of each species in the state, instead of distinguishing them individually. Thus, they allow to gracefully transition between an individual-based representation and a representation by counts. The goal of this thesis was to devise novel inference algorithms for MRSs that make use of this property.

**Lifted Marginal Filtering** The main contribution of this thesis is an efficient Bayesian filtering algorithm for systems with MRS dynamics. We started by showing how (ground) Bayesian filtering can be performed for MRS (which is a contribution in itself, as previously, MRS were only used for simulation studies).

Then, we derived a more efficient representation for distributions over multisets. This was achieved by introducing a suitable decomposition of multisets  $x$  into pairs  $(s, \mathbf{v})$ , where  $s$  is a multiset structure and  $\mathbf{v}$  is a value tuple. This made it possible to use standard mechanisms to represent the distribution over  $\mathbf{v}$  more efficiently: Independence could be exploited by maintaining the distribution in factorized form, and exchangeability could be exploited by representing the distribution by sufficient statistics. This procedure, which is an instance of Rao-Blackwellization, directly lead to the concept of *lifted state*: A pair of structure  $s$ , and a representation  $\gamma$  of a distribution over  $\mathbf{v}$ . A lifted state represents a distribution over ground states that all have the same structure and values distributed according to  $\gamma$ .

Next, we showed how Bayesian filtering can be performed directly on this representation. We made use of the fact that the structures  $s$  are still multisets, so that multiset rewriting can (in principle) still be applied. Whenever a constraint is not determined in a lifted state, because it is satisfied for some part of the support of the lifted state and not satisfied for some other part, a split (conceptually similar to splitting in lifted inference) is required.

Even without further modifications or improvements, we could show that this lifted Bayesian filtering algorithm can sometimes be orders of magnitude more efficient (in terms of number of maintained states and thus runtime) than ground filtering.

The algorithm directly lends itself to an approximate version, by limiting the maximum number of maintained states. In the approximate case, the lifted algorithm could achieve a lower approximation error and lower variance of the estimate than ground filtering.

**Approximating the System Dynamics Using MCMC** Even though the algorithm worked well for some models, we identified two scalability issues that severely limit the usefulness of the basic algorithm for real-world applications.

The first problem is the fact that the number of parallel actions (compound actions) can grow exponentially with respect to the number of entities in each state. Therefore, even for a moderate number of entities, the number of compound actions can become excessively large, so that complete enumeration (as done by the exact algorithm) is infeasible. We proposed a Markov chain Monte Carlo (MCMC) algorithm that samples compound actions. The proposal works by backtracking in the search tree of the exact algorithm, and then sampling a completion of the remaining, non-maximal compound action (which is much simpler to find than constructing a maximal compound action from scratch). Using this algorithm, maximally parallel multiset rewriting can be performed for systems with thousands of entities, where the exact algorithm is infeasible.

**Lifted Marginal Filtering for Asymmetrical Models** The other problem that prevents the direct application of the algorithm to real-world domains is the fact that symmetry breaks require repeated splitting until the representation degenerates to the ground form. We introduced *merging* to overcome this problem – identifying subsets of lifted states that can be represented by a single lifted state. As this problem is hard in general, we provided merging algorithms for a number of relevant special cases: For multinomial value distributions, we considered the case where lifted states can be merged because they represent approximately the same ground distribution. The intuition is that due to the stochasticity of the process, the distributions described by two lifted states will become more similar over time, increasing the chance of finding states that are sufficiently similar to merge them.

For hypergeometric value distributions, we considered the orthogonal case, where a set of disjoint states can be merged because the states complement each other, so that they can be replaced by a single lifted state. We also considered the case of Gaussian value distributions, where different entities in a state could be merged by Gaussian mixture reduction methods.

Finally, we explored opportunities for merging that arise from knowledge about the causal structure of the system: When a certain property never needs to be distinguished in the future by any of the preconditions, this property can be forgotten. Using these merging methods, the lifted filtering algorithm required a substantially lower number of states than ground filtering even in a real-world application scenario.

## 7.2. Discussion: Why Lifting Works Here

*Symmetry-breaking evidence* is viewed as one of the major issues that prevents the application of lifted inference to real-world domains [219, 104, 233, 29]. Even if the model is symmetric, the symmetry breaks easily when asymmetric evidence about the random variables are obtained, so that lifted inference coincides with standard, propositional inference. It has been formally shown that exact inference in a weighted first-order knowledge base becomes intractable when evidence on binary atoms is present [90].

Somewhat surprisingly though, we could still retain a lifted representation for the human activity recognition (HAR) scenarios considered in this thesis. For example, in the tracking scenario (where agents move between multiple locations and are observed by presence sensors), splitting was never required, although observations were obtained at each time step.

## 7. Discussion & Conclusion

Where does the discrepancy between this empirical observation and the fact that evidence can be highly problematic in lifted inference come from? To get an intuition, consider collective classification in the WebKB dataset [44] as an example of a typical lifted inference application domain. The dataset consists of websites from computer science departments, containing information about the words that appear on each website, and links between them. The collective classification task is to predict the page labels, given information about words on each page and the link structure. Evidence on the words that appear on each page makes all pages distinct, so that the symmetry in the model breaks. Exact lifted inference is intractable in this case.

In contrast, in the tracking scenario, the observations are of a different kind: The presence sensors just observe that there is *any* agent at the corresponding location, instead of observing a specific agent. More formally, the observation likelihood depends on *constraints* of the state. For example, when the presence sensor at location  $A$  is active, the likelihood depends on the constraint  $e.Loc == A$ .

A model of the tracking scenario in a relational graphical model (e.g. a parfactor graph) might contain a par-RV  $\text{at-Location}(L, N)$  with logical variables  $L$  (describing the location) and  $N$  (describing the name). Then, the likelihood of the presence sensor observation would depend on the constraint “there is an  $n \in N$  so that  $\text{at-Location}(a, n)$  is true”. Thus, the observations in the Tracking model do not correspond to evidence on ground atoms, but to constraints on the first-order terms.

In summary, the problem of symmetry-breaking evidence seems to be less prevalent in HAR as compared to domains like link prediction, collective classification, or entity resolution, which makes lifted inference-based methods well suited to this domain. Of course, HAR is not completely free from this problem, as we saw in the kitchen scenario, where it was necessary to distinguish the different items in the kitchen. Merging can alleviate this problem for a number of cases, shown in Chapter 6. Additionally, ideas for directly *preventing* splitting due to symmetry-breaking evidence (instead of recovering the lifted representation later on, as done by merging) are presented below.

### 7.3. Future Work

**Algorithmic Extensions** The lifted marginal filtering (LiMa) algorithm directly lends itself to a number of algorithmic extensions, that are well-established for ground Bayesian filtering, e.g. for hidden Markov models (HMMs) or computational state-space models (CSSMs).

- **Smoothing and MAP inference:** In this thesis, we only considered filtering, i.e. computing  $p(x_t | y_{1:t})$ , but not smoothing (computing  $p(x_t | y_{1:T})$  for  $t < T$ ) and maximum a-posteriori (MAP) inference (i.e. computing  $\arg\max_{x_{1:T}} p(x_{1:T} | y_{1:T})$ ). Extending LiMa to these tasks is straightforward, by appropriately adapting the forward-backward algorithm to compute the smoothing distribution, and the Viterbi algorithm to obtain the MAP sequence. Unfortunately, for smoothing or MAP inference, asymmetries at time  $t + \delta$  can propagate back to time  $t$  (i.e. a split at time  $t + \delta$  can require splitting at time  $t$ ). Thus, appropriate merging strategies – or other methods for avoiding grounding, as discussed below – are even more relevant for smoothing and MAP inference.
- **Durative actions:** In LiMa, the transition model has a discrete-time Markov chain

semantics. Thus, the durations of all (compound) actions implicitly follow a geometric distribution – the probability that the (compound) action at time  $t$  continues at time  $t + 1$  does not depend on the duration of that action. In real-world domains, however, action durations do not necessarily follow a geometric distribution. Thus, to better describe reality, it is necessary to explicitly model the duration distribution of actions (as done in some CSSMs [123]), so that the underlying formalism becomes a semi-Markov model. This is complicated by the fact that LiMa is based on a *maximally parallel* MRS, so that the question arises how parallel actions and semi-Markov semantics can be combined appropriately.

- **Parameter and structure learning:** So far, we have only been concerned with inference, but not with learning. In contrast to sub-symbolic models (e.g. neural networks), symbolic models, as considered here, can be constructed directly based on prior domain knowledge, so that learning is not of outmost relevance. Still, parameter learning (e.g. learning the weights of actions) via expectation maximization is an interesting topic for future work. Another challenging goal is structure learning of LiMa models, e.g. learning the (compound) actions from state trajectories or even observation sequences. Process mining [226] might provide first ideas for this task. Another interesting direction is to derive possible actions from semi-structured data (e.g. textual descriptions of the domain), as demonstrated by Yordanova for the case of PDDL actions [238].

**A Theory of Lifted Filtering** For relational probabilistic models, a detailed theory is available that describes under which conditions (lifted) inference is tractable. The formal results are based on the notion of domain-lifted algorithms [217]: An algorithm is domain-lifted for a problem class, if and only if for all instances of this problem class, inference is polynomial in the domain size of the logical variables. Jaeger and Van den Broeck [90] showed that inference in relational models is intractable in general, but domain-lifted algorithms exist when the model adheres to certain constraints.

A similar theory is not yet available for LiMa models, or more generally, Bayesian filtering tasks with MRS dynamics. LiMa is based on a completely different, computational specification of the probabilistic model, so that applying existing results is not straightforward. Therefore, a possible topic for future work is to derive such formal results: Which properties of the model (e.g. of the multiset rewriting rules) guarantee tractable inference?

**Neural Observation Models** In this thesis, we focused mainly on a formalism for the system model  $p(X_t | X_{t-1})$  and an efficient representation of the filtering distribution  $p(X_t | y_{1:t})$ . For the observation model  $p(Y_t | X_t)$ , we only considered hand-crafted models or simple parametric distributions (like a conditional normal distribution in the Kitchen scenario). However, accurate, flexible observation models are required to achieve state-of-the-art prediction performance (e.g. in terms of activity recognition accuracy). Furthermore, opposed to the system model that can often be constructed from prior domain knowledge, the observation model typically needs to be estimated from training data.

Therefore, an immediate direction for future work is to extend LiMa by powerful methods to learn observation models, i.e. density estimators that can model the complex relationship between system states and observations. *Neural density estimators*, like the (real-valued) neural autoregressive density estimator [216] or the masked autoregressive flow density estimator [168], are promising methods for this purpose. They make use of recent advances

## 7. Discussion & Conclusion

in (deep) neural networks to allow tractable density estimation in high-dimensional data. Tractable probabilistic graphical model, like sum-product networks [176] (that also support continuous distributions [148]) are another promising class of methods for this purpose. This research could eventually lead to a *hybrid* method that uses recent advances in deep architectures to learn complex, high-dimensional observation models, and the symbolic methods developed in this thesis to model the system dynamics based on domain knowledge.

**Symmetry-aware Observation Models** One of the recurring themes throughout this thesis has been that asymmetry in the transition or observation model can lead to a complete grounding of the filtering distribution, and thus intractable inference. Specifically, asymmetric evidence can be highly problematic for lifted inference, as discussed in Section 7.2. We approached this problem by *merging* methods that restore a lifted representation of the distribution. However, merging can simply come too late when the distribution is already completely ground.

An alternative idea is to avoid splitting due to asymmetric evidence in the first place, by approximating the evidence by more “symmetrical” evidence, so that fewer splits are required. Specifically, Van den Broeck and Darwiche propose to approximate evidence by a low-rank Boolean matrix factorization [219].

It is not obvious how these methods could be employed for LiMa, that does not use relational graphical models to specify the probabilistic model. Additionally, the observations in most interesting, real-world domains that we are concerned with are typically continuous rather than boolean. For example, HAR tasks like the kitchen scenario typically involve sensor data from wearables or environmental sensors. For the continuous case, methods that avoid observation splits have not been proposed so far.

To achieve this goal, one could exploit the fact that there is a clear separation between observations and hidden states in Bayesian filtering models like LiMa. This should allow us to handle asymmetrical observations already in the observation model, so that the asymmetry does not propagate to the lifted representation over hidden states. Intuitively, this could be achieved by regularizing the observation model with the (known) symmetries from the system model. For example, posterior regularization [69] allows estimation of probabilistic models subject to additional constraints (that represent the symmetries).

**Other Distribution Representations** In this thesis, we made use of *Rao-Blackwellization* – the idea that some of the RVs can be handled analytically during filtering, while others need to be represented explicitly. This was done by introducing the notion of *representations* of distributions – specifically, of the factors of the value distribution  $p(\mathbf{v} \mid s, \gamma)$ . Here, we considered representations by *parameters* of parametric distributions, and by sufficient statistics (for exchangeable factors).

In general, other, more flexible methods for representing distributions could be employed, as long as they provide tractable marginal inference (which is necessary for testing constraints and performing splitting), and are closed under conditioning (to ensure that split results can still be represented). Most notably, tractable probabilistic models like sum-product networks [176] satisfy these conditions. They even support continuous distributions by using piecewise polynomials as leaf distributions [148]. Another option for this purpose are exchangeable variable models (EVMs) [158]: Interestingly, the *context*  $\gamma$  in LiMa can be seen as a specific instance of an EVM. This would also provide us with methods for *learning* distribution



representations from training data.

**Distributions over Multiplicities** Even in the lifted state representation, the multiplicities of entities are still represented explicitly, so that when different numbers of entities are possible, all possible multiplicities need to be enumerated. In some scenarios, this can contribute substantially to the overall number of states, e.g. in the predator-prey scenario in Chapter 5.

A future research direction is to investigate how distributions over multiplicities could be represented more efficiently, e.g. parametrically, or by other methods to represent distributions over counts, like Poisson sum-product networks [147]. The challenge here is the computation of (compound) actions – the algorithms presented in this thesis require exact multiplicities instead of distributions. Either, situations where compound actions can be computed on the parametric level could be identified, or approximation strategies, e.g. appropriate sampling schemes, need to be devised.

**Noncommutative Fourier Theory** An important special case of distributions that arose in many of the application scenarios we considered are distributions over permutations. For this case, specialized methods [88] have been developed that allow efficient Bayesian filtering in state spaces of permutations. They utilize a Fourier transform over the symmetric group (the group of permutations) to approximate a distribution over permutations by its low-frequency components. As a first step, using these methods to represent factors of the value distribution where permutations are involved could lead to substantial efficiency gains for those cases.

Additionally, algebraic structures are ubiquitous in MRSs: For example, compound actions with the effect composition operator form a group, and multisets with multiset union are a monoid. From here, the following research question naturally arises: Can ideas from noncommutative Fourier theory be used to devise an efficient Bayesian filtering algorithm for systems with MRS dynamics, e.g. by providing low-frequency approximations of distributions over compound actions or multiset states?



# Bibliography

- [1] B. Ahmadi, K. Kersting, and F. Hadiji. Lifted belief propagation: Pairwise marginals and beyond. In *Proceedings of the 5th European Workshop on Probabilistic Graphical Models*, pages 9–16, 2010. (Cited on page 169)
- [2] B. Ahmadi, K. Kersting, M. Mladenov, and S. Natarajan. Exploiting symmetries for scaling loopy belief propagation and relational training. *Machine Learning*, 92(1):91–132, 2013. (Cited on pages 166 and 169)
- [3] B. Ahmadi, K. Kersting, and S. Natarajan. MapReduce lifting for belief propagation. In *AAAI Workshop - Technical Report*, volume WS-13-16, pages 2–7, 2013. (Cited on page 169)
- [4] B. Ahmadi, M. Mladenov, K. Kersting, and S. Sanner. On lifted pagerank, kalman filter and towards lifted linear program solving. In *Technical Report of the Symposium "Lernen, Wissen, Adaptivitat - Learning, Knowledge, and Adaptivity 2011" of the GI Special Interest Groups KDML, IR and WM*, pages 35–42, 2011. (Cited on page 169)
- [5] A. Anand, A. Grover, Mausam, and P. Singla. Contextual symmetries in probabilistic graphical models. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, 2016. (Cited on page 169)
- [6] U. Apsel and R. Brafman. Extended Lifted Inference with Joint Formulas. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, UAI’11, pages 11–18, Barcelona, Spain, 2011. AUAI Press. (Cited on pages 29 and 169)
- [7] U. Apsel, K. Kersting, and M. Mladenov. Lifting relational MAP-LPs using cluster signatures. In *AAAI Workshop - Technical Report*, volume WS-14-13, pages 2–8, 2014. (Cited on page 169)
- [8] D. Arnaud, N. de Freitas, and Neil Gordon. *Sequential Monte Carlo Methods in Practice*. Springer-Verlag New York, 2001. (Cited on pages 13 and 14)
- [9] R. Barbuti, F. Levi, P. Milazzo, and G. Scatena. Maximally Parallel Probabilistic Semantics for Multiset Rewriting. *Fundamenta Informaticae*, 112(1):1–17, 2011. (Cited on pages 1, 19, 23, 46, 47, and 169)
- [10] R. Barbuti, F. Levi, P. Milazzo, and G. Scatena. Probabilistic model checking of biological systems with uncertain kinetic rates. *Theoretical Computer Science*, 419:2 – 16, 2012. (Cited on pages 18 and 169)

