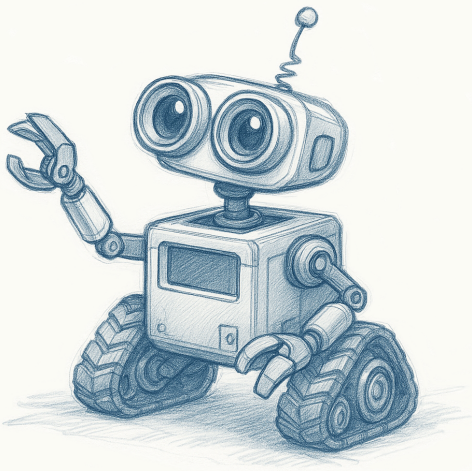
KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

# Relational Neurosymbolic Sequential Reasoning



**Gabriele Venturato**

Dissertation presented in partial fulfillment
of the requirements for the degree of
Doctor of Engineering Science (PhD):
Computer Science

Supervisor:
Prof. dr. Luc De Raedt

December 2025

# RELATIONAL NEUROSYMBOLIC SEQUENTIAL REASONING

Gabriele VENTURATO

Supervisor:
Prof. dr. Luc De Raedt

Members of the
Examination Committee:
Prof. dr. Michiel Steyaert, chair
Prof. dr. Tinne De Laet
Prof. dr. Giuseppe Marra
Prof. dr. Hector Geffner
(RWTH Aachen)
Prof. dr. Kurt Driessens
(Maastricht University)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Computer Science

December 2025

Uitgegeven in eigen beheer, Gabriele Venturato, Parkstraat 16, 3000 Leuven (Belgium)

# Acknowledgments

> "You miss 100% of the shots you don't take. -*Wayne Gretzky*"
>
> *Michael Scott*

Doing a PhD was tougher than I expected. There were many moments when I doubted myself, my abilities, and my choices. There were times when I felt like giving up. If I didn't, it's probably because of a fine mix of optimism and delusion, and because I had incredible people around me who made it all worthwhile. To them, I owe my deepest thanks.

I am very grateful to my supervisor, Luc De Raedt. He is not only a great professor, but also a remarkably supportive and understanding person who values balance as much as achievement, and who trusts people to find their own way. Luc, thank you for believing in me, for giving me the freedom to explore my ideas, and for your constant support and encouragement. Your guidance has been invaluable throughout this journey.

I would like to thank Prof. Steyaert for chairing the jury, and Profs. De Laet, Marra, Geffner, and Driessens for their insightful questions and comments, which helped improve this thesis. I am honoured to have you on my examination committee.

I could not have done this work without my co-authors. There is a well-known idea in jazz circles that if you are the best musician in the band, you are in the wrong band. With each of you, I never felt I was in the wrong band. I would like to believe that you learned something from me, just as I learned from you, but I am certain the balance is tilted in your favour. Thank you, Vincent, Pedro, Lennert, David, Rishi, Luca, Nicola, Gavin, Angelo, and Giuseppe.

I had the fortune of sharing this journey with many friends and colleagues who made the hard times easier and the good times unforgettable. While thinking

about whom to thank, I realised how hard it is to draw a line between friends and colleagues, and how impossible it is to thank each of you for something specific, since so many of you are part of multiple corners of my life. I will do my best to mention as many of you as I can, but please forgive me if I forget anyone. At my age, my memory is not what it used to be.

To the people I have shared an office with over the years: Adem, we started on the same day, cleaned up the office together, and helped each other keep our mental sanity during COVID times; Sieben, we shared many laughs and frustrations, as well as a passion for board games, music, and even our birthdays, although you are still a youngster compared to me; Jessa, you are my flour dealer, my gossip source, and the reason why I know far too much about breast pumps; David, when you joined I liked to joke with Matilde that I was probably spending more time with you than with her, and that didn't get better once we started playing games together, thank you for reminding me every morning what games I should play, and for the free Dutch lessons; Nick, even if you abandoned us and sold your soul to finance, I forgive you, because your tool said I am the second-best expert in neurosymbolic AI at KU Leuven; and Quinten, the last to join our office, you always have the perfect English term and pull out the best unexpected jokes, one day you will manage to work on constraints, I'm sure. To Pietro and Lorenzo "the Wise", you literally helped me carry my luggage to my place on my first day in Leuven, and that was just the first of many times you helped and advised me over the years. Thank you for always being there, and to all the other Italians who joined afterwards: Lorenzo, Luca, Marco, Stefano, Andrea, Eleonora, and many, many more. To all the other colleagues and friends at DTAI and beyond, Robin, Jaron, Laurens, Mariette, Rik, Lucas, Thomas W., Thomas "chess", Thomas S., Ying, Annegret, Ignace, Orestis, Senne, Jay, Irfan, Daan, Simiao, Pieter R., Pieter D., Timo, Jose, Kostis, Jens, Stella, Wannes, Maaike, Loren, Dimos, Jonas, Paolo, Kshitij, Mohit, Nitesh, Emanuele, Ilias, and many more. Thank you for the stimulating discussions, the fun coffee breaks, the travelling, the parties, the DnD sessions, the (board) games, the dinners, the climbing, the running, the swimming, the skiing, and all the other activities that made my time here so enjoyable.

To the friends I made along the way, Bennie and Matthias (we still need to do the *LotR* marathon for the extended editions), Roberto, Niccolò (sorry for all the times I was too busy to climb), Sara, Dominique, Roger, Athina, Jackie, Hande, Lucas, Renata, Florian (thank you for introducing me to DnD and for being an amazing DM), Giovanni (you taught me so much about cinema, thank you for all the movie nights), Sylvia, Caterina (every time I see something Shrek-related, I think of you), Adrià, Melissa, Björn, Justine, Robbe, and Ine (one day I will learn how to dance salsa). Thank you all for countless hours spent together, whether playing games, watching movies, having dinners, or just eating animal

cookies. To the *Pistolëëri*, Mara, Monica, Giorgio, Alessia Fenicottero, Francis, Alice, Hector, Marco, Matteo, Alessia. Above all, I have to thank you because thanks to you I met Matilde. But beyond that, thank you for all the parties, karaoke nights, and dinners that made my time in Leuven so special. Even if I am not the most active member, you've been an important part of my life here. To the *Porto* gang, Simo, Maestra P., Dedo, Gio, and Eli. I got to know you through Matilde, but you quickly became my friends too. Thank you for all the fun times, the dinners, the games, and the summer holidays. To my bandmates, past and present, Claudio, Serena, Giacomo, Walteriño, Walter, Mariette, Mattiash, and Nico. Thank you for the music, the rehearsals, the gigs, and all the memories we created together, on stage and beyond.

To the friends from my high school and university years. To *PlankUniud*, Nico, Letizia, Checco, Andrea, Erica, Matteo, and Rebecca. To my favourite couple, Greta and Ale. To my favourite cousin, Alice (don't tell the others). To *I Migliori*, Giulia and Buri. To the *Gruppo Mensa*, Leo, Samuel, Thea, Gian, and Emma. When I decided to move to Belgium for my PhD, I severely underestimated how much I would miss you all. It hurts not being able to see you as often as I would like, but I love how, whenever we meet, it feels like no time has passed. Thank you all, from the bottom of my heart, for being there for me, even from afar. And to all the other people who were part of my life at some point: you know who you are, and I thank you too.

I would like to thank my parents, because if I am here today, it is first and foremost thanks to them. *Grazie mamma per avermi insegnato l'attenzione per i dettagli, grazie papà per avermi insegnato il valore del lavoro.* My family is not only my parents though. I was lucky enough to grow up with a sister and two brothers. Michela, Riccardo, and Samuele, I cannot think of a better way to thank you than by quoting a Disney film: "Ohana means family. Family means nobody gets left behind or forgotten". There will always be a place for you *nel porcile* whenever you want to visit. My family is also my extended family. To Do, and to the whole "Ruzze's family", thank you for making me feel at home from the very first day I met you.

Finally, to Matilde. I "kiss my elbows" every day for having such an amazing girlfriend by my side. You have been my rock, my confidant, and my biggest supporter throughout this journey. Your love, patience, and understanding have meant the world to me. Thank you for believing in me, for encouraging me when I doubted myself, and for celebrating every milestone with me. I am incredibly lucky to have you in my life, and I look forward to many more adventures together.

# Popularized Abstract

Artificial Intelligence has made tremendous progress in recent years. From self-driving cars to smart assistants, machines are becoming better at seeing the world, understanding it, and making decisions. But many of these systems rely purely on data-driven approaches, like deep learning, which, while powerful, often behave like black boxes. They struggle to explain their decisions, cannot easily incorporate prior knowledge, and often fail when faced with unfamiliar situations.

This dissertation explores a different approach, one that combines the strengths of traditional symbolic AI (based on logic and rules) with the learning capabilities of modern neural networks. The aim is to create AI agents that can perceive their environment (for example, by interpreting camera images), reason about what is happening using structured knowledge, and make decisions even when things are uncertain or partially unknown.

To achieve this, the dissertation introduces new models and tools that help these so-called "neurosymbolic" agents operate effectively in dynamic and unpredictable environments. One project develops a planning system that uses logic and probability to make decisions, while still allowing the model to learn from experience. Another contribution introduces a model that learns how things change over time while following logical constraints, like predicting how objects move or how goals can be achieved step by step. A third system enhances reinforcement learning agents by enforcing logical safety rules during training, helping them learn safer and more interpretable behaviours.

Together, these contributions point toward a new generation of AI systems: ones that can learn from data like a neural network but reason and generalise like a human. While many challenges remain, this work takes key steps toward making AI not just more powerful, but also more trustworthy, interpretable, and reliable in real-world scenarios.

# Gepopulariseerde Samenvatting

Kunstmatige intelligentie heeft de afgelopen jaren enorme vooruitgang geboekt. Van zelfrijdende auto's tot slimme assistenten: machines worden steeds beter in het waarnemen van de wereld, het begrijpen van situaties en het nemen van beslissingen. Veel van deze systemen zijn echter puur gebaseerd op datagedreven methoden, zoals deep learning. Hoewel deze technieken krachtig zijn, functioneren ze vaak als een soort black box. Ze kunnen hun beslissingen moeilijk uitleggen, het is lastig om er bestaande kennis in te verwerken, en ze falen vaak in onbekende situaties.

Dit proefschrift onderzoekt een andere aanpak, die de sterke punten van traditionele symbolische AI (gebaseerd op logica en regels) combineert met het leervermogen van moderne neurale netwerken. Het doel is om AI-agenten te ontwikkelen die hun omgeving kunnen waarnemen (bijvoorbeeld door camerabeelden te interpreteren), logisch kunnen redeneren over wat er gebeurt, en beslissingen kunnen nemen, zelfs wanneer de situatie onzeker of onvolledig is.

Om dat te bereiken, worden er in dit werk nieuwe modellen en tools voorgesteld die zulke zogenaamde "neurosymbolische" agenten helpen om effectief te opereren in dynamische en onvoorspelbare omgevingen. Eén project ontwikkelt een planningssysteem dat logica en waarschijnlijkheid combineert om beslissingen te nemen, waarbij het model toch van ervaring kan blijven leren. Een ander model leert hoe dingen in de tijd veranderen, terwijl het tegelijkertijd logische beperkingen respecteert — zoals het voorspellen van hoe objecten zich verplaatsen of hoe doelen stap voor stap bereikt kunnen worden. Een derde systeem versterkt reinforcement learning-agenten door tijdens het leren logische veiligheidsregels af te dwingen, zodat ze veiliger en beter te interpreteren gedrag aanleren.

Samen wijzen deze bijdragen in de richting van een nieuwe generatie AI-systemen: systemen die leren van data zoals neurale netwerken, maar redeneren en generaliseren zoals mensen. Hoewel er nog veel uitdagingen zijn, zet dit werk belangrijke stappen richting AI die niet alleen krachtig is, maar ook betrouwbaarder, beter uitlegbaar en geschikter voor gebruik in de echte wereld.

# Abstract

Developing intelligent agents that can operate reliably in uncertain and dynamic environments requires more than raw predictive power. These agents must be able to perceive the world from high-dimensional sensory data, such as images or text, sequentially reason over structured representations of their environment, make informed decisions under uncertainty, and generalise across novel tasks, objects, or contexts. While deep learning excels at pattern recognition from data, it struggles to incorporate prior knowledge, provide interpretability, or guarantee robustness outside the training distribution. On the other hand, symbolic reasoning offers structure and guarantees but lacks scalability and the ability to deal with raw perceptual inputs.

This dissertation addresses these limitations by advancing neurosymbolic AI (NeSy) towards sequential decision making. We explore how to unify probabilistic reasoning, logical structure, and neural perception into cohesive models that can act, learn, and reason over time. The focus is on building agents that can combine subsymbolic perception with structured probabilistic models, handle partial or missing knowledge, and operate under logical constraints, all within a principled and differentiable framework.

The first part of the dissertation introduces *dynamic decision circuits* (DDCs), a framework for planning under uncertainty using compiled arithmetic circuits derived from sequential probabilistic graphical models. These circuits represent sequential decision problems in a symbolic and algebraic form while allowing for efficient and differentiable inference. This enables agents to reason about future states and rewards, and to learn model parameters from data through gradient-based optimisation. This contribution demonstrates how knowledge compilation techniques can be extended to sequential settings, bridging the gap between model-based decision-making and deep learning.

Next, we propose *relational neurosymbolic Markov models* (NeSy-MMs), a new class of differentiable models that combine neural perception, relational

logic, and probabilistic reasoning in time-dependent domains. NeSy-MMs use neural modules to map raw inputs to symbolic representations, which are then processed by a probabilistic logical model that evolves over time. The system supports both generative and discriminative tasks, accommodates partially specified domain knowledge, and allows for the end-to-end learning of both neural and symbolic components via a differentiable particle filter. Logical constraints are enforced during inference, allowing for interpretability, formal guarantees, and strong generalisation across entity configurations.

Finally, the third part of the dissertation investigates how neurosymbolic reasoning can be applied in online learning settings, such as reinforcement learning. We introduce a codebase that integrates *probabilistic logic shields* with deep RL agents in the MiniHack environment. These shields act as runtime guards, renormalising unsafe or undesirable actions based on symbolic constraints. This approach improves both safety and sample efficiency, while preserving compatibility with standard RL algorithms. A web-based platform showcases the resulting agents and allows broader access to neurosymbolic RL techniques.

Collectively, these contributions offer building blocks for a new generation of AI systems that learn from data, reason under uncertainty, adapt to new situations, and provide structured, interpretable behaviour. They demonstrate how symbolic and subsymbolic methods can be effectively combined in sequential settings, and lay the groundwork for future AI agents that are robust, safe, and capable of principled generalisation.

# Beknopte Samenvatting

Het ontwikkelen van intelligente agenten die betrouwbaar kunnen opereren in onzekere en dynamische omgevingen vereist meer dan puur voorspellend vermogen. Deze agenten moeten in staat zijn om de wereld waar te nemen aan de hand van hoog-dimensionale sensorgegevens, zoals beelden of tekst, sequentieel te redeneren over gestructureerde representaties van hun omgeving, geïnformeerde beslissingen te nemen onder onzekerheid, en te generaliseren naar nieuwe taken, objecten of contexten. Hoewel deep learning uitblinkt in patroonherkenning op basis van data, is het moeilijk om voorafgaande kennis te integreren, interpretaties te bieden of robuustheid buiten de trainingsdata te garanderen. Symbolisch redeneren daarentegen biedt structuur en garanties, maar mist schaalbaarheid en het vermogen om met ruwe waarnemingen om te gaan.

Dit proefschrift pakt deze beperkingen aan door neurosymbolische AI (NeSy) uit te breiden naar sequentiële besluitvorming. We onderzoeken hoe probabilistisch redeneren, logische structuur en neurale perceptie kunnen worden geïntegreerd in samenhangende modellen die kunnen handelen, leren en redeneren over tijd. De focus ligt op het bouwen van agenten die subsymbolische waarnemingen kunnen combineren met gestructureerde probabilistische modellen, die met gedeeltelijke of ontbrekende kennis kunnen omgaan, en die onder logische beperkingen kunnen opereren, alles binnen een principieel en differentieerbaar kader.

Het eerste deel van het proefschrift introduceert *dynamische beslissingscircuits* (DDCs), een kader voor planning onder onzekerheid op basis van gecompileerde rekenkundige circuits afgeleid van sequentiële probabilistische grafische modellen. Deze circuits representeren sequentiële besluitvormingsproblemen in symbolische en algebraïsche vorm, terwijl ze efficiënte en differentieerbare inferentie mogelijk maken. Dit stelt agenten in staat om te redeneren over toekomstige toestanden en beloningen, en om modelparameters te leren op basis van data via gradiëntgebaseerde optimalisatie. Deze bijdrage toont aan hoe kenniscompilatie kan worden uitgebreid naar sequentiële contexten, en overbrugt zo de kloof tussen

modelgebaseerde besluitvorming en deep learning.

Vervolgens introduceren we *relationele neurosymbolische Markov-modellen* (NeSy-MMs), een nieuwe klasse differentieerbare modellen die neurale perceptie, relationele logica en probabilistisch redeneren combineren in tijdsafhankelijke domeinen. NeSy-MMs gebruiken neurale modules om ruwe invoer te vertalen naar symbolische representaties, die vervolgens worden verwerkt door een probabilistisch logisch model dat zich in de tijd ontwikkelt. Het systeem ondersteunt zowel generatieve als discriminerende taken, kan omgaan met gedeeltelijk gespecificeerde domeinkennis, en maakt end-to-end leren van zowel neurale als symbolische componenten mogelijk via een differentieerbaar deeltjesfilter. Logische beperkingen worden afgedwongen tijdens inferentie, wat leidt tot interpreteerbaarheid, formele garanties en sterke generalisatie over entiteiten heen.

Ten slotte onderzoekt het derde deel van het proefschrift hoe neurosymbolisch redeneren kan worden toegepast in online leeromgevingen, zoals reinforcement learning. We introduceren een codebase die *probabilistische logische schilden* integreert met diepe RL-agenten binnen de MiniHack-omgeving. Deze schilden fungeren als runtime-waakhonden en hernormaliseren onveilige of ongewenste acties op basis van symbolische beperkingen. Deze aanpak verbetert zowel de veiligheid als de sample-efficiëntie, terwijl de compatibiliteit met standaard-RL-algoritmen behouden blijft. Een webplatform toont de resulterende agenten en maakt bredere toegang tot neurosymbolische RL-technieken mogelijk.

Gezamenlijk bieden deze bijdragen bouwstenen voor een nieuwe generatie AI-systemen die leren uit data, redeneren onder onzekerheid, zich aanpassen aan nieuwe situaties en gestructureerd, interpreteerbaar gedrag vertonen. Ze tonen aan hoe symbolische en subsymbolische methoden effectief kunnen worden gecombineerd in sequentiële contexten, en vormen zo de basis voor toekomstige robuste en veilige AI-agenten die principieel kunnen generaliseren.

# List of Abbreviations

**mapl-cirup** Markov planning with circuit updates. xxii, xxvii, 7, 29, 35–44, 47

**2-SAT** 2-Satisfiability problem. 84

**3-SAT** 3-Satisfiability problem. xxiv, xxv, 83–88, 95

**ADD** Algebraic decision diagram. 40, 47

**AI** Artificial intelligence. 1–7, 11, 12, 49, 56, 85, 93, 97

**AMC** Algebraic model counting. 17–20, 33–35

**BN** Bayesian network. 3, 17, 21, 24, 28, 29

**BROPOP** Belief Re-use in Online Partially Observable Planning. xxii, 39, 40

**CDCL** Conflict-driven clause learning. 86

**CNF** Conjunctive normal form. xxiv, xxv, 85, 87, 90

**CoT** Chain of thought. 84

**DAG** Directed acyclic graph. 18

**DBN** Markovian dynamic Bayesian network. 4, 21, 51

**DDC** Dynamic decision circuit. ix, xi, 29, 34–36, 38, 47, 94

**DDN** Dynamic decision network. xxii, 7, 24, 28–38, 42

**Deep-HMM** Deep hidden Markov model. xxv, xxvii, xxviii, 62–64, 66, 67, 99, 100, 104–106

**DPLL** Davis-Putnam-Logemann-Loveland algorithm. xxv, 86, 87

**HMM** Hidden Markov model. xxii, 4, 21, 51

**KC** Knowledge compilation. xxii, xxvii, 4, 17, 18, 20, 28–30, 41, 42, 44

**LLM** Large language model. 6, 83–89, 95

**LRM** Large reasoning model. 6, 83, 85, 95

**LSH** Locality-sensitive hashing. xxii, 39, 40

**LTL** Linear temporal logic. 97

**MCTS** Monte Carlo tree search. 39, 40

**MDP** Markov decision process. xxii, 4, 7, 20, 21, 23–25, 28, 29, 35, 37, 38, 40, 42, 71

**MEU** Maximum expected utility. 19

**ML** Machine learning. 2

**MLP** Multi-layer perceptron. 100, 108

**NDF** Neural distributional fact. 14

**NeSy** Neurosymbolic AI. ix, xi, xxi, xxiv, 5, 7, 11, 12, 20, 49–53, 55, 56, 60, 61, 63, 65–67, 80, 81, 93, 97

**NeSy-MM** Relational neurosymbolic Markov model. ix, x, xii, xxiv, xxviii, 7, 51–53, 55, 56, 59–62, 64, 65, 67, 80, 81, 93, 95, 96, 99–101, 104–106

**OOD** Out of distribution. xxvii, 63, 64, 66, 67

**PCF** Probabilistic comparison formula. 14

**PGM** Probabilistic graphical model. 3, 4, 12

**PLP** Probabilistic logic programming. 12–14, 72

**PLPG** Probabilistic logic policy gradient. 72, 73

**PLS** Probabilistic logic shield. 71

**POMDP** Partially observable Markov decision process. 4, 38–40

**PPO** Proximal policy optimization. xxiv, 26, 72, 77, 82, 108

**RA** Reconstruction accuracy. xxvii, 62, 64

**RBPF** Rao-Blackwellized particle filter. 23, 56, 58, 60

**RE** Reconstruction error. xxvii, 62, 64

**ReLU** Rectified linear unit activation function. 100, 101, 108

**RL** Reinforcement learning. xxiv, 4, 8, 25, 26, 69–72, 76, 80, 81, 94, 96, 97

**RLOO** REINFORCE leave-one-out estimator. 57, 58

**SAT** Boolean satisfiability problem. xxiv, xxv, 85–87

**SPUDD** Symbolic planning using decision diagrams. xxii, xxvii, 40–43, 47

**StarAI** Statistical relational AI. 4, 5, 11, 12

**VAE** Variational autoencoder. 53, 99

**VI** Value iteration. xxii, xxvii, 35, 41, 42, 44

**VT** Variational transformer. 62, 64, 65, 104, 105

**WMC** Weighted model counting. 4, 17, 18

**WMI** Weighted model integration. 12

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

> "We choose to go to the moon in this decade and do the other things, not because they are easy, but because they are hard."
>
> *John F. Kennedy*

Artificial Intelligence (AI) has long aimed to build machines capable of intelligent behaviour (Russell and Norvig 2020). This means we would like to have agents that can perceive and understand the world, reason about it, and eventually make decisions that guide their actions within the environments where they are deployed. Crucially, this must be achieved while dealing with the inherent uncertainty arising from noisy sensors, or from lack of knowledge about the underlying rules that govern real-world dynamics. Figure 1.1 provides a high-level schematic of the main components that we consider essential for building an intelligent agent. In this thesis, we will delve into each of these components, examining how they work together to address all the crucial aspects of perception, reasoning, learning, and decision-making under uncertainty.