## Bibliography

- [11] M. Baum and U. Hanebeck. Association-free tracking of two closely spaced targets. In *Proceedings of the IEEE Conference on Multisensor Fusion and Integration for Intelligent Systems*, pages 62–67. IEEE, 2010. (Cited on page 169)
- [12] M. Baum and U. Hanebeck. Using symmetric state transformations for multi-target tracking. In *Proceedings of the 14th International Conference on Information Fusion*, pages 1–8. IEEE, 2011. (Cited on page 169)
- [13] M. Baum and U. Hanebeck. The kernel-sme filter for multiple target tracking. In *Proceedings of the 16th International Conference on Information Fusion*, pages 288–295. IEEE, 2013. (Cited on page 169)
- [14] M. Baum, P. Ruoff, D. Itte, and U. Hanebeck. Optimal Point Estimates for Multi-target States based on Kernel Distances. In *Proceedings of the 51st IEEE Conference on Decision and Control*, Maui, Hawaii, USA, December 2012. (Cited on page 169)
- [15] M. Baum, P. Willett, and U. Hanebeck. MMOSPA-based track extraction in the PHD filter—a justification for k-means clustering. In *Proceedings of the IEEE 53rd Annual Conference on Decision and Control*, pages 1816–1821. IEEE, 2014. (Cited on page 169)
- [16] P. Beame, G. Van den Broeck, E. Gribkoff, and D. Suciu. Symmetric Weighted First-Order Model Counting. In *Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 313–328, 2015. (Cited on pages 29 and 169)
- [17] V. Belle, A. Passerini, and G. Van den Broeck. Probabilistic inference in hybrid domains by weighted model integration. In *Proceedings of 24th International Joint Conference on Artificial Intelligence*, pages 2770–2776, 2015. (Cited on pages 31 and 169)
- [18] V. Belle, G. Van den Broeck, and A. Passerini. Hashing-based approximate probabilistic inference in hybrid domains. In *Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 141–150, 2015. (Cited on page 169)
- [19] V. Belle, G. Van den Broeck, and A. Passerini. Component Caching in Hybrid Domains with Piecewise Polynomial Densities. In *AAAI*, pages 3369–3375, 2016. (Cited on page 169)
- [20] Thomas Bengtsson, Peter Bickel, Bo Li, et al. Curse-of-dimensionality revisited: Collapse of the particle filter in very large scale systems. In *Probability and statistics: Essays in honor of David A. Freedman*, pages 316–334. Institute of Mathematical Statistics, 2008. (Cited on page 14)
- [21] Eli Bingham, Jonathan P Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D Goodman. Pyro: Deep universal probabilistic programming. *The Journal of Machine Learning Research*, 20(1):973–978, 2019. (Cited on page 165)
- [22] S. Bistarelli, I. Cervesato, G. Lenzini, R. Marangoni, and F. Martinelli. On representing biological systems through multiset rewriting. In *International Conference on Computer Aided Systems Theory*, pages 415–426. Springer, 2003. (Cited on page 169)

- [23] Wayne D Blizard et al. Multiset theory. *Notre Dame Journal of formal logic*, 30(1):36–66, 1988. (Cited on page 18)
- [24] C. Boutilier, R. Reiter, and B. Price. Symbolic dynamic programming for first-order MDPs. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, volume 1, pages 690–700, 2001. (Cited on page 166)
- [25] Xavier Boyen and Daphne Koller. Tractable inference for complex stochastic processes. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 33–42, 1998. (Cited on pages 13, 106, and 127)
- [26] Tanya Braun and Ralf Möller. Lifted junction tree algorithm. In *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, pages 30–42. Springer, 2016. (Cited on pages 29 and 163)
- [27] Peter Buchholz. Exact and ordinary lumpability in finite markov chains. *Journal of applied probability*, pages 59–75, 1994. (Cited on page 164)
- [28] ST Buckland, KB Newman, L Thomas, and NB Koesters. State-space models for the dynamics of wild animal populations. *Ecological modelling*, 171(1-2):157–175, 2004. (Cited on page 14)
- [29] H. Bui, T. Huynh, and R. De Braz. Exact lifted inference with distinct soft evidence on every object. In *Proceedings of the National Conference on Artificial Intelligence*, volume 3, pages 1875–1881, 2012. (Cited on pages 135 and 169)
- [30] H. Bui, T. Huynh, and S. Riedel. Automorphism groups of graphical models and lifted variational inference. In *Uncertainty in Artificial Intelligence - Proceedings of the 29th Conference*, pages 132–141, 2013. (Cited on pages 30 and 169)
- [31] H. Bui, T. Huynh, and D. Sontag. Lifted tree-reweighted variational inference. In *Uncertainty in Artificial Intelligence - Proceedings of the 30th Conference*, pages 92–101, 2014. (Cited on page 169)
- [32] W. Buntine. Operations for learning with graphical models. *Journal of artificial intelligence research*, 2:159–225, 1994. (Cited on page 10)
- [33] Luca Cardelli, Mirco Tribastone, Max Tschaikowski, and Andrea Vandin. Syntactic markovian bisimulation for chemical reaction networks. In *Models, Algorithms, Logics and Tools*, pages 466–483. Springer, 2017. (Cited on page 164)
- [34] Carlos M Carvalho and Hedibert F Lopes. Simulation-based sequential analysis of markov switching stochastic volatility models. *Computational Statistics & Data Analysis*, 51(9):4526–4542, 2007. (Cited on page 14)
- [35] Iliano Cervesato, Nancy A Durgin, Patrick D Lincoln, John C Mitchell, and Andre Scedrov. A meta-notation for protocol analysis. In *Computer Security Foundations Workshop, 1999. Proceedings of the 12th IEEE*, pages 55–69. IEEE, 1999. (Cited on page 17)
- [36] Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6-7):772–799, 2008. (Cited on page 47)

## Bibliography

- [37] J. Choi and E. Amir. Lifted Relational Variational Inference. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*, UAI'12, pages 196–206, Catalina Island, CA, 2012. AUAI Press. (Cited on page 169)
- [38] J. Choi, E. Amir, and D. Hill. Lifted Inference for Relational Continuous Models. In *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, UAI'10, pages 126–134, Catalina Island, CA, 2010. AUAI Press. (Cited on pages 32, 34, and 169)
- [39] J. Choi, E. Amir, T. Xu, and A. Valocchi. Learning Relational Kalman Filtering. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 2539–2546, 2015. (Cited on pages 32, 35, 96, and 169)
- [40] J. Choi, R. de Salvo Braz, and H. Bui. Efficient Methods for Lifted Inference with Aggregate Factors. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011. (Cited on page 169)
- [41] J. Choi, A. Guzman-Rivera, and E. Amir. Lifted Relational Kalman Filtering. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, pages 2092–2099, 2011. (Cited on pages 32, 34, and 169)
- [42] Edmund M Clarke Jr, Orna Grumberg, Daniel Kroening, Doron Peled, and Helmut Veith. *Model checking*. MIT press, 2018. (Cited on page 127)
- [43] M.C. Cooper, S. de Givry, M. Sanchez, T. Schiex, M. Zytnicki, and T. Werner. Soft arc consistency revisited. *Artificial Intelligence*, 174:449–478, 2010. (Cited on page 172)
- [44] Mark Craven and Seán Slattery. Relational learning with statistical predicate invention: Better models for hypertext. *Machine Learning*, 43(1-2):97–119, 2001. (Cited on page 136)
- [45] Nilesh Dalvi, Karl Schnaitter, and Dan Suciu. Computing query probability with incidence algebras. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 203–214. ACM, 2010. (Cited on page 169)
- [46] Nilesh Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. *The International Journal on Very Large Data Bases*, 16(4):523–544, 2007. (Cited on page 169)
- [47] Vincent Danos, Jérôme Feret, Walter Fontana, Russell Harmer, and Jean Krivine. Rule-based modelling of cellular signalling. In *International conference on concurrency theory*, pages 17–41. Springer, 2007. (Cited on pages 17 and 42)
- [48] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17(1):229–264, 2002. (Cited on page 165)
- [49] M. Das, Y. Wu, T. Khot, K. Kersting, and S. Natarajan. Scaling lifted probabilistic inference and learning via graph databases. In *Proceedings of the 16th SIAM International Conference on Data Mining 2016*, pages 738–746, 2016. (Cited on page 169)

- [50] L. De Raedt, K. Kersting, S. Natarajan, and D. Poole. Statistical relational artificial intelligence: Logic, probability, and computation. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 10(2):1–189, 2016. (Cited on page 22)
- [51] R. de Salvo Braz, E. Amir, and D. Roth. Lifted first-order probabilistic inference. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 1319–1325, 2005. (Cited on pages 23, 29, 71, 169, and 179)
- [52] R. de Salvo Braz, E. Amir, and D. Roth. MPE and partial inversion in lifted probabilistic variable elimination. In *Proceedings of the National Conference on Artificial Intelligence*, volume 2, pages 1123–1130, 2006. (Cited on page 169)
- [53] R. de Salvo Braz, S. Natarajan, H. Bui, J. Shavlik, and S. Russell. Anytime lifted belief propagation. In *International Workshop on Statistical Relational Learning*, volume 9, 2009. (Cited on page 169)
- [54] T. Dean and R. Givan. Model minimization in Markov decision processes. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Conference on Innovative Applications of Artificial Intelligence*, pages 106–111, 1997. (Cited on page 166)
- [55] Pierre Del Moral. Non linear filtering: Interacting particle solution. *Markov Processes and Related Fields*, 2:555–580, 03 1996. (Cited on page 2)
- [56] Salem Derisavi, Holger Hermanns, and William H Sanders. Optimal state-space lumping in markov chains. *Information Processing Letters*, 87(6):309–315, 2003. (Cited on page 164)
- [57] Persi Diaconis and David Freedman. De finetti’s generalizations of exchangeability. *Studies in inductive logic and probability*, 2:233–249, 1980. (Cited on page 57)
- [58] P. Domingos and W. Webb. A tractable first-order probabilistic logic. In *Proceedings of the National Conference on Artificial Intelligence*, volume 3, pages 1902–1909, 2012. (Cited on page 169)
- [59] A. Doucet, N. De Freitas, K. Murphy, and S. Russell. Rao-Blackwellised particle filtering for dynamic Bayesian networks. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 176–183. Morgan Kaufmann Publishers Inc., 2000. (Cited on pages 14, 34, and 58)
- [60] Arnaud Doucet. On sequential simulation-based methods for bayesian filtering. Technical report, CUED/F-INFENG/TR 310, Department of Engineering, Cambridge University., 1998. (Cited on page 14)
- [61] IL Dryden and KV Mardia. *Statistical analysis of shape*. Wiley, 1998. (Cited on page 103)
- [62] Maximilian Dylla, Iris Miliaraki, and Martin Theobald. Top-k query processing in probabilistic databases with non-materialized views. In *29th International Conference on Data Engineering*, pages 122–133. IEEE, 2013. (Cited on page 169)

## Bibliography

- [63] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996. (Cited on pages 100 and 104)
- [64] James R Faeder, Michael L Blinov, and William S Hlavacek. Rule-based modeling of biochemical systems with bionetgen. In *Systems biology*, pages 113–167. Springer, 2009. (Cited on pages 17 and 42)
- [65] Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. Inference and learning in probabilistic logic programs using weighted boolean formulas. *Theory and Practice of Logic Programming*, 15(3):358–401, 2015. (Cited on page 165)
- [66] Peter A Flach and Nicolas Lachiche. Decomposing probability distributions on structured individuals. In *ILP Work-in-progress reports*, 2000. (Cited on page 52)
- [67] Maria Fox and Derek Long. Pddl2.1: An extension to pddl for expressing temporal planning domains. *Journal of artificial intelligence research*, 20:61–124, 2003. (Cited on page 15)
- [68] Keinosuke Fukunaga. *Introduction to statistical pattern recognition*. Elsevier, 2013. (Cited on page 122)
- [69] Kuzman Ganchev, João Graça, Jennifer Gillenwater, and Ben Taskar. Posterior regularization for structured latent variable models. *Journal of Machine Learning Research*, 11(67):2001–2049, 2010. (Cited on page 138)
- [70] Timon Gehr, Sasa Misailovic, and Martin Vechev. Psi: Exact symbolic inference for probabilistic programs. In *International Conference on Computer Aided Verification*, pages 62–83. Springer, 2016. (Cited on page 165)
- [71] Marcel Gehrke, Tanya Braun, and Ralf Möller. Lifted Dynamic Junction Tree Algorithm. In *Proceedings of the International Conference on Conceptual Structures*, pages 55–69. Springer, 2018. (Cited on page 163)
- [72] Marcel Gehrke, Tanya Braun, and Ralf Möller. Taming Reasoning in Temporal Probabilistic Relational Models. In *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI 2020)*, 2020. (Cited on page 96)
- [73] T. Geier and S. Biundo. Approximate online inference for dynamic markov logic networks. In *23rd IEEE International Conference on Tools with Artificial Intelligence*, pages 764–768. IEEE, 2011. (Cited on page 169)
- [74] L. Getoor and B. Taskar. *Introduction to Statistical Relational Learning*. MIT press, 2007. (Cited on page 166)
- [75] Daniel T Gillespie. Exact stochastic simulation of coupled chemical reactions. *The journal of physical chemistry*, 81(25):2340–2361, 1977. (Cited on page 18)
- [76] R. Givan, T. Dean, and M. Greig. Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, 147(1-2):163–223, July 2003. (Cited on page 166)



- [77] V. Gogate and P. Domingos. Exploiting logical structure in lifted probabilistic inference. In *AAAI Workshop - Technical Report*, volume WS-10-06, pages 19–25, 2010. (Cited on page 169)
- [78] V. Gogate, A. Jha, and D. Venugopal. Advances in Lifted Importance Sampling. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012. (Cited on page 169)
- [79] Vibhav Gogate and Pedro Domingos. Probabilistic Theorem Proving. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence*, pages 256–265. AUAI Press, 2011. (Cited on pages 23, 29, 62, and 169)
- [80] Noah Goodman, Vikash Mansinghka, Daniel M Roy, Keith Bonawitz, and Joshua Tenenbaum. Church: a language for generative models with non-parametric memoization and approximate inference. In *Uncertainty in Artificial Intelligence*, 2008. (Cited on page 165)
- [81] Neil J Gordon, David J Salmond, and Adrian FM Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. In *IEE proceedings F (radar and signal processing)*, volume 140, pages 107–113. IET, 1993. (Cited on pages 14 and 16)
- [82] F. Hadiji, B. Ahmadi, and K. Kersting. Efficient sequential clamping for lifted message passing. In *Lecture Notes in Computer Science*, volume 7006 LNAI, pages 122–133, 2011. (Cited on page 169)
- [83] F. Hadiji and K. Kersting. Reduce and re-lift: Bootstrapped lifted likelihood maximization for MAP. In *AAAI Workshop - Technical Report*, volume WS-13-16, pages 8–14, 2013. (Cited on page 169)
- [84] U. Hanebeck and M. Baum. Association-free direct filtering of multi-target random finite sets with set distance measures. In *18th International Conference on Information Fusion*, pages 1367–1374. IEEE, 2015. (Cited on page 169)
- [85] S. Hölldobler and O. Skvortsova. A logic-based approach to dynamic programming. In *Proceedings of the Workshop on “Learning and Planning in Markov Processes—Advances and Challenges” at the Nineteenth National Conference on Artificial Intelligence*, pages 31–36, 2004. (Cited on page 166)
- [86] R. Holte and G. Fan. State Space Abstraction in Artificial Intelligence and Operations Research. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015. (Cited on page 24)
- [87] J. Huang, C. Guestrin, and L. Guibas. Efficient inference for distributions on permutations. In *Advances in Neural Information Processing Systems*, 2009. (Cited on pages 35, 131, and 169)
- [88] J. Huang, C. Guestrin, and L. Guibas. Fourier Theoretic Probabilistic Inference over Permutations. *Journal of Machine Learning Research*, 10:997–1070, June 2009. (Cited on pages 23, 32, 35, 139, and 169)

## Bibliography

- [89] J. Huang, C. Guestrin, X. Jiang, and L. Guibas. Exploiting Probabilistic Independence for Permutations. In *International Conference on Artificial Intelligence and Statistics*, pages 248–255, 2009. (Cited on pages 35 and 169)
- [90] M. Jaeger and G. Van den Broeck. Liftability of Probabilistic Inference: Upper and Lower Bounds. In *Proceedings of StarAI*, 2012. (Cited on pages 135 and 137)
- [91] S. Jagabathula and D. Shah. Inferring rankings using constrained sensing. *IEEE Transactions on Information Theory*, 57(11):7288–7306, 2011. (Cited on pages 35 and 169)
- [92] A. Jaimovich, O. Meshi, and N. Friedman. Template based inference in symmetric relational Markov random fields. In *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, 2007. (Cited on page 169)
- [93] A. Jha, V. Gogate, A. Meliou, and D. Suciu. Lifted inference seen from the other side: The tractable features. In *24th Annual Conference on Neural Information Processing Systems*, 2010. (Cited on pages 29 and 169)
- [94] Abhay Jha and Dan Suciu. Probabilistic databases with MarkoViews. *Proceedings of the VLDB Endowment*, 5(11):1160–1171, 2012. (Cited on page 169)
- [95] X. Jiang, J. Huang, and L. Guibas. Fourier-information duality in the identity management problem. In *Lecture Notes in Computer Science*, volume 6912 LNAI, pages 97–113, 2011. (Cited on pages 36 and 169)
- [96] Mathias John, Cédric Lhoussaine, Joachim Niehren, and Cristian Versari. Biochemical reaction rules with constraints. In *European symposium on programming*, pages 338–357. Springer, 2011. (Cited on pages 17 and 42)
- [97] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960. (Cited on pages 2 and 13)
- [98] B. Kang and K. Kim. Exploiting symmetries for single- and multi-agent Partially Observable Stochastic Domains. *Artificial Intelligence*, 182–183:32 – 57, 2012. (Cited on page 166)
- [99] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972. (Cited on page 113)
- [100] Martin Kasparick and Frank Krüger. Probabilistic action selection - tracking multiple persons in indoor environments. [http://dx.doi.org/10.18453/rosdok\\_id000000114](http://dx.doi.org/10.18453/rosdok_id000000114), 2013. (Cited on page 76)
- [101] S. Kazemi, A. Kimmig, G. Van den Broeck, and D. Poole. New liftable classes for first-order probabilistic inference. In *Advances in Neural Information Processing Systems*, pages 3117–3125, 2016. (Cited on page 169)
- [102] S. Kazemi, A. Kimmig, G. Van Den Broeck, and D. Poole. Domain Recursion for Lifted Inference with Existential Quantifiers. In *arXiv Preprint arXiv:1707.07763*, 2017. (Cited on page 169)

- [103] S. Kazemi and D. Poole. Elimination ordering in lifted first-order probabilistic inference. In *Proceedings of the National Conference on Artificial Intelligence*, volume 2, pages 863–870, 2014. (Cited on page 169)
- [104] K. Kersting. Lifted Probabilistic Inference. In *Proceedings of the 20th European Conference on Artificial Intelligence*, volume 242, pages 33–38. IOS Press, 2012. (Cited on pages 23, 28, 96, and 135)
- [105] K. Kersting, B. Ahmadi, and S. Natarajan. Counting belief propagation. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence*, pages 277–284, 2009. (Cited on pages 30, 57, and 169)
- [106] K. Kersting, Y. El Massaoudi, F. Hadji, and B. Ahmadi. Informed Lifting for Message-Passing. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010. (Cited on page 169)
- [107] K. Kersting, M. Van Otterlo, and L. De Raedt. Bellman goes relational. In *Proceedings of the Twenty-First International Conference on Machine Learning*, page 59. ACM, 2004. (Cited on page 166)
- [108] Kristian Kersting, Martin Mladenov, Roman Garnett, and Martin Grohe. Power iterated color refinement. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014. (Cited on page 96)
- [109] R. Khardon and S. Sanner. Stochastic planning and lifted inference. *arXiv preprint*, arXiv:1701.01048, 2017. (Cited on pages 166 and 167)
- [110] C. Kiddon and P. Domingos. Leveraging ontologies for lifted probabilistic inference and learning. In *AAAI Workshop - Technical Report*, volume WS-10-06, pages 40–45, 2010. (Cited on page 169)
- [111] C. Kiddon and P. Domingos. Coarse-to-fine inference and learning for first-order probabilistic models. In *Proceedings of the National Conference on Artificial Intelligence*, volume 2, pages 1049–1056, 2011. (Cited on page 169)
- [112] Doga Kisa, Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. Probabilistic sentential decision diagrams. In *Fourteenth International Conference on the Principles of Knowledge Representation and Reasoning*, 2014. (Cited on page 165)
- [113] J. Kisiński and D. Poole. Constraint processing in lifted probabilistic inference. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence*, pages 293–302, 2009. (Cited on page 169)
- [114] J. Kisiński and D. Poole. Lifted aggregation in directed first-order probabilistic models. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence*, pages 1922–1929, 2009. (Cited on page 169)
- [115] Barbara Kitchenham. Procedures for Performing Systematic Reviews. In *Keele University Technical Report TR/SE-0401*. 2004. (Cited on page 22)
- [116] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT press, 2009. (Cited on page 8)

- [117] R. Kondor, A. Howard, and T. Jebara. Multi-object tracking with representations of the symmetric group. *Journal of Machine Learning Research*, 2:211–218, 2007. (Cited on pages 35 and 169)
- [118] Parisa Kordjamshidi, Dan Roth, and Kristian Kersting. Systems ai: A declarative learning based programming perspective. In *IJCAI*, pages 5464–5471, 2018. (Cited on page 134)
- [119] E. Krishnamurthy, V. Murthy, and V. Krishnamurthy. Biologically inspired rule-based multiset programming paradigm for soft-computing. In *Proceedings of the 1st Conference on Computing Frontiers*, pages 140–149, 2004. (Cited on page 169)
- [120] Frank Krüger. *Activity, Context, and Plan Recognition with Computational Causal Behaviour Models*. PhD thesis, University of Rostock, 2016. (Cited on pages 15 and 188)
- [121] Frank Krüger, Albert Hein, Kristina Yordanova, and Thomas Kirste. Recognising the actions during cooking task (Cooking task dataset), 2015. (Cited on page 76)
- [122] Frank Krüger, Martin Kasparick, Thomas Mundt, and Thomas Kirste. Where are my colleagues and why? Tracking multiple persons in indoor environments. In *10th International Conference on Intelligent Environments (IE), 2014*, Shanghai, China, July 2014. (Cited on page 76)
- [123] Frank Krüger, Martin Nyolt, Kristina Yordanova, Albert Hein, and Thomas Kirste. Computational State Space Models for Activity and Intention Recognition. A Feasibility Study. *PLOS ONE*, 9(11):e109381, November 2014. (Cited on pages 15, 75, 76, 77, 137, and 189)
- [124] M. Kwiatkowska, G. Norman, and D. Parker. Symmetry reduction for probabilistic model checking. In *Lecture Notes in Computer Science*, volume 4144 LNCS, pages 234–248, 2006. (Cited on page 23)
- [125] Steffen L Lauritzen and David J Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society: Series B (Methodological)*, 50(2):157–194, 1988. (Cited on page 163)
- [126] W. Leven and A. Lanterman. Multiple target tracking with symmetric measurement equations using unscented Kalman and particle filters. In *Proceedings of the Thirty-Sixth Southeastern Symposium on System Theory*, pages 195–199. IEEE, 2004. (Cited on page 169)
- [127] W. Leven and A. Lanterman. Unscented Kalman Filters for Multiple Target Tracking With Symmetric Measurement Equations. *IEEE Transactions on Automatic Control*, 54(2):370–375, February 2009. (Cited on page 169)
- [128] David A Levin and Yuval Peres. *Markov chains and mixing times*, volume 107. American Mathematical Soc., 2017. (Cited on pages 87, 88, and 92)
- [129] Stefan Lüdtke, Marcel Gehrke, Tanya Braun, Ralf Möller, and Thomas Kirste. Lifted marginal filtering for asymmetric models by clustering-based merging. In *Proceedings of the 24th European Conference on Artificial Intelligence*. IOS Press, 2020. (Cited on page 93)

- [130] Stefan Lüdtke and Thomas Kirste. Lifted bayesian filtering in multiset rewriting systems. *Journal of Artificial Intelligence Research*, 2020. accepted. (Cited on pages 7 and 39)
- [131] Stefan Lüdtke, Alejandro Molina, Kristian Kersting, and Thomas Kirste. Gaussian lifted marginal filtering. In *KI 2019: Advances in Artificial Intelligence*, pages 230–243. Springer, 2019. (Cited on page 93)
- [132] Stefan Lüdtke, Max Schröder, Sebastian Bader, Kristian Kersting, and Thomas Kirste. Lifted Filtering via Exchangeable Decomposition. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, 2018. (Cited on page 39)
- [133] Stefan Lüdtke, Max Schröder, and Thomas Kirste. Approximate probabilistic parallel multiset rewriting using MCMC. In *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, pages 73–85. Springer, 2018. (Cited on pages 39 and 85)
- [134] Stefan Lüdtke, Max Schröder, Frank Krüger, Sebastian Bader, and Thomas Kirste. State-space abstractions for probabilistic inference: a systematic review. *Journal of Artificial Intelligence Research*, 63:789–848, 2018. (Cited on pages 7 and 21)
- [135] Stefan Lüdtke, Max Schröder, Frank Krüger, and Thomas Kirste. Where are my colleagues? Tracking and Counting Multiple Persons using Lifted Marginal Filtering. In *Proceedings of the 4th International Workshop on Sensor-Based Activity Recognition and Interaction*, 2017. (Cited on page 75)
- [136] Stefan Lüdtke, Kristina Yordanova, and Thomas Kirste. Human activity and context recognition using lifted marginal filtering. In *IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 83–88, 2019. (Cited on pages 39 and 93)
- [137] R. Mahler. Multitarget Bayes filtering via first-order multitarget moments. *IEEE Transactions on Aerospace and Electronic systems*, 39(4):1152–1178, 2003. (Cited on page 169)
- [138] C. Maus, S. Rybacki, and A. Uhrmacher. Rule-based multi-level modeling of cell biological systems. *BMC Systems Biology*, 5(1):166, 2011. (Cited on page 169)
- [139] Andrew McCallum, Karl Schultz, and Sameer Singh. Factorie: Probabilistic programming via imperatively defined factor graphs. In *Advances in Neural Information Processing Systems*, pages 1249–1257, 2009. (Cited on page 165)
- [140] W. Meert, G. Van Den Broeck, and A. Darwiche. Lifted Inference for Probabilistic Logic Programs. In *Workshop on Probabilistic Logic Programming*, 2014. (Cited on page 169)
- [141] B. Milch, L. Zettlemoyer, K. Kersting, M. Haimes, and L. Kaelbling. Lifted probabilistic inference with counting formulas. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, pages 1062–1068, 2008. (Cited on pages 23, 29, and 169)

## Bibliography

- [142] H. Mittal, P. Goyal, V. Gogate, and P. Singla. New rules for domain independent lifted MAP inference. In *Advances in Neural Information Processing Systems*, pages 649–657, 2014. (Cited on page 169)
- [143] M. Mladenov, B. Ahmadi, and K. Kersting. Lifted Linear Programming. In *International Conference on Artificial Intelligence and Statistics*, pages 788–797, 2012. (Cited on page 169)
- [144] M. Mladenov, A. Globerson, and K. Kersting. Efficient lifting of MAP LP relaxations using k-locality. *Journal of Machine Learning Research*, 33:623–632, 2014. (Cited on page 169)
- [145] M. Mladenov, A. Globerson, and K. Kersting. Lifted Message Passing as Reparametrization of Graphical Models. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, pages 603–612, 2014. (Cited on page 169)
- [146] M. Mladenov and K. Kersting. Lifted inference via k-locality. In *AAAI Workshop - Technical Report*, volume WS-13-16, pages 25–30, 2013. (Cited on page 169)
- [147] Alejandro Molina, Sriraam Natarajan, and Kristian Kersting. Poisson sum-product networks: A deep architecture for tractable multivariate poisson distributions. In *AAAI*, pages 2357–2363, 2017. (Cited on page 139)
- [148] Alejandro Molina, Antonio Vergari, Nicola Di Mauro, Sriraam Natarajan, Floriana Esposito, and Kristian Kersting. Mixed sum-product networks: A deep architecture for hybrid domains. In *Thirty-second AAAI conference on artificial intelligence*, 2018. (Cited on page 138)
- [149] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012. (Cited on pages 11, 12, and 13)
- [150] Kevin Patrick Murphy and Stuart Russell. *Dynamic bayesian networks: representation, inference and learning*. PhD thesis, University of California, Berkeley Berkeley, CA, 2002. (Cited on page 163)
- [151] A. Nath and P. Domingos. Efficient Belief Propagation for Utility Maximization and Repeated Inference. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, volume 4, page 3, 2010. (Cited on page 169)
- [152] A. Nath and P. Domingos. Efficient lifting for online probabilistic inference. In *AAAI Workshop - Technical Report*, volume WS-10-06, pages 64–69, 2010. (Cited on page 169)
- [153] K. Ng and J. Lloyd. Probabilistic reasoning in a classical logic. *Journal of Applied Logic*, 7(2):218–238, 2009. (Cited on page 169)
- [154] K. Ng, J. Lloyd, and W. Uther. Probabilistic modelling, inference and learning using logical theories. *Annals of Mathematics and Artificial Intelligence*, 54(1-3):159–205, 2008. (Cited on page 169)

- [155] M. Niepert. Markov chains on orbits of permutation groups. In *Uncertainty in Artificial Intelligence - Proceedings of the 28th Conference*, pages 624–633, 2012. (Cited on pages 23, 30, and 169)
- [156] M. Niepert. Symmetry-aware marginal density estimation. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence*, pages 725–731, 2013. (Cited on page 169)
- [157] Mathias Niepert. Lifted probabilistic inference: An mcmc perspective. In *Proceedings of the 2nd International Workshop on Statistical Relational AI*, 2012. (Cited on page 164)
- [158] Mathias Niepert and Pedro Domingos. Exchangeable variable models. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 271–279, 2014. (Cited on pages 2 and 138)
- [159] Mathias Niepert and Guy Van den Broeck. Tractability through exchangeability: A new perspective on efficient probabilistic inference. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 2467–2475, 2014. (Cited on page 11)
- [160] D. Nitti, T. De Laet, and L. De Raedt. A particle filter for hybrid relational domains. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2764–2771. IEEE, November 2013. (Cited on pages 32, 33, and 169)
- [161] D. Nitti, T. De Laet, and L. De Raedt. Relational object tracking and learning. In *IEEE International Conference on Robotics and Automation*, pages 935–942, 2014. (Cited on pages 32, 33, and 169)
- [162] D. Nitti, T. De Laet, and L. De Raedt. Probabilistic logic programming for hybrid relational domains. *Machine Learning*, 103(3):1–43, 2016. (Cited on pages 32, 33, and 169)
- [163] Martin Nyolt and Thomas Kirste. On Resampling for Bayesian Filters in Discrete State Spaces. In *Proceedings of the 27th International Conference on Tools with Artificial Intelligence*, pages 526–533, Vietri sul Mare, Italy, November 2015. IEEE Computer Society. (Cited on pages 17, 71, 72, and 80)
- [164] Martin Nyolt, Frank Krüger, Kristina Yordanova, Albert Hein, and Thomas Kirste. Marginal filtering in large state spaces. *International Journal of Approximate Reasoning*, 61:16–32, June 2015. (Cited on pages 2, 16, 17, and 78)
- [165] Kenji Okuma, Ali Taleghani, Nando De Freitas, James J Little, and David G Lowe. A boosted particle filter: Multitarget detection and tracking. In *European conference on computer vision*, pages 28–39. Springer, 2004. (Cited on page 14)
- [166] N. Oury and G. Plotkin. Multi-level modelling via stochastic multi-level multiset rewriting. *Mathematical Structures in Computer Science*, 23(02):471–503, April 2013. (Cited on page 169)
- [167] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., 1982. (Cited on page 103)

## Bibliography

- [168] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pages 2338–2347, 2017. (Cited on page 137)
- [169] Matthew Parker and Alex Kamenev. Extinction in the Lotka-Volterra model. *Physical Review E*, 80(2), August 2009. (Cited on page 89)
- [170] Gheorghe Paun. *Membrane Computing: An Introduction*. Springer Science & Business Media, 2012. (Cited on page 19)
- [171] Dario Pescini, Daniela Besozzi, Giancarlo Mauri, and Claudio Zandron. Dynamical probabilistic P systems. *International Journal of Foundations of Computer Science*, 17(01):183–204, 2006. (Cited on page 17)
- [172] Avi Pfeffer. Figaro: An object-oriented probabilistic programming language. *Charles River Analytics Technical Report*, 137:96, 2009. (Cited on page 165)
- [173] D. Poole. First-order probabilistic inference. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 985–991, 2003. (Cited on pages 10, 11, 22, 23, 25, 28, 29, 57, and 169)
- [174] D. Poole, F. Bacchus, and J. Kisynski. Towards completely lifted search-based probabilistic inference. *arXiv preprint*, arXiv:1107.4035, 2011. (Cited on pages 29 and 169)
- [175] H. Poon, P. Domingos, and M. Sumner. A General Method for Reducing the Complexity of Relational Inference and its Application to MCMC. In *Proceedings of the 23rd National Conference on Artificial Intelligence*, volume 8, pages 1075–1080, 2008. (Cited on page 169)
- [176] Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, UAI’11, page 337–346, Arlington, Virginia, USA, 2011. AUAI Press. (Cited on pages 138 and 165)
- [177] M Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014. (Cited on page 166)
- [178] Miquel Ramírez and Hector Geffner. Goal recognition over POMDPs: Inferring the intention of a POMDP agent. In *Proceedings of the 22nd IJCAI*, pages 2009–2014. AAAI Press, 2011. (Cited on page 16)
- [179] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62(1-2 SPEC. ISS.):107–136, 2006. (Cited on page 8)
- [180] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine learning*, 62(1-2):107–136, 2006. (Cited on pages 22 and 25)
- [181] Andrew R Runnalls. Kullback-leibler approach to gaussian mixture reduction. *IEEE Transactions on Aerospace and Electronic Systems*, 43(3), 2007. (Cited on pages 96, 97, 120, and 121)



- [182] Stuart Russell and Peter Norvig. Artificial intelligence: a modern approach. 2002. (Cited on page 8)
- [183] Adam Sadilek and Henry Kautz. Location-based Reasoning about Complex Multi-Agent Behavior. *Journal of Artificial Intelligence Research*, 43(1):87–133, January 2012. (Cited on page 16)
- [184] S. Sanner and E. Abbasnejad. Symbolic Variable Elimination for Discrete and Continuous Graphical Models. In *AAAI*, 2012. (Cited on pages 31 and 169)
- [185] S. Sanner and C. Boutilier. Practical solution techniques for first-order MDPs. *Artificial Intelligence*, 173(5-6):748–788, April 2009. (Cited on page 166)
- [186] S. Sarkhel and V. Gogate. Lifting WALKSAT-based local search algorithms for MAP inference. In *AAAI Workshop - Technical Report*, volume WS-13-16, pages 64–67, 2013. (Cited on page 169)
- [187] S. Sarkhel, D. Venugopal, P. Singla, and V. Gogate. Lifted MAP Inference for Markov Logic Networks. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, pages 859–867, 2014. (Cited on page 169)
- [188] Simo Särkkä. *Bayesian Filtering and Smoothing*, volume 3. Cambridge University Press, 2013. (Cited on page 1)
- [189] Taisuke Sato and Yoshitaka Kameya. New advances in logic-based probabilistic modeling by prism. In *Probabilistic inductive logic programming*, pages 118–155. Springer, 2008. (Cited on page 165)
- [190] Thomas Schiex, Helene Fargier, and Gerard Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1995. (Cited on page 172)
- [191] Max Schröder, Stefan Lüdtke, Sebastian Bader, Frank Krüger, and Thomas Kirste. An Office Scenario Dataset for Benchmarking Observation-equivalent Entities, December 2016. (Cited on page 75)
- [192] Max Schröder, Stefan Lüdtke, Sebastian Bader, Frank Krüger, and Thomas Kirste. Abstracting from Observation-equivalent Entities in Human Behavior Modeling. In *AAAI Workshop: Plan, Activity, and Intent Recognition*, February 2017. (Cited on page 75)
- [193] Dominic Schuhmacher, Ba-Tuong Vo, and Ba-Ngu Vo. A consistent metric for performance evaluation of multi-object filters. *IEEE transactions on signal processing*, 56(8):3447–3457, 2008. (Cited on page 103)
- [194] B. Schumitsch, S. Thrun, G. Bradski, and K. Olukotun. The information-form data association filter. In *NIPS*, pages 1193–1200, 2005. (Cited on pages 32, 35, and 169)
- [195] Luca Scrucca, Michael Fop, T Brendan Murphy, and Adrian E Raftery. mclust 5: Clustering, classification and density estimation using gaussian finite mixture models. *The R journal*, 8(1):289, 2016. (Cited on pages 123 and 190)