Early AI systems relied heavily on *symbolic methods*: formal logic (Gallier 1985; Genesereth and Nilsson 1987; Lloyd 1984), expert systems (Buchanan et al. 1969; Feigenbaum et al. 1970), and structured planning algorithms (Fikes and Nilsson 1971). These systems provided a high degree of interpretability and could enforce correctness through formal proofs. However, their rigidity and inability to handle raw subsymbolic inputs, *e.g.* images, limited their applicability, leading eventually to the so-called "AI winter" in the late 1980s. Then, it became clear that the complexity of the real-world was difficult to tame. Partially, because

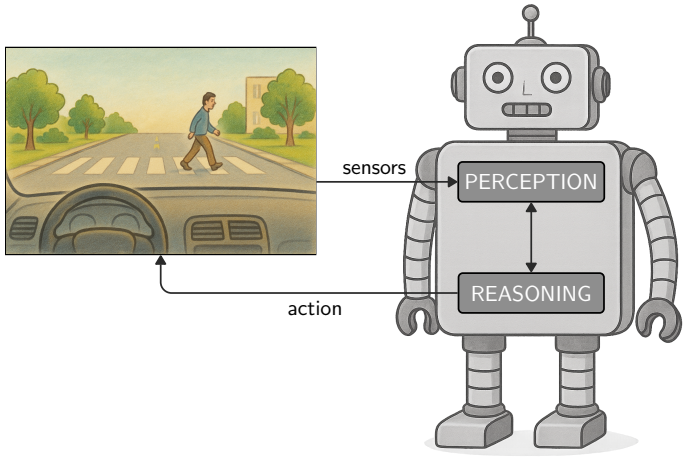



Figure 1.1: Schematic representation of the main components we consider in this thesis to build an intelligent agent. The perception module processes the raw data from the environment, mapping subsymbolic inputs (*e.g.* images from an autonomous vehicle) to symbolic representations such as 'pedestrian' or 'crossroad'. Given that sensors can be noisy, this process introduces uncertainty about the observed objects. The reasoning module then uses this probabilistic symbolic information, along with prior knowledge and rules, to infer the current state of the world and plan sequences of future actions that adhere to desired constraints, such as safety. However, the agent's knowledge can be incomplete, and it must be capable of learning from experience to fill these gaps. For example, the dynamics of a pedestrian crossing might be initially unknown but can be learned from interactions over time. This reasoning should also be relational, allowing the agent to generalize across entities and contexts, such as applying the same rule to stop regardless of the number of pedestrians present. Additionally, notice that the perception module can be influenced by the reasoning module, allowing the agent to refine its predictions based on prior knowledge, such as anticipating a pedestrian near a crossroad.

these systems struggled when faced with the uncertainty of the real world, and partially because they were not able to learn from experience (Russell and Norvig 2020).

With the advent of machine learning (ML) and, more recently, deep learning, AI experienced a paradigm shift. Instead of relying on symbols and hand-crafted rules, ML systems learn from high-dimensional *subsymbolic inputs*, automatically discovering patterns and representations that enable them to

perform complex tasks. Deep learning models have shown remarkable success in a variety of domains, including computer vision (Szeliski 2022), natural language processing (Jurafsky and Martin 2025), and reinforcement learning (Sutton and Barto 2018). However, they often operate as black boxes, lacking interpretability and the ability to incorporate prior knowledge. Moreover, they poorly generalise to domains that differ significantly from their training data. This raises concerns about their reliability and safety, especially in high-stakes applications such as healthcare, autonomous driving, or robotics.

In this thesis we will focus on the interaction between symbolic reasoning and (subsymbolic) deep learning, and how to leverage the strengths of both paradigms to build more capable and trustworthy AI systems.

**Probabilistic graphical models and Bayesian reasoning.** A first necessary step in this direction is to provide a principled way to reason about uncertainty. In the late 1980s, Judea Pearl's work on Bayesian networks (BNs) (Pearl 1988), led to a rigorous and efficient framework for representing and reasoning about uncertain knowledge. More generally, probabilistic graphical models (PGMs) (Koller and Friedman 2009) are either directed or undirected graphs that encode a joint probability distribution $P(X_1, \ldots, X_n)$ over $n$ random variables. The key insight is that nodes represent random variables, edges represent probabilistic dependencies, and the structure of the graph tells us how to factorise the probability distribution, so that it can be exploited in probabilistic inference algorithms, like belief propagation (Pearl 1988) or variable elimination (Dechter 1999). Therefore, PGMs can be used to model complex relationships between variables and perform inference to compute probabilities of interest.

**The challenge of sequential decision making.** Many real-world problems are inherently sequential. An autonomous vehicle must plan a safe route through changing traffic conditions; a robotic assistant needs to sequence its actions to clean a room or assist a human; a dialogue system must choose its next response based on prior conversation. In such domains, an agent must make a series of interdependent decisions under uncertainty, balancing short-term outcomes against long-term objectives. These problems are challenging, and they notoriously suffer from two main curses. First, the state space grows exponentially with respect to the variables considered. This is the so-called *curse of dimensionality* (Kaelbling et al. 1998). Second, the number of possible future trajectories increases exponentially with the planning horizon. This is known as the *curse of history* (Pineau et al. 2006).

Markov decision processes (MDPs) (Puterman 1994), together with their partially observable counterpart (POMDPs) (Kaelbling et al. 1998), are powerful mathematical frameworks for modelling sequential decision making under uncertainty. They consist of a set of states, actions, transition probabilities, and rewards, which together define the dynamics of the environment. The goal is to find a policy that maximises the expected cumulative reward over time. Sutton (1988) introduced the connection between the theory of MDPs and reinforcement learning (RL), which allows agents to learn optimal policies approximately through trial and error, using feedback from the environment. In the same years, researchers were already using hidden Markov models (HMMs) (Baum and Petrie 1966) to model sequential data, such as speech recognition. HMMs are a specific flavour of PGM that can be used to model sequential problems, where the underlying state is hidden and only partially observable through noisy observations. More expressive PGMs, such as dynamic Bayesian networks (DBNs) (Dean and Kanazawa 1989), extend HMMs to allow for more complex dependencies between states and observations.

**Relational representations for generalisation.** While PGMs provide a powerful framework for reasoning under uncertainty, their propositional nature limits their ability to generalise across objects or entities. In contrast, early AI systems, grounded in formal logic, offered expressive relational representations that naturally captured relational structure and abstractions. The field of statistical relational AI (StarAI) emerged from the recognition that these two perspectives, logical and probabilistic, could be unified to combine the expressiveness of relational logic with the robustness of statistical reasoning (De Raedt, Kersting, et al. 2016). StarAI frameworks, such as probabilistic logic programs (De Raedt and Kimmig 2015; Poole 1993), generalise PGMs to relational domains by extending logic programs (or clausal theories) with probabilistic semantics. In these models, logical rules are annotated with weights that quantify their uncertainty. Then, probabilistic inference corresponds to computing the likelihood of a query given all possible worlds, *i.e.* the assignments that satisfy the logical theory. Specifically, one can use weighted model counting (WMC), where the weights are probabilities, to perform inference in these models. This yields powerful modelling capabilities, but also presents a major computational challenge: probabilistic inference in these models is #P-hard (Roth 1996). Knowledge compilation (KC) (Darwiche 2002) is a technique that aims at amortising this cost over multiple inference queries by transforming logical theories into tractable circuit representations. These circuits then allow efficient inference, in polynomial time, over complex structured domains. This line of work not only makes inference feasible in practice, but also provides a unifying "assembly language" for probabilistic

logic. Together, these advances show how relational symbolic knowledge and statistical inference can be effectively combined.

**The neurosymbolic paradigm.** The successes of machine learning and deep learning could not leave the field of StarAI untouched. The new capabilities of dealing with subsymbolic data, such as images or text, led to the creation of neurosymbolic AI (NeSy) (Towell and Shavlik 1994), which emerged as a promising approach to combine the strengths of both paradigms (Garcez and Lamb 2023; Marra et al. 2024). By integrating deep learning with symbolic reasoning, NeSy aims to build systems that can learn from data while also being able to reason about it in a structured way. This allows for more interpretable models that can leverage prior knowledge and provide guarantees about their behaviour.

**Towards relational neurosymbolic sequential models.** An attentive reader may have noticed that a crucial piece is missing in the puzzle: how to combine all these ideas in a single framework that can deal with sequential settings under uncertainty, while being able to leverage the power of deep learning and relational representations. This is the main objective of this dissertation. We aim to build a framework that can be used to reason about sequential problems in a principled way, while also being able to learn from data and generalise across objects or entities. This is a challenging task, as it requires us to combine the strengths of sequential probabilistic models, relational representations, and deep learning, while also addressing the limitations of each approach. In particular, we need to find ways to scale up NeSy techniques to handle large state spaces and long planning horizons, while also ensuring that our models are interpretable and can provide guarantees about their behaviour.

Therefore, using Figure 1.1 as a guide, in this thesis we will consider agents that

**(i)** can (learn to) perceive the world,

**(ii)** can reason over multiple steps,

**(iii)** can learn from interactions with the environment,

**(iv)** can learn to fill in knowledge gaps,

**(v)** can reason about the world in a relational way.

All of this, grounded in solid probabilistic semantics.

In the sections that follow, we formalise the central research questions that guide this dissertation and present an overview of the technical contributions that address them.

## 1.1 Dissertation Statement

This dissertation pushes the boundaries of neurosymbolic AI towards expressive and efficient sequential decision making. As previously stated, the main goal is to develop agents that **(i)** can learn to perceive the world, while **(ii)** sequentially reasoning and **(iii)** interacting with it, **(iv)** compensating for missing knowledge, and that **(v)** can generalise across objects or entities. Specifically, we aim to answer the following questions:

**RQ1** How can sequential symbolic reasoning incorporate subsymbolic perception in a principled and probabilistic way?

**RQ2** How can probabilistic reasoning be extended to support sequential decision making and planning under uncertainty?

**RQ3** Can neurosymbolic methods be effectively applied in online learning settings?

**RQ4** Is it possible to compensate for partial domain knowledge through learning, while retaining coherent reasoning capabilities?

**RQ5** How can relational rules be leveraged to enable generalisation across entities, tasks, or environments?

Finally, given the recent surge of interest in Large Language and Reasoning Models (LLMs, LRMs), we include a complementary investigation in Chapter 6. While not central to the main research questions, this chapter puts our work into the current perspective of generative AI, and it explores to what extent such models can serve as intelligent agents capable of perception, reasoning, and interaction.

## 1.2 Contributions

This dissertation addresses its statement through three main contributions. All my publications, including those not covered in this dissertation, are listed at the end of the thesis.

### Dynamic Decision Circuits

Decision making under uncertainty in dynamic environments is a fundamental AI problem in which agents need to determine which decisions (or actions) to

make at each time step to maximise their expected utility. Dynamic decision networks (DDNs) are an extension of dynamic Bayesian networks with decisions and utilities, and can be used to compactly represent Markov decision processes (MDPs). With our **first main contribution**, we propose a novel algorithm called `mapl-cirup` that leverages knowledge compilation techniques developed for (dynamic) Bayesian networks to perform inference and gradient-based learning in DDNs (**RQ2**). Specifically, we knowledge-compile the Bellman update present in DDNs into dynamic decision circuits and evaluate them within an (algebraic) model counting framework. In contrast to other exact symbolic MDP approaches, we obtain differentiable circuits that enable gradient-based parameter learning (**RQ4**), making a first step towards the integration of deep learning and probabilistic reasoning in sequential decision making.

This contribution is based on the following peer-reviewed conference publication.

G. Venturato, V. Derkinderen, P. Zuidberg Dos Martires, and L. De Raedt (2024). "Inference and learning in dynamic decision networks using knowledge compilation". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 38. 18, pp. 20567–20576

We describe our approach in detail in Chapter 3. The code is available at `https://github.com/ML-KULeuven/mapl-cirup`.

## Relational Neurosymbolic Markov Models

Sequential problems are ubiquitous in AI, such as in reinforcement learning or natural language processing. State-of-the-art deep sequential models, like transformers, excel in these settings but fail to guarantee the satisfaction of constraints necessary for trustworthy deployment. In contrast, neurosymbolic AI (NeSy) provides a sound formalism to enforce constraints in deep probabilistic models but scales exponentially on sequential problems. To overcome these limitations, with our **second main contribution**, we introduce relational neurosymbolic Markov models (NeSy-MMs), a new class of end-to-end differentiable sequential models that integrate and provably satisfy relational logical constraints (**RQ1**, **RQ4**). We propose a strategy for inference and learning that scales on sequential settings, and that combines approximate Bayesian inference, automated reasoning, and gradient estimation. Experiments show that NeSy-MMs can solve problems beyond the current state-of-the-art in neurosymbolic AI and still provide strong guarantees with respect to desired properties. Moreover, we show that our models are more interpretable and that constraints can be adapted at test time to out-of-distribution scenarios (**RQ5**).

This contribution is mainly based on the following peer-reviewed conference publication.

L. De Smet*, G. Venturato*, L. De Raedt, and G. Marra (2025). "Relational neurosymbolic Markov models". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 39. 15, pp. 16181–16189

We describe our approach in detail in Chapter 4. The code is available at `https://github.com/ML-KULeuven/nesy-mm`.

## Neurosymbolic Reinforcement Learning

Probabilistic logic shields (W.-C. Yang et al. 2023) integrate deep reinforcement learning (RL) with probabilistic logic reasoning to train agents that operate in uncertain environments while giving strong guarantees with respect to logical constraints, such as safety properties. With our **third main contribution**, we introduce a codebase that streamlines the design of custom MiniHack environments where neurosymbolic RL agents leverage probabilistic logic shields to learn safe and interpretable policies with strong guarantees (**RQ3**, **RQ5**). Our framework allows expert users to easily define and train agents that integrate deep neural policies with probabilistic logic in arbitrarily complex games: from simple exploration to planning and interacting with enemies. Additionally, we provide a web-based platform that showcases our application, offering an interactive interface for the broader community to experiment with and explore the capabilities of neurosymbolic reinforcement learning. This lowers the barrier for researchers and developers, making it accessible for a wider audience to engage with safety-critical RL scenarios.

This contribution is mainly based on the following peer-reviewed conference publication, that won the *Best Technical Demonstration Award* at AAAI 2025.

D. Debot*, G. Venturato*, G. Marra, and L. De Raedt (2025). "Neurosymbolic Reinforcement Learning: Playing MiniHack With Probabilistic Logic Shields". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 39. 28, pp. 29631–29633

We describe our approach in detail in Chapter 5. The code is available at `https://github.com/ML-KULeuven/nesy-minihack`.

* These authors contributed equally to this work.

# Chapter 2

# Background

In this chapter we introduce the relevant background knowledge. We first cover neurosymbolic AI and its representation via probabilistic logic programming. We then present two orthogonal directions: the decision-theoretical setting and the sequential setting, whose combination forms the core of the contributions of this thesis. Finally, we provide a brief overview of reinforcement learning.

## 2.1 Probabilistic Neurosymbolic AI

Probabilistic NeSy methods originate from the field of statistical relational AI (StarAI) that integrates statistical AI with logic (De Raedt, Kersting, et al. 2016; Marra et al. 2024). This integration leads to systems capable of performing inference and learning with uncertainty over *symbolic*, *i.e.* logical, knowledge. For example, the logical relations `player_at(player1, location1)` and `monster_at(monster1, location1)` can be used to apply the rule `hit(P,M) :- player_at(P,L), monster_at(M,L)` and deduce that `player1` is hit by `monster1` because they are in the same location. Moreover, this knowledge is often uncertain in practice, resulting in uncertainty on whether the deduced logical relations hold. For example, consider the case of a sneaky monster. If we are unsure whether `monster_at(monster1, location1)` is true or not, we will also be uncertain whether the player is hit or not. Notice that uncertain logical relations can be modelled as binary random variables, which justifies the integration with statistics.

While StarAI assumes knowledge to be neatly represented as a symbolic


Figure 2.1: Probabilistic graphical model of a NeSy system. **N** represents the subsymbolic state, while **S** represents the symbolic state. The neural predicate $\boldsymbol{\varphi}$ maps the subsymbolic state to a probability distribution over the symbolic state.

state **S**, such an assumption does not always hold. Images, sound waves or natural language are usually represented as *subsymbolic* data, *i.e.* tensors, that are not directly usable by relational AI. Therefore, probabilistic NeSy methods use neural predicates $\boldsymbol{\varphi}$ to map subsymbolic data to a probability distribution over symbolic representations that can be used by StarAI. Figure 2.1 depicts a probabilistic graphical model (PGM) (Koller and Friedman 2009) representation of a NeSy system. While formal details will follow in the next sections, it is helpful to already have a high-level view of how inference in NeSy works. Specifically, given a boolean variable $Y$ from **S** with domain $\{y, \neg y\}$ and a set of rules $\mathcal{R}$ on the symbols in **S**, the probability that the query $Y$ holds, given subsymbolic input **n**, is computed via weighted model integration (WMI) (Morettin et al. 2021)

$$p_{\boldsymbol{\varphi}}(y \mid \mathbf{n}) = \int \mathbb{1}_{\mathbf{s} \models_{\mathcal{R}} y} \; p_{\boldsymbol{\varphi}}(\mathbf{s} \mid \mathbf{n}) \; \mathrm{d}\mathbf{s}, \tag{2.1}$$

where the distribution of **S** is parametrised by a neural network $\boldsymbol{\varphi}$ from the subsymbolic state **n**. All the symbols used here will be introduced later in the thesis, but the intuition is that the neural predicate $\boldsymbol{\varphi}$ maps the subsymbolic state **n** to a probability distribution over the symbolic state **S**, and the integral computes the probability that $Y$ holds in all possible worlds **s** that satisfy the rules $\mathcal{R}$.

A prominent way of representing neurosymbolic models is via probabilistic logic programming (PLP) (De Raedt, Kimmig, and Toivonen 2007).

### 2.1.1 Probabilistic Logic Programming

Logic programming is a declarative programming paradigm rooted in formal logic. One of its most known languages is Prolog (Sterling and Shapiro 1986).

A logic program is composed of a set $\mathcal{R}$ of rules (or definite clauses), and a set $\mathcal{S}$ of symbols (or atoms).

**Definition 2.1.1** (Atom)**.** An *atom* is a logical expression of the form $\texttt{p(t}_1\texttt{,...,t}_K\texttt{)}$, where $\texttt{p}$ is a predicate symbol of arity $K$ and each $\texttt{t}_i$ is a term.

**Definition 2.1.2** (Term)**.** A *term* is recursively defined as follows:

- A constant symbol is a term;
- A variable symbol is a term;
- If $\texttt{f}$ is a functor of arity $K$ and $\texttt{t}_1\texttt{,...,t}_K$ are terms, then $\texttt{f(t}_1\texttt{,...,t}_K\texttt{)}$ is a term.

An atom is either true or false depending on the values of its terms. Moreover, when an atom has arity $K > 1$, it expresses a *relation* among its terms. Atoms are used in the definition of rules of the form $\texttt{h:- b}_1\texttt{,...,b}_K$, where $\texttt{h}$ is an atom and each $\texttt{b}_i$ is a *literal*, *i.e.* an atom or the negation of an atom. We call $\texttt{h}$ the *head* of the rule while the conjunction $\texttt{b}_1\texttt{,...,b}_K$ is the body of the rules. If the body of the rule is logically true, then the rule expresses that the head of the rule is true as well by logical consequence. If the body of the rule is empty, the rule is called a *fact*.

**Example 2.1.1** (Logic Program)**.** Algorithm 1 expresses the background knowledge necessary to find out if someone is a grandparent of someone else. There is a shared knowledge base at the top in the form of two atoms that express that `george` is the father of `alice` and that `alice` is the mother of `william`. The first two rules define the parent relation as being either the mother or the father of someone. Finally, the last rule defines that `X` is a grandparent of `Y` if there exists an intermediate person `Z` that is the parent of `Y` and whose parent is `X`.

**Algorithm 1** Logic programming encoding of Example 2.1.1.

```
father(george, alice). mother(alice, william).

parent(X, Y) :- father(X, Y).
parent(X, Y) :- mother(X, Y).

grandparent(X, Y) :- parent(X, Z), parent(Z, Y).
```

Traditional logic programming assumes that atoms are either true or false. However, in many real-world settings, such binary knowledge is too restrictive. Probabilistic logic programming (PLP) extends logic programming by allowing facts and rules to be annotated with probabilities, thus enabling reasoning

under uncertainty (De Raedt, Kimmig, and Toivonen 2007). Probabilistic logic programs typically define a distribution over all possible symbols, often called *possible worlds*. Inference then consists of computing the marginal probability that a certain query holds across all such worlds.

For example, suppose we are not fully certain that `george` is `alice`'s father. We can represent this uncertainty as `0.42::father(george, alice)`. This expresses that `father(george, alice)` holds with probability 0.42. Probabilistic inference in PLP now computes the probability that derived atoms, such as `grandparent(george, william)`, are true given the uncertain facts.

This framework provides a flexible and principled way to incorporate symbolic reasoning and probabilistic uncertainty. It is particularly powerful when combined with neural components to bridge subsymbolic data and symbolic representations, as we explain in the next section.

### 2.1.2 Deep Probabilistic Logic Programming

Modern implementations of probabilistic logic programming (Manhaeve, Dumancic, et al. 2018) allow for the probabilities of atoms to be parameterised by neural networks to achieve a neurosymbolic integration. Apart from parametrising atoms representing finite random variables, these implementations have also been extended to the infinite domain (De Smet, Zuidberg Dos Martires, et al. 2023). To facilitate the definition and inclusion of such atoms, two additional building blocks were added. First, there is the *neural distributional fact* (NDF), an expression of the form `x ~ distribution(`$n_1$`,...,`$n_K$`)`. Here, `x` is a term representing a random variable distributed according to `distribution` filled in with the numeric terms $n_1,\ldots,n_K$. These numeric terms can be constant numerical values or the output of a neural network. Second, to use neural distributional facts in a logical expression, which is always either true or false, *probabilistic comparison formulae* (PCF) were also introduced. These are specific atoms that take the terms coming from a neural distributional fact as argument.

**Example 2.1.2** (DeepSeaProbLog program)**.** In Algorithm 2, we show how NDFs and PCFs work together to write a probabilistic logic program that represents a simple model of the weather. Two finite variables first model whether it is cloudy and what degree of humidity currently holds. The `temperature` NDF models a normal distribution with mean `15` and standard deviation `3`. Because of the uncertain nature of these variables, the truth value of the atoms `rain` and `good_weather` will also be uncertain. These atoms are defined under the NDFs; it is rainy when it is cloudy and humid, and the weather is good when it does

not rain and the temperature is high or if it does rain, but the temperature is low.