## Bibliography

- [196] P. Sen, A. Deshpande, and L. Getoor. Exploiting shared correlations in probabilistic databases. *Proceedings of the VLDB Endowment*, 1(1):809–820, 2008. (Cited on page 169)
- [197] P. Sen, A. Deshpande, and L. Getoor. Bisimulation-based approximate lifted inference. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 496–505. AUAI Press, 2009. (Cited on page 169)
- [198] P. Shenoy and J. West. Inference in hybrid Bayesian networks using mixtures of polynomials. *International Journal of Approximate Reasoning*, 52(5):641–657, July 2011. (Cited on pages 31 and 169)
- [199] Kyle Siegrist. Random: Probability, mathematical statistics, stochastic processes, 2020. Accessed: 2020-10-01. (Cited on pages 65 and 66)
- [200] P. Singla and P. Domingos. Lifted first-order belief propagation. In *Proceedings of the National Conference on Artificial Intelligence*, volume 2, pages 1094–1099, 2008. (Cited on pages 23 and 169)
- [201] P. Singla, A. Nath, and P. Domingos. Approximate Lifted Belief Propagation. In *AAAI Workshop - Technical Report*, pages 92–97, 2010. (Cited on page 169)
- [202] P. Singla, A. Nath, and P. Domingos. Approximate Lifting Techniques for Belief Propagation. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 2497–2504, 2014. (Cited on pages 31 and 169)
- [203] Parag Singla, Aniruddh Nath, and Pedro M Domingos. Approximate lifting techniques for belief propagation. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014. (Cited on page 96)
- [204] Harold W Sorenson and Daniel L Alspach. Recursive bayesian estimation using gaussian sums. *Automatica*, 7(4):465–479, 1971. (Cited on page 13)
- [205] R. L. Stratonovich. Conditional markov processes. *Theory of Probability & Its Applications*, 5(2):156–178, 1960. (Cited on page 1)
- [206] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. Probabilistic Databases. *Synthesis Lectures on Data Management*, 3(2):1–180, May 2011. (Cited on page 29)
- [207] N. Taghipour, J. Davis, and H. Blockeel. Generalized counting for lifted variable elimination. In *23rd International Conference on Inductive Logic Programming*, volume 8812, pages 107–122, 2014. (Cited on pages 11, 29, 163, and 169)
- [208] N. Taghipour, D. Fierens, J. Davis, and H. Blockeel. Lifted variable elimination with arbitrary constraints. *Journal of Machine Learning Research*, 22:1194–1202, 2012. (Cited on page 169)
- [209] N. Taghipour, D. Fierens, J. Davis, and H. Blockeel. Lifted variable elimination: Decoupling the operators from the constraint language. *Journal of Artificial Intelligence Research*, 47:393–439, 2013. (Cited on page 169)

- [210] N. Taghipour, D. Fierens, G. Van Den Broeck, J. Davis, and H. Blockeel. Completeness results for lifted variable elimination. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*, 2013. (Cited on pages 29 and 169)
- [211] N. Taghipour, D. Fierens, G. Van Den Broeck, J. Davis, and H. Blockeel. On the completeness of lifted variable elimination. In *AAAI Workshop - Technical Report*, volume WS-13-16, pages 74–80, 2013. (Cited on page 169)
- [212] F. Takiyama and F. Cozman. Inference with Aggregation Parfactors: Lifted Elimination with First-Order d-Separation. In *Proceedings of the Brazilian Conference on Intelligent Systems*, pages 384–389, October 2014. (Cited on page 169)
- [213] I. Thon, N. Landwehr, and L. De Raedt. Stochastic relational processes: Efficient inference and applications. *Machine Learning*, 82(2):239–272, February 2011. (Cited on pages 119, 122, and 124)
- [214] Sebastian Thrun, Michael Montemerlo, Daphne Koller, Ben Wegbreit, Juan Nieto, and Eduardo Nebot. Fastslam: An efficient solution to the simultaneous localization and mapping problem with unknown data association. *Journal of Machine Learning Research*, 4(3):380–407, 2004. (Cited on pages 14 and 15)
- [215] M. Toussaint and A. Storkey. Probabilistic inference for solving discrete and continuous state Markov Decision Processes. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 945–952. ACM, 2006. (Cited on page 166)
- [216] Benigno Uria, Iain Murray, and Hugo Larochelle. Rnade: The real-valued neural autoregressive density-estimator. In *Advances in Neural Information Processing Systems*, pages 2175–2183, 2013. (Cited on page 137)
- [217] G. Van Den Broeck. On the completeness of first-order knowledge compilation for lifted probabilistic inference. In *25th Annual Conference on Neural Information Processing Systems*, 2011. (Cited on pages 137 and 169)
- [218] G. Van Den Broeck, A. Choi, and A. Darwiche. Lifted relax, compensate and then recover: From approximate to exact lifted probabilistic inference. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*, pages 131–141, 2012. (Cited on page 169)
- [219] G. Van Den Broeck and A. Darwiche. On the complexity and approximation of binary evidence in lifted inference. In *Advances in Neural Information Processing Systems*, pages 2868–2876, 2013. (Cited on pages 135, 138, and 169)
- [220] G. Van Den Broeck and J. Davis. Conditioning in first-order knowledge compilation and lifted probabilistic inference. In *Proceedings of the National Conference on Artificial Intelligence*, volume 3, pages 1961–1967, 2012. (Cited on page 169)
- [221] G. Van Den Broeck, W. Meert, and A. Darwiche. Skolemization for weighted first-order model counting. In *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning*, 2014. (Cited on page 169)

## Bibliography

- [222] G. Van Den Broeck and M. Niepert. Lifted probabilistic inference for asymmetric graphical models. In *Proceedings of the National Conference on Artificial Intelligence*, volume 5, pages 3599–3605, 2015. (Cited on pages 31 and 169)
- [223] G. Van Den Broeck, N. Taghipour, W. Meert, J. Davis, and L. De Raedt. Lifted probabilistic inference by first-order knowledge compilation. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, pages 2178–2185, 2011. (Cited on pages 29, 165, and 169)
- [224] Guy Van den Broeck. *Lifted inference and learning in statistical relational models*. PhD thesis, PhD thesis, KU Leuven, 2013. (Cited on page 27)
- [225] Guy Van den Broeck and Adnan Darwiche. On the complexity and approximation of binary evidence in lifted inference. In *Advances in Neural Information Processing Systems*, pages 2868–2876, 2013. (Cited on page 96)
- [226] Wil Van Der Aalst. Process mining. *Communications of the ACM*, 55(8):76–83, 2012. (Cited on page 137)
- [227] J.a Van Haaren, G.a b Van den Broeck, W.a Meert, and J.a Davis. Lifted generative learning of Markov logic networks. *Machine Learning*, 103(1):27–55, 2016. (Cited on page 166)
- [228] D. Venugopal and V. Gogate. On lifting the Gibbs sampling algorithm. In *Advances in Neural Information Processing Systems*, volume 3, pages 1655–1663, 2012. (Cited on pages 30 and 169)
- [229] D. Venugopal and V. Gogate. Evidence-based clustering for scalable inference in Markov logic. In *Lecture Notes in Computer Science*, volume 8726 LNAI, pages 258–273, 2014. (Cited on pages 31 and 169)
- [230] D. Venugopal and V. Gogate. Scaling-up importance sampling for Markov logic networks. In *Advances in Neural Information Processing*, volume 4, pages 2978–2986, 2014. (Cited on page 169)
- [231] D. Venugopal, S. Sarkhel, and K. Cherry. Non-parametric domain approximation for scalable Gibbs sampling in MLNs. In *32nd Conference on Uncertainty in Artificial Intelligence*, pages 745–754, 2016. (Cited on page 169)
- [232] D. Venugopal, S. Sarkhel, and V. Gogate. Just count the satisfied groundings: Scalable local-search and sampling based inference in MLNs. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, volume 5, pages 3606–3612, 2015. (Cited on page 169)
- [233] Deepak Venugopal and Vibhav Gogate. Evidence-based clustering for scalable inference in markov logic. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 258–273. Springer, 2014. (Cited on pages 96 and 135)
- [234] J. Vlasselaer, A. Kimmig, A. Dries, W. Meert, and L. De Raedt. Knowledge Compilation and Weighted Model Counting for Inference in Probabilistic Logic Programs. In *Workshops at the Thirtieth AAAI Conference on Artificial Intelligence*, 2016. (Cited on page 169)

- [235] C. Wang, S. Joshi, and R. Khardon. First order decision diagrams for relational MDPs. *Journal of Artificial Intelligence Research*, 31:431–472, 2008. (Cited on page 166)
- [236] T. Warnke, T. Helms, and A. Uhrmacher. Syntax and Semantics of a Multi-Level Modeling Language. In *Proceedings of the 3rd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, pages 133–144. ACM Press, 2015. (Cited on page 169)
- [237] Frank Wood, Jan Willem Meent, and Vikash Mansinghka. A new approach to probabilistic programming inference. In *Artificial Intelligence and Statistics*, pages 1024–1032, 2014. (Cited on page 165)
- [238] Kristina Yordanova. Extracting planning operators from instructional texts for behaviour interpretation. In *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, pages 215–228. Springer, 2018. (Cited on page 137)
- [239] Lei Yu, Tianyu Yang, and Antoni B Chan. Density-preserving hierarchical em algorithm: simplifying gaussian mixture models for approximate inference. *IEEE transactions on pattern analysis and machine intelligence*, 41(6):1323–1337, 2018. (Cited on page 96)
- [240] Shun-Zheng Yu. Hidden semi-markov models. *Artificial intelligence*, 174(2):215–243, 2010. (Cited on page 77)
- [241] Zeynep Yucel, Francesco Zanlungo, Claudio Feliciani, Adrien Gregorj, and Takayuki Kanda. Identification of social relation within pedestrian dyads. *PloS one*, 14(10):e0223656, 2019. (Cited on page 99)
- [242] L. Zettlemoyer, H. Pasula, and L. Kaelbling. Logical particle filtering. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2008. (Cited on pages 32 and 169)
- [243] Kai Zhang and James T Kwok. Simplifying mixture models through function approximation. In *Advances in Neural Information Processing Systems*, pages 1577–1584, 2007. (Cited on page 96)
- [244] Nevin L. Zhang and David Poole. A simple approach to Bayesian network computations. In *Proceedings of the Tenth Canadian Conference on Artificial Intelligence*, 1994. (Cited on page 9)





## 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** The type of a map (partial function) taking elements from  $X$  and mapping to elements from  $Y$  is denoted as  $X \rightarrow Y$ . Furthermore,  $\text{dom}$  denotes the domain of a map (the set of all elements  $x \in X$  such that  $f(x)$  is defined),  $\text{ran}$  denotes the range (codomain) of the map (i.e.  $Y$  in the previous example), and  $\text{img}$  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  $m(x_1) = y_1, \dots, m(x_i) = y_i$  is denoted as  $m = \langle x_1 : y_1, \dots, x_i : y_i \rangle$ . To distinguish *entities* from other maps, we use the notation  $e.K$  to denote the value corresponding to the key  $K$  of entity  $e$ .

Two maps can be combined by the operator  $\odot$ , which is defined as:

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

Note that this operator is not commutative: When a key  $k$  exists in both  $m_1$  and  $m_2$ , the map  $m_1 \odot m_2$  contains the value  $m_2(k)$ . We use  $m \setminus k$  to denote removing the key-value pair with key  $k$  from the map  $m$ .

It is sometimes convenient to have another map combination operator  $\oplus$  at hand that directly computes the sum of the elements in the range of the map (requiring that a sum is defined for those elements):

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

Similarly, an operator  $\ominus$  can be defined, that computes the difference of elements in the range of the maps.

## A. Notation

**Sequences** Sequences are maps of indices to elements. The expression  $\text{seq } X$  denotes the set of finite sequences over  $X$ . We use  $\langle a_1, \dots, a_n \rangle$  as shorthand notation for  $\langle 1 : a_1, \dots, n : a_n \rangle$ . The concatenation of two sequences  $s_1$  and  $s_2$  is denoted by  $s_1 \odot s_2$ , i.e.  $\langle a_1, \dots, a_i \rangle \odot \langle a_{i+1}, \dots, a_j \rangle = \langle a_1, \dots, a_j \rangle$ . 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.  $\langle a_1, \dots, a_i, \dots, a_n \rangle_i = a_i$ . We use  $a \in s$  to denote that an element  $a$  is occurring in the sequence  $s$ . We use  $s \setminus a_i$  to denote removing element  $a_i$  from the sequence  $s$ , i.e.  $\langle a_1, \dots, a_{i-1}, a_i, a_{i+1}, \dots, a_n \rangle \setminus a_i = \langle a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_n \rangle$ .

**Multisets** Multisets are maps from elements to natural numbers (multiplicities). The expression  $\text{mset } X$  denotes the set of multisets over  $X$ . We write  $\llbracket n_1 a_1, \dots, n_i a_i \rrbracket$  to denote the multiset  $\langle a_1 : n_1, \dots, a_i : n_i \rangle$ . The expression  $x \# a$  denotes the multiplicity of element  $a$  in  $x$ , i.e.  $x \# a = x(a)$ . We use  $x_1 \uplus x_2$  to denote multiset union, and  $x_1 \Downarrow x_2$  to denote multiset difference, i.e.  $x_1 \uplus x_2 = x_1 \oplus x_2$  and  $x_1 \Downarrow x_2 = x_1 \ominus x_2$  (we use different symbols for clarity). The function  $\text{items}(s)$  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.  $\text{items}(s) = \biguplus_{a \in s} a$ .

Symbol	Description
$\text{dom}$	domain
$\text{ran}$	codomain
$\text{img}$	image
$\text{seq } X$	Set of finite sequences over $X$
$\odot$	For maps: map combination; for sequences: concatenation
$\setminus$	For sets: set difference; for maps: removing key-value pairs from a map; for sequences: removing an element from a sequence
$\oplus$	Map combination with summation of values
$\ominus$	Map combination with subtraction of value
$\text{mset } X$	Set of multiset over $X$
$\uplus$	Multiset union
$\Downarrow$	Multiset difference
$\#$	Multiplicity of an element in a multiset
$\mathbb{1}$	Indicator function

Table A.1.: List of notation symbols.



## More Related Work

In the following, we present additional work related to LiMa that was not discussed in the literature survey (Chapter 3).

### B.1. The Lifted (Dynamic) Junction Tree Algorithm

Using the variable elimination algorithm to answer multiple queries on the same graphical model is unnecessarily inefficient, as some intermediate results need to be computed repeatedly for each query. The idea of the *junction tree algorithm* [125] is to perform these computations a priori and store the results in a suitable form, so that queries can be answered efficiently afterwards. Intuitively, the nodes of the junction tree represent subsets of the complete model that share close relations, and information is propagated between the junction tree nodes so that at the end, each node has all information it needs to compute marginals of the random variables it contains. Braun and Möller [26] introduced a lifted version of the junction tree algorithm by introducing first-order junction trees and using first-order variable elimination [207] as the underlying inference algorithm.

The interface algorithm [150] is an inference algorithm for dynamic Bayesian networks (DBNs). It constructs a junction tree from the DBN and performs inference on this junction tree. Furthermore, it exploits the fact that only the random variables with outgoing edges from time  $t - 1$  are required for computing the belief states at time  $t$ . The lifted dynamic junction tree (LDJT) algorithm [71] combines the lifted junction tree algorithm with the interface algorithm to perform efficient inference in dynamic relational models.

The LDJT algorithm is closely related to LiMa: Both algorithm perform inference in dynamic systems, and exploit symmetries in the underlying distribution to increase efficiency. However, they are using completely different formalisms to specify the dynamic system: The LDJT algorithm uses relational graphical models to specify the dynamic system, whereas LiMa uses a symbolic, rule-based description (specifically, a MRS). The LDJT algorithm (and conventional lifted inference algorithms in general) cannot be applied directly to MRSs, which was the reason for devising LiMa in the first place.

## B.2. Lumpability and Syntactic Markovian Bisimulation

The concept of *lumpability* describes under which conditions states in a Markov chain can be aggregated to obtain an equivalent Markov chain with fewer states. A Markov chain with transition model  $p$  is called *lumpable* with respect to a partition  $T$  of states, when for each subset  $T_i$  and  $T_j$  in  $T$  and each states  $x_i, x'_i \in T_i$ ,

$$\sum_{x_j \in T_j} p(x_j | x_i) = \sum_{x_j \in T_j} p(x_j | x'_i).$$

Intuitively, when a Markov chain is lumpable with respect to partition  $T$ , we can construct a smaller Markov chain, where state  $x_i$  represents the partition  $T_i$  in  $T$ , and the transition from  $x_i$  to  $x_j$  are the summed transition probabilities from states in  $T_i$  to  $T_j$ . The chain constructed this way is the *quotient Markov chain*.

Lumping can lead to a substantial reduction in state space size. Instead of working with the original chain with large state space size, we can compute a lumping and work with the quotient chain (as long as we are not interested in differentiating between states that are in the same subset of  $T$ ). Unfortunately, most of the methods for generating the partition require complete generation of the state space of the Markov chain, so that they are intractable for large state spaces [56, 27].

Lumpability was also discussed in the context of lifted MCMC [157]: In this context, the quotient chain does not actually need to be computed. Instead, the idea is to introduce new transitions between states that are equivalent with respect to some equivalence relation, so that the resulting *orbital Markov chain* behaves as if it was the quotient chain.

It is important to note that this algorithm does not perform Bayesian filtering, but generates samples of a single, static distribution – the Markov chain that is used in the MCMC algorithm should not be confused with the Markov chain that is induced by the transition model of Bayesian filtering. Thus, the algorithm only considers symmetries that are present in the filtering distribution *at a specific point in time*.

Other approaches apply the concept of lumpability to stochastic processes in the context of stochastic simulation of MRSs [33]. They avoid generating the complete state space of the Markov chain by working on the MRS syntax. Specifically, they introduce an equivalence relation of *species* (called *Syntactic Markovian Bisimulation*, SMB), based on properties that can be checked statically by inspecting the MRS rules. Intuitively, two species  $X$  and  $Y$  are equivalent when all reactions that have identical products and whose reactants are only different in  $X$  and  $Y$  have identical reaction rates.

An SMB is fundamentally different from the symmetry property in LiMa, because it is a *global* property that must hold for all points in time and regardless of the current distribution over system states. In contrast, in LiMa, entities are grouped at specific points in time, when the corresponding random variables are exchangeable in the current filtering distribution. The grouping of entities can change over time due to splitting and merging. When a split of some property could be required in the system at some point (due to action preconditions or observations), entities that contain this property will not be equivalent in the sense of SMB.

### B.3. Probabilistic Programming Languages

Probabilistic Programming Languages (PPLs) describe, similar to computational state-space models (CSSMs) like LiMa, a complex distribution by the algorithmic process that generates the distribution. There are two branches of PPLs: *Probabilistic Logic Programs* (like ProbLog [65] or Prism [189]) add probabilistic annotations to facts of a logic program. Imperative and functional PPLs (e.g. Pyro [21] or Church [80]) 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 [237, 70], 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 [139, 172] 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.

### B.4. Knowledge Compilation and Tractable Models

Knowledge compilation refers to a class of propositional reasoning methods. The idea is to transform a propositional theory into a structure that allows more efficient inference, like a binary decision diagram (BDD), or d-DNNF (see [48] for an overview of compilation targets). The motivation is to perform this (potentially costly) transformation up-front, and then be able to answer a large number of queries on the compiled representation quickly.

These methods can be used for probabilistic inference, by transforming the probabilistic inference task to a weighted model counting (WMC) task. Given a propositional theory  $\Delta$  and a weight assigned to each literal (that define the weight of each model), WMC asks for the sum of all models of  $\Delta$ . After transforming the probabilistic inference to a WMC task, WMC can be performed efficiently by knowledge compilation methods.

Recently, tractable probabilistic models (like sum-product networks [176] or probabilistic sentential decision diagrams [112]) have received attention. They are closely related to knowledge compilation target languages, but take a more direct approach: They encode distributions in such a way that inference is directly tractable.

These methods can be seen as orthogonal to lifted inference methods: While knowledge compilation methods and tractable models can exploit contextual independence, the focus of lifted inference is on exploiting exchangeability. Indeed, a method that combines knowledge compilation with lifted inference has been devised [223], that works by transforming inference to a first-order WMC problem, and using a first-order compilation target language.

### B.5. Statistical Relational Learning

A task that has not been discussed in this thesis is that of *learning* a LiMa model. Specifically, the parameters of the MRS that defines the transition model (i.e. the action weights) or the parameters of the observation model could be estimated from data.

## B. More Related Work

A closely related task is considered in *statistical relational learning* [74], that aims at learning the parameters of relational probabilistic models from data. In parameter learning, the goal is to optimize the likelihood of the model, given the data. This is, as in the propositional setting, typically done by *expectation maximization*: The parameters are computed in an iterative process, consisting of computing the expected likelihood of the model, given the current parameters, and choosing new parameters that maximize this expectation. In contrast to propositional models, however, multiple parameters may be tied in relational models (thus effectively reducing the total number of parameters). Parameter learning is a difficult task, as it requires to perform probabilistic inference (which is itself a hard problem) each time the expectation is computed. Thus, approximate methods are typically used, that optimize easier to compute measures than the likelihood. Recently, exact [227] and approximate [2] lifted inference methods have been used for parameter learning.

## B.6. First-Order Markov Decision Processes

A Markov decision process (MDP) is a model for sequential decision making where an agent has to select actions based on the current environment state. Each action is associated with a reward. Given an MDP, the task is to compute an optimal *policy*, i.e. a function that assigns each state a corresponding action such that the long-term reward is maximized. The optimal policy can be obtained by computing the *value function* (that assigns a value to each state) using dynamic programming, see Puterman [177] for a more thorough introduction.

Similar to Bayesian filtering in MRSs, MDPs also suffer from the combinatorial explosion in the number of states, such that algorithms that need to enumerate all states can become infeasible. Methods for retaining a compact representation of the state space follow two basic ideas. The first approach is to find symmetries in the state space of an MDP and group symmetric state, thus obtaining a smaller state space [54, 76, 98]. The second approach is to directly use a first-order formalism to represent states, and perform all operations within this formalism [24, 107, 85, 185, 235]. In these approaches, states, actions, reward functions, value functions and policies are all based on first-order logic. This way, the resulting policy can be independent of the actual domain objects, and the computations to obtain this policy can be independent of the domain size. A problem is that the (logical) representation of the value function can easily become complex and redundant, requiring expensive first-order simplification. Proposed solutions to this problem include first-order ADDs (that can represent such a value function more compactly), and the use of approximate methods [185].

Conceptually, first-order MDPs (and their solution techniques) bear strong relationships to lifted probabilistic inference: Both are concerned with first-order models, where parts of the model are redundant or identical. They both exploit these symmetries to achieve more efficient algorithms, by performing operations “in bulk” for entire sets of redundant components.

However, there is also a more technical, intimate relationship between MDPs and probabilistic inference: It has been shown that decision problems (in terms of an MDP) can be cast into a probabilistic inference problem [215]. Thus, any probabilistic inference algorithm can be used to solve MDPs. This relationship also holds for first-order MDPs: Recently, Khaldon and Sanner [109] showed that the probabilistic inference problem that can be derived from a first-order MDP inherits its symmetric structure. This structure can be exploited by lifted

inference, avoiding redundant computations. Due to the complex structure of the query, it is however not possible to use standard lifted inference algorithms here. Instead, the first-order dynamic programming approaches for solving first-order MDPs can be seen as performing some specialized lifted inference algorithm. An interesting perspective for future research is to combine the distinct innovations from both domains, and close the gap between the respective lines of research – a paradigm termed *generalized lifted inference* by Khardon and Sanner [109].



# Assignment of Papers to Groups

The following table shows the specific papers associated with each of the groups defined in the systematic literature review (Chapter 3).

Name	References
Top-down LI	[173] [113] [51] [52] [141] [6] [207] [210] [211] [49] [208] [209] [154] [153] [212] [114] [40] [200] [53] [201] [202] [79] [78] [223] [220] [217] [221] [140] [16] [234] [29] [77] [37] [93] [174] [103] [101] [102] [110] [111] [175] [186] [187] [232] [142] [58] [45] [46] [62] [94]
Bottom-up LI	[106] [92] [105] [3] [2] [231] [230] [228] [229] [218] [83] [196] [197] [30] [31] [155] [5] [222] [156] [144] [7] [143] [146] [145] [219] [152] [151] [1] [82] [4] [73]
Continuous Inference	[17] [18] [19] [184] [198]
Logical Particle Filter	[242]
Relational Particle Filter	[160] [162] [161]
Relational Kalman Filter	[38] [41] [39]
Data Association	[194] [87] [88] [89] [91] [117] [95] [11] [12] [13] [14] [15] [84] [126] [127] [137]
Prob. Multiset Rewriting	[9] [10] [119] [236] [22] [166] [138]





# AMCA Computation as Constraint Satisfaction

The central computational challenge in PMPMRs is the enumeration of all AMCAs  $k$  of a state  $l$ , which is required for computing the partition function  $Z = \sum_{k \in K} v_l(k)$ , where  $v_l(k)$  is the weight of  $k$  in  $l$  (see Definition 12). In Section 4.1.5, we presented a backtracking search algorithm for this purpose, that exploits commutativity of multiset insertion to prune subtrees of the search tree (that correspond to the different insertion orders).

Alternatively, the problem of identifying AMCAs can be formalized as a constraint satisfaction problem (CSP), which allows us to employ the mature and well-understood theory of CSPs to analyze the problem. Specifically, for a given set of action instances  $A$  and state  $l = (s, \gamma)$ , a CSP  $\Omega = (X, D, C)$ <sup>1</sup> can be created, such that each solution of the CSP is an AMCA of  $(A, l)$  and vice versa, as follows:

- For each action instance  $(a, i) \in A$ , there is a CSP variable  $x \in X$ . The domain of  $x$  is  $\{0, \dots, \min_{e \in i}(s \# e)\}$ , i.e. the variable  $x$  models the multiplicity of  $(a, i)$  in the compound action.
- For each entity  $e \in \text{dom}(s)$  with multiplicity  $n_e$ , there is a constraint  $c \in C$  involving all variables  $x_i$  whose corresponding action instances  $a_i$  bind  $e$ . Let  $m_{i,e}$  be the number of times the action instance  $a_i$  binds  $e$ . The constraint then is  $\sum_i m_{i,e} x_i = n_e$ . This constraint ensures that the corresponding compound action is applicable and maximal.

Note that the constraint language consists only of sums and equality, independently of the constraint language of action preconditions, which have been resolved before when computing action instances.

A solution  $\sigma$  of the CSP  $\Omega$  is an assignment of all variables in  $X$  that satisfies all constraints. Each solution  $\sigma$  of  $\Omega$  corresponds to a compound action  $k_\sigma$ , where the value  $\sigma(x)$  of a variable  $x$  indicates the multiplicity of the corresponding action instance  $(a, i)$  in  $k_\sigma$ . Furthermore,  $k_\sigma$  is obviously applicable and maximal due to the constraints of  $\Omega$ , as illustrated by the following example.

**Example 46.** Recall the population model introduced in Example 33, where prey  $x = \langle \text{Type} = X \rangle$  and predators  $y = \langle \text{Type} = Y \rangle$  exist. Predators can eat other animals (prey

<sup>1</sup> $X$  is a set of variables,  $D$  is a set of domains, and  $C$  is a set of constraints

#### D. AMCA Computation as Constraint Satisfaction

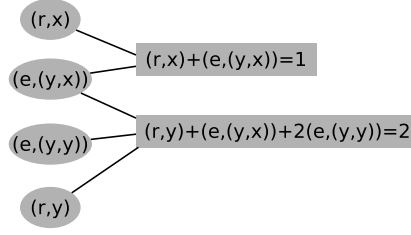


Figure D.1.: The CSP for Example 46. Circles represent variables, rectangles represent constraints.

or other predators, action  $e$ ), and all animals can reproduce (action  $r$ ). For the state  $l = \llbracket 1x, 2y \rrbracket$ , the following applicable action instances exist:  $(r, \langle y \rangle)$ ,  $(r, \langle x \rangle)$ ,  $(e, \langle y, x \rangle)$ ,  $(e, \langle y, y \rangle)$ .

The corresponding CSP is shown in Figure D.1. For example, it contains the constraint  $(r, y) + (e, (y, x)) + 2(e, (y, y)) = 2$  because the entity  $y$  needs to be bound exactly 2 times (i.e. the sum of multiplicities of action instances that bind  $y$  must be 2). Obviously, this CSP has three solutions that correspond directly to the AMCAs.

**Theorem 2.** Let  $A$  be a set of action instances, and let  $l$  be a lifted state. Let  $\Omega$  be the CSP constructed from  $(A, l)$  as outlined above. Let  $k_\sigma$  denote the compound action constructed from a CSP solution  $\sigma$ . Then, for each solution  $\sigma$  of  $\Omega$ ,  $k_\sigma$  is an AMCA of  $(A, l)$ , and for each AMCA  $k$  of  $(A, l)$ , there is a solution  $\sigma$  of  $\Omega$  so that  $k_\sigma = k$ .

*Proof.* Let  $\sigma$  be a solution of  $\Omega$ , and  $k_\sigma$  be the corresponding compound action. Consider an arbitrary entity  $e$  of  $l$  with multiplicity  $n_e$ . Due to the constraint  $\sum_i m_{e,i} x_i = n_e$ , the entity  $e$  is bound exactly  $n_e$  times in  $k_\sigma$ . Thus,  $k_\sigma$  is an AMCA.

Let  $k$  be an AMCA of  $l$ . We can construct an assignment  $\sigma$  of the CSP variables where the value of variable  $x_i$  is the multiplicity of the corresponding action instance  $a_i$  in  $k$ . As  $k$  is applicable and maximal,  $\sigma$  satisfies all constraints of  $\Omega$  and thus is a solution of  $\Omega$ .  $\square$

Thus, we can obtain all AMCAs of  $(A, l)$  by enumerating all solutions of the corresponding CSP  $\Omega$ . We can use standard backtracking for this purpose. This is sufficient, as the problem here is not that *finding* each solution is difficult, but that there are factorially many solutions.

Note that the CSP we are considering is *not* an instance of a *valued* (or *weighted*) CSP [43, 190]: They assume that each satisfied constraint has a value, and the goal is to find the *optimal* variable assignment, whereas in our approach, only *solutions* have a value, and we are interested the *distribution* of solutions.

# Expressiveness of Sequential and Maximally Parallel Multiset Rewriting

In this section, we investigate the relationship between maximally parallel MRS semantics (as defined in Section 4.1) and sequential MRS semantics (as defined in Section 2.3.1). Specifically, we are concerned with the expressiveness of both formalisms. Obviously, each sequential MRS can be emulated by a maximally parallel MRS, by adding a mutex entity to the state and all preconditions, so that each compound action consists of exactly a single action.

Here, we investigate their relationship in the other direction: Is it possible, for each maximally parallel MRS, to construct an equivalent sequential MRS that shows the same system behavior? If this would be the case, the involved definitions of the maximally parallel MRS in Section 4.1 would be superfluous, and we could have used a sequential MRS as the foundation of the efficient BF algorithm.

Below, we show that this is *not* the case: Maximally parallel MRS are strictly more expressive than sequential MRS, i.e. there are systems that cannot be expressed by a sequential MRS.

**Example 47.** Consider an MRS with the initial state  $x = \llbracket 1A, 1B, 1X, 1Y \rrbracket$  and actions<sup>1</sup>

$$\begin{aligned} AX &= \llbracket 1A, 1X \rrbracket \rightarrow \llbracket \rrbracket \\ AY &= \llbracket 1A, 1Y \rrbracket \rightarrow \llbracket \rrbracket \\ BX &= \llbracket 1B, 1X \rrbracket \rightarrow \llbracket \rrbracket \\ BY &= \llbracket 1B, 1Y \rrbracket \rightarrow \llbracket \rrbracket \end{aligned}$$

For this MRS, action instances coincide with actions, i.e. there is one action instance for each of the actions. Suppose that the weights of the action instances are  $w_{AX} = w_{BX} = w_{BY} = 3$  and  $w_{AY} = 1$ .

---

<sup>1</sup>For simplicity, we used a simple reactant-product notation for the actions, but of course, the actions could just as well be denoted in precondition-effect notation.

### E. Expressiveness of Sequential and Maximally Parallel Multiset Rewriting

Note that this MRS models a simple assignment problem over the sets  $\{A, B\}$  and  $\{X, Y\}$ , where the assignment  $AY$  has higher cost than the others (when interpreting weights as negative costs).

There are two possible AMCAs  $k_1 = \llbracket 1AX, 1BY \rrbracket$  and  $k_2 = \llbracket 1AY, 1BX \rrbracket$ . Using Definition 12, the compound actions have probabilities  $p(k_1 | x) = 3 * 3/12 = 0.75$  and  $p(k_2 | x) = 3/12 = 0.25$ .

We are concerned with the question whether there is a sequential MRS that “behaves similar” to the maximally parallel MRS outlined in the example. More precisely, can we define a distribution over actions  $p(a | x)$  so that the probability of sampling a specific sequence of actions is identical to the probability of sampling the corresponding AMCA?

Before we can answer this question, we need to formalize it more. Consider a sequence of actions  $s = \langle a_1, \dots, a_T \rangle$ . Under sequential MRS semantics, the probability of the sequence is simply given by the product of the individual actions’ probabilities. These probabilities will in general depend on the weight of the actions, and (by the chain rule) on the previously chosen actions (not directly, but via the state that is manipulated by the previously chosen actions).

$$p(s) = p(A_1 = a_1, \dots, A_T = a_T) = \prod_{t=1}^T p_{\mathbf{w}}(A_t = a_t | A_1 = a_1, \dots, A_{t-1} = a_{t-1}), \quad (\text{E.1})$$

where  $\mathbf{w}$  is the vector of all action weights. Here, the random variable  $A_t$  denotes the action chosen at time  $t$ .

Obviously, multiple of such sequences correspond to the same compound actions, as the elements in the compound action do not have an order. Specifically, all sequences  $s$  with  $\text{items}(s) = k$  correspond to the compound action  $k$ . In the example above, the sequences  $s_{11} = \langle AX, BY \rangle$  and  $s_{12} = \langle BY, AX \rangle$  both correspond to  $k_1$ , and the sequences  $s_{21} = \langle AY, BX \rangle$  and  $s_{22} = \langle BX, AY \rangle$  both correspond to  $k_2$ .