**Algorithm 2** Logic programming encoding of Example 2.1.2.

```
cloudy ~ bernoulli(0.7)
humid ~ categorical([0.3, 0.5, 0.2], [dry, moist, wet]).
temperature ~ normal(15, 3).

rain :- cloudy, not (humid =:= dry).

good_weather :- not rain, temperature > 20.
good_weather :- rain, temperature < 0.
```

Finally, let us introduce a running example to further illustrate the concepts introduced so far, and that will be useful later in the thesis.

**Example 2.1.3.** Consider a simple game where a monster `M` and player `P` interact with each other (Algorithm 3). Both entities are represented by their normally distributed locations, which are parametrised by neural networks from a given image `Im`. Their interactions are in the form of `hits`, where the monster can hit the player if it is close and not `clumsy`. The clumsiness of the monster is uncertain and modelled by a separate Bernoulli random variable. Finally, the game ends whenever the monster succeeds in hitting the player. Because of the uncertainty on locations and clumsiness, it also follows that whether the game is over is uncertain.

Example 2.1.3 is encoded in Algorithm 3. The first two lines are *neural predicates* that represent deep random variables modelling the normally distributed locations of the monster `M` and the player `P`. Each of the neural predicates has a named neural network that takes the image `Im` as input and outputs the parameters of its random variable. The third line introduces a Bernoulli random variable `clumsy` indicating that the monster will be clumsy with a 75% chance. The final two lines express two rules of the game that determine when the monster `M` `hits` the player `P` and when the image `Im` depicts a lost game, *i.e.* when the image depicts the monster hitting the player.

Figure 2.2 shows the probabilistic graphical model view of Example 2.1.3. Notice how the two rules H (player hit by the monster) and G (game lost) are represented as binary random variables in the graphical model. The logical relations between them and the other variables are implicitly defined by the topology of the model itself. Specifically, the rules are encoded in the conditional probability tables, which we omit for brevity. In mathematical terms, Equation 2.1 tells us that

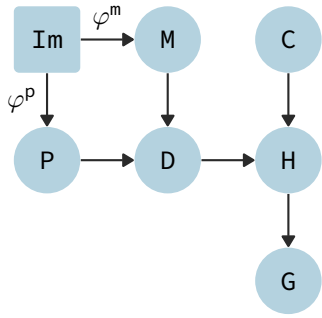



Figure 2.2: Probabilistic graphical model view of Example 2.1.3.

the probability that `image.png` represents a lost game is

$$p(\texttt{game\_over(image.png)}) = \int_{(\mathsf{P},\mathsf{M},\mathsf{C}) \models_{\mathsf{H},\mathsf{D}} \mathsf{g}} p(\mathsf{P} \mid \texttt{image.png}) p(\mathsf{M} \mid \texttt{image.png}) p(\mathsf{C}) \, \mathrm{d}\mathsf{P}\mathrm{d}\mathsf{M}\mathrm{d}\mathsf{C},$$

with a small abuse of notation for the discrete variable $\mathsf{C}$.

**Algorithm 3** Logic programming encoding of Example 2.1.3.

```
player(Im, P) ~ normal(noisy_player(Im)).
monster(Im, M) ~ normal(noisy_monster(Im)).
clumsy ~ bernoulli(0.75).

hits(M, P) :- distance(M, P, D), D < 2, not clumsy.
game_over(Im) :- player(Im, P), monster(Im, M), hits(M, P).
```

## 2.2 Decision Making

Many intelligent agents must not only infer knowledge from uncertain data but also make decisions under uncertainty. This requires evaluating different possible actions and choosing those that maximise a given utility function. Probabilistic logic programming and neurosymbolic models offer a powerful language for describing such decision problems in a structured and interpretable way. However, solving these problems requires efficient inference techniques.

In this section, we focus on non-sequential decision making. Specifically, on how inference in probabilistic logic programs (introduced in Section 2.1.1) can

be reduced to *weighted model counting* (WMC), a core computational concept in probabilistic reasoning. We then present *knowledge compilation* (KC) as a scalable method to support repeated inference over complex logical theories. Finally, we show how decision-theoretic quantities such as expected utility can be computed via *algebraic model counting* (AMC), a generalisation of WMC over semirings.

### 2.2.1 Weighted Model Counting

A prominent technique to perform probabilistic inference with discrete random variables is reducing it to weighted model counting (WMC) (Darwiche 2009). That is, encoding a specific probabilistic inference problem as a weighted propositional logic formula. Computing the weight of the formula is then equivalent to computing the target probability. Chavira and Darwiche (2008), for instance, used this technique to perform inference in BNs.

**Definition 2.2.1** (Weighted Model Counting)**.** Let $T$ be a propositional logic theory (or formula) over a set of literals $\mathcal{L}$, and let $w : \mathcal{L} \to \mathbb{R}$ be a weight function assigning a real weight to each literal. The *weighted model count* of $T$ under $w$ is defined as:

$$WMC(T, w) = \sum_{m \in \mathcal{M}(T)} \prod_{\ell \in m} w(\ell), \tag{2.2}$$

where $\mathcal{M}(T)$ denotes the set of all satisfying assignments (models) of $T$, and each $m$ is interpreted as a set of literals that are true in the model.

Notice that Equation 2.1 reduces to Equation 2.2 when there are no neural predicates, *i.e.* when the subsymbolic state $\mathbf{n}$ is empty, the domain of the symbolic state $\mathbf{S}$ is finite, and the weights assigned by $w$ are probabilities. This shows that probabilistic inference can be cast as a special case of neurosymbolic reasoning.

**Example 2.2.1** (Weighted Model Counting)**.** Consider the theory $T = C \leftrightarrow A \vee B$, where we use weight function $w : \{a \mapsto 0.1, \neg a \mapsto 1 - 0.1, b \mapsto 0.2, \neg b \mapsto 1 - 0.2, c \mapsto 0.3, \neg c \mapsto 1 - 0.3\}$. The weighted model count of $T$ is then:

$$\begin{aligned} WMC(T, w) &= w(a)w(b)w(c) + w(a)w(\neg b)w(c) + \\ &\quad w(\neg a)w(b)w(c) + w(\neg a)w(\neg b)w(\neg c) \\ &= 0.006 + 0.024 + 0.054 + 0.504 = 0.588 \end{aligned}$$

In case the weights have a probabilistic interpretation, the weight 0.588 is the probability of the theory being satisfied.

### 2.2.2 Knowledge Compilation

Unfortunately, performing WMC on propositional logic formulas is a computational hard task, #P-hard to be precise (Valiant 1979). We can see this intuitively in Example 2.2.1 where we explicitly enumerate all the satisfiable states. Of course it is infeasible in practice to enumerate all satisfiable states one by one. So instead, a state-of-the-art technique to solving this is through knowledge compilation (KC) (Darwiche 2002) — an approach also deployed by Chavira and Darwiche (2008). The core idea is simple: take a propositional logic formula and compile it into a representation where WMC, and consequently probabilistic inference, can be performed in polytime (with respect to the size of the target representation). A key advantage of this approach is that the compilation is weight agnostic. This means that multiple WMC problems can be addressed using a single compilation step, as long as those problem instances share the same logic formula. This amortises the cost of (offline) compilation, which is still computationally hard, over the multiple (online) query steps that follow. This is especially useful for repeated inference.

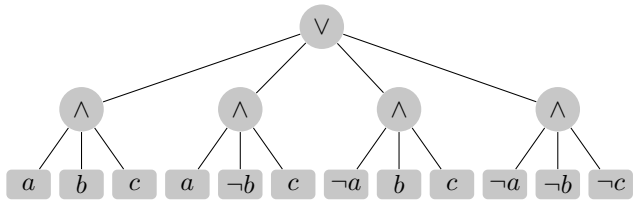
**Example 2.2.2** (KC)**.** The logic theory from Example 2.2.1 can be represented as a directed acyclic graph (DAG) trivially enumerating all the possible models (Figure 2.3a). However, using knowledge compilation one can represent the same theory in a more compact way (Figure 2.3b), while preserving the efficient WMC computation. Substituting the $\wedge$ and $\vee$ connectives with multiplication and addition operations respectively, and the literals with the according weights, we obtain the arithmetic circuit for $WMC(T, w)$ (Figure 2.3c).

### 2.2.3 Algebraic Model Counting

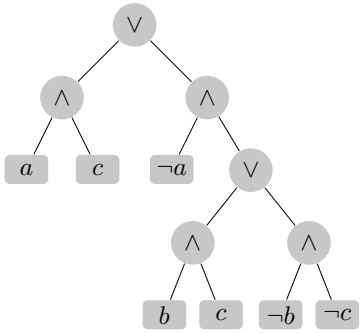
A limitation of WMC is its restriction to the semiring of positive real valued numbers, for instance probabilities. Algebraic model counting (AMC) (Kimmig et al. 2017) alleviates this issue by defining meaningful counting problems on (knowledge-compiled) logic formulas using an arbitrary commutative semiring.

**Definition 2.2.2** (Commutative Semiring)**.** A commutative semiring is an algebraic structure $(\mathcal{A}, \oplus, \otimes, e^{\oplus}, e^{\otimes})$ such that 1) $\mathcal{A}$ is a set of domain elements; 2) addition $\oplus$ and multiplication $\otimes$ are binary operations $\mathcal{A} \times \mathcal{A} \to \mathcal{A}$ that are both associative and commutative; 3) $\otimes$ distributes over $\oplus$; 4) $e^{\oplus} \in \mathcal{A}$ is the neutral element of $\oplus$; 5) $e^{\otimes} \in \mathcal{A}$ is the neutral element of $\otimes$; and 6) $e^{\oplus}$ is an annihilator for $\otimes$.

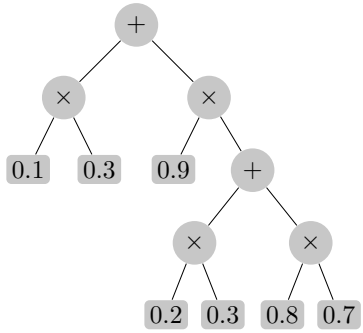
**Definition 2.2.3** (Algebraic Model Counting)**.** Given a commutative semiring $\mathcal{S} = (\mathcal{A}, \oplus, \otimes, e^{\oplus}, e^{\otimes})$, a propositional logic theory $T$, and a labelling function




(a) Explicit enumeration




(b) Knowledge compilation




(c) Arithmetic circuit

Figure 2.3: From the logic theory in Example 2.2.1 to the circuit, via knowledge compilation. Evaluating the circuit in Figure 2.3c yields the weighted model count.

$\alpha : \mathcal{L} \mapsto \mathcal{A}$, mapping literals $\mathcal{L}$ of the variables in $T$ to values from the semiring domain $\mathcal{A}$, the task of algebraic model counting (AMC) is to compute:

$$AMC(\mathcal{S}, T, \alpha) = \bigoplus_{m \in \mathcal{M}(T)} \bigotimes_{\ell \in m} \alpha(\ell), \tag{2.3}$$

where $\otimes_{\ell \in m} \alpha(\ell)$ is the label of a model $m$ represented as a set of literals $\ell$ true in $m$, and $\mathcal{M}(T)$ are the models of $T$.

A plethora of problems in artificial intelligence can be formulated as algebraic model counting problems (ibid.). For instance, computing gradients using the gradient semiring (Manhaeve, Dumancic, et al. 2018), which is useful for learning tasks. Derkinderen and De Raedt (2020) adapted the AMC framework to perform decision making under uncertainty by formulating the computation of maximum expected utilities (MEUs) as algebraic model counts. First, they define an appropriate labelling function for the set of literals $\mathcal{L}$ of a logic formula:

$$\{\alpha(\ell) = (p_\ell, eu_\ell, D_\ell) \mid \ell \in \mathcal{L}\} \subset \mathcal{A}, \tag{2.4}$$

where $p_\ell$ is the probability of $\ell$, $eu_\ell$ its expected utility, and $D_\ell = \{\ell\}$ if $\ell$ is a decision literal, or $D_\ell = \emptyset$ otherwise. Second, they adapt the expected utility semiring to incorporate maximisation. This results in the algebraic structure $\mathcal{S}_{meu} = (\mathcal{A}, \oplus, \otimes, e^{\oplus}, e^{\otimes})$ with

$$a_1 \oplus a_2 = \begin{cases} max(a_1, a_2) & \text{if } D_1 \neq D_2 \\ (p_1 + p_2, eu_1 + eu_2, D_1) & \text{otherwise} \end{cases}$$

$$a_1 \otimes a_2 = (p_1 \cdot p_2, p_1 \cdot eu_2 + p_2 \cdot eu_1, D_1 \cup D_2)$$

$$e^{\oplus} = (0, 0, \mathcal{D}) \quad \text{and} \quad e^{\otimes} = (1, 0, \emptyset)$$

where $a_1, a_2 \in \mathcal{A}$ and $\mathcal{D} \subset \mathcal{L}$ being the set of all decision variables and their negation. Furthermore, $max(a_1, a_2)$ returns the element ($a_1$ or $a_2$) with the highest expected utility ($eu_1/p_1$ or $eu_2/p_2$, where the divisor is a technical detail related to normalizing under the presence of constraints). Note that a semiring contains two operations, while decision making under uncertainty involves three: max, sum, and product. This considerably complicates inference but is solved in $\mathcal{S}_{meu}$ by the input-dependent $\oplus$-operation. Consequently, $\mathcal{S}_{meu}$ is not a semiring in general, and must only be used while constraining the variable ordering within the AMC computations. This is supported by several knowledge compilation tools. Intuitively, $\mathcal{S}_{meu}$ extends the standard expected utility semiring (Eisner 2002) by also keeping track of the set of decision literals $D$. This allows the algebraic circuit to perform maximisation whenever two elements correspond to different decision assignments, and summation otherwise. The $\otimes$-operation aggregates probabilities and expected utilities along conjunctions, while the $\oplus$-operation compares alternative decision outcomes based on their normalised expected utility $eu/p$. As Derkinderen and De Raedt (2020) point out, this extension generalises the expectation semiring by coupling probabilistic reasoning (through $p$), utility accumulation (through $eu$), and decision comparison (through $D$) into a single algebraic structure. This design enables algebraic circuits to compute maximum expected utilities directly, while preserving the declarative nature of the AMC framework. While Derkinderen and De Raedt (ibid.) were able to solve decision making problems, their technique does not consider temporal or sequential dependencies between decisions. Such scenarios are usually modelled using MDPs (Puterman 1994).

## 2.3 Sequential Models

The concepts introduced so far (NeSy, KC, AMC) operate mainly on logic formulas for single-step (decision) problems. However, many real-world scenarios

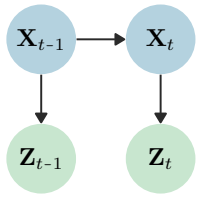



Figure 2.4: Probabilistic graphical model of an HMM. Blue nodes represent the (hidden) states, green the observations.

involve sequences of observations, decisions, and latent states evolving over time. To reason under uncertainty in such settings, we must extend our probabilistic models to capture temporal dynamics.

Sequential probabilistic models allow us to track and predict the evolution of hidden variables over time, given a stream of noisy observations. This provides a foundation for state estimation and planning in partially observable and uncertain environments. In this section, we introduce hidden Markov models (HMMs), particle filters, and Markov decision processes (MDPs) as core tools for modelling temporal structure in probabilistic and decision-theoretic settings.

Hidden Markov models (HMMs) are sequential probabilistic models for discrete-time Markov processes (Baum and Petrie 1966). Given sequences of *states* $\mathbf{X} = (\mathbf{X}_t)_{t\in\mathbb{N}}$ and *observations* $\mathbf{Z} = (\mathbf{Z}_t)_{t\in\mathbb{N}}$, an HMM factorises the joint probability distribution $p(\mathbf{X}, \mathbf{Z})$ as,

$$p(\mathbf{X}_0)p(\mathbf{Z}_0 \mid \mathbf{X}_0) \prod_{t\in\mathbb{N}} p(\mathbf{X}_{t+1} \mid \mathbf{X}_t)p(\mathbf{Z}_{t+1} \mid \mathbf{X}_{t+1}), \tag{2.5}$$

where $\mathbf{X}_t$ is a fully latent state. Figure 2.4 depicts a probabilistic graphical representation of an HMM. If $\mathbf{X}_t$ has a known factorisation in the form of a Bayesian network (BN) (Pearl 1988), then the process and its observations encode a *Markovian dynamic Bayesian network* (DBN) (Dean and Kanazawa 1989). Note that $\mathbf{X}_t$ and $\mathbf{Z}_t$ are random vectors that can have both discrete and continuous components. In all that follows, a specific assignment of a random variable or vector will be written in lowercase. For example, $\boldsymbol{x}_t = (x_{t,1}, \ldots, x_{t,D})$ is an assignment of the random vector $\mathbf{X}_t = (X_{t,1}, \ldots, X_{t,D})$ of dimension $D$.

### 2.3.1 Particle Filters

In many sequential settings, the state of the environment is not directly observable. Instead, agents rely on noisy and partial observations. Probabilistic inference in sequential models requires ad-hoc techniques to deal with the additional complexity of reasoning through time. In real-world scenarios, computing an exact estimation of the hidden state is often not practical, due to the number of state variables involved, that make exact inference intractable. This motivates the use of state estimation techniques to maintain a belief over the latent state. Widely used methods for this purpose are particle filters (N. J. Gordon et al. 1993; Handschin and Mayne 1969), also known as sequential Monte Carlo methods, or sequential importance sampling with resampling.

Particle filters approximate the belief distribution $p(\mathbf{X}_t \mid \mathbf{Z}_{0:t})$ over the current state $\mathbf{X}_t$ given a sequence of observations $\mathbf{Z}_{0:t}$ by representing it with a set of $N$ weighted samples (particles)

$$\mathcal{B}_t = \{(\boldsymbol{x}_t^{(i)}, w_t^{(i)})\}_{i=1}^{N}.$$

Each particle $\boldsymbol{x}_t^{(i)}$ represents a sampled hypothesis about the state, and $w_t^{(i)}$ is its corresponding importance weight.

Particle filters starts from a set of particles $\mathcal{B}_0$, that are sampled from the prior $p(\mathbf{X}_0)$. Then, they operate recursively through three main steps:

1. **Prediction:** Each particle $\boldsymbol{x}_{t-1}^{(i)}$ is propagated forward using the transition model $p(\mathbf{X}_t \mid \mathbf{X}_{t-1})$ to obtain $\boldsymbol{x}_t^{(i)}$.
2. **Update:** The weight of each particle is updated based on the likelihood of the new observation, $w_t^{(i)} \propto p(\mathbf{Z}_t \mid \boldsymbol{x}_t^{(i)})$.
3. **Resampling:** To avoid weight degeneracy, the new particles are resampled from $\mathcal{B}_t$ according to their importance weights.

Notice that the resampling step is crucial to ensure that particles with low weights do not dominate the belief representation. However, resampling also breaks the differentiability of the inference process, which is a limitation for gradient-based learning methods.

In some problems, the state $\mathbf{X}_t$ can be partitioned, by exploiting conditional independencies among state variables, into components that admit tractable inference $\mathbf{D}_t$ and those that do not $\mathbf{C}_t$. If this is the case, one can use the statistical Rao-Blackwell trick to obtain Rao-Blackwellised particle filters

(RBPFs) (K. Murphy and Russell 2001). They are based on the simple intuition that exact inference is always more accurate than sampling, even if only for a partition of the state. Therefore, by marginalising out the tractable components of the state analytically, while sampling only over the intractable components, they reduce variance and the number of samples required, making RBPFs more efficient for certain classes of problems. In practice, they split

$$p(\mathbf{X}_{0:t} \mid \mathbf{Z}_{0:t}) = p(\mathbf{C}_{0:t}, \mathbf{D}_{0:t} \mid \mathbf{Z}_{0:t}) \tag{2.6}$$

$$= p(\mathbf{D}_{0:t} \mid \mathbf{C}_{0:t}, \mathbf{Z}_{0:t}) p(\mathbf{C}_{0:t} \mid \mathbf{Z}_{0:t}), \tag{2.7}$$

where $p(\mathbf{C}_t \mid \mathbf{Z}_{0:t})$ is sampled, forming the set of particles

$$\mathcal{B}_t = \left\{ \left( \boldsymbol{c}_t^{(i)}, w_t^{(i)} \right) \mid \boldsymbol{c}_t^{(i)} \sim p(\mathbf{C}_t \mid \boldsymbol{c}_{t-1}^{(i)}), w_t^{(i)} \propto p(\mathbf{Z}_t \mid \boldsymbol{c}_{0:t}^{(i)}, \mathbf{Z}_{0:t-1}) \right\}, \tag{2.8}$$

and then computing the posteriors $p(\mathbf{D}_t \mid \boldsymbol{c}_{0:t}^{(i)}, \mathbf{Z}_{0:t})$ analytically via exact filtering. Therefore, obtaining the belief over the state $\mathbf{X}_t$ as

$$p(\mathbf{X}_t \mid \mathbf{Z}_{0:t}) = p(\mathbf{C}_t, \mathbf{D}_t \mid \mathbf{Z}_{0:t}) \tag{2.9}$$

$$\approx \sum_{i=1}^{N} p(\mathbf{D}_t \mid \boldsymbol{c}_{0:t}^{(i)}, \mathbf{Z}_{0:t}) w_t^{(i)} \delta(\boldsymbol{C}_t - \boldsymbol{c}_t^{(i)}), \tag{2.10}$$

where $\delta$ is the Dirac delta function.

As it turns out, RBPFs are particularly well-suited for neurosymbolic models, where one typically combines discrete logical structures with continuous sensor information. The discrete components can be represented as the tractable part $\mathbf{D}_t$, while the continuous components are sampled as the intractable part $\mathbf{C}_t$. This allows RBPFs to efficiently handle hybrid models that combine symbolic and subsymbolic reasoning. We will exploit these ideas in Chapter 4.

### 2.3.2 Markov Decision Processes

MDPs are formal models for dynamic decision problems in a fully observable and stochastic environment, with additive rewards. They consist of a set of (explicit) states $\mathbf{S}$, a set of actions $\mathbf{A}$, a reward function $R(s, a)$, and a probabilistic transition function $T(s, a, s') = P(s' \mid s, a)$ expressing the probability of ending up in state $s'$ given the current state $s$ and the action $a$ performed in it.

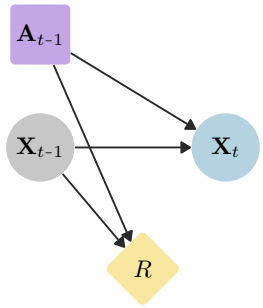



Figure 2.5: Probabilistic graphical model of a MDP. Blue nodes represent the states, grey observed states, purple the decisions (*i.e.* action), and yellow the reward function.

Solving an MDP, *i.e.* finding the best action for each state, corresponds to solving the following recursive equation for every state $s \in \mathbf{S}$

$$U(s) = \max_{a \in \mathbf{A}} \left\{ R(s,a) + \gamma \sum_{s' \in \mathbf{S}} P(s'|s,a) U(s') \right\} \tag{2.11}$$

This is the renowned *Bellman equation*, which can be solved using the *value iteration* algorithm (Bellman 1957). Value iteration typically starts with $U(s) = 0$, $\forall s \in \mathbf{S}$, and iteratively updates the value of $U$ (also called the utility) according to Equation 2.11. A single value iteration step is referred to as a *Bellman update*. Value iteration is guaranteed to converge to the optimal solution for a discount factor $\gamma \in [0, 1)$. That is, for each state, the action yielding the maximum expected reward.

Graphically, MDPs can be represented using dynamic decision networks (DDNs) —a dynamic and decision-theoretic extension of standard BNs (Figure 2.5). Just as in BNs, DDNs use circular nodes to represent random variables. Additionally, DDNs also contain reward nodes (diamond shaped) and decision nodes (rectangular shaped). Note that in DDNs 'actions' are called 'decisions'.

Contrary to MDPs, where states ($s$ and $s'$) are modelled explicitly, DDNs instead model the random variables that constitute the state. This is illustrated in Figure 2.5. An advantage of representing MDPs as DDNs, *i.e.* using random variables $\mathbf{X}_t$ instead of explicit states, is a (potential) exponential reduction in the representation size (Russell and Norvig 2020, Section 17.1.4). This is analogous to the space efficiency of Bayesian networks over flattened state encodings. Representing the state as a Bayesian network is equivalent to using factored MDPs (Boutilier et al. 2000). We will heavily exploit this structure in Chapter 3.

## 2.4 Reinforcement Learning

Reinforcement learning (RL) (Sutton and Barto 2018) is a learning paradigm concerned with training agents to act optimally in an environment through interaction. It differs from both supervised and unsupervised learning. In fact, supervised learning assumes access to an optimally labelled dataset, while unsupervised learning focus on pattern recognition without explicit feedback. In contrast, RL agents learn from trial and error, receiving feedback in the form of rewards or penalties based on their actions.

RL problems are formally modelled as Markov Decision Processes (MDPs), introduced in Section 2.3.2. At each time step $t$, the agent observes the current state $\mathbf{X}_t$, selects an action $\mathbf{A}_t$, and receives a reward $R_t$, which reflects the quality of the chosen action. The environment then transitions to a new state $\mathbf{X}_{t+1}$ according to a (possibly stochastic) transition model. The agent's objective is to learn a *policy* $\pi$ that maximises the expected total reward.

The transition dynamics of the environment are typically unknown to the agent. In *model-based* RL, the agent attempts to learn a model of these dynamics, and can use this model for planning or simulating future outcomes. In contrast, *model-free* RL bypasses explicit modelling of the environment and instead learns a policy (or value function) directly from experience, relying solely on observed transitions and rewards.

Formally, a policy is a function mapping states to probabilities of selecting each possible action, $\pi(\mathbf{a} \mid \mathbf{x}) = p(\mathbf{A}_t = \mathbf{a} \mid \mathbf{X}_t = \mathbf{x})$. In this thesis we will focus on *policy gradient* methods, where the policy $\pi_\theta(\mathbf{a} \mid \mathbf{x})$, parametrised by $\theta$, maximises the expected cumulative reward, also called the return:

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t R_t\right], \tag{2.12}$$

where $\gamma \in [0, 1)$ is a discount factor that prioritises immediate rewards over future ones. This objective is maximised by repeatedly updating the policy parameters $\theta$ in the direction of the gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{x}_t)\right], \tag{2.13}$$

where $\Psi_t$ is a return-related advantage signal, such as the total return, temporal difference, or baseline-subtracted return (Schulman, Moritz, et al. 2018). This gradient can be estimated using Monte Carlo sampling, which forms the basis for many policy gradient algorithms. If the samples are collected from the same policy $\pi_\theta$ that is being optimised, the method is called *on-policy*,

such as REINFORCE (Williams 1992), A2C (Mnih, Badia, et al. 2016), or PPO (Schulman, Wolski, et al. 2017). If the samples are collected from a different behaviour policy, the method is called *off-policy*, such as Q-learning (Watkins and Dayan 1992), or DQN (Mnih, Kavukcuoglu, et al. 2015).

When the policy $\pi_\theta$ is represented by a deep neural network we talk about deep RL. This allows the agent to map raw observations directly to actions, typically learning an end-to-end function from perceptual inputs (*e.g.* images) to decisions. In the context of Figure 1.1, deep RL systems can be seen as implementing a direct connection from the environment's observation to the agent's action, subsuming both the "perception" and "reasoning" modules into a single black-box function approximator. However, such systems often lack an explicit reasoning component and are prone to poor generalisation and brittle behaviour under distributional shifts. This gap motivates the integration of structured symbolic reasoning into the RL loop. At this point it will come with no surprise that neurosymbolic approaches can help to bridge this gap by combining the perceptual strengths of deep learning with the generalisability and interpretability of symbolic models. We will explore this integration in detail in Chapter 5.

# Chapter 3

# Dynamic Decision Circuits

This chapter is based on the following peer-reviewed conference publication.

> G. Venturato, V. Derkinderen, P. Zuidberg Dos Martires, and L. De Raedt (2024). "Inference and learning in dynamic decision networks using knowledge compilation". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 38. 18, pp. 20567–20576

The core idea for this work was developed jointly with L. De Raedt, who also supervised the research. Together, we designed the theoretical framework and discussed the algorithmic approach. I was responsible for the detailed development of the framework, the implementation of the algorithms, and the experimental evaluation. P. Zuidberg Dos Martires and V. Derkinderen contributed to parts of the implementation, the experimental methodology, and theory, and provided additional supervision. I wrote the paper, with contributions and feedback from all co-authors.

The preliminaries have been fully moved to Chapter 2. Some figures have been updated to match the thesis style. The appendices have been integrated into the main text.

In this chapter, we focus on the reasoning module from Figure 1.1. Specifically, we address research questions **RQ2** and **RQ4**, exploring how knowledge compilation can be leveraged to efficiently perform inference in dynamic decision

networks (DDNs), and how this enables parameter learning in Markov decision processes (MDPs). As a result, we obtain an agent capable of reasoning sequentially **(ii)** while compensating for missing knowledge **(iv)**.

## 3.1 Introduction

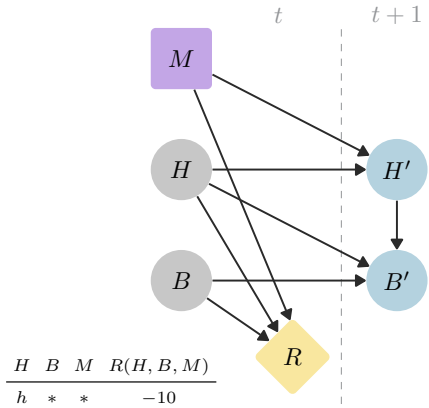
Bayesian networks (BNs) have been widely adopted to model real-world processes under uncertainty (Koller and Friedman 2009; Russell and Norvig 2020). Unfortunately, inference in BNs is computationally hard (#P-hard) (Roth 1996). To deal with this hardness, state-of-the-art techniques apply knowledge compilation (KC), exploiting conditional independencies as well as local structures (Chavira and Darwiche 2008). More recently, KC has also been successfully applied for inference in dynamic models (Vlasselaer et al. 2016), exploiting not only local, but also repeated structure over time.

*Dynamic decision networks* (DDNs) combine dynamic (Dean and Kanazawa 1989; K. P. Murphy 2002) and decision-theoretic[1] (Bhattacharjya and Shachter 2007; Howard and Matheson 1984) Bayesian networks into a single modelling language (Kanazawa and Dean 1989; Russell and Norvig 2020), capable of representing, for instance, Markov decision processes (MDPs) (Bellman 1957).

**Example 3.1.1** (Monkey DDN, Figure 3.1)**.** A monkey tries to hit ($H$) you with suspicious mud. You can decide to move ($M$) or not. If you get hit, you might smell bad ($B$), but the monkey celebrates and is less likely to hit you in the next time step. Your decision and current state hence influence the next state (primed variables). If the monkey misses, it gets angrier and focuses more on its task, increasing the probability to hit you. Moving decreases the probability of being hit. There is a reward ($R$) associated with every state and decision.

Derkinderen and De Raedt (2020) explored a KC approach to model and solve a single-stage maximisation problem under uncertainty. Crucially, they leveraged the algebraic model counting framework (Kimmig et al. 2017) where compiling once allows for solving a variety of tasks on top of the same compiled diagram. In this work we investigate how to apply a similar approach to the dynamic setting, solving dynamic decision networks. To illustrate the benefit of applying the algebraic model counting framework, we then use the same compiled diagram of the decision task for parameter learning.

[1] Note that non-sequential decision networks were originally introduced as 'influence diagrams' (Howard and Matheson 1984), but have more recently been referred to as 'decision networks' (Russell and Norvig 2020, Chapter 15.5).




| $H$ | $M$ | $P(h'\|H, M)$ |
|---|---|---|
| $h$ | $*$ | 0.2 |
| $\neg h$ | $m$ | 0.5 |
| $\neg h$ | $\neg m$ | 0.8 |

| $H$ | $H'$ | $B$ | $P(b'\|H, H'B)$ |
|---|---|---|---|
| $h$ | $h'$ | $b$ | 0.54 |
| $h$ | $\neg h'$ | $b$ | 0.3 |
| $\neg h$ | $h'$ | $b$ | 0.48 |
| $\neg h$ | $\neg h'$ | $b$ | 0.06 |
| $*$ | $*$ | $\neg b$ | 0 |

| $H$ | $B$ | $M$ | $R(H, B, M)$ |
|---|---|---|---|
| $h$ | $*$ | $*$ | $-10$ |
| $*$ | $b$ | $*$ | $-4$ |
| $*$ | $*$ | $m$ | $-1$ |
| $*$ | $*$ | $*$ | 0 |

Figure 3.1: DDN for Example 3.1.1. DDNs provide a graphical representation of the dependencies between variables at time $t$ and $t+1$, the latter represented by using a primed symbol such as $B'$. For DDNs representing MDPs, the current state is fully observed, *i.e.* all non primed state variables are evidence (gray nodes). The '$*$' in the tables represent unspecified truth values. The reward table represents an additive reward function (Russell and Norvig 2020, Chapter 16.4.2). The truth values of each variable are represented with, for instance, $h$ and $\neg h$ for $H$ being true and false, respectively. This is to align with the notation used in the encoding in Section 3.2.

Our main contribution is the introduction of `mapl-cirup` (`Ma`rkov `pl`anning with `cir`cuit `up`dates), read as 'maple syrup'. `mapl-cirup` is a variation of the classic value iteration algorithm that uses KC to exploit dependencies and repeated temporal structures. We adapt the idea by Chavira and Darwiche (2008) to encode BNs as weighted logic formulas and knowledge-compile them into arithmetic circuits for efficient inference. Specifically, we introduce dynamic decision circuits (DDCs) and use them as a compilation target for DDNs. This reduces planning to probabilistic inference in DDCs. As a second contribution we show that the resulting differentiable representation can be used for gradient-based parameter learning in MDPs.

While `mapl-cirup` focuses on planning in fully observable settings using exact inference, real-world applications often involve partial observability. In such settings, agents must reason over belief states, which introduces additional complexity. To highlight how some of the ideas in this chapter extend to this more general setting, at the end of Section 3.3 we will provide a brief intermezzo

on our recent work on belief reuse in online partially observable planning.

## 3.2 Method

The main advantage of KC lies in the possibility of amortising the computationally hard compilation step over multiple circuit evaluations. Assuming that the structure of a DDN does not change over time, the idea is to compile only once the logic formula representing the transition from one time step to the next, and to reuse the resulting compiled circuit to solve the underlying DDN (*i.e.* exploit the repeated structure). This idea is inspired by the work of Vlasselaer et al. (2016), who compile the transition function of dynamic Bayesian networks into a transition circuit. This allows them to perform efficient filtering.

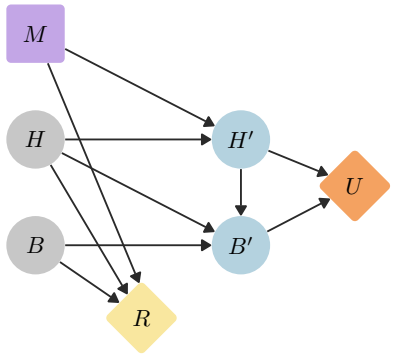
Instead of applying knowledge compilation to the filtering process, however, we introduce a novel encoding to knowledge-compile the Bellman equation (cf. Equation 2.11). More concretely, we want to compile the structure underlying the transition in a DDN into a circuit such that evaluating it using the $\mathcal{S}_{meu}$ algebraic structure (cf. Section 2.2.3) corresponds to a Bellman update. Evaluating the circuit involves three operations (max, sum, and product), in contrast to only two (sum and product) in the work of Vlasselaer et al. (ibid.).

In order to represent a Bellman update, every term in Equation 2.11 should have a corresponding graphic element in its DDN. Comparing Equation 2.11 and the DDN in Figure 3.1, we can see that this is indeed the case for the terms $R(s, a)$ and $P(s' \mid s, a)$. However, there does not exist a graphical equivalent for the accumulated expected reward coming from the future ($U(s')$). Therefore, we augment the DDN with a utility node $U$. In Figure 3.2 we perform this augmentation for the DDN in Figure 3.1.

### 3.2.1 Encoding

Below we describe how to encode DDNs as propositional logic formulas, assuming Boolean random variables. This can easily be extended to multivalued random variables using the encoding from Chavira and Darwiche (2008). Similarly, our encoding also consists of Boolean indicator variables $\lambda$ and Boolean parameter variables $\theta$.

First, we generate *indicator clauses*. Intuitively, they encode the values a variable can assume, and that it must assume exactly one of them. For non-Boolean




| $H'$ | $B'$ | $U(H', B')$ |
|---|---|---|
| $h'$ | $b'$ | $u_1$ |
| $h'$ | $\neg b'$ | $u_2$ |
| $\neg h'$ | $b'$ | $u_3$ |
| $\neg h'$ | $\neg b'$ | $u_4$ |

Figure 3.2: Dynamic decision network (DDN) for Example 3.1.1 augmented with the utility node $U$. For each state, which corresponds to a specific instantiation of the primed variables, we associate a *utility parameter* $u_i$.

variables $Y$ with domain $\{y_1, \ldots, y_n\}$, we have

$$\left(\lambda_{y_1} \vee \ldots \vee \lambda_{y_n}\right) \bigwedge_{\text{for } i<j} \left(\neg\lambda_{y_i} \vee \neg\lambda_{y_j}\right) \tag{3.1}$$

When a variable is instead Boolean, we simply encode it with $\lambda_x$ such that the truth value corresponds to that of $x$.

Second, we introduce *parameter clauses*. Each *parameter variable* $\theta$ corresponds to a value in the probability, reward, or utility tables of a DDN.

- Probabilities $P(x_i'|pa_1, \ldots, pa_n)$

$$\bigvee_{pa_1,\ldots,pa_n} \left(\lambda_{pa_1} \wedge \ldots \wedge \lambda_{pa_n} \wedge \theta_{x_i'|pa_1,\ldots,pa_n}\right) \leftrightarrow \lambda_{x_i'} \tag{3.2}$$

- Rewards $R(x_1, \ldots, x_n) = r_i$

$$\lambda_{x_1} \wedge \ldots \wedge \lambda_{x_n} \wedge \lambda_{r_i} \leftrightarrow \theta_{r_i} \tag{3.3}$$

- Decisions $D_i$

$$\lambda_{d_i} \leftrightarrow \theta_{d_i} \tag{3.4}$$

- Utilities $U(x_1', \ldots, x_n') = u_i$

$$\lambda_{x_1'} \wedge \ldots \wedge \lambda_{x_n'} \wedge \lambda_{u_i} \leftrightarrow \theta_{u_i} \tag{3.5}$$

Therefore, the indicator and parameter clauses together form a propositional logic formula that represents the DDN. Here, the symbol $\leftrightarrow$ denotes a bi-implication, that is, $p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$. Although the encoding is fully Boolean, the indicator and parameter variables will later be assigned different semantics during the labelling step. Specifically, indicator variables $\lambda$ are assigned Boolean values (0 or 1, corresponding to *false* and *true*)whereas parameter variables $\theta$ are assigned the actual numerical values (*e.g.* probabilities) they represent in the conditional probability tables.

**Example 3.2.1** (Encoding of the Monkey DDN)**.** We encode every element of the DDN in Figure 3.2 as described above. The decision variable ($M$): $\lambda_m \leftrightarrow \theta_m$.
The reward ($R$):

$$\begin{array}{l} \lambda_{r_1} \vee \lambda_{r_2} \vee \lambda_{r_3}, \\ \neg\lambda_{r_1} \vee \neg\lambda_{r_2}, \quad \neg\lambda_{r_1} \vee \neg\lambda_{r_3}, \quad \neg\lambda_{r_2} \vee \neg\lambda_{r_3}, \\ \lambda_h \wedge \lambda_{r_1} \leftrightarrow \theta_{r_1}, \quad \lambda_b \wedge \lambda_{r_2} \leftrightarrow \theta_{r_2}, \quad \lambda_m \wedge \lambda_{r_3} \leftrightarrow \theta_{r_3} \end{array}$$

The transition for $H'$:

$$\begin{aligned} \lambda_{h'} \leftrightarrow & (\lambda_h \wedge \theta_{h'|h}) \vee (\neg\lambda_h \wedge \lambda_m \wedge \theta_{h'|\neg h,m}) \\ & \vee (\neg\lambda_h \wedge \neg\lambda_m \wedge \theta_{h'|\neg h,\neg m}) \end{aligned}$$

Note that $\theta_{h'|h,m}$ and $\theta_{h'|h,\neg m}$ were merged together into $\theta_{h'|h}$, effectively exploiting context-specific independence.

The transition for $B'$:

$$\begin{aligned} \lambda_{b'} \leftrightarrow & (\lambda_h \wedge \lambda_{h'} \wedge \theta_{b'|h,h'}) \vee (\neg\lambda_h \wedge \neg\lambda_{h'} \wedge \theta_{b'|\neg h,\neg h'}) \\ & \vee (\lambda_h \wedge \neg\lambda_{h'} \wedge \theta_{b'|h,\neg h'}) \vee (\neg\lambda_h \wedge \lambda_{h'} \wedge \theta_{b'|\neg h,h'}) \\ & \vee (\lambda_b \wedge \theta_{b'|b}) \vee (\neg\lambda_b \wedge \theta_{b'|\neg b}) \end{aligned}$$

Finally, the utility variable ($U$):

$$\begin{array}{l} \lambda_{u_1} \vee \lambda_{u_2} \vee \lambda_{u_3} \vee \lambda_{u_4}, \\ \neg\lambda_{u_1} \vee \neg\lambda_{u_2}, \quad \neg\lambda_{u_1} \vee \neg\lambda_{u_3}, \quad \neg\lambda_{u_1} \vee \neg\lambda_{u_4}, \\ \neg\lambda_{u_2} \vee \neg\lambda_{u_3}, \quad \neg\lambda_{u_2} \vee \neg\lambda_{u_4}, \quad \neg\lambda_{u_3} \vee \neg\lambda_{u_4}, \\ \lambda_{h'} \wedge \lambda_{b'} \wedge \lambda_{u_1} \leftrightarrow \theta_{u_1}, \\ \lambda_{h'} \wedge \neg\lambda_{b'} \wedge \lambda_{u_2} \leftrightarrow \theta_{u_2}, \\ \neg\lambda_{h'} \wedge \lambda_{b'} \wedge \lambda_{u_3} \leftrightarrow \theta_{u_3}, \\ \neg\lambda_{h'} \wedge \neg\lambda_{b'} \wedge \lambda_{u_4} \leftrightarrow \theta_{u_4} \end{array}$$

### 3.2.2 Labelling

We now introduce the labelling function that maps literals to elements of $\mathcal{S}_{meu}$ (cf. Section 2.2.3). Concretely, we define a triple label $(p_\ell, eu_\ell, \mathcal{D}_\ell)$ for each literal $\ell$, representing the probability, the expected utility, and the set of decisions, respectively. This enables us to solve DDNs using AMC.

First, all indicator variables are labelled as:

$$\alpha(\lambda_\ell) = \alpha(\neg\lambda_\ell) = (1, 0, \emptyset), \tag{3.6}$$

which corresponds to the neutral element $e^{\otimes}$ from $\mathcal{S}_{meu}$.

Second, we label the parameter variables.

- Probabilities with $p = P(x_i'|pa_1, \ldots, pa_n)$:

$$\alpha(\theta_{x_i'|pa_1,\ldots,pa_n}) = (p, 0, \emptyset) \tag{3.7}$$

$$\alpha(\neg\theta_{x_i'|pa_1,\ldots,pa_n}) = (1-p, 0, \emptyset) \tag{3.8}$$

- Rewards, where $R(x_1, \ldots, x_n) = r_i$:

$$\alpha(\theta_{r_i}) = (1, R(x_1, \ldots, x_n), \emptyset) \tag{3.9}$$

$$\alpha(\neg\theta_{r_i}) = (1, 0, \emptyset) \tag{3.10}$$

- Decisions:

$$\alpha(\theta_d) = (1, 0, \{d\}), \quad \alpha(\neg\theta_d) = (1, 0, \{\neg d\}) \tag{3.11}$$

- Utilities, where $U(x_1', \ldots, x_n') = u_i$:

$$\alpha(\theta_{u_i}) = (1, U(x_1', \ldots, x_n'), \emptyset) \tag{3.12}$$

$$\alpha(\neg\theta_{u_i}) = (1, 0, \emptyset) \tag{3.13}$$

Note that Equation 3.12 reflects the recursive nature of the Bellman equation. In the context of value iteration we initialise the label

$$\alpha(\theta_{u_i}) = (1, 0, \emptyset) \tag{3.14}$$

**Example 3.2.2** (Labelling of the Monkey encoding)**.** We provide the labelling function for the encoding in Example 3.2.1. All the indicator variables are set as specified by Equation 3.6.

The decision parameter is:

$$\alpha(\theta_m) = (1, 0, \{d\}), \quad \alpha(\neg\theta_m) = (1, 0, \{\neg d\})$$

The reward parameters are:

$$\alpha(\theta_{r_1}) = (1, -10, \emptyset), \quad \alpha(\theta_{r_2}) = (1, -4, \emptyset),$$

$$\alpha(\theta_{r_3}) = (1, -1, \emptyset),$$

$$\alpha(\neg\theta_{r_1}) = \alpha(\neg\theta_{r_2}) = \alpha(\neg\theta_{r_3}) = (1, 0, \emptyset)$$

The probabilities for $H'$:

$$\alpha(\theta_{h'|h}) = (0.2, 0, \emptyset), \quad \alpha(\neg\theta_{h'|h}) = (0.8, 0, \emptyset),$$

$$\alpha(\theta_{h'|\neg h,m}) = (0.5, 0, \emptyset), \quad \alpha(\neg\theta_{h'|\neg h,m}) = (0.5, 0, \emptyset),$$

$$\alpha(\theta_{h'|\neg h,\neg m}) = (0.8, 0, \emptyset), \quad \alpha(\neg\theta_{h'|\neg h,\neg m}) = (0.2, 0, \emptyset)$$

The labelling function for $B'$ can be produced in a similar way. The utility parameters are initially set as specified by Equation 3.14.