We say that a sequential MRS and parallel-action MRS behave similar, when for all  $k$ ,

$$p(k) = \sum_{\{s | \text{items}(s)=k\}} p(s) \quad (\text{E.2})$$

Now, we come back to Example 47, and investigate whether we can find action probabilities for a sequential MRS so that it is equivalent to the parallel MRS in the sense of Equation E.2.

**Example 47, cont’d.** In this example, both compound actions consist of exactly two actions, thus we look at action sequences of length two, and try to find distributions  $p(A_1)$  and  $p(A_2 | A_1)$  so that Equation E.2 holds. If distributions  $p(A_1)$  and  $p(A_2 | A_1)$  exist, the following needs to hold:

$$\begin{aligned} p(k_1) &= p(s_{11}) + p(s_{12}) \\ &= p(A_1=AX) p(A_2=BY | A_1=AX) + p(A_1=BY) p(A_2=AX | A_1=BY) \end{aligned} \quad (\text{E.3})$$

Note that in this example, the choice of the second action is always deterministic: Once the first action is chosen, there are only two entities left in the state, leaving only a single action that can be executed. Thus,  $p(A_2=BY | A_1=AX) = p(A_2=AX | A_1=BY) = 1$ , so that

$$p(k_1) = p(A_1=AX) + p(A_1=BY). \quad (\text{E.4})$$

The same reasoning applies to  $p(k_2)$ , for which we get

$$p(k_2) = p(A_1=AY) + p(A_1=BX). \quad (\text{E.5})$$

Now, we make use of the fundamental assumption for sequential MRS stated above: The probability of an action depends only on the action's weight and previously executed actions (via the current state). From  $w_{AX} = w_{BX} = w_{BY}$ , it thus follows that  $p(A_1=AX) = p(A_1=BX) = p(A_1=BY)$ . Using the symbolic names  $\alpha = p(A_1=AX) = p(A_1=BX) = p(A_1=BY)$  and  $\beta = p(A_1=AY)$ , we get

$$p(k_1) = 2\alpha \quad (\text{E.6})$$

$$p(k_2) = \alpha + \beta \quad (\text{E.7})$$

Inserting the compound action probabilities  $p(k_1) = 0.75$  and  $p(k_2) = 0.25$  and solving for  $\alpha$  and  $\beta$  gives  $\alpha = 0.375$  and  $\beta = -0.125$ . As  $\beta$  must be a probability, we cannot find a distribution  $p(A_1)$  so that the resulting sequential MRS has a similar dynamics as the parallel MRS. We can generalize this observation in the following theorem.

**Theorem 3.** [non-serializability] Let  $A$  be a set of actions and let  $x_0$  be a state. Let  $K$  be the set of AMCAs of  $(A, x_0)$ , and let  $p(k | x_0)$  be a distribution over AMCAs. It is not possible to fix distributions  $p(A_t | A_1, \dots, A_{t-1})$  that only depend on the action weights and satisfy Equation E.2.

The counterexample given above directly proofs this theorem.

The theorem shows that it is not always possible to find a sequential MRS that shows the same behavior as a maximally parallel MRS. Specifically, this means that although two actions have identical weight, they need to have different probabilities to emulate the behavior of a maximally parallel MRS. Of course, this violates the fundamental assumption for sequential MRS: The weight are the only information available to construct the action probabilities.

From a certain perspective, this result is not surprising: If it would be possible to construct a sequential MRS with equivalent behavior to a given maximally parallel MRS, the global optimum of the assignment problem that we started with in the example above could be solved by a *greedy* strategy, by simply selecting the most likely action (i.e. most likely assignment) in each step.

Another reason for the potential disagreement of maximally parallel and sequential MRS can be seen as follows: Consider a given joint distribution over actions  $p(A_1, A_2)$ . The marginal probability for selecting the first action is, of course  $p(A_1) = \sum_{a_2} p(A_1, A_2=a_2)$ . Thus, intuitively, to faithfully model the compound action distribution  $p(A_1, A_2)$ , the marginal distribution  $p(A_1)$  must contain “knowledge” about the future, since it needs to consider all choices of future actions.

In summary, parallel and serial systems defined on the same set of local weights will represent disagreeing distributions over compound actions, no matter what mapping from weights to probabilities is chosen.

Finally, this result does not mean that maximally parallel MRSs are inherently *better* than sequential MRSs. Our goal here was not a “competition” between different MRS semantics, but rather showing that maximally parallel MRS can model different situations than the (more common) sequential MRS. Which semantics is more suitable for a given situation is a different question that might not always be easy to answer.



## Disjointness of Lifted States

Here, we discuss the following problem: Given two lifted states  $l_1$  and  $l_2$ , test whether there is a ground state that has a non-zero probability in both  $l_1$  and  $l_2$ , i.e. whether  $\text{region}(l_1) \cap \text{region}(l_2) \neq \emptyset$ , without completely enumerating the ground states. Additionally, when such a ground state exists, perform appropriate splitting operations, such that all split results are pairwise disjoint.

**Example 48.** As an example of two lifted states whose regions are not disjoint, consider the situation shown in Figure F.1. The regions of the lifted states  $l_1$  and  $l_2$  with

$$\begin{aligned} l_1 &= (\llbracket 1\langle N: \mathbf{N}_1, L: X \rangle, 1\langle N: \mathbf{N}_2, L: X \rangle, 1\langle N: \mathbf{N}_1, L: Y \rangle \rrbracket, \langle \mathbf{N}_1: \mathcal{U}(A, B), \mathbf{N}_2: \mathcal{U}(A, B) \rangle) \\ l_2 &= (\llbracket 1\langle N: \mathbf{N}_1, L: X \rangle, 1\langle N: \mathbf{N}_2, L: X \rangle, 1\langle N: \mathbf{N}_1, L: Y \rangle \rrbracket, \langle \mathbf{N}_1: \mathcal{U}(A, B), \mathbf{N}_2: \mathcal{U}(A) \rangle) \end{aligned}$$

overlap on the ground state

$$x = \llbracket 1\langle N: A_{\mathbf{N}_1}, L: X_{\mathbf{X}} \rangle, 1\langle N: B_{\mathbf{N}_2}, L: X_{\mathbf{X}} \rangle, 1\langle N: A_{\mathbf{N}_1}, L: Y_{\mathbf{Y}} \rangle \rrbracket.$$

As in Section 6.2, we consider overlapping states for two situations: We start with the simple case, where the regions of two lifted states are considered to be overlapping when there is a *typed* ground state that is an element of both regions (as in the example above).

Afterwards, we discuss a more general case, where overlap of regions is defined in terms of *untyped* ground states: The regions of the states  $l_1$  and  $l_2$  are considered to be overlapping when there are ground states  $x_1 \in \text{region}(l_1)$  and  $x_2 \in \text{region}(l_2)$  so that  $x_1$  is identical to  $x_2$  under an arbitrary renaming of the distribution types of  $x_1$ . This case is relevant, for example, for the MERGE-DISJOINT algorithm (Section 6.3), that requires states to be disjoint in this sense.

### F.1. Disjointness of Typed States

In this section, we investigate the case where the distribution types of ground states are taken into account to define overlap of regions. We first show how we can test whether  $\text{region}(l_1) \cap \text{region}(l_2) \neq \emptyset$  for given  $l_1$  and  $l_2$ , without enumeration of all ground states. Afterwards, we show how appropriate splits can be applied to  $l_1$  and  $l_2$ , so that the regions of all split results are disjoint.

## F. Disjointness of Lifted States

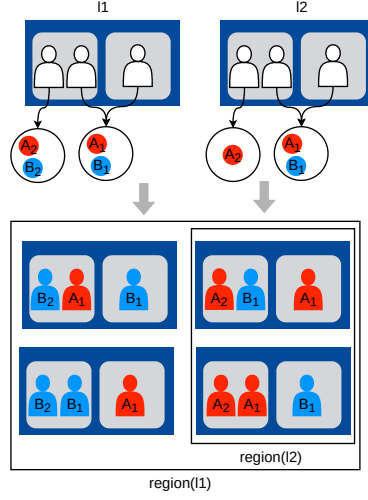


Figure F.1.: Example of lifted states with overlapping regions (see Example 48).

### F.1.1. Identifying Overlap

We start by showing that lifted states can only have overlapping regions when their structures are identical. We call the states  $l_1 = (s_1, \gamma_1)$  and  $l_2 = (s_2, \gamma_2)$  disjoint when  $\text{region}(l_1) \cap \text{region}(l_2) = \emptyset$ , i.e. when for each ground state  $x$ , either  $p(x | s_1, \gamma_1) = 0$  or  $p(x | s_2, \gamma_2) = 0$ . In other words, the states are disjoint when the conditional distributions  $p(x | s_1, \gamma_1)$  and  $p(x | s_2, \gamma_2)$  are *orthogonal*, i.e. when

$$0 = \sum_x p(x | s_1, \gamma_1) p(x | s_2, \gamma_2).$$

By using the decomposition  $\phi(x) = (s, \mathbf{v})$ , we get:

$$\begin{aligned} &= \sum_s \sum_{\mathbf{v}} p(s | s_1, \gamma_1) p(\mathbf{v} | s, s_1, \gamma_1) p(s | s_2, \gamma_2) p(\mathbf{v} | s, s_2, \gamma_2) \\ &= \sum_s p(s | s_1, \gamma_1) p(s | s_2, \gamma_2) \sum_{\mathbf{v}} p(\mathbf{v} | s_1, \gamma_1) p(\mathbf{v} | s_1, \gamma_1) \end{aligned}$$

Only a single term of the first sum can be non-zero, when  $s = s_1 = s_2$ , in which case  $p(s | s_1, \gamma_1) = 1$  and  $p(s | s_2, \gamma_2) = 1$ , i.e.

$$= \mathbb{1}(s_1 = s_2) \sum_{\mathbf{v}} p(\mathbf{v} | s_1, \gamma_1) p(\mathbf{v} | s_2, \gamma_2).$$

Thus,  $l_1$  and  $l_2$  can only be non-orthogonal when their structures are identical.

Next, we decompose the other sum further. When the structures  $s_1$  and  $s_2$  are identical, the contexts  $\gamma_1$  and  $\gamma_2$  have identical distribution types. Using this fact (and the fact that



$p(\mathbf{v} \mid s, \gamma)$  decomposes into the product shown in Equation 4.9), the sum can be rewritten as

$$\begin{aligned}
& \sum_{\mathbf{v}} p(\mathbf{v} \mid s_1, \gamma_1) p(\mathbf{v} \mid s_2, \gamma_2) \\
&= \sum_{\mathbf{v}} \left( \prod_{d \in \text{dom}(\gamma_1)} p_{\gamma_1(d)}(\mathbf{v}^{(d)} \mid s_1) \right) \left( \prod_{d \in \text{dom}(\gamma_2)} p_{\gamma_2(d)}(\mathbf{v}^{(d)} \mid s_2) \right) \\
&= \sum_{\mathbf{v}} \prod_{d \in \text{dom}(\gamma_1)} p_{\gamma_1(d)}(\mathbf{v}^{(d)} \mid s_1) p_{\gamma_2(d)}(\mathbf{v}^{(d)} \mid s_2).
\end{aligned}$$

This expression is non-zero when at least one of the terms of the sum is non-zero. This is the case when each pair of factors  $\gamma_1(d)$  and  $\gamma_2(d)$  have an assignment for which they are both non-zero, i.e. when each pair of factors  $\gamma_1(d)$  and  $\gamma_2(d)$  has overlapping support.

In summary, the regions of lifted states  $l_1$  and  $l_2$  overlap when (i) they have the same structure, and (ii) when all factors with identical types have overlapping support. Both properties can be tested on the syntactic level, without enumerating all groundings.

### F.1.2. Shattering

To ensure disjointness, states with overlapping regions need to be split until all split results are disjoint. Specifically, a pair of states  $l_1$  and  $l_2$  with overlapping regions will be split into at least three split results: A state representing  $\text{region}(l_1) \setminus \text{region}(l_2)$ , a state representing  $\text{region}(l_2) \setminus \text{region}(l_1)$ , and a state representing  $\text{region}(l_1) \cap \text{region}(l_2)$ .

For the case discussed here, identifying appropriate splits is straightforward: We can simply select a pair of factors  $\gamma_1(d)$  and  $\gamma_2(d)$  that have overlapping (but not identical) support. Such a factor must exist: If all factors have identical support, the states  $l_1$  and  $l_2$  would be identical, and if a pair of factors would have disjoint support, the states would be disjoint.

For such a pair  $\gamma_1(d)$  and  $\gamma_2(d)$ , we split both factors on a value that is in the support of both  $\gamma_1(d)$  and  $\gamma_2(d)$ . This way, after splitting, in all split results, the factors can no longer overlap on that value. This procedure then needs to be applied recursively to all split results, as they can still overlap on other factors (or on other values of the same factor). The procedure is guaranteed to terminate, as the regions of the states become smaller with each split. In the worst case, all resulting states are completely ground (and thus guaranteed to be disjoint). We call this process *shattering* (named after shattering in FOVE [51]).

**Example 49.** Consider the states  $l_1 = (s_1, \gamma_1)$  and  $l_2 = (s_2, \gamma_2)$  from Example 48. The support of factors  $\gamma_1(\mathbf{N}_2)$  and  $\gamma_2(\mathbf{N}_2)$  overlap on the value  $A$ . Thus, we split  $l_1$  on the entity  $e = \langle \mathbf{N}: \mathbf{N}_2, \mathbf{L}: X \rangle$  and the constraint  $e.N == A$ . Splitting  $l_1$  leads to the following two split results:

$$\begin{aligned}
l'_1 &= (\llbracket 1\langle \mathbf{N}: \mathbf{N}_1, \mathbf{L}: X \rangle, 1\langle \mathbf{N}: \mathbf{N}_2, \mathbf{L}: X \rangle, 1\langle \mathbf{N}: \mathbf{N}_1, \mathbf{L}: Y \rangle \rrbracket, \quad \langle \mathbf{N}_1: \mathcal{U}(A, B), \mathbf{N}_2: \mathcal{U}(A) \rangle) \\
l''_1 &= (\llbracket 1\langle \mathbf{N}: \mathbf{N}_1, \mathbf{L}: X \rangle, 1\langle \mathbf{N}: \mathbf{N}_2, \mathbf{L}: X \rangle, 1\langle \mathbf{N}: \mathbf{N}_1, \mathbf{L}: Y \rangle \rrbracket, \quad \langle \mathbf{N}_1: \mathcal{U}(A, B), \mathbf{N}_2: \mathcal{U}(B) \rangle)
\end{aligned}$$

In  $l_2$ , the factor  $\gamma_2(\mathbf{N}_2)$  is already ground and thus, the state is not split. Now, all resulting states are either disjoint or identical: The region of  $l''_1$  is disjoint from the regions of  $l_2$  as well as  $l'_1$ , and  $l'_1$  and  $l_2$  are identical so they can be trivially merged.

Finally, shattering can be integrated into the filtering algorithm. Specifically, the test for disjointness can be performed iteratively during the prediction step, when inserting elements

## F. Disjointness of Lifted States

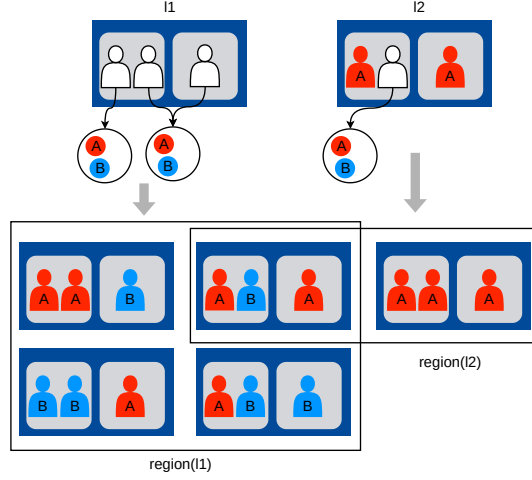


Figure F.2.: Example of lifted states with overlapping regions, when distribution types are ignored.

into the map that represents the filtering distribution (second to last line in Algorithm 3). Condition (i) – states can only overlap when they have the same structure – can be exploited here, by representing the filtering distribution by a map  $P : \mathcal{S} \rightarrow (\mathcal{V} \rightarrow \mathbb{R}_{>0})$ , so that states with potential overlap can be identified quickly.

## F.2. Disjointness of Untyped States

Next, we consider a more general notion of *overlap*: Lifted states have overlapping region when there are ground states  $x_1 \in \text{region}(l_1)$  and  $x_2 \in \text{region}(l_2)$  so that  $x_1$  is identical to  $x_2$  except for the distribution types. For example, the MERGE-DISJOINT algorithm (see Section 6.3) requires that all lifted states are disjoint in this sense. The underlying intuition is that ground states that have identical values are identical, even when these values were sampled from different factors (indicated by different distribution types). The following example illustrates this situation.