### 3.2.3 Compiling

Given the encoding of a DDN as a propositional logic formula, and given the proper labelling function, we can proceed to compiling the circuit. To this end, we can use an off-the-shelf knowledge compiler, label the leaves in the compiled structure, and evaluate the circuit using AMC:

$$U(\mathbf{x}) = AMC(\mathcal{S}_{meu}, \Delta, \alpha_{|\mathbf{x}}), \tag{3.15}$$

which yields the maximum expected utility for the current time step. Here we denote the compiled circuit by $\Delta$, and use the subscript $|\mathbf{x}$ to indicate instantiating the state variables $\mathbf{X}$ to the values $\mathbf{x}$. In the circuit this translates to changing the indicator variables' weights. For instance in the monkey example, to condition on $H = h$, we set $\alpha(\neg\lambda_h) = (0, 0, \emptyset)$.

**Definition 3.2.1** (Decision Circuit, by Bhattacharjya and Shachter (2007))**.** A decision circuit is a rooted, directed, acyclic graph whose leaf nodes are labelled with variables or constants, and whose other nodes are either summation, multiplication, or maximisation.

**Definition 3.2.2** (Dynamic Decision Circuit)**.** A dynamic decision circuit (DDC) is a decision circuit labelled with a recursive labelling function, which

associates each leaf with a value that can depend on the evaluation of the circuit itself.

With Equation 3.15 we can compute the maximum expected utility only if we know the utility coming from the future, *i.e.* $U(s')$. In the next section we show how to perform this AMC call recursively.

## 3.3 `mapl-cirup`

We explained how to obtain a (dynamic decision) circuit $\Delta$ from a DDN, and how to evaluate it —via algebraic model counting— with the labelling function $\alpha$. Next, we provide a variation of the classic value iteration algorithm in which the Bellman update is substituted with an AMC call on $\Delta$.

### 3.3.1 Bellman Update Using Circuits

Algorithm 4 shows `mapl-cirup`'s value iteration approach. Initially, the value of each state is set to zero in Line 4. Then, it iteratively performs Bellman updates until convergence (Line 5 to 11). The Bellman update itself (Equation 2.11) is replaced by an AMC call in Line 7, corresponding to Equation 3.15. Similar to classic value iteration, this update is performed for each state $s \in \mathbf{S}$, *i.e.* for each possible instantiation $\mathbf{x}$ (Line 6). The newly computed values $U'(\mathbf{x})$ are then used in the next update step by updating the utility labels $\alpha(\theta_{\mathbf{u}})$ in Line 8. Note that Line 7 is the core difference between `mapl-cirup` and the classic value iteration algorithm for planning in MDPs: the loop over the states is explicit (Line 6), but the Bellman update is encoded as a DDC. Importantly, the compilation of the circuit is amortised over $|\text{VI}|2^{|\mathbf{X}|}$ steps. Where, $|\text{VI}|$ is the number of iterations required to converge (to the optimal policy), and $|\mathbf{X}|$ is the number of state variables in the dynamic decision network (DDN) given in input.

**Theorem 3.3.1** (Correctness of `mapl-cirup`)**.** Algorithm 4 correctly computes the optimal solution for the problem encoded via the DDC $\Delta$ and labelling function $\alpha$.

*Proof.* (Sketch) The proof follows from the correctness of `mapl-cirup`'s components. Since the convergence of value iteration has already been proven, we must only prove that we correctly use the AMC framework to perform a Bellman update. We use $\mathcal{S}_{meu}$ as it was defined originally (Derkinderen and De Raedt 2020), leaving only the labelling function and the encoding to be

**Algorithm 4** Value Iteration with DDCs

**inputs:** the DDC $\Delta$, the labelling function $\alpha$, the convergence error to terminate $\epsilon$
**local variables:** $U$, $U'$, vectors of utilities for states $\mathbf{x}$; $\delta$ the infinite norm of the change in the utilities
**procedure** MAPL-CIRUP($\Delta, \alpha, \epsilon$)
    $U \leftarrow \mathbf{0}$
    **repeat**
        **for each** instantiation $\mathbf{x}$ **do**
            $U'(\mathbf{x}) \leftarrow AMC(\mathcal{S}_{meu}, \Delta, \alpha_{|\mathbf{x}})$
        $\alpha(\theta_{\mathbf{u}}) \leftarrow U'$
        $\delta \leftarrow ||U' - U||$
        $U \leftarrow U'$
    **until** $\delta < \epsilon$
    **return** $U$

analysed. The labelling function is a mere application of the function described along $\mathcal{S}_{meu}$. We map the DDN parameters to the appropriate labels, and update the parameters $U$ similarly to the Bellman update (Equation 2.11) while the correspondence between states and utility parameters is guaranteed by the encoding. The remainder of the encoding is an adaptation of the encoding by Chavira and Darwiche (2008). □

A computational drawback of `mapl-cirup` is the number of utility parameters $U$. Namely, we introduce one such parameter for each state, to store the state value that is used during the next update step (Figure 3.2, Section 3.2). This negatively affects the number of variables involved in the compiled representation $\Delta$, as well as its size. Moreover, this encoding forces `mapl-cirup` to iterate over all the explicit states. Investigating a more compact representation for the utility function, for example similarly to Hoey et al. (1999), is therefore a promising direction for future work that would solve both problems.

A key benefit of `mapl-cirup` is that $\Delta$ is only compiled once, effectively exploiting the repeating temporal structure and amortizing the compilation cost over multiple iterations. Indeed, $\Delta$ remains constant throughout the whole value iteration process. This also means that it is useful to dedicate more computational resources to achieving a more compact representation $\Delta$ before starting the value iteration process.

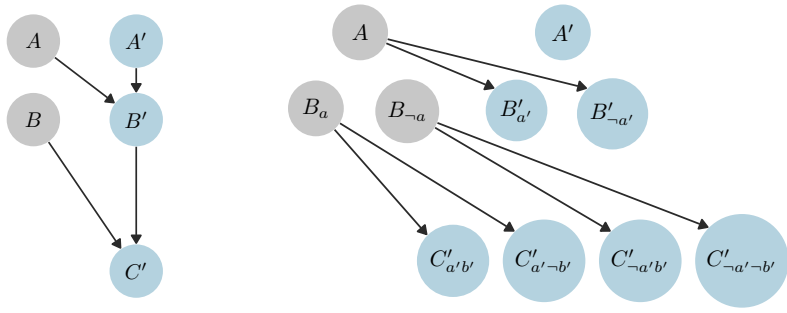



Figure 3.3: Intra-state chain structure (left) leading to an exponential explosion in its MDP formulation (right). Decisions, rewards, and some transitions are omitted for clarity.

### 3.3.2 Intra-state Dependencies

Intra-state dependencies —also called 'correlated action effects' or 'synchronic constraints' (Boutilier et al. 2000)— are dependencies between state variables within the same time slice. An example of this is given by the DDN in Figure 3.1. The probability distribution $P(B'|H, H')$ contains an intra-state dependency where a variable such as $B'$ depends not only on variables from the previous time step, but also on variables from the current time step, *e.g.* $H'$.

To encode the DDN of Figure 3.1 in a symbolic MDP solver that can not manage intra-state dependencies, they must first be removed by adding extra variables (Guestrin et al. 2003). This means splitting $B'$ into $B'_{h'}$ and $B'_{\neg h'}$, where

$$P(B'_{h'}|H, M) = P(B'|H, H' = h) \text{ and}$$

$$P(B'_{\neg h'}|H, M) = P(B'|H, H' = \neg h),$$

for all instantiations of $M$. Flattening out a DDN with a rich intra-state structure may lead to an exponential blow-up, as we exemplify in Figure 3.3. `mapl-cirup` does not suffer from this ailment as it leverages knowledge compilation to exploit exactly these complex dependencies.

### 3.3.3 Learning

As `mapl-cirup` follows the compile+evaluate paradigm, it provides access to the plethora of techniques developed within the algebraic model counting framework. Therefore we can re-use the circuit $\Delta$ to learn for example reward

parameters in the following learning task. We are given a data set $\mathbf{E}$ of trajectories $\tau = \langle s_0, a_{0:k}, r_{0:k} \rangle$, each composed of the initial known state $s_0$, the $k+1$ consecutive actions $a_{0:k}$ taken from that state, and the rewards $r_{0:k}$ obtained after each of those actions. Additionally, we are given the corresponding DDN where each variable may have an unknown reward parameter, *i.e.* the reward is an additive function where (unknown) rewards are associated with state variables. The task consists of learning those unknown reward parameters, while the intermediate states are unobserved.

We tackle this task using a gradient-based approach, exploiting the algebraic framework to compute gradients on top of $\Delta$. To this aim, we introduce a mean squared error loss function:

$$\frac{1}{|\mathbf{E}|} \sum_{\tau \in \mathbf{E}} \sum_{t=0}^{k} \left( ceu_{t;\theta}(s_0, a_{0:t}) - r_t \right)^2 \tag{3.16}$$

$$ceu_{t;\theta}(s_0, a_{0:t}) = \sum_{s_t} P(s_t | s_0, a_{0:t}) R_\theta(s_t, a_t) \tag{3.17}$$

where we parametrise $R_\theta$ with learnable parameters; and use $ceu_{t;\theta}(s_0, a_{0:t})$ as the expected utility at time $t$, given the current parameters $\theta$, the initial state and actions leading up to $t$. By using this loss function, we minimise the difference between the expected utility *ceu* and the actually observed reward at time $t$, $r_t$. This loss function is readily computable from the already compiled $\Delta$. Additionally, probability parameters can be simultaneously learned by integrating the work of Gutmann et al. (2008) on top of our method. To do that, we must account for the rewards and decisions, and therefore use the expected utility semiring on top of DDCs.

## Intermezzo: Belief Re-use in Online Partially Observable Planning

Partially Observable Markov Decision Processes (POMDPs) (Smallwood and Sondik 1973) extend MDPs to account for incomplete state information. They are notoriously used to model problems where an agent needs to make sequential decision-making in a partially observable stochastic environment. In these problems, the agent only has a 'belief' about the current state that is represented by a belief state: a probability distribution over all possible explicit states. While `mapl-cirup` deals with exact inference in fully observable MDPs, here we consider how similar ideas can be achieved when planning over belief spaces in POMDPs. Specifically, we investigate how structural similarities in belief

states can be exploited to improve planning efficiency. Unlike `mapl-cirup`, this approach does not rely on exact knowledge compilation, but instead leverages approximate techniques such as locality-sensitive hashing.

POMDPs are hard problems (Mundhenk et al. 2000; Papadimitriou and Tsitsiklis 1987). Their belief state space grows exponentially on the number of state variables (curse of dimensionality), and also the number of action-observation histories suffers from an exponential growth (curse of history) (Kaelbling et al. 1998). Monte Carlo tree search (MCTS) is a standard technique to tackle these 'curses', both in planning (Browne et al. 2012; Silver and Veness 2010) and in reinforcement learning (Schrittwieser et al. 2020). However, when different action-observation sequences lead to the same state, they are still treated as separate nodes in the MCTS tree, meaning they are expanded and evaluated multiple times. This is further complicated by the fact that POMDPs work on an uncountable set of belief states, where a slight variation in the belief distribution can correspond to the same explicit state.

In Theeuwes et al. (2025) we introduce *Belief Re-use in Online Partially Observable Planning* (BROPOP), which addresses this inefficiency by transforming the MCTS tree into a graph: nodes with similar beliefs are merged and reused as *transposition nodes*. This reduces redundancy in the search while maintaining accuracy. To detect belief similarity, BROPOP introduces a tailored *locality-sensitive hashing* (LSH) scheme that efficiently clusters similar belief states into hash buckets, significantly reducing the number of belief comparisons.

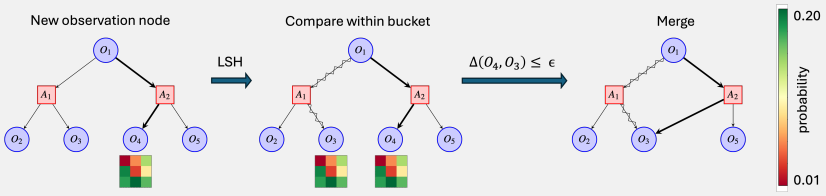



Figure 3.4: BROPOP merges observation nodes when belief similarity falls below a threshold $\varepsilon$, using locality-sensitive hashing (LSH).

Merging nodes introduces the risk of *information leakage* (Saffidine et al. 2012), where value updates propagate inconsistently due to the graph structure. To address this, BROPOP adopts an *update-descent backpropagation* scheme inspired by recent work on graph-based

MCTS (Czech et al. 2021). This mechanism propagates corrected values with minimal overhead and mitigates spurious value accumulation.
Empirical evaluations on classic POMDP benchmarks, such as RockSample (Smith and Simmons 2012), DroneSurveillance (Svoreňová et al. 2015), and Tag (Pineau et al. 2003), show that BROPOP consistently improves reward performance while keeping time complexity under control. Particularly, the use of LSH restores the time efficiency lost in belief comparisons. While dense reward settings (*e.g.* Tag) require careful tuning to avoid distorted value updates, the method demonstrates strong robustness across problem types.
Overall, BROPOP introduces a general and efficient mechanism for belief re-use in POMDP planning, combining hash-based similarity detection with principled value propagation in a unified framework.

## 3.4 Related Work

Similar to `mapl-cirup`, SPUDD (Hoey et al. 1999) is a variation of the classic value iteration algorithm using knowledge compilation. It performs the Bellman update symbolically by replacing its elements with algebraic decision diagrams (ADDs) (Bahar et al. 1997). These ADDs exploit shared values to form compact representations, and support multiplication and addition between each other. During its value iteration process, SPUDD performs multiple compilation operations. `mapl-cirup`, on the other hand, compiles only once and reuses the diagram $\Delta$ multiple times. Moreover, we perform parameter learning, which is not tackled by SPUDD, on top of the same $\Delta$.

Although SPUDD was introduced over two decades ago, it is still considered the state-of-the-art approach for solving factored MDPs *exactly*, as mentioned in multiple recent publications on approximate methods (Dudek et al. 2022; Hayes et al. 2021; Heß et al. 2021; Moreira et al. 2021; Tan and Nejat 2022).

Vlasselaer et al. (2016) investigate how dynamic Bayesian networks (K. P. Murphy 2002) benefit from knowledge compilation techniques. However, they do neither investigate the decision-theoretic setting, nor perform learning. In an orthogonal way, Derkinderen and De Raedt (2020) use algebraic model counting with circuits in order to compute expected utilities and optimise decisions. Their work is restricted to non-temporal problems, while we consider a setting with decisions spanning over time.

Finally, *recurrent sum-product-max networks* (Tatavarti et al. 2021) have recently

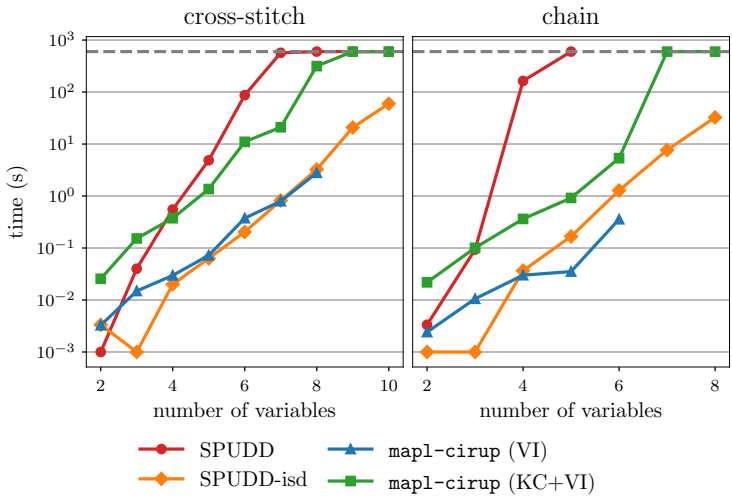



Figure 3.5: Comparison between `mapl-cirup` and SPUDD. It reports the value iteration time (VI) and the total time (KC+VI) including the compilation. SPUDD does not distinguish them because it interleaves compilation and VI. The dashed line indicates the timeout of 600s. Note that when knowledge compilation times out, VI is not performed, hence the absence of the last two points for `mapl-cirup` (VI).

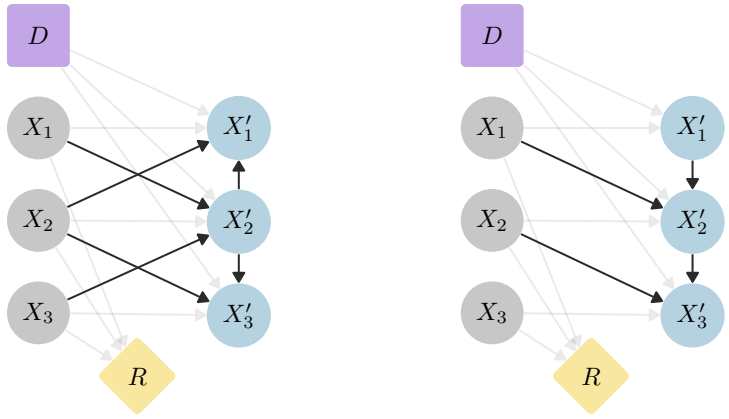
been introduced. These are circuits whose structure and parameters are learned directly from data and not compiled from a model. They learn from fully-observed trajectories where the total accumulated reward is provided as a signal, making it a different learning task than ours. In addition, to the best of our understanding, their approach does not perform an exact Bellman update. They update the utility value per-variable, instead of considering all the explicit states, making it more similar to a linear approximation (Guestrin et al. 2003).

## 3.5 Experiments

We performed our experimental evaluation with an Intel CPU E3-1225v3 @3.20 GHz and 32 GB of memory. All experiments ran 10 times and we report the average run time. We omit the variance when negligible. The hyperparameters for `mapl-cirup` were as follows: discount factor $\gamma = 0.9$ and tolerance $\epsilon = 0.1$. As a timeout, we use 600s of total run time (indicated by a dashed line on the figures). On the implementation side we used the PySDD

| | | SPUDD | | mapl-cirup | | |
|---|---|---|---|---|---|---|
| model | $\lvert\mathbf{X}\rvert$ | $\lvert\Delta\rvert$ | VI [s] | $\lvert\Delta\rvert$ | KC [s] | VI [s] |
| monkey | 2 | 11 664 | $< 0.01$ | 163 | 0.01 | 0.005 |
| elevator | 4 | 5 794 | $< 0.01$ | 277 | 0.02 | 0.003 |
| coffee | 6 | 142 519 | 0.03 | 2542 | 0.6 | 0.054 |
| factory | 7 | 38 163 | 0.01 | 2932 | 0.93 | 0.105 |

Table 3.1: Comparison between mapl-cirup and SPUDD on the circuit size ($|\Delta|$), *i.e.* the total number of nodes, the compilation time (KC), the time to find the best solution (VI). SPUDD reports time with a precision of 0.01s. It does not provide the knowledge compilation time.




(a) Cross-stitch (b) Chain

Figure 3.6: Dynamic decision networks with structure (in bold) and $n = 3$ variables.

package (Darwiche et al. 2018). In order to mitigate some of the performance issues with Python3 we JIT-compiled the circuit for the Bellman update using Numba (Lam et al. 2015). As a comparison we include SPUDD (version 3.6.2, written in C++), with the same hyperparameters.[2]

**(Q1) How does mapl-cirup perform in general?** To address this question, we evaluate mapl-cirup on three MDP instances of different sizes from the SPUDD repository: elevator, coffee, and factory. We additionally include the monkey instance of Example 3.1.1, and design two DDN families

[2] https://github.com/ML-KULeuven/mapl-cirup

that are parametric in the number of state variables $|\mathbf{X}|$ as to better illustrate the scalability. The first family has a cross-stitch-like structure, while the second forms a chain of dependencies. These are depicted in Figure 3.6. We devised them also to investigate how the structure imposed by intra-state dependencies can be exploited by `mapl-cirup` and SPUDD. In particular, the chain-like structure represents an extreme case, because the cascade of dependencies leads to an exponential blow-up (cf. Section 3.3.2). On the other hand, the cross-stitch-like structure represents a more realistic scenario, where only some of the state variables depend on others. As a baseline and to verify correctness, we compare to the SPUDD algorithm. Notice that SPUDD, as it was introduced by Hoey et al. (1999), can not exploit these dependencies. In order to handle them, it requires a non-trivial extension, as described by Boutilier et al. (2000), which is implemented in the official release of SPUDD. Therefore, for the parametric instances we distinguish between SPUDD and SPUDD-isd, even though it is a single implementation. Where, the former corresponds to the original algorithm, and it requires the flattening described in Section 3.3.2 to handle intra-state dependencies, while the latter includes the extension that can manage them. The results reported in Table 3.1 and Figure 3.5 indicate that `mapl-cirup` is able to solve dynamic decisions problems up to reasonable sizes. Both `mapl-cirup` and SPUDD suffer from the exponential explosion inherent to the hardness of decision making problems. Due to the utility function representation, this impacts `mapl-cirup` more than SPUDD. `mapl-cirup` produces a much smaller circuit ($|\Delta|$) than SPUDD since it only knowledge-compiles a single time step and reuses the same circuit $\Delta$ over time. In contrast, SPUDD manipulates the circuits, compiling new ones at each iteration resulting in a larger total node count.

**(Q2) How does the exponential representation of the utility function influence performance?** The main drawback of `mapl-cirup` is how it currently represents the utility function explicitly inside the circuit itself (cf. Section 3.3.2). Therefore, we explored the effect of removing the exponential encoding of the utility function by means of a well-known linear approximation (Guestrin et al. 2003), which reduces the number of utility nodes in the circuit from $2^{|\mathbf{X}|}$ to $|\mathbf{X}|$ (cf. Section 3.3.2).

**Definition 3.5.1.** A linear value function over a set of basis functions $H = \{h_1, \ldots, h_k\}$ is a function $V$ that can be written as $V(\mathbf{x}) = \sum_{j=1}^{k} w_j h_j(\mathbf{x})$ for some weights $\mathbf{w} = (w_1, \ldots, w_k)'$.

Normally these weights are learned, but we randomly initialised them, without focusing on the accuracy of the approximation, because it was out of the scope of our work. However, since in this way the approximation breaks the

| | cross-stitch | | | chain | | |
|---|---|---|---|---|---|---|
| $\lvert\mathbf{X}\rvert$ | $\lvert\Delta\rvert$ | $\lvert\text{VI}\rvert$ | $\lvert\Delta_{\approx}\rvert$ | $\lvert\Delta\rvert$ | $\lvert\text{VI}\rvert$ | $\lvert\Delta_{\approx}\rvert$ |
| 2 | 327 | 20 | 352 | 259 | 17 | 205 |
| 3 | 815 | 45 | 554 | 632 | 39 | 461 |
| 4 | 2 220 | 21 | 1 244 | 1 851 | 37 | 648 |
| 5 | 4 230 | 37 | 1 277 | 3 872 | 19 | 756 |
| 6 | 12 606 | 52 | 1 738 | 11 005 | 51 | 999 |
| 7 | 17 365 | 49 | 2 364 | – | – | 1 265 |
| 8 | 39 623 | 53 | 2 753 | – | – | 2 223 |
| 9 | – | – | 3 242 | – | – | 1 481 |
| 10 | – | – | 2 989 | – | – | 1 816 |

Table 3.2: Comparison of `mapl-cirup`'s circuit size, with ($\lvert\Delta_{\approx}\rvert$) and without ($\lvert\Delta\rvert$) the linearly approximated utility function. Included is the number of iterations until convergence ($\lvert\text{VI}\rvert$), and the number of variables ($\lvert\mathbf{X}\rvert$).

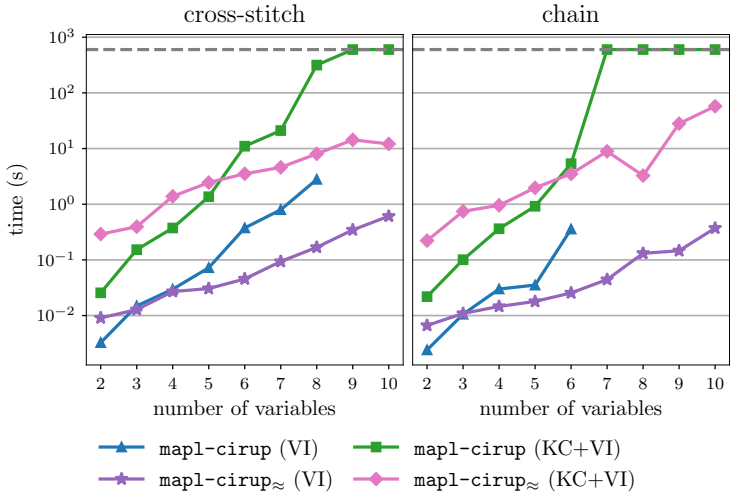



Figure 3.7: Comparison with and without a linearly approximated utility function. We report the value iteration time (VI) and the total time (KC+VI) including the compilation. The dashed line indicates the timeout of 600s. Note that when knowledge compilation times out, VI is not performed, hence the absence of the last two points for `mapl-cirup` (VI).

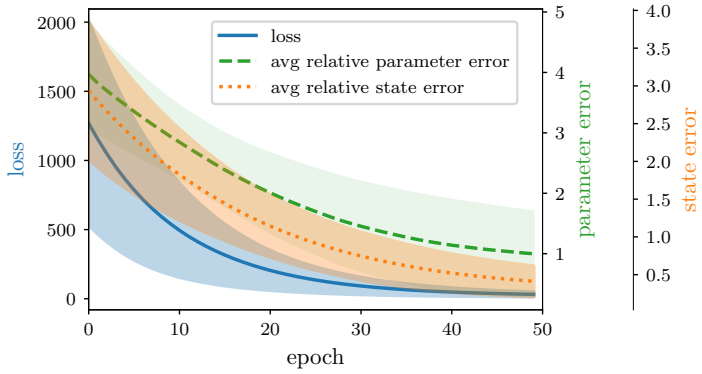
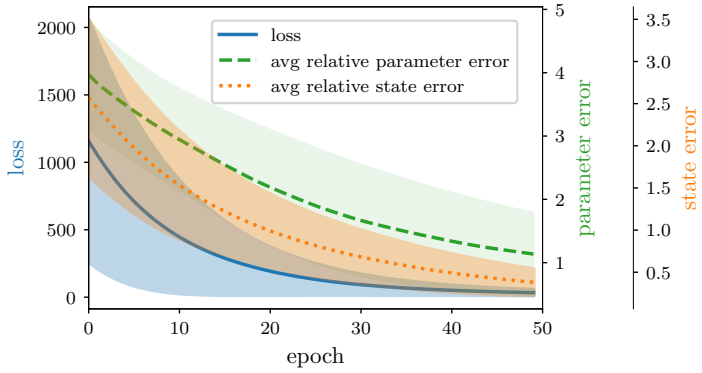


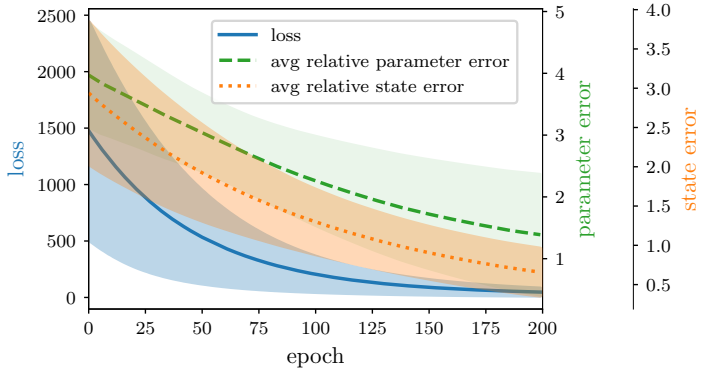

Figure 3.8: Results of learning reward parameters on the `coffee` example. It plots the average of each metric over 10 runs (line) and the standard deviations (coloured region).

convergence property, we set an horizon limit of 50 iterations ($\sim$ the maximum number of iterations required by the exact algorithm to convergence) during our experiments. We report the circuit sizes $|\Delta_{\approx}|$ in Table 3.2 and the run times in Figure 3.7. These results show that indeed the utility function encoding impacts the size of $\Delta$ and thereby the run time. It also demonstrates that improvements are possible, and that these can be compatible with our compilation approach that exploits structure and enables differentiable learning (cf. Q3).

**(Q3) Is the loss function a good indicator for learning the rewards?** We consider two evaluation metrics: the average relative error on i) the reward parameters, and ii) the reward of each state. The latter metric is important for planning, because the policy is computed on the total reward of each state. We use the previously outlined learning method, using the expected utility semiring to compute the loss function, and TensorFlow's automatic differentiation abilities (Martín Abadi et al. 2015) to extract the gradients. The Adam optimiser (Kingma and Ba 2015) was used with learning rate $\alpha = 0.1$, $\hat{\epsilon} = 10^{-7}$, and the rest of the parameters set as default. Moreover, we initialised the reward parameters with values sampled uniformly from the interval of integers $[-30, 30]$. The dataset contains 100 trajectories ($|\mathbf{E}| = 100$) each of length 5 ($k = 5$). The training was performed on batches of size 10. We demonstrate the ability of the method by focusing on the `coffee` example, with additional reward parameters. To generate the dataset we sampled the `coffee` example enriched with extra reward parameters to make it more challenging, and initialised them with values sampled uniformly from the interval of integers




(a) Highly stochastic transition function




(b) Small dataset (10 trajectories)

Figure 3.9: Results of learning reward parameters on the `coffee` example. It plots the average of each metric over 10 runs (line) and the standard deviations (coloured region).

[1, 10]. When learning, we assume that we do not know the distribution of the reward parameters. We ran the learning method 10 times, reporting the average and standard deviation in Figure 3.8. These results show the parameter and state error decreasing along with the loss, affirmatively answering Q3 that the loss function acts as an approximate signal for learning the rewards. Specifically, we significantly decrease the relative state error from 2.94 to 0.41 on average. The relative parameter error is slightly larger, which is possible due to the additive nature of the reward function, and the additional freedom this yields to represent the state rewards.

In order to further evaluate the capabilities of our learning task we executed two extra experiments. First, we changed the transition function of the `coffee` example to be less deterministic. The results are reported in Figure 3.9a. The quality of the learned parameters is only slightly influenced, but a higher variance is also noticeable. This is expected since we do not observe the intermediate states of a trajectory. Second, we perform the training on a smaller dataset with 10 trajectories instead of 100 (still of length $k = 5$), and batches of size 5 (instead of 10). The results are reported in Figure 3.9b. Interestingly, despite the fact that, of course, more epochs are required, and more variance is associated with the relative parameters error, even with such a small amount of data the learned parameters have a good quality, close to the ones learned from 100 examples.

## 3.6 Conclusions and Future Work

We introduced dynamic decision circuits, together with a value-iteration-based algorithm called `mapl-cirup` that reduces Markov planning to inference in DDCs. Thanks to the compile+evaluate paradigm, the compiled diagram can be used within the algebraic model counting framework to solve other tasks. We specifically identified and solved a learning task, where the reward parameters are learned from trajectories, in an offline reinforcement learning fashion. Our approach goes beyond planning for factored representations. It is the starting point to integrate exact inference methods with new approximate approaches such as policy gradient reinforcement learning.

As future work we consider investigating more efficient representations for the utility function, *e.g.* by integrating the ADD operations used within SPUDD, since it alleviates the computational cost while keeping the method exact. Orthogonal to that, approximation methods (St-Aubin et al. 2000) can be combined with `mapl-cirup`, leading again to improved run times and the possibility to scale to larger domains. In addition to the explored learning setting, exploiting the compiled, differentiable representation for other tasks, such as sensitivity analysis (Bhattacharjya and Shachter 2010), may be of interest.

# Chapter 4

# Relational Neurosymbolic Markov Models

This chapter is based primarily on the following peer-reviewed conference publication.

> L. De Smet*, G. Venturato*, L. De Raedt, and G. Marra (2025). "Relational neurosymbolic Markov models". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 39. 15, pp. 16181–16189

L. De Raedt and G. Marra posed the question as to whether one could develop a neurosymbolic extension of sequential probabilistic models. L. De Smet and I jointly devised the concept, combining my expertise from the work in Chapter 3 with his background in gradient estimation and hybrid (continuous-discrete) inference. Together, we developed the theoretical framework, implemented the algorithms, and conducted the experiments. G. Marra and L. De Raedt provided valuable input throughout the project, contributing to both the conceptual development, the refinement of the methodology and supervised the research. The paper was written by L. De Smet and me, with contributions and feedback from all co-authors.

The preliminaries have been fully moved to Chapter 2 and expanded with additional details on particle filters and probabilistic logic programming to

ensure self-containment. Section 4.3.5 is new and provides insights into the complexity of the inference algorithm.

In this chapter, we focus on both the reasoning and perception modules from Figure 1.1. Specifically, we address research questions **RQ1**, **RQ4**, and **RQ5**, exploring how Markov models can be integrated with neurosymbolic AI to obtain sequential neurosymbolic models capable of both discriminative and generative tasks. As a result, we obtain an agent that can perceive the world **(i)**, while sequentially reason about it **(ii)**, compensate for missing knowledge **(iv)**, and generalise across objects or entities **(v)**.

## 4.1 Introduction

Markov models are the theoretical foundation for many successful applications of artificial intelligence, such as speech recognition (Juang and Rabiner 1991), meteorological predictions (Khiatani and Ghose 2017), games (Schrittwieser et al. 2020), music generation (Austin et al. 2021), sports analytics (Van Roy et al. 2023) and many more (Mor et al. 2020). They are so popular mainly because they naturally factorise a sequential problem into step-wise probability distributions. Such a decomposition leads to better predictions in terms of bias and variance compared to models that do not incorporate the sequential nature of the problem (Bishop 2006).

Neurosymbolic AI (NeSy) has also enjoyed a tremendous increase in attention. Its general goal is to combine the generalisation potential of symbolic, *i.e.* logical, reasoning with the representational learning prowess of neural networks. This integration can improve interpretability (Ciatto et al. 2024) and provably satisfy logical constraints. For example, to guarantee the safety of an autonomous agent (W.-C. Yang et al. 2023), to constrain autoregressive language generation (H. Zhang, Dang, et al. 2023) or to impose physical modelling into temporal forecasting (Reichstein et al. 2019).

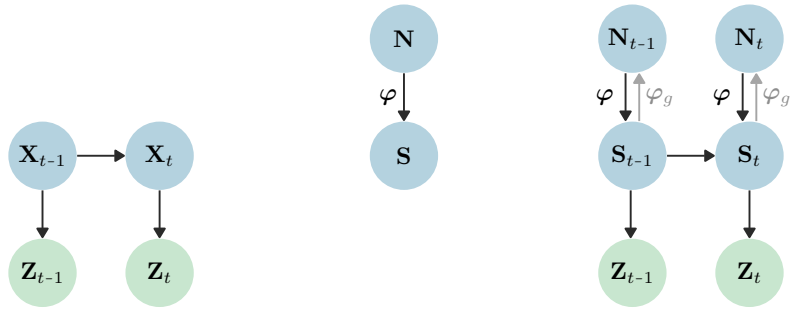
Such a combination already exists in many different flavours, using either fuzzy logic (Badreddine et al. 2022) or probabilistic logic (De Smet, Zuidberg Dos Martires, et al. 2023; J. Huang et al. 2021; Manhaeve, Dumančić, et al. 2021; Z. Yang et al. 2020), and either propositional or relational logic (Marra et al. 2024). The probabilistic case is of special interest, as probabilistic NeSy systems provide a sound semantics to handle uncertainty, as well as to tackle generative tasks. The relational case is also of special interest as relational logic is a popular and very expressive representation for

representing states in, for instance, databases and planning (Russell and Norvig 2020). Moreover, relational representations facilitate strong generalisation behaviour (Hummel and Holyoak 2003). Unfortunately, existing probabilistic or relational NeSy models can not exploit the sequential decomposition inherent to temporal reasoning tasks, thereby limiting their applicability in complex sequential problems. Therefore, there are still no inference algorithms for such NeSy models that are tailored to scale in sequential settings.

In order to overcome these limitations, we identify four desiderata that a model and its inference algorithm should satisfy. **(D.I)** It must be able to model and exploit relational logical constraints on states and transition functions. It should use relational states as in planning, and ideally it can cope with both continuous and discrete aspects of reality. **(D.II)** It has to exploit sequential dependencies without restricting the modelling power, allowing it to scale further than existing NeSy systems. **(D.III)** It must be properly neurosymbolic, that is, it must support transition functions that are logical, neural or purely probabilistic in nature, or any combination thereof. Moreover, it must be end-to-end differentiable to allow for the optimisation of any neural components of the model. **(D.IV)** It can tackle both discriminative and generative tasks in a probabilistic fashion.

Both existing probabilistic techniques and neurosymbolic AI are insufficient. On the neurosymbolic side, scalability **(D.II)** remains the biggest problem, and generative capabilities **(D.IV)** are also lacking. Purely exact techniques (Manhaeve, Dumančić, et al. 2021; Z. Yang et al. 2020) do not scale to non-trivial time horizons, while approximate techniques (J. Huang et al. 2021; Krieken et al. 2024; Penning et al. 2011; Tran and Garcez 2023; Tran 2021) are still limited, *e.g.* they are statistically biased, lack guarantees, or do not support generative tasks. Only few exact NeSy systems can tackle generative tasks (De Smet, Zuidberg Dos Martires, et al. 2023; Misino et al. 2022) and those that can, are very limited in scalability. Moreover, most NeSy systems use neural predicates only for perception. While some NeSy methods for structure learning exist (Towell and Shavlik 1994; F. Yang et al. 2017), we want to be able to learn both perception and structure at the same time through parameter learning. On the side of probabilistic models, only desideratum **(D.IV)** can be fully met. Non-parametric techniques can infer any generic hidden Markov model (Koller and Friedman 2009) and have been applied in the statistical relational setting (Nitti et al. 2016). However, their integration with the neural paradigm is often paired with strong distributional assumptions (Krishnan et al. 2017), such as requiring Gaussian densities. In particular, gradient-based optimisation is often difficult for general approximate Bayesian inference methods (Corenflos et al. 2021; Ścibior et al. 2021; Younis and E. B. Sudderth 2023).

To fulfil all desiderata, we introduce *relational neurosymbolic Markov models*




(a) HMM (b) NeSy (c) NeSy-MM

Figure 4.1: Probabilistic graphical model representations of the different systems considered in this work. Blue represents the states, green the observations.

(NeSy-MMs), the first integration of deep sequential probabilistic models with NeSy. In particular, (i) we provide a formal definition of NeSy-MMs, (ii) we introduce a new differentiable neurosymbolic particle filter that combines Rao-Blackwellised (Liu et al. 2019) inference and state-of-the-art discrete gradient estimation, (iii) we provide an implementation of such models, and (iv) we introduce two new benchmarks for generative and discriminative learning, and run an extensive experimental analysis on both. The results show that NeSy-MMs satisfy all the desiderata **(D.I)** - **(D.IV)**.

## 4.2 Method

*Relational neurosymbolic Markov models (NeSy-MMs)* combine the sequential and partially observable nature of HMMs and DBNs (Figure 4.1a) with neurally parametrised relational probability distributions (Figure 4.1b). That is, we consider Markov processes $\mathbf{X} = (\mathbf{X}_t)_{t\in\mathbb{N}}$ with observations $\mathbf{Z} = (\mathbf{Z}_t)_{t\in\mathbb{N}}$ where the state $\mathbf{X}_t$ is now a *neurosymbolic state* $\mathbf{X}_t = (\mathbf{N}_t, \mathbf{S}_t)$. Here, $\mathbf{N}_t$ denotes the *subsymbolic* component, typically a continuous vector $\mathbf{N}_t \in \mathbb{R}^n$ representing low-level perceptual or feature-based aspects of the world. Conversely, $\mathbf{S}_t$ denotes the *symbolic* component, comprising discrete or continuous random variables that represent high-level concepts and the relations between them. We refer to a neurosymbolic state as *finite* when its symbolic part $\mathbf{S}_t$ involves only finite-domain variables (e.g., Boolean or categorical symbols), and as *infinite* when it involves only continuous or countably infinite-domain variables. Unless otherwise specified, we consider the general case where both types of variables coexist. Figure 4.1c depicts the graphical model of this novel integration.

NeSy-MMs represent joint probability distributions $p_{\boldsymbol{\varphi}}(\mathbf{N}, \mathbf{S}, \mathbf{Z})$ that factorise as

$$p_{\boldsymbol{\varphi}}(\mathbf{S}_0 \mid \mathbf{N}_0) p(\mathbf{N}_0) p(\mathbf{Z}_0 \mid \mathbf{S}_0)$$
$$\prod_{t \in \mathbb{N}} p_{\boldsymbol{\varphi}}(\mathbf{S}_{\mathbf{t+1}} \mid \mathbf{S}_t, \mathbf{N}_{t+1}) p(\mathbf{N}_{t+1}) p(\mathbf{Z}_{t+1} \mid \mathbf{S}_{t+1}). \tag{4.1}$$

Despite the similarity with Eq. 2.5, NeSy-MMs are complex models that define a wide variety of distributions, taking into account our four desiderate of interest **(D.I)** - **(D.IV)**.

**NeSy-MMs explicitly model symbols and their relations.** Having a NeSy state means we perform inference in a symbolic state space where *relational logic rules* $\mathcal{R}$ govern the relationship between symbols, both within a single time slice and in the transition between states. This relational symbolic space allows NeSy-MMs to incorporate human knowledge into our reasoning process, giving guarantees on how the sequential process evolves, *e.g.* we can guarantee safety properties throughout the entire sequence (see Example 4.2.1). Additionally, the relational aspect significantly enhanced the out-of-distribution generalisation potential (Section 4.4).

**NeSy-MMs factorise symbols over sequences.** Standard NeSy systems (Figure 4.1b) must model the full joint distribution over time, *i.e.* $p(\mathbf{S}) = p(\mathbf{S}_1, \ldots, \mathbf{S}_t)$. In contrast, we can factorise the distribution thanks to the Markovian neurosymbolic transition function $p_{\boldsymbol{\varphi}}(\mathbf{S}_{\mathbf{t+1}} \mid \mathbf{S}_t, \mathbf{N}_{t+1})$, allowing for the definition of probabilistic temporal relations between symbols. Moreover, such a factorisation dramatically simplifies the symbolic space by exploiting the sequential dependencies that standard NeSy systems ignore.

**NeSy-MMs integrate neural and logical parametrisations.** The symbols of a NeSy-MM and their transitions need not be purely logical and can be parametrised by neural networks. This flexibility in parametrisation not only bridges the gap between subsymbols and symbols, but also allows for neural nets to fill in gaps in background knowledge. For example, when faced with learning the behaviour of another entity in a game while being constrained by the rules of the game (Section 4.4.2). In essence, NeSy-MMs place symbols and logic where knowledge is available, while using neural nets to parametrise symbols and structure where necessary.

**NeSy-MMs express discriminative and generative neurosymbolic models.** When given a target variable $Y$, which can be any of the symbols in $\mathbf{S}$ or a logical derivation thereof, a NeSy-MM can answer conditional *discriminative NeSy queries* of the form $p_{\boldsymbol{\varphi}}(y \mid \mathbf{n}, \mathbf{z})$ via

$$\int_{\mathbf{s} \models_{\mathcal{R}} y} p_{\boldsymbol{\varphi}}(\mathbf{s}_0 \mid \mathbf{n_0}, \mathbf{z_0}) \prod_{t \in \mathbb{N}} p_{\boldsymbol{\varphi}}(\mathbf{s_{t+1}} \mid \mathbf{s}_t, \mathbf{n}_{t+1}, \mathbf{z}_{t+1}) \, \mathrm{d}\mathbf{s}. \tag{4.2}$$

Alternatively, we can assume a generative perspective (Dinh et al. 2016; Goodfellow et al. 2020; Ho et al. 2020), *i.e.* the inverted $\boldsymbol{\varphi}_g$ edges in Figure 4.1c. This leads to the factorisation

$$\int p_{\boldsymbol{\varphi}}(\mathbf{s}_0, \mathbf{N}_0 \mid \mathbf{z}_0) \prod_{t \in \mathbb{N}} p_{\boldsymbol{\varphi}}(\mathbf{s_{t+1}}, \mathbf{N}_{t+1} \mid \mathbf{s}_t, \mathbf{z}_{t+1}) \, \mathrm{d}\mathbf{s}, \tag{4.3}$$

of $p_{\boldsymbol{\varphi}}(\mathbf{N} \mid \mathbf{z})$. That is, NeSy-MMs can tackle *generative tasks* where samples $\mathbf{n}$ from $p_{\boldsymbol{\varphi}}(\mathbf{N} \mid \mathbf{z})$ that satisfy the possibly logical evidence $\mathbf{z}$ are asked. We showcase this functionality in Section 4.4.3, where we use a VAE (Kingma and Welling 2013) to generate sequences of images of a game that adhere to the rules of the game.