**Example 50.** Consider the states  $l_1$  and  $l_2$  shown in Figure F.2, where

$$l_1 = (\llbracket 2\langle N: \mathbf{N}_1, L: X \rangle, 1\langle N: \mathbf{N}_2, L: Y \rangle \rrbracket, \langle \mathbf{N}_1: \mathcal{U}(A, B), \mathbf{N}_2: \mathcal{U}(A, B) \rangle)$$

$$l_2 = (\llbracket 1\langle N: \mathbf{N}_1, L: X \rangle, 1\langle N: \mathbf{N}_2, L: X \rangle, 1\langle N: \mathbf{N}_3, L: Y \rangle \rrbracket, \langle \mathbf{N}_1: \mathcal{U}(A, B), \mathbf{N}_2: \mathcal{U}(A), \mathbf{N}_3: \mathcal{U}(A) \rangle)$$

The ground states  $x_1 \in \text{region}(l_1)$  and  $x_2 \in \text{region}(l_2)$  where

$$x_1 = \llbracket 1\langle N: B_{\mathbf{N}_1}, L: X \rangle, 1\langle N: A_{\mathbf{N}_1}, L: X \rangle, 1\langle N: A_{\mathbf{N}_2}, L: Y \rangle \rrbracket$$

$$x_2 = \llbracket 1\langle N: B_{\mathbf{N}_1}, L: X \rangle, 1\langle N: A_{\mathbf{N}_2}, L: X \rangle, 1\langle N: A_{\mathbf{N}_3}, L: Y \rangle \rrbracket$$

are identical, except for the distribution types (i.e. when “removing” the distribution types from  $x_1$  and  $x_2$ , they are both mapped to the state

$$x = \llbracket 1\langle N: B, L: X \rangle, 1\langle N: A, L: X \rangle, 1\langle N: A, L: Y \rangle \rrbracket.$$

Identifying states that overlap in this sense is more challenging than for the case before. Intuitively, the problem is that we do not know in advance how the entities from  $l_1$  and  $l_2$  are associated to the entities in  $x$  – in contrast to the case above, where the distribution types directly provided this information. The general strategy is to search a possible matching of the entities from  $l_1$  and  $l_2$ , so that a ground state  $x$  that is in the region of both  $l_1$  and  $l_2$  can be constructed from the matching.

In the following, we start by defining the notion of states that are “similar except for distribution types” more precisely. Afterwards, we define *matchings* of lifted states  $l_1$  and  $l_2$ , which indicate how the entities from  $l_1$  and  $l_2$  are associated, when this information is not provided by the distribution type, as before. Finally, we show how a state  $x \in \text{region}(l_1) \cup \text{region}(l_2)$  can be constructed from a matching, and how the necessary splits (so that all split results are disjoint) can be read off  $x$ .

### F.2.1. Untyped States

The concept of “ignoring distribution types” is formally captured by *untyped states*.

**Definition 22.** [untyped entity, untyped state] Let  $e = \langle q_1 : (d_1, v_1), \dots, q_n : (d_n, v_n) \rangle$  be a typed entity. We call  $u(e) = \langle q_1 : v_1, \dots, q_n : v_n \rangle$  the untyped entity of  $e$ . Similarly, given a typed (ground) state  $x = \llbracket m_1 e_1, \dots, m_n e_n \rrbracket$ , we call  $u(x) = \llbracket m_1 u(e_1), \dots, m_n u(e_n) \rrbracket$  the untyped state of  $x$ .

Similarly, an *untyped* entity structure  $u(e)$  is a set of property names of the entity  $e$ , and an untyped structure  $u(s)$  is a multiset of the untyped entities of the structure  $s$ . Based on this definition, we can define the *untyped region* of an entity as  $\text{uregion}_l(e) = \{u(e_x) \mid e_x \in \text{region}_l(e)\}$ , and similarly the untyped region of a state  $l$  as  $\text{uregion}(l) = \{u(x) \mid x \in \text{region}(l)\}$ . Finally, we can specify the goal of this section more precisely: Given two lifted states  $l_1$  and  $l_2$ , identify whether  $\text{uregion}(l_1) \cap \text{uregion}(l_2) \neq \emptyset$ .

### F.2.2. Matchings

In the following, we discuss how we can test whether the untyped regions of lifted states overlap, without enumerating all untyped ground states. Conceptually, the idea is to construct a *matching* of entities from  $l_1$  and  $l_2$ , so that each pair of entities have overlapping (untyped) regions. As we will see below, the existence of such a matching is only a necessary condition for the overlap of  $l_1$  and  $l_2$ , as it ignores the dependencies between entities (that arise when values in multiple entities are drawn from a joint factor).

**Definition 23.** [multiset matching] Let  $s_1$  and  $s_2$  be multisets. We call a sequence of pairs

$$m = \langle (e_1^{(i)}, e_2^{(i)}) \rangle_{i=1}^n$$

a *matching* of  $s_1$  and  $s_2$ , when

- for each pair  $(e_1, e_2) \in m$  :  $e_1 \in \text{dom}(s_1)$  and  $e_2 \in \text{dom}(s_2)$ ,
- the multiplicity of each entity  $e \in s_1$  is identical to the number of occurrences of  $e$  in  $m$ , i.e.  $s_1 \# e = \sum_{(e_1, e_2) \in m} \mathbb{1}(e_1 = e)$ , and
- the multiplicity of each entity  $e \in s_2$  is identical to the number of occurrences of  $e$  in  $m$ , i.e.  $s_2 \# e = \sum_{(e_1, e_2) \in m} \mathbb{1}(e_2 = e)$ .

## F. Disjointness of Lifted States

The set of all matchings of the entities in  $s_1$  and  $s_2$  (that are distinct up to ordering of the pairs) is denoted as  $\mathcal{M}(s_1, s_2)$ .

For each ground state  $x \in \text{region}(l)$ , there is a matching between  $x$  and  $l$  so that for each pair  $(e_x, e_l)$ ,  $e_x \in \text{uregion}_l(e_l)$ , as stated by the following theorem.

**Theorem 4.** Let  $l = (s, \gamma)$  be a lifted state, and let  $x \in \text{uregion}(l)$ . Then, there is a matching  $m$  of  $s$  and  $x$ , with the property that  $\forall (e_s, e_x) \in m: e_x \in \text{uregion}_l(e_s)$ .

*Proof.* Consider the sampling semantics to obtain a sample  $x \in \text{uregion}(l)$ . For each entity  $e_s \in \text{dom}(s)$ , a new entity  $e_x$  is created as follows: For each property  $q$  in  $e_s$ , sample a value  $v$  and set  $e_x.q = v$ . Obviously,  $e_x \in \text{uregion}_l(e_s)$ . Thus, when collecting exactly all those pairs  $(e_s, e_x)$  in  $m$ , the sequence  $m$  will be a matching with the claimed property.  $\square$

**Example 51.** Consider the lifted state

$$l_1 = (\llbracket 2\langle N: \mathbf{N}_1, L: X \rangle, 1\langle N: \mathbf{N}_2, L: Y \rangle \rrbracket, \quad \langle \mathbf{N}_1: \mathcal{U}(A, B), \mathbf{N}_2: \mathcal{U}(A, B) \rangle)$$

and the ground state  $x = \llbracket 1\langle N: A, L: X \rangle, 1\langle N: B, L: X \rangle, 1\langle N: A, L: Y \rangle \rrbracket$  from Example 50 above. As  $x \in \text{uregion}(l_1)$ , we can find a matching  $m$  with the property described in Theorem 4:

$$\begin{aligned} m = & \langle (\langle N: \mathbf{N}_1, L: X \rangle, \langle N: A, L: X \rangle), \\ & (\langle N: \mathbf{N}_1, L: X \rangle, \langle N: B, L: X \rangle), \\ & (\langle N: \mathbf{N}_2, L: Y \rangle, \langle N: A, L: Y \rangle) \rangle \end{aligned}$$

This is true for *any* lifted state  $l$  that has  $x$  in its untyped region. Thus, for two lifted states  $l_1$  and  $l_2$  that both have  $x$  in their untyped region (i.e. that are overlapping), there is also a matching of entities from  $l_1 = (s_1, \gamma_1)$  and  $l_2 = (s_2, \gamma_2)$ , such that for each pair  $(e_1, e_2)$  with  $e_1 \in \text{dom}(s_1)$  and  $e_2 \in \text{dom}(s_2)$ , they both have an entity  $e \in \text{dom}(x)$  in their region.

**Theorem 5.** Let  $l_1$  and  $l_2$  be two lifted states with  $\text{uregion}(l_1) \cap \text{uregion}(l_2) \neq \emptyset$ . Then, there is a matching  $m$  of  $l_1$  and  $l_2$  where

$$\forall (e_1, e_2) \in m : \text{uregion}_{l_1}(e_1) \cap \text{uregion}_{l_2}(e_2) \neq \emptyset. \quad (\text{F.1})$$

This theorem provides us with a necessary condition for overlapping regions of lifted states that can be tested on individual entities. Fortunately, deciding whether Equation F.1 holds is simple: Regions of entities overlap when they have the same properties, and when for each property  $q$ , the corresponding distributions  $\gamma_1(e_1.q)$  and  $\gamma_2(e_2.q)$  do not have disjoint support.

However, *finding* a matching with the property of Equation F.1 can still be difficult. Naively, we need to construct all matchings  $m$ , and check each of them. Below, we outline a search-based approach that can be more efficient.

Before, we introduce a another necessary condition for overlapping regions that can be tested without finding a matching of entities: Similar to the case above, two lifted states  $l_1$  and  $l_2$  can only have overlapping regions when they have the same untyped structure. This is the case because all ground states from the region of  $l_1$  and  $l_2$  need to share that structure.

---

**Algorithm 17** Test disjointness of lifted states (for the case where distribution types are ignored).

---

```

1: function IS-DISJOINT( $l_1 = (s_1, \gamma_1)$ ,  $l_2 = (s_2, \gamma_2)$ )
2:   if  $u(s_1) \neq u(s_2)$  then
3:     return true
4:   Let  $E_1$  be the sequence of entities of  $s_1$ , and  $E_2$  be the sequence of entities of  $s_2$ 
5:   return SEARCH-OVERLAP( $E_1, E_2, \langle \rangle$ )
6: function SEARCH-OVERLAP( $E_1 = \langle e_1^{(1)}, \dots, e_1^{(2)} \rangle$ ,  $E_2 = \langle e_2^{(1)}, \dots, e_2^{(n)} \rangle$ ,  $m$ )
7:   if  $E_1 == \langle \rangle$  then ▷ All entities are used in  $m$ , i.e.  $m$  is candidate matching
8:     Construct a CSP  $\Gamma$  from  $m$  that has a solution when  $\exists x \in \text{region}(l_1) \cap \text{region}(l_2)$ 
9:     if  $\Gamma$  has a solution then return false else return true
10:  for  $i = 1, \dots, n$  do
11:    if  $\text{uregion}_{l_1}(e_1^{(1)}) \cap \text{uregion}_{l_2}(e_2^{(i)}) \neq \emptyset$  then
12:       $r \leftarrow \text{SEARCH-OVERLAP}(E_1 \setminus e_1^{(1)}, E_2 \setminus e_2^{(i)}, m \odot \langle (e_1^{(1)}, e_2^{(i)}) \rangle)$ 
13:      if  $r$  is false then
14:        return false
15:  return true

```

---

**Theorem 6.** Let  $l_1 = (s_1, \gamma_1)$  and  $l_2 = (s_2, \gamma_2)$  be two lifted states. When  $\text{uregion}(l_1) \cap \text{uregion}(l_2) \neq \emptyset$ , then  $u(s_1) = u(s_2)$ .

*Proof.* Via Theorem 5, there is a matching between  $l_1$  and  $l_2$  such that  $\forall (e_1, e_2) \in m$ :  $\text{uregion}_{l_1}(e_1) \cap \text{uregion}_{l_2}(e_2) \neq \emptyset$ . Thus,  $\forall (e_1, e_2) \in m : u(e_1) = u(e_2)$ , and therefore,  $u(s_1) = u(s_2)$ .  $\square$

These properties allow to check potential overlap directly on the syntactic level: First, two lifted states can only overlap when their untyped structures are identical (Theorem 6). When this necessary condition is satisfied, we can proceed to check the necessary condition given by Theorem 5. Unfortunately, this property needs to be checked for *all* possible matches of the two lifted states. Although this number is large in the worst case (when both states have  $n$  entity structures each, there can be up to  $n!$  matches), the number of matches that actually needs to be checked is typically much lower: First, the number of *different* entity structures is typically much lower than the overall multiplicity, and thus, the number of distinct matches will also be much lower. Furthermore, the matchings can be constructed iteratively in a DFS-based approach (as outlined in Algorithm 17), such that when the condition of Theorem 6 is violated, we do not need to explore that subtree any further.

Finally, note that these are only *necessary*, but not *sufficient* conditions. Intuitively, these conditions ignore the correlation of entities that can exist when values of different entities follow a joint distribution. For example, the entities  $\langle N: \mathbf{N}_1, L: X \rangle$  and  $\langle N: \mathbf{N}_2, L: Y \rangle$  from the state  $l_1$  (Example 50) need to have a distinct  $N$  slot, as that property is distributed according to an urn without replacement with unique values.

Thus, for each matching that satisfies the necessary conditions, we need to check whether there actually exists a ground state  $x$  with  $x \in \text{region}(l_1) \cap \text{region}(l_2)$ . Here, this is done by transformation into a constraint satisfaction problem (CSP), as outlined next.

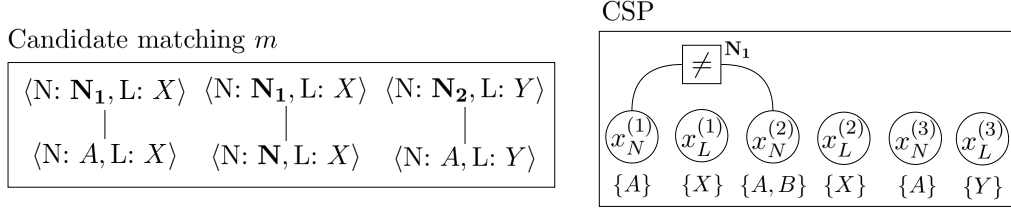


Figure F.3.: Example of a candidate matching (left), and the corresponding CSP (right). Circles denote CSP variables (with domain written below each variable), and rectangles denote constraints.

### F.2.3. Reduction to a Constraint Satisfaction Problem

Here, we discuss the remaining problem for identifying whether  $\text{uregion}(l_1) \cap \text{uregion}(l_2) \neq \emptyset$ : Given a candidate matching  $m$ , construct a ground state  $x$  with  $x \in \text{uregion}(l_1) \cap \text{uregion}(l_2)$ , if one exists.

This can be done by transforming the problem into a constraint satisfaction problem (CSP), so that a solution to the CSP directly corresponds to a state  $x \in \text{uregion}(l_1) \cap \text{uregion}(l_2)$ . The general idea is to create a CSP variable  $x_q^{(i)}$  for each property  $q$  of each pair of entities  $(e_1^{(i)}, e_2^{(i)}) \in m$ , with domain identical to the intersection of the support of the distributions  $\gamma_1(e_1.q)$  and  $\gamma_2(e_2.q)$ . This way, we ensure that an assignment of these CSP variables corresponds to a ground entity  $e^{(i)}$  with  $e^{(i)} \in \text{uregion}_{l_1}(e_1^{(i)}) \cap \text{uregion}_{l_2}(e_2^{(i)})$ .

Additionally, we need to consider the constraints that arise due to the distributions in  $\gamma$ : For example, for an urn without replacement with unique values, all values that reference that distribution need to be unique. This is modeled by pairwise inequality constraints of the corresponding CSP variables.

When the CSP has a solution, then the regions of  $l_1$  and  $l_2$  are not disjoint, and a state  $x$  that is an element of both regions can be directly constructed from the solution.

**Example 52.** Consider the example shown in Figure F.3. Suppose we have identified the matching

$$m = \langle \langle \langle N: \mathbf{N}_1, L: X \rangle, \langle N: A, L: X \rangle \rangle, \\ \langle \langle N: \mathbf{N}_1, L: X \rangle, \langle N: \mathbf{N}, L: X \rangle \rangle, \\ \langle \langle N: \mathbf{N}_2, L: Y \rangle, \langle N: A, L: Y \rangle \rangle \rangle$$

for the states  $l_1$  and  $l_2$  (see Example 50). The corresponding CSP is shown in Figure F.3 (right). From the solution of the CSP, the ground state

$$x = \llbracket 1\langle N: A, L: X \rangle, 1\langle N: B, L: X \rangle, 1\langle N: A, L: Y \rangle \rrbracket$$

can be constructed directly.

Finally, the CSP solution directly provides us with a matching  $m_x$  between entities of  $x$  and  $l_1$  (and also with a matching  $m_2$  between entities of  $x$  and  $l_2$ , that is consistent with  $m_x$ ), which is necessary for shattering, explained next.

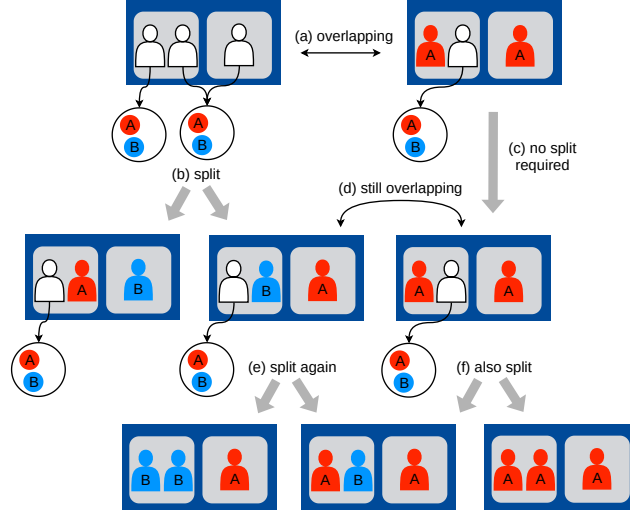


Figure F.4.: Example of shattering (see Example 53 for a more detailed explanation). The two states at the top are not disjoint (a), so they need to be split (b,c). Unfortunately, two of the split results are still overlapping (d), so they need to be split again (e,f). The resulting four states are disjoint.

#### F.2.4. Shattering

As for the case of states with identical structure, we are again interested in identifying appropriate splits so that all split results are disjoint.

Fortunately, the function IS-DISJOINT (Algorithm 17) already provides us with a ground state  $x \in \text{region}(l_1) \cap \text{region}(l_2)$  and a mapping  $m$  of the entities in  $l_1$  and  $l_2$ , which can be used to guide the splitting process.