**Algorithm 5** Logic programming encoding of Example 4.2.1.

```
player(Im, P)₀ ~ normal(noisy_player(Im)).
player(Im, P)ₜ ~ normal(Next) :-
    player(Im, P)ₜ₋₁, monster(Im, M),
    player_move(P, M, Next).
monster(Im, M) ~ normal(noisy_monster(Im)).
clumsy ~ bernoulli(0.75).

hits(M, P)ₜ :-
    distance(M, P, D)ₜ, D < 2, not clumsy.

game_overₜ(Im) :-
    player(Im, P)ₜ, monster(Im, M),
    hits(M, P)ₜ.

safeₜ(Im, P) :-
    player(Im, P)ₜ, monster(Im, M),
    distance(M, P, D)ₜ, D > 2.

observe(safe₀:T, true).
```

**Example 4.2.1.** Figure 4.2 shows a new version of the game from Example 2.1.3, and Algorithm 5 its logic programming encoding. The player can now move in the environment with a Markovian transition function `player_move` based on the player's previous location and the static monster's position. The observation

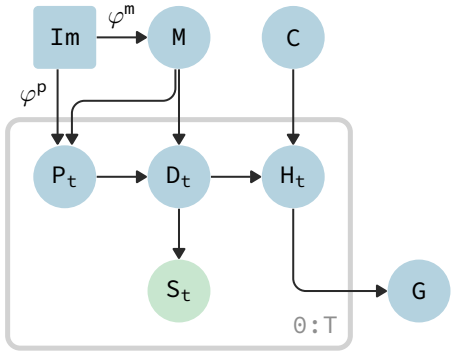



Figure 4.2: Graphical model of Example 4.2.1. We use plate notation to indicate a Markov transition. A rolled-out version is available in Figure 4.3.

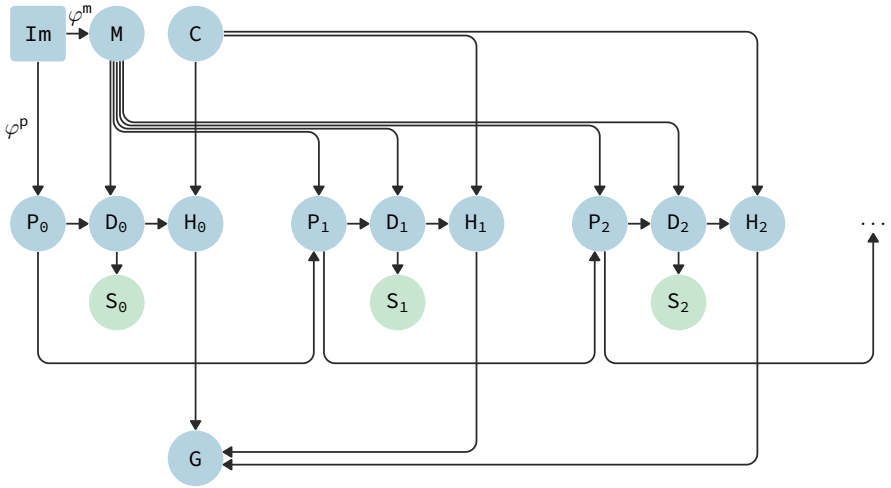



Figure 4.3: Rolled-out graphical model of Figure 4.2, from Example 4.2.1.

rule `safe` guarantees the player's safety at every time step within the horizon `0:T`. Finally, we can query $\texttt{game\_lost(image.png)}_{\texttt{t}}$ for any $\texttt{t} \in \{\texttt{0}, \ldots, \texttt{T}\}$. Notice that this NeSy-MM depends only on the first image at time `t=0` and that the projection into the future is done via the logical transition rules.

## 4.3 Inference and Learning

To bridge the gap between NeSy and sequential probabilistic models, we propose a new, differentiable inference technique that combines non-parametric approximate Bayesian inference with exact NeSy inference. In the following sections, we will distinguish between random variables with finite and infinite domains. The latter includes both countably infinite and continuous (uncountable) domains.

### 4.3.1 Neurosymbolic Particle Filtering

Let us focus on the global NeSy state variable $\mathbf{X} = (\mathbf{N}, \mathbf{S})$ and how it evolves over time. The recursive equation of a particle filter applied to a NeSy-MM $(\mathbf{X}, \mathbf{Z})$ computing the probability of a state $\mathbf{X}_{t+1}$ at time step $t{+}1$ given observations $\mathbf{Z}_{0:t+1}$ from time steps 0 to $t{+}1$ is

$$p_{\boldsymbol{\varphi}}(\mathbf{X}_{t+1} \mid \mathbf{Z}_{0:t+1}) = \frac{p_{\boldsymbol{\varphi}}(\mathbf{Z}_{t+1} \mid \mathbf{X}_{t+1})}{p_{\boldsymbol{\varphi}}(\mathbf{Z}_{t+1} \mid \mathbf{Z}_{0:t})} \int p_{\boldsymbol{\varphi}}(\mathbf{X}_{t+1} \mid \mathbf{x}_t) p_{\boldsymbol{\varphi}}(\mathbf{x}_t \mid \mathbf{Z}_{0:t}) \, \mathrm{d}\mathbf{x}_t. \quad (4.4)$$

Practical implementations of these recursive equations require three steps. First, a set $\{\mathbf{x}_t^{(n)}\}_{n=1}^N$ of $N$ samples is drawn from the current time $t$ distribution, *i.e.* $\mathbf{x}_t^{(n)} \sim p_{\boldsymbol{\varphi}}(\mathbf{X}_t \mid \mathbf{Z}_{0:t})$. Then, this set is transitioned to the next time step via the transition distribution $p_{\boldsymbol{\varphi}}(\mathbf{X}_{t+1} \mid \mathbf{x}_t)$. Finally, each of these samples is reweighted according to the observation probabilities $p_{\boldsymbol{\varphi}}(\mathbf{Z}_{t+1} \mid \mathbf{X}_{t+1})$. The resulting set of weighted samples is therefore approximately distributed according to $p_{\boldsymbol{\varphi}}(\mathbf{X}_{t+1} \mid \mathbf{Z}_{0:t+1})$.

Instead of keeping a set of samples $\{\mathbf{x}_t^{(n)}\}_{n=1}^N$ weighted by their observations $p_{\boldsymbol{\varphi}}(\mathbf{Z}_{0:t+1} \mid \mathbf{x}_t^{(n)})$, practical particle filters use *resampling* to avoid the sample set from collapsing to samples with very low probability. Concretely, they use the observation probabilities $p_{\boldsymbol{\varphi}}(\mathbf{Z}_{0:t+1} \mid \mathbf{x}_t^{(n)})$ as the weights of a finite random variable with $N$ outcomes, one for each sample $\mathbf{x}_t^{(n)}$ and take $N$ samples from the distribution of this variable to use as the recursive set of samples for the next filtering step. Unfortunately, while the original set of weights could be differentiated and used to approximate gradients for every sample,

the introduction of this auxiliary finite random variable is not differentiable, preventing a resampling particle filter from directly being used in our setting.

### 4.3.2 Differentiable Neurosymbolic Particle Filtering

As we just discussed, traditional particle filters are not differentiable because they perform resampling. Resampling is needed because the observations $\mathbf{Z}_t$ are separated from the transitions $p_{\boldsymbol{\varphi}}(\mathbf{X}_{t+1} \mid \mathbf{X}_t)$, which means the conditional distribution $p_{\boldsymbol{\varphi}}(\mathbf{X}_{t+1} \mid \mathbf{X}_t, \mathbf{Z}_{t+1})$ is not readily available. The current state-of-the-art solution is to recover the differentiability of resampling (Corenflos et al. 2021; Ścibior et al. 2021; Younis and E. B. Sudderth 2023). On the contrary, we propose a novel solution that takes advantage of the neurosymbolic nature of a NeSy-MM. In particular, we circumvent the problem of differentiating through resampling by using a Rao-Blackwellised particle filter (RBPF) (K. Murphy and Russell 2001). A RBPF assumes $p_{\boldsymbol{\varphi}}(\mathbf{X}_{t+1} \mid \mathbf{X}_t, \mathbf{Z}_{t+1})$ can be computed exactly and uses it to recursively compute $p_{\boldsymbol{\varphi}}(\mathbf{X}_{t+1} \mid \mathbf{Z}_{0:t+1})$ as

$$\propto \int p_{\boldsymbol{\varphi}}(\mathbf{X}_{t+1} \mid \mathbf{x}_t, \mathbf{Z}_{t+1}) p_{\boldsymbol{\varphi}}(\mathbf{Z}_{t+1} \mid \mathbf{x}_t) p_{\boldsymbol{\varphi}}(\mathbf{x}_t \mid \mathbf{Z}_{0:t}) \, \mathrm{d}\mathbf{x}_t. \tag{4.5}$$

We claim it is viable to compute $p_{\boldsymbol{\varphi}}(\mathbf{X}_{t+1} \mid \mathbf{X}_t, \mathbf{Z}_{t+1})$ in our NeSy setting because, when $\mathbf{X}_t$ is purely discrete, computing these probabilities can leverage the advances in exact inference from both neurosymbolic AI (Kisa et al. 2014; Tsamoura et al. 2021) and probabilistic AI (Darwiche 2020; Holtzen et al. 2020). Note that, computing $p_{\boldsymbol{\varphi}}(\mathbf{X}_{t+1} \mid \mathbf{X}_t, \mathbf{Z}_{t+1})$ involves computing $p_{\boldsymbol{\varphi}}(\mathbf{Z}_{t+1} \mid \mathbf{X}_t)$. Therefore, we will omit the latter from the rest of the exposition.

By removing resampling and having access to the exact transition probabilities, we can exploit an up-until-now unexplored synergy with gradient estimation. State-of-the-art unbiased discrete gradient estimation algorithms (De Smet, Sansone, et al. 2023; Kool et al. 2019) use samples and the gradients of the probability of those samples to approximate the gradients of finite distributions. In other words, they need the exact probabilities of these distributions to function. Hence, since our RBPF computes $p_{\boldsymbol{\varphi}}(\mathbf{X}_{t+1} \mid \mathbf{X}_t, \mathbf{Z}_{t+1})$ exactly, gradient estimation can be used to recursively get approximate gradients for the distribution $p_{\boldsymbol{\varphi}}(\mathbf{X}_{t+1} \mid \mathbf{Z}_{0:t+1})$. For example, using the Log-Derivative trick (Williams 1992)

$$\begin{aligned} &\nabla_{\boldsymbol{\varphi}} p_{\boldsymbol{\varphi}}(\mathbf{X}_{t+1} \mid \mathbf{Z}_{0:t+1}) \\ &= \mathbb{E}_{\mathbf{X}_t \sim p_{\boldsymbol{\varphi}}(\mathbf{X}_t \mid \mathbf{Z}_{0:t})} \left[ \nabla_{\boldsymbol{\varphi}} p_{\boldsymbol{\varphi}}(\mathbf{X}_{t+1} \mid \mathbf{X}_t, \mathbf{Z}_{t+1}) \right] \\ &+ \mathbb{E}_{\mathbf{X}_t \sim p_{\boldsymbol{\varphi}}(\mathbf{X}_t \mid \mathbf{Z}_{0:t})} \left[ p_{\boldsymbol{\varphi}}(\mathbf{X}_{t+1} \mid \mathbf{X}_t, \mathbf{Z}_{t+1}) \nabla_{\boldsymbol{\varphi}} \log p_{\boldsymbol{\varphi}}(\mathbf{X}_t \mid \mathbf{Z}_{0:t}) \right]. \end{aligned} \tag{4.6}$$

In our implementation, we opted for the state-of-the-art performance of RLOO (Kool et al. 2019) for gradient estimation.

### 4.3.3 Gradient Estimation for NeSy-MMs

For brevity and ease of exposition, we assume without loss of generality that each $\mathbf{X}_t$ is finite in nature. Our exposition will be fully general by considering the problem of computing

$$\nabla_{\boldsymbol{\varphi}} \mathbb{E}_{\mathbf{X}_{0:T} \sim p_{\boldsymbol{\varphi}}(\mathbf{X}_{0:T} | \mathbf{Z}_{0:T})} \left[ f(\mathbf{X}_{0:T}) \right]. \tag{4.7}$$

In particular, all cases described in Section 4.2 are covered by the expectation in this equation. Consider for example the discriminative task of computing $p_{\boldsymbol{\varphi}}(y \mid \mathbf{n}_{0:T}, \mathbf{Z}_{0:T})$ where the target random variable $Y$ depends on the symbols $\mathbf{S}_t$ for every $0 \leq t \leq T$. By definition of a probability, we can write

$$p_{\boldsymbol{\varphi}}(y \mid \mathbf{n}_{0:T}, \mathbf{Z}_{0:T}) = \mathbb{E}_{\mathbf{S}_{0:T} \sim p_{\boldsymbol{\varphi}}(\mathbf{S}_{0:T} | \mathbf{n}_{0:T}, \mathbf{Z}_{0:T})} \left[ \mathbb{1}_{\mathbf{S}_{0:T} \models y} \right], \tag{4.8}$$

and this expression is indeed an expectation like the one in Equation 4.7, modulo given values for the neural variables $\mathbf{N}_{0:T}$.

To provide an unbiased approximation of the gradient in Equation 4.7, we start by applying the Log-Derivative Trick (Williams 1992), resulting in the sum of the two terms

$$\mathbb{E}_{\mathbf{X}_{0:T} \sim p_{\boldsymbol{\varphi}}(\mathbf{X}_{0:T} | \mathbf{Z}_{0:T})} \left[ \nabla_{\boldsymbol{\varphi}} f(\mathbf{X}_{0:T}) \right], \tag{4.9}$$

and

$$\mathbb{E}_{\mathbf{X}_{0:T} \sim p_{\boldsymbol{\varphi}}(\mathbf{X}_{0:T} | \mathbf{Z}_{0:T})} \left[ f(\mathbf{X}_{0:T}) \nabla_{\boldsymbol{\varphi}} \log p_{\boldsymbol{\varphi}}(\mathbf{X}_{0:T} \mid \mathbf{Z}_{0:T}) \right]. \tag{4.10}$$

In all that follows, we will ignore the first term (Equation 4.9) as $\nabla_{\boldsymbol{\varphi}} f(\mathbf{x}_{0:T})$ can easily be computed for any instance $\mathbf{x}_{0:T}$ using automatic differentiation algorithms. The second term is where many of the problems in gradient estimation arise. While the infamously high variance of a direct Monte Carlo estimate of Equation 4.10 is problematic, we additionally have to deal with the fact that the gradient $\nabla_{\boldsymbol{\varphi}} \log p_{\boldsymbol{\varphi}}(\mathbf{X}_{0:T} \mid \mathbf{Z}_{0:T})$ could prove problematic. To minimise the former problem of variance, we use the unbiased RLOO estimator (Kool et al. 2019) of Equation 4.10.

$$\frac{1}{N-1} \sum_{i=1}^{N} \left( f(\mathbf{x}_{0:T}^{(i)}) - \overline{f} \right) \nabla_{\boldsymbol{\varphi}} \log p_{\boldsymbol{\varphi}}(\mathbf{x}_{0:T}^{(i)} \mid \mathbf{Z}_{0:T}), \tag{4.11}$$

where $\overline{f} = \frac{1}{N}\sum_{j=1}^{N} f(\mathbf{x}_{0:T}^{(j)})$ is an empirical estimate of the average of $f$ with respect to $p_{\boldsymbol{\varphi}}(\mathbf{X}_{0:T} \mid \mathbf{Z}_{0:T})$.

Next, we can start decomposing $\nabla_{\boldsymbol{\varphi}} \log p_{\boldsymbol{\varphi}}(\mathbf{x}_{0:T}^{(i)} \mid \mathbf{Z}_{0:T})$ and show that we are able to compute it exactly. Notice how the joint distribution follows a recursive factorisation similar to the recursive integration of the RBPF (Equation 4.5)

$$p_{\boldsymbol{\varphi}}(\mathbf{x}_{0:T}^{(i)} \mid \mathbf{Z}_{0:T}) = p_{\boldsymbol{\varphi}}(\mathbf{x}_{T}^{(i)} \mid \mathbf{x}_{T-1}^{(i)}, \mathbf{Z}_{t+1}) p_{\boldsymbol{\varphi}}(\mathbf{x}_{0:T-1}^{(i)} \mid \mathbf{Z}_{0:T-1}). \tag{4.12}$$

Because we assume that $p_{\boldsymbol{\varphi}}(\mathbf{X}_T \mid \mathbf{x}_{T-1}^{(i)}, \mathbf{Z}_{t+1})$ can be computed exactly, as necessitated by the use of our RBPF, it follows that we can also exactly compute the gradient of $p_{\boldsymbol{\varphi}}(\mathbf{x}_{T}^{(i)} \mid \mathbf{x}_{T-1}^{(i)}, \mathbf{Z}_{t+1})$. Consequently, we can simply apply automatic differentiation on

$$\log p_{\boldsymbol{\varphi}}(\mathbf{x}_{0:T}^{(i)} \mid \mathbf{Z}_{0:T}) = \sum_{t=0}^{T} \log p_{\boldsymbol{\varphi}}(\mathbf{x}_{t}^{(i)} \mid \mathbf{x}_{t-1}^{(i)}, \mathbf{Z}_{t+1}), \tag{4.13}$$

to compute the gradient $\nabla_{\boldsymbol{\varphi}} \log p_{\boldsymbol{\varphi}}(\mathbf{x}_{0:T}^{(i)} \mid \mathbf{Z}_{0:T})$. Hence, as alluded to in the previous section, we can now clearly see how the assumption of our RBPF makes it possible to apply gradient estimation following Equation 4.11.

To end this section, we further elaborate on the example given in Equation 4.6 as is an interesting special case of Equation 4.11. Indeed, gradients of the marginal $p_{\boldsymbol{\varphi}}(\mathbf{X}_t \mid \mathbf{Z}_{0:t})$ often appear when the target query variable $Y$ only depends on a single time step. For example, assume we are interested in knowing what the probability is that the game from Example 2.1.3 is over *now* given that we have observed a series of hits in the *past*. In this case, we would have an expectation of the form

$$\mathbb{E}_{\mathbf{X}_T \sim p_{\boldsymbol{\varphi}}(\mathbf{X}_T \mid \mathbf{Z}_{0:T})} \left[ f(\mathbf{X}_T) \right], \tag{4.14}$$

leading to an application of RLOO that requires $\nabla_{\boldsymbol{\varphi}} p_{\boldsymbol{\varphi}}(\mathbf{x}_{T}^{(i)} \mid \mathbf{Z}_{0:T})$. Since we only have samples from the joint distribution $p_{\boldsymbol{\varphi}}(\mathbf{X}_{0:T} \mid \mathbf{Z}_{0:T})$, we approximate $\nabla_{\boldsymbol{\varphi}} p_{\boldsymbol{\varphi}}(\mathbf{x}_{T}^{(i)} \mid \mathbf{Z}_{0:T})$ via a Monte Carlo estimate of Equation 4.6. To again mitigate the high variance of the recursive term $\mathbb{E}_{\mathbf{X}_{T-1}} \Big[ p_{\boldsymbol{\varphi}}(\mathbf{x}_{T}^{(i)} \mid \mathbf{X}_{T-1}, \mathbf{Z}_T) \nabla_{\boldsymbol{\varphi}} \log p_{\boldsymbol{\varphi}}(\mathbf{X}_{T-1} \mid \mathbf{Z}_{0:T}) \Big]$ we apply RLOO *recursively* through the expression

$$\frac{1}{N-1} \sum_{j=1}^{N} \Big( p_{\boldsymbol{\varphi}}(\mathbf{x}_{T}^{(i)} \mid \mathbf{x}_{T-1}^{(j)}, \mathbf{Z}_T) - \overline{p}_{\boldsymbol{\varphi}}(\mathbf{x}_{T}^{(i)} \mid \mathbf{Z}_T) \Big) \nabla_{\boldsymbol{\varphi}} \log p_{\boldsymbol{\varphi}}(\mathbf{x}_{T-1}^{(j)} \mid \mathbf{Z}_{0:T-1}), \tag{4.15}$$

where now

$$\overline{p}_{\boldsymbol{\varphi}}(\mathbf{x}_T^{(i)} \mid \mathbf{Z}_T) = \frac{1}{N}\sum_{k=1}^{N} p_{\boldsymbol{\varphi}}(\mathbf{x}_T^{(i)} \mid \mathbf{x}_{T-1}^{(k)}, \mathbf{Z}_T), \tag{4.16}$$

with $x_{T-1}^{(k)} \sim p_{\boldsymbol{\varphi}}(\mathbf{X}_{T-1} \mid \mathbf{Z}_{0:T-1})$.

### 4.3.4 NeSy Inference via Cluster Factorisation

Unfortunately, computing $p_{\boldsymbol{\varphi}}(\mathbf{X}_{t+1} \mid \mathbf{x}_t, \mathbf{Z}_{t+1})$ exactly when $\mathbf{X}_{t+1}$ also contains variables with an infinite domain is generally only possible under strict assumptions such as Gaussian densities. Moreover, it can still become prohibitively expensive in the purely finite case when ignoring the internal dependency structure of $\mathbf{X}_{t+1}$. We mitigate these problems by factorising the NeSy-MM further into clusters of variables that become independent when conditioning on $\mathbf{Z}$. Specifically, a conditional probability distribution $p(\mathbf{X} \mid \mathbf{Z})$ can be factorised as

$$p(\mathbf{X} \mid \mathbf{Z}) = \prod_{i=1}^{B} p(\mathbf{X}^i \mid \mathbf{Z}), \tag{4.17}$$

where $B$ is the maximal number of clusters. Intuitively, variables within the same cluster must always be sampled together and hence comprise minimal subproblems to be solved. The distribution $p(\mathbf{X}^i \mid \mathbf{Z})$ of each of the subproblems can be computed separately to alleviate the computational bottleneck of computing $p(\mathbf{X} \mid \mathbf{Z})$ exactly.

Applying the cluster factorisation to the conditional probability distribution $p_{\boldsymbol{\varphi}}(\mathbf{X}_{t+1} \mid \mathbf{x}_t, \mathbf{Z}_{t+1})$ with clusters $\{\mathbf{X}_{t+1}^i\}_{i=1}^B$ yields

$$p_{\boldsymbol{\varphi}}(\mathbf{X}_{t+1} \mid \mathbf{x}_t, \mathbf{Z}_{t+1}) = \prod_{i=1}^{B} p_{\boldsymbol{\varphi}}(\mathbf{X}_{t+1}^i \mid \mathbf{x}_t, \mathbf{Z}_{t+1}). \tag{4.18}$$

If we split every cluster $\mathbf{X}_t^i$ into a finite part $\mathbf{F}_t^i$ and infinite part $\mathbf{I}_t^i$, this factorisation can be further refined into

$$\prod_{i=1}^{B} p_{\boldsymbol{\varphi}}(\mathbf{F}_{t+1}^i \mid \mathbf{I}_{t+1}^i \mathbf{x}_t, \mathbf{Z}_{t+1}) p_{\boldsymbol{\varphi}}(\mathbf{I}_{t+1}^i \mid \mathbf{x}_t, \mathbf{Z}_{t+1}). \tag{4.19}$$

By first obtaining samples for the infinite random variables $\mathbf{I}_t^i$ in every $i^{\text{th}}$ cluster using a traditional particle filter, we are again left with a purely finite

probability distribution $p_{\boldsymbol{\varphi}}(\mathbf{F}^i_{t+1} \mid \mathbf{I}^i_{t+1}\mathbf{x}_t, \mathbf{Z}_{t+1})$ that we compute exactly such that discrete gradient estimation can be applied.

For infinite random variables, where our RBPF-based method cannot be applied, we can recover differentiability using any of the proven and tailored gradient estimation algorithms (Corenflos et al. 2021; Ścibior et al. 2021; Younis and E. B. Sudderth 2023). In our case, we followed the work of Ścibior et al. (2021) as it provides strong baseline performance. In summary, we recover gradient-based optimisation of infinite, finite and binary logical variables by joining local exact inference with specialised gradient estimation. *The result is a novel Rao-Blackwellised particle filter for NeSy-MMs that handles hybrid domains and exploits the inner conditional dependency structure of the NeSy states* $\mathbf{X}_t$.

### 4.3.5 Computational Complexity

We now provide further insights into the computational complexity of the inference process in our framework, focusing on how RBPF and sampling are employed.

The first and most direct use of RBPF corresponds to the application of Equation 2.10, where RBPF is used to marginalise out variables with infinite domains. This results in a purely discrete posterior distribution $p(\mathbf{X}_{t+1} \mid \mathbf{x}_t^{(i)}, \mathbf{Z}_{t+1})$ that can be computed exactly.

Given a set of samples $\{\mathbf{x}_t^{(i)}\}_{i=1}^N$, we can apply a non-standard use of RBPF to exactly compute this posterior distribution, conditioned on the evidence and current samples $\mathbf{x}_t^{(i)}$, and then draw new samples from it

$$\mathbf{x}_{t+1}^{(i)} \sim p(\mathbf{X}_{t+1} \mid \mathbf{x}_t^{(i)}, \mathbf{Z}_{t+1}).$$

This procedure is repeated at every time step to propagate the particle set forward, while the exact probabilities computed in this step are used to recover differentiability via gradient estimation, as explained in Section 4.3.3.

We now analyse how sampling affects the computational complexity of this process. Notice that we can decompose

$$p(\mathbf{X}_{t+1} \mid \mathbf{x}_t^{(i)}, \mathbf{Z}_{t+1}) \propto p(\mathbf{X}_{t+1} \mid \mathbf{x}_t^{(i)}) \cdot p(\mathbf{Z}_{t+1} \mid \mathbf{X}_{t+1}). \tag{4.20}$$

The sampling then affects only the transition term $p(\mathbf{X}_{t+1} \mid \mathbf{x}_t^{(i)})$, while the observation model $p(\mathbf{Z}_{t+1} \mid \mathbf{X}_{t+1})$ remains unchanged.

Assuming each state variable $X_{t,j}$ has domain $|X_{t,j}|$ and there are $n$ such variables, the total number of possible discrete instantiations is $|X_{t,j}|^n = |\mathbf{X}_t|$.

Without sampling, evaluating the transition model exhaustively leads to a complexity of $\mathcal{O}(|\mathbf{X}_t|^2)$ per step, due to the need to compute a full joint transition matrix.

However, using samples reduces this cost. In fact, the transition requires computing a distribution over $\mathbf{X}_{t+1}$, for $N$ samples, which costs $\mathcal{O}(N \cdot |\mathbf{X}_t|)$. Being $N$ a constant, this equals to a total complexity of $\mathcal{O}(|\mathbf{X}_t|)$, leading to a quadratic improvement over the exact (non-sampled) case.

Moreover, the cluster factorisation introduced in Equation 4.17 allows further improvements. If the transition function factorises over $B$ independent clusters $\mathbf{X}_t^i$, then the complexity reduces to

$$\mathcal{O}\left(\sum_{i=1}^{B} |\mathbf{X}_t^i|\right),$$

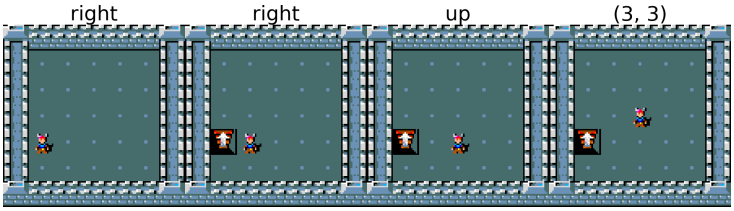
where $\sum_{i=1}^{B} |\mathbf{X}_t^i|$ is always $\leq |\mathbf{X}_t|$, and the improvement depends on the sparsity and structure of the model.

## 4.4 Experiments

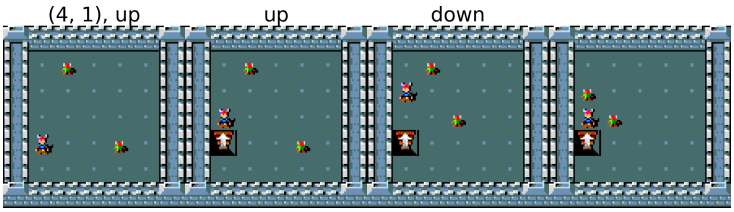
We present here our generative and discriminative benchmarks, and show that NeSy-MMs are capable of tackling both settings **(D.IV)**. We also clearly show how the presence of relational logic in NeSy-MMs significantly and positively impacts both in- and out-of-distribution performance compared to state-of-the-art deep (probabilistic) models **(D.I)**. In doing so, we show that NeSy-MMs scale to problem settings far beyond the horizon of existing NeSy methods **(D.II)**. In total, NeSy-MMs are successful neurosymbolic models capable of optimising various neural components while adhering to logical constraints **(D.III)**.

### 4.4.1 Generative

Our generative experiment is inspired by the Mario experiment of Misino et al. (2022), extended using MiniHack (Samvelyan et al. 2021), a flexible framework to define environments of the open-ended game NetHack (Küttler et al. 2020). The dataset consists of trajectories of images of length $T$ representing an agent moving $T$ steps in a grid world of size $N \times N$ surrounded by walls. The starting position of the agent is randomly initialised and the actions the agent takes are uniformly sampled among the four cardinal directions, *i.e.* up, down, left, right (Figure 4.4a). The actions the agent took at every time step are also given in




(a) Generative dataset




(b) Discriminative dataset

Figure 4.4: Example trajectories of length 4 in a 5×5 grid for the generative (a) and discriminative (b) datasets, with the corresponding labels above the images. In the generative setting, each trajectory contains the images, the actions taken (*e.g.* right, right, up), and the final location of the agent, *e.g.* (3, 3). For the discriminative task, the images are provided here for visualization purposes only. In fact, the models do not take images as input but rather the symbolic state, that is, the agent's initial location, and the executed actions.

the trajectory. During training, the model takes sequences of both MiniHack images and actions and learns to reconstruct the given images. At test time, the model should then be able to generate sequences of MiniHack images that follow a given sequence of actions and satisfy the rules of NetHack.

We use VAEL (Misino et al. 2022) as a neurosymbolic baseline and two other fully neural baselines: a variational transformer architecture (VT); and a NeSy-MM without logical rules and with neural networks as transition function (Deep-HMM). Since Deep-HMMs are subsumed by NeSy-MMs, the baseline was implemented in our framework to allow it to benefit from our Rao-Blackwellised inference and learning strategy.

We consider two metrics for the evaluation. First, the reconstruction error (RE), measured by the mean absolute difference in pixel values, which is first averaged over the images, then separately averaged over all images of the sequence. Second, the reconstruction accuracy (RA), which uses a pre-trained classifier for the location of the agent and measures how much the reconstructed trajectory

aligns with the ground truth. This is crucial to understand whether the agent is moving according to the actual rules of the game.

### 4.4.2 Discriminative

The next setting consists of a discriminative task, where the goal is to classify trajectories of symbolic states. The main challenge is that the transition function is now partially unknown and needs to be learned from examples. That is, we do not use neural networks for perception as is usually done in NeSy, but have a transition that is both neural and logical. More concretely, the dataset for the discriminative task consists of trajectories similar to the generative dataset. However, there are now also enemies present that are trying to kill the agent (Figure 4.4b). The input to the model in this case is fully symbolic, meaning we do not input images, but rather the precise starting coordinate of the agent and the list of actions performed by the agent. On top of that, we observe if one of the enemies hits the agent, *i.e.* the observations $Z_t$ are binary random variables. The discriminative task is binary classification, where a trajectory has label 1 if the agent dies somewhere in the trajectory and 0 otherwise. While we know the basic rules of NetHack, such as permitted movements and damage mechanics, we do not know the transition function of the enemy. That is, we don't know the behaviour of the enemies and fill this gap in knowledge with a neural network that should respect the known rules of NetHack. We assume that all the enemies share the same behaviour.

Similar to the generative experiment, we use a transformer and a Deep-HMM as baselines, this time in a discriminative configuration. To specifically gauge the out-of-distribution (OOD) generalisation capabilities of all methods, we train only using simple sequences of length 10 containing just one enemy moving on a 10×10 grid and we test on more complex sequences. The OOD cases consider different combinations of sequences on grids of size 10×10 or 15×15, length 10 or 20, and with 1 or 2 enemies. When more enemies are present or the sequences are longer, it is naturally easier for the agent to be killed. Conversely, the enemies might need more steps to reach the agent when the grid is bigger. These differences lead to a difference in class balance from one configuration to the other (Table 4.1), posing an additional significant learning challenge. Ideally, we want a model that is able to counterbalance the bias inherent to its training data. To test such abilities, we evaluate all methods in terms of both balanced accuracy and F1-score.

| $N$ | $T$ | $E$ | **Death (%)** | **OOD** |
|---|---|---|---|---|
| 10 | 10 | 1 | 17.2 | – |
| | | 2 | 60.9 | ✓ |
| | 20 | 1 | 89.0 | ✓ |
| | | 2 | 99.0 | ✓ |
| 15 | 10 | 1 | 9.9 | ✓ |
| | | 2 | 32.1 | ✓ |
| | 20 | 1 | 75.1 | ✓ |
| | | 2 | 96.7 | ✓ |

Table 4.1: Percentage of trajectories leading to the death of the agent for the discriminative experiment, based on grid size ($N$), trajectory length ($T$) and number of enemies ($E$).

| | | **Methods** | | |
|---|---|---|---|---|
| **Metric** | $N$ | VT | Deep-HMM | NeSy-MM |
| RE (↓) | 5 | 3.30 ± 0.04 | 4.97 ± 0.37 | **3.32 ± 1.80** |
| | 10 | **2.23 ± 0.01** | 4.66 ± 0.08 | 3.78 ± 0.07 |
| RA (↑) | 5 | 91.39 ± 0.54 | 44.55 ± 5.75 | **97.17 ± 1.00** |
| | 10 | 30.62 ± 1.24 | 1.54 ± 0.00 | **89.63 ± 3.59** |

Table 4.2: Reconstruction error (RE) and accuracy (RA) for the generative experiment on different grid sizes ($N \times N$). RE is multiplied by 1000, and RA is in percentage.

### 4.4.3 Results

Results for the generative experiment are reported in Table 4.2 while the results for the discriminative task can be found in Table 4.3 and Table 4.4. We report the mean and standard error for all the metrics. We discuss the main findings of both experiments by highlighting the advantages of NeSy-MMs.

**Better generative consistency.** Integrating knowledge about the environment is clearly advantageous to the generation process in terms of logical consistency, as can be seen from the reconstruction accuracies. NeSy-MMs significantly outperform both baselines in this regard, especially on larger grids. In terms of reconstruction error, the variational transformer performs on-par or even better than NeSy-MMs. While this shows how transformers are very capable of optimising their losses, the sub-par reconstruction accuracy questions the degree of transfer from optimised solutions to desired solutions. The Deep-

Figure 4.5: Generated trajectory for actions: right, down, left, up, left, up, right, down.

Figure 4.6: Generated trajectory for actions: *right*, down, left, up, right, *right*, up, *right*; but with the test-time constraint that the area to the right of the start position should not be entered. When the agent is asked to move in the unsafe area (*i.e.* actions in italics) it, instead, stays in the safe zone, and then it continues following the rest of the instructions.

HMM generally underperforms compared to both the VT and the NeSy-MM, illustrating the challenge of tackling sequential tasks without logic (NeSy-MM) or a longer time dependency (VT).

**Logical interpretability and intervenability.** One of the biggest advantages of neurosymbolic generation is its ability to induce interpretable and intervenable logical consistency into subsymbolic generation. As an example, consider the generation in Figure 4.5 where the generative model was asked to generate a trajectory for the agent, following a given sequence of actions, while adhering to the movement rules of the game. Because the symbolic rules of the game are an inherent part of the generative model, NeSy-MMs generate sequences that perfectly adhere to the mechanics of the game and the provided actions. Other methods lack the necessary semantics or symbolic knowledge to fully guarantee this sort of logical consistency. Moreover, NeSy-MMs allow imposing constraints at test time in addition to the ones used during learning, which corresponds to zero-shot adherence to new queries (Figure 4.6).

**Scaling NeSy to non-trivial time horizons.** NeSy methods are known for their scalability issues. When sequential generative settings are considered, the situation is even more dramatic. VAEL fails to perform inference on a single sequence of length 10 even on a smaller grid of size 3×3, with $6h$ timeout. On the contrary, we manage to perform inference and generative learning that does not deteriorate over time, even compared to the neural baselines. In fact, NeSys

| | | | Methods | | |
|---|---|---|---|---|---|
| $N$ | $T$ | $E$ | Transformer | Deep-HMM | NeSy |
| 10 | 10 | 1 | **75.72 ± 1.35** | 59.88 ± 0.28 | 64.45 ± 1.10 |
| | | 2 | 68.61 ± 0.78 | 38.43 ± 0.21 | **78.13 ± 0.67** |
| | 20 | 1 | 50.40 ± 0.19 | 49.99 ± 0.39 | **67.66 ± 0.92** |
| | | 2 | 16.09 ± 1.33 | 49.33 ± 0.17 | **57.47 ± 0.56** |
| 15 | 10 | 1 | **78.53 ± 3.09** | — | 57.22 ± 1.78 |
| | | 2 | 67.85 ± 1.79 | — | **75.57 ± 0.42** |
| | 20 | 1 | 50.47 ± 0.18 | — | **71.85 ± 0.58** |
| | | 2 | 41.54 ± 2.39 | — | **77.13 ± 2.14** |

Table 4.3: Balanced accuracy (%) for the discriminative experiment for grid sizes $N \times N$, with trajectory length $T$ and $E$ enemies. The first line is in-distribution performance, the rest is OOD. Deep-HMM cannot be applied to bigger grids.

| | | | Methods | | |
|---|---|---|---|---|---|
| $N$ | $T$ | $E$ | Transformer | Deep-HMM | NeSy |
| 10 | 10 | 1 | **0.61 ± 0.02** | 0.25 ± 0.01 | 0.41 ± 0.03 |
| | | 2 | 0.59 ± 0.02 | 0.56 ± 0.00 | **0.79 ± 0.01** |
| | 20 | 1 | 0.02 ± 0.01 | 0.79 ± 0.01 | **0.92 ± 0.01** |
| | | 2 | 0.02 ± 0.01 | 0.98 ± 0.00 | **0.99 ± 0.00** |
| 15 | 10 | 1 | **0.57 ± 0.03** | — | 0.15 ± 0.02 |
| | | 2 | 0.52 ± 0.03 | — | **0.65 ± 0.01** |
| | 20 | 1 | 0.02 ± 0.01 | — | **0.78 ± 0.02** |
| | | 2 | 0.02 ± 0.01 | — | **0.98 ± 0.01** |

Table 4.4: F1-Score for the discriminative experiment. This follows the same notation as Table 4.3.

perform a forward and backward pass over a batch of sequences in $\approx 0.25s$, for both grid sizes. In the discriminative setting, the presence of a neural network as transition prevented us from applying any existing NeSy system, as they do not provide effective strategies to integrate neural networks except as perception.

**Better out-of-distribution generalisation.** Pivoting the attention to the discriminative experiment, we can see that NeSys exploit their relational expressivity to perform well in all the out-of-distribution settings. Both the

accuracy (Table 4.3) and F1-score (Table 4.4) paint a similar picture: the transformer is able to achieve better performance when staying in distribution ($N = 10$, $H = 10$ and $E = 1$). However, the out-of-distribution settings deteriorate the transformer's performance. Only when the balance between positive and negative classes is closest to the balance of the training data, *i.e.* when only $N$ is increased to 15 (Table 4.1), the transformer is able to keep a good accuracy. In contrast, NeSys show that their relational representations are much more robust to distribution shifts. Deep-HMMs land somewhere in the middle between transformers and NeSys as their performance is always lower than NeSys, but depending on the case they can be more robust than the transformers. Finally, notice that Deep-HMMs cannot be applied to larger grid sizes, limiting their OOD capabilities.

## 4.5 Conclusion

We introduced relational neurosymbolic Markov models (NeSy-MMs), a powerful new class of relational probabilistic models that can incorporate neural networks beyond just perception modules. These models are provided with a novel scalable and differentiable particle filtering technique for inference and learning, facilitating the bidirectional flow of information necessary for a proper neurosymbolic model **(D.III)**. Our empirical results show that the integration of relational symbolic knowledge into deep Markov models leads to significant improvements in generative and discriminative tasks **(D.IV)**, while also providing guarantees that neural models alone cannot achieve. Importantly, we stressed the relational aspect of NeSy-MMs by showing that purely neural models and even deep probabilistic models struggle to learn representations that generalise to unseen data and settings **(D.I)**. While such generalisation behaviour is inherent to many neurosymbolic approaches, our experiments showed that NeSy-MMs scale to sequential settings beyond the reach of existing NeSy systems **(D.II)**.

Future work will focus on further applying NeSy-MMs to new settings, *e.g.* reinforcement learning, and applications where continuous random variables are used differently from image generation, *e.g.* physical systems.

# Chapter 5

# Neurosymbolic Reinforcement Learning

This chapter is based primarily on the following peer-reviewed conference publication.

> D. Debot*, G. Venturato*, G. Marra, and L. De Raedt (2025). "Neurosymbolic Reinforcement Learning: Playing MiniHack With Probabilistic Logic Shields". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 39. 28, pp. 29631–29633

As the publication is a demo paper, it includes only the core contributions developed by D. Debot and me, under the active supervision of G. Marra and L. De Raedt. Both provided valuable guidance and contributed to shaping the design and positioning of the demo. This chapter has therefore been extended with additional material based on the earlier paper by W.-C. Yang et al. (2023). Specifically, Section 5.2 provides background on probabilistic logic shields; Section 5.2.4 is new to this thesis and discusses the $\alpha$ hyperparameter; Section 5.5 presents the results featured on the accompanying website, which is part of the demo contribution. Finally, Section 5.6 concludes the chapter with material from the extended abstract

> L. De Smet*, G. Venturato*, G. Marra, and L. De Raedt (2024). *Neurosymbolic Reinforcement Learning With Sequential Guarantees*.

Extended Abstract. Joint International Scientific Conferences on AI and Machine Learning (BNAIC/BeNeLearn 2024)

and current work in progress that is not yet published.

In this chapter, we focus on the full intelligent agent presented in Figure 1.1, using however a reasoning module that does not yet support sequential reasoning. Specifically, we address research questions **RQ3** and **RQ5**, exploring how neurosymbolic methods can be applied in reinforcement learning (RL) to design agents that adhere to safety specifications while learning to act in their environment.

As a result, we obtain agents that can perceive the world **(i)** via pre-trained sensors, while single-step reasoning and interacting **(iii)** with it, and that can generalise across objects or entities **(v)**. In Section 5.6, we outline future directions toward incorporating sequential reasoning and completing the remaining capabilities **(i)** - **(v)** required for a fully intelligent agent.

## 5.1 Introduction

In recent years, deep reinforcement learning (RL) has achieved remarkable successes across a variety of domains, from mastering complex games (Schrittwieser et al. 2020) to enabling robotic control (Schulman, Levine, et al. 2015) and autonomous driving (Kiran et al. 2021). While deep RL excels in finding effective policies through large-scale exploration, applying it to safety-critical environments remains a challenge. In such settings, agents must not only learn to maximise long-term rewards but also ensure compliance with given constraints, such as avoiding hazardous states or ensuring regulatory adherence.

Reward shaping (Ng et al. 1999) is a way of tackling this problem by changing the reward function to guide the behaviour of the agent. However, it does not provide guarantees, and it is often difficult to engineer the right reward function in practice. In contrast, probabilistic logic shields (W.-C. Yang et al. 2023) have emerged as a promising approach, because they rely on logical reasoning to re-normalise the policy function and prevent unsafe actions.

Although the concept of combining symbolic reasoning with deep learning is not new (Marra et al. 2024), practical implementations of safe RL through probabilistic logic shields remain limited. Most existing systems require highly specialised knowledge, which creates barriers for wider adoption. Our work

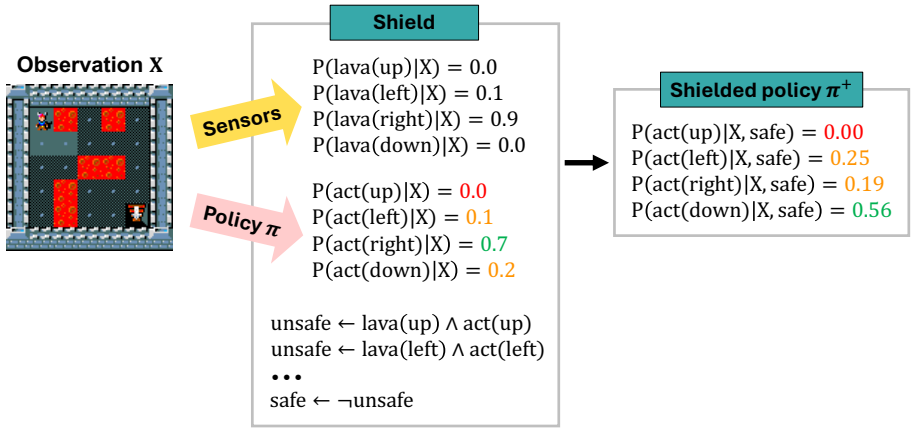



Figure 5.1: Example of Probabilistic Logic Shielding. The observation of the environment is used to extract symbolic information, which is then combined with the base policy to produce a shielded policy that avoids unsafe actions. The shielded policy is computed by renormalising the base policy conditioned on the safety model, ensuring that the agent acts safely while remaining close to its original behaviour. In this case the base policy would point the agent to the right, leading to death, but thanks to the rules and the sensors information, the shield renormalises the action distribution to point down instead.

builds on W.-C. Yang et al. (2023), and aims to lower these barriers by providing a flexible and user-friendly codebase. We extend W.-C. Yang et al. (ibid.) by integrating it into the popular MiniHack (Samvelyan et al. 2021) environment, a framework derived from the NetHack Learning Environment (Küttler et al. 2020), which offers rich, procedurally generated challenges ideal for testing complex RL agents.

By integrating probabilistic logic shields into MiniHack, our framework not only supports the development of safe and interpretable RL agents but also democratises the deployment of neurosymbolic RL in real-world and research settings. We believe this toolset will play a crucial role in advancing safe RL practices and fostering broader adoption of logic-augmented neural agents in safety-critical domains.

## 5.2 Probabilistic Logic Shields

Probabilistic Logic Shields (PLS) provide a differentiable and symbolic interface to enforce safety in reinforcement learning (RL) policies. They combine a probabilistic abstraction of the environment, a logical safety specification, and a stochastic base policy to produce a shielded policy that avoids unsafe decisions while remaining close to the original behaviour.

Figure 5.1 illustrates how the shielding technique works. As in standard deep RL, an observation from the environment is passed to a neural network to produce an initial policy $\pi