Specifically, we select a pair  $(e_1, e_2)$  of entities from  $m$ , where for a property  $q$ ,  $\gamma_1(e_1.q) \neq \gamma_2(e_2.q)$ . By the same reasoning as in Section F.1.2, such a pair of entities must exist, as otherwise, the states  $l_1$  and  $l_2$  would be identical. Also, the distributions  $\gamma_1(e_1.q)$  and  $\gamma_2(e_2.q)$  cannot have disjoint support, due to Theorem 5. Thus, the distributions  $\gamma_1(e_1.q)$  and  $\gamma_2(e_2.q)$  are neither disjoint nor identical, i.e. they need to be split to make the states  $l_1$  and  $l_2$  disjoint.

The value on which to split is given directly by the entity  $e_x \in \text{region}_{l_1}(e_1) \cap \text{region}_{l_2}(e_2)$  of ground state  $x \in \text{region}(l_1) \cap \text{region}(l_2)$ : We split entity  $e_1$  of  $l_1$ , as well as  $e_2$  of  $l_2$ , on the constraint  $q == e_x.q$ . Intuitively, by splitting on the value  $e_x.q$ , we are dividing the lifted state into those ground states that are elements of  $\text{uregion}(l_1) \cap \text{uregion}(l_2)$ , and those that are just elements of *either*  $\text{uregion}(l_1)$  *or*  $\text{uregion}(l_2)$ . This procedure is then applied recursively on all split results, as the resulting states can still overlap, due to other properties.

**Example 53.** Consider the lifted states  $l_1 = (s_1, \gamma_1)$  and  $l_2 = (s_2, \gamma_2)$  shown in Figure F.4 (top) with

$$\begin{aligned} l_1 &= (\llbracket 2\langle N: \mathbf{N}_1, L: X \rangle, 1\langle N: \mathbf{N}_2, L: Y \rangle \rrbracket, & \langle \mathbf{N}_1: \mathcal{U}(A, B), \mathbf{N}_2: \mathcal{U}(A, B) \rangle) \\ l_2 &= (\llbracket 1\langle N: \mathbf{N}, L: X \rangle, 1\langle N: A, L: X \rangle, 1\langle N: A, L: Y \rangle \rrbracket, & \langle \mathbf{N}: \mathcal{U}(A, B) \rangle) \end{aligned}$$

Their regions overlap on the ground state

$$x = \llbracket 1\langle N: A, L: X \rangle, 1\langle N: B, L: X \rangle, 1\langle N: A, L: Y \rangle \rrbracket.$$

### F. Disjointness of Lifted States

The shattering algorithm selects  $q = N$ ,  $e_1 = \langle N: \mathbf{N}_2, L: Y \rangle$  and  $e_2 = \langle N: A, L: Y \rangle$  for splitting. Here,  $e_2$  does not need to be split, as  $e_2.q$  is already a constant. Splitting  $l_1$  on  $e_1$  with the constraint  $e_1.N == A$  results in two states

$$\begin{aligned} l'_1 &= (\llbracket 2\langle N: \mathbf{N}_1, L: X \rangle, 1\langle N: A, L: Y \rangle \rrbracket, & \langle \mathbf{N}_1: \mathcal{U}(A, B) \rangle) \\ l''_1 &= (\llbracket 2\langle N: \mathbf{N}_1, L: X \rangle, 1\langle N: B, L: Y \rangle \rrbracket, & \langle \mathbf{N}_1: \mathcal{U}(A, B) \rangle) \end{aligned}$$

Unfortunately,  $l'_1$  and  $l_2$  still have overlapping regions, so the shattering process needs to be applied recursively, as shown in Figure F.4. Finally, we arrive at four disjoint states (as opposed to the five ground states in  $\text{region}(l_1) \cap \text{region}(l_2)$ ).



# Details of Experiments

## G.1. Kitchen Scenario

Here, we provide more details on the LiMa model for the kitchen scenario (see Sections 4.5 and 6.5.4). The state is modeled so that it consists of one entity for each object that is used during the cooking process, as well as an entity for the subject, and an entity for each hand of the subject (the latter allow to easily track whether each hand is already occupied).

For the lifted state representation, the identities of the glass, plate, pot, spoon and wooden spoon are represented by an urn without replacement. The reasoning is that these objects do not always need to be distinguished, e.g. when cleaning the utensils or when setting the table. Specifically, the initial state is  $l = (s, \gamma)$  with

$$\begin{aligned}
 s = \llbracket & 3\langle \mathbf{N}: \mathbf{N}, \text{Dirty: No}, \text{Pos: Cupboard} \rangle, \\
 & 2\langle \mathbf{N}: \mathbf{N}, \text{Dirty: No}, \text{Pos: Counter} \rangle, \\
 & 1\langle \mathbf{N}: \text{Knife}, \text{Pos: Counter} \rangle, \\
 & 1\langle \mathbf{N}: \text{Cutting board}, \text{Pos: Counter} \rangle, \\
 & 1\langle \mathbf{N}: \text{Sponge}, \text{Pos: Sink} \rangle, \\
 & 1\langle \mathbf{N}: \text{Bottle}, \text{Pos: Counter}, \text{Closed: Yes} \rangle, \\
 & 1\langle \mathbf{N}: \text{Carrot}, \text{Pos: Counter}, \text{Clean: No}, \text{Cut: No}, \text{Cooked: No} \rangle, \\
 & 1\langle \mathbf{N}: \text{Cupboard}, \text{Closed: Yes} \rangle, \\
 & 1\langle \mathbf{N}: \text{Stove}, \text{SwitchedOn: No} \rangle, \\
 & 1\langle \mathbf{N}: \text{Person}, \text{Pos: Sink}, \text{Hungry: Yes}, \text{Thirsty: Yes}, \text{Seated: No} \rangle, \\
 & 2\langle \mathbf{N}: \text{Hand}, \text{Free: Yes} \rangle \rrbracket, \\
 \gamma = & \langle \mathbf{N}: \mathcal{U}(\text{Glass}, \text{Plate}, \text{Pot}, \text{Spoon}, \text{Wooden Spoon}) \rangle.
 \end{aligned}$$

For each of the 16 action classes that subjects can perform (see Figure G.2 for all action classes), there is a corresponding action in the PMPMRS that describes the transition model. Additionally, a number of more specific actions have been designed, that cover behaviors that have been observed but are not already modeled by the 16 general-purpose actions. For example, the general *put* action assumes that an object is put *at* a specific location (the current location of the agent), but there are also cases where an object *a* is put *inside*

## G. Details of Experiments

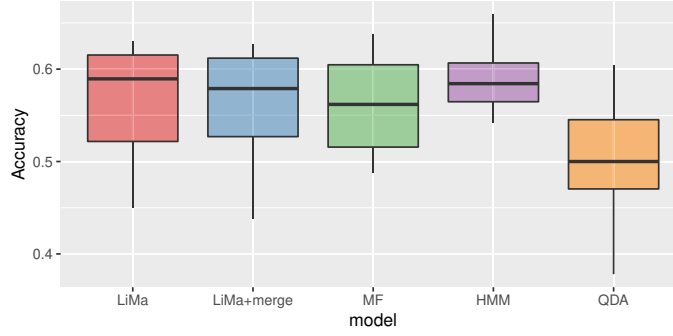


Figure G.1.: Accuracy (of predicting the activity class) of LiMa (using at most 2000 states), the ground marginal filter, and the baseline classifiers for the kitchen scenario.

another object  $b$ , such that moving  $b$  will also move  $a$ , which is modeled by an additional *put-inside* action.

The weights of actions have been chosen according to a goal distance heuristic, that is based on the task script shown in Table G.1. The intuition here is that people act goal-directed in this scenario, i.e. they have a higher chance to execute an action that will decrease the goal distance. The same concepts are used in Computational Causal Behavior Models (CCBMs) [120].

Specifically, such an action selection mechanism requires to slightly generalize the definition of actions (Definition 6): Instead of positive real numbers, a weight is a function  $\kappa : \mathcal{S} \times \Gamma \times \langle \mathcal{E} \rangle \rightarrow \mathbb{R}_{\geq 0}$  of the prior state and the bound entities to the positive real numbers. Here, we set  $\kappa(s, \gamma, e) = \exp(-d(f(s, \gamma, e)))$ , where  $f$  is the effect function of the action (i.e.  $f(s, \gamma, e)$  is the posterior state after applying the action to state  $l$  with bound entities  $e$ ), and  $d$  is the step of the task script shown in Table G.1.

In addition, we compared the accuracy of action class recognition between LiMa and two baseline classifiers: Quadratic discriminant analysis (QDA) and a hidden Markov model (HMM). The parameters of the QDA and the HMM have been estimated by the standard maximum likelihood estimators, using the labeled training data. As the rest of the evaluation is not done as a cross validation (which would have been methodologically infeasible considering the knowledge-based construction of the PMPMRS), the baseline models are also trained on the complete dataset. Figure G.2 shows the confusion matrices of LiMa (with and without merging) and the baseline classifiers, and Figure G.1 shows corresponding accuracies. There is no significant difference of accuracy between LiMa, the ground marginal filter (MF), and the HMM. The fact that there is no significant difference between LiMa and MF comes from the fact that both algorithms are saturated with 2000 particles (see Figure 4.11). The HMM (with 16 states) has 272 parameters – as no cross validation is performed, we suspect that the HMM is subject to overfitting, and thus accuracy is overestimated.

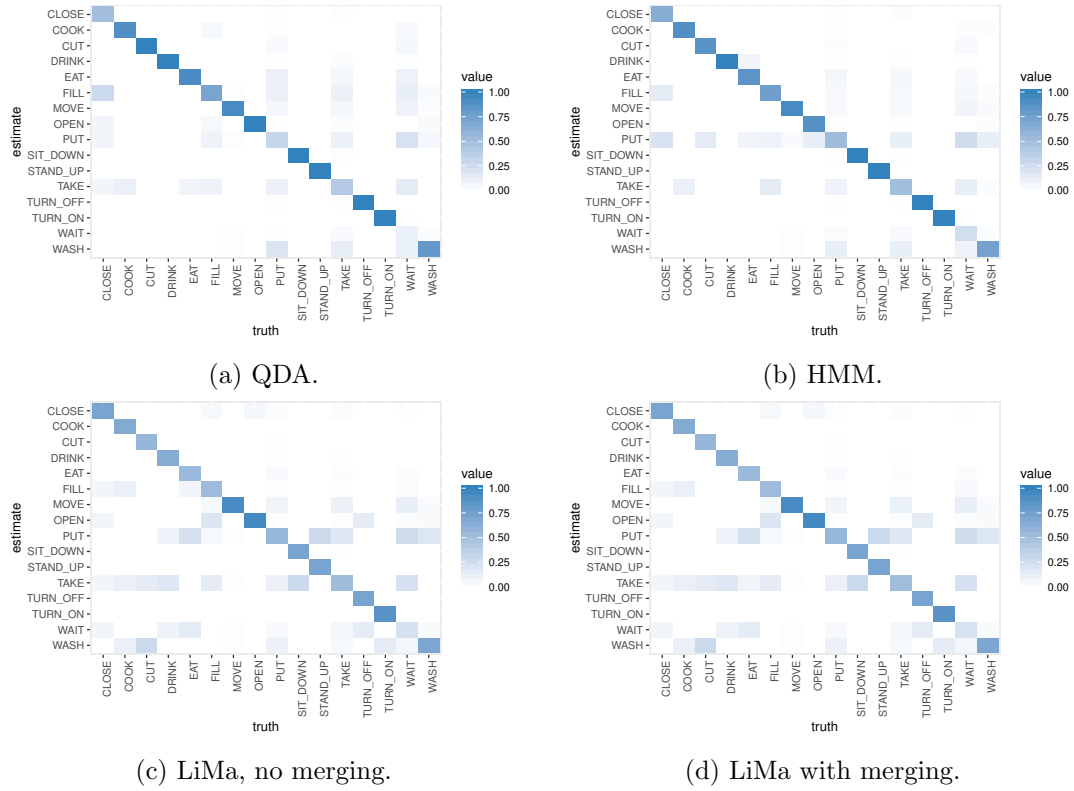


Figure G.2.: Confusion matrices for the kitchen scenario.

Step	$d$	Task	State predicate if task not fulfilled
1	14	Clean hands	– (no state predicate with lower step count matches)
2	13	Get food to sink	not clean food, holds food, not at sink
3	12	Clean food	not clean food, holds food, at sink
4	11	Cut food	clean food, not food prepared
5	10	Turn stove on	food prepared, not cooked, stove off
6	9	Cook food	food prepared, not cooked, stove on
7	8	Turn oven off	hungry, cooked, stove on
8	7	Finish setting table & Sit down	hungry, cooked, stove off, not seated
9	6	Enjoy meal	hungry, stove off, seated
10	5	Get up	not hungry, seated
11	4	clean kitchen utensil (ku)	not hungry, not seated, 4 ku dirty
12	3	clean ku	not hungry, not seated, 3 ku dirty
13	2	clean ku	not hungry, not seated, 2 ku dirty
14	1	clean ku	not hungry, not seated, 1 ku dirty
15	0	done	

Table G.1.: Task script for the kitchen scenario. Reproduced from Krüger et al. [123].

## G. Details of Experiments

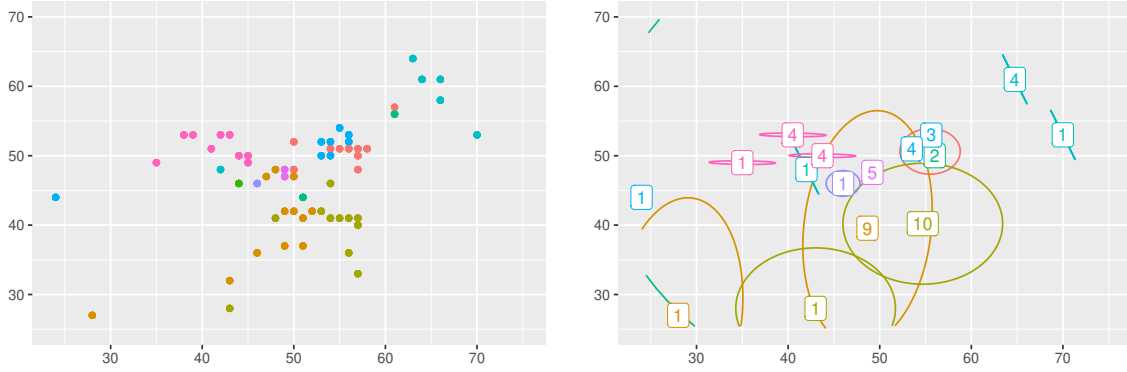


Figure G.3.: Example of fitting a Gaussian mixture to obtain the lifted representation for the Travian scenario. Left: Original village locations, different colors denote different players. Right: Lifted state representation. Each ellipse denotes location distribution of an entity structure, the numbers denote their multiplicities.

### G.2. Learning a Lifted Representation by Gaussian Mixture Fitting for the Travian Scenario

In this section, we describe in more detail how the lifted initial state is estimated in the Travian scenario (Section 6.4).

The dataset  $D$  that was obtained from a real game server consists of tuples  $(v, o)$ , where  $v$  denotes the two-dimensional position of a village on the grid map, and  $o$  denotes the owner of the village. This situation could be modeled as a ground state  $x$ , where each tuple  $(v, o)$  corresponds to an entity  $e = \langle P: o, L: v \rangle$ . Our goal here is to obtain a lifted state  $l$  that is a “good” description of this game state, but has fewer distinct entities.

This is done by assuming that the data  $D$  is a sample of an underlying distribution of village locations, and that this distribution can be described by a lifted state. The lifted state can then be obtained by estimating the parameters of this distribution.

Recall (as shown in Equation 6.20) that the distribution over all data points  $v_{1:N}$  with a fixed owner  $o$  can be approximated as:

$$p(v_{1:N} | s, \gamma) = \prod_{j=1}^N \sum_{(d_i, \rho_i) \in \gamma} \frac{n_i}{N} p_{\rho_i}(v_j),$$

with  $p_{\rho} \sim \mathcal{N}(\mu_i, \Sigma_i)$ . This distribution is a Gaussian mixture, and parameters (i.e. mixture weights  $n_i$ , mean vectors  $\mu_i$  and covariance matrices  $\Sigma_i$ ) can be estimated by expectation maximization for each player  $o$ . Specifically, we used the R package `mclust` [195] for expectation maximization, and assumed that all components of a given owner share a single covariance matrix  $\Sigma$ .

Constructing the corresponding lifted state  $l = (s, \gamma)$  from those parameters is straightforward: For each mixture component with parameters  $\mu_i$  and  $\Sigma$ ,

- (i) create a representation  $\rho_i = \mathcal{N}(\mu_i, \Sigma)$ ,
- (ii) create a new distribution label  $d_i$ ,
- (iii) insert  $\langle d_i : \rho_i \rangle$  into  $\gamma$ ,

*G.2. Learning a Lifted Representation by Gaussian Mixture Fitting for the Travian Scenario*

- (iv) create an entity structure  $e_i = \langle \text{P: } o, \text{L: } d_i \rangle$ ,
- (v) set the multiplicity  $m_i$  of  $e_i$  as the number of samples  $(v, o)$  that most likely belong to mixture component  $i$ , i.e. for which the likelihood  $p_{\rho_i}(v)$  is maximal, and
- (vi) insert  $e_i$  with multiplicity  $m_i$  into  $s$ .

Figure G.3 visualizes this estimation procedure for one of the datasets that was actually used in the experiments with 10 players and 71 villages (only a part of the map is shown).



# Curriculum Vitae

Name	Stefan Lüdtke
Date of Birth	August 05, 1991
Place of Birth	Rostock
Nationality	German

## Experience and Education

since 10/2016	Research Associate, University of Rostock, Germany
10/2014 – 10/2016	Master in Computer Science, University of Rostock
10/2011 – 10/2014	Bachelor in Computer Science, University of Lübeck
2002 – 2010	High-school diploma ( <i>Abitur</i> ), Gymnasium Sanitz





## *G.2. Learning a Lifted Representation by Gaussian Mixture Fitting for the Travian Scenario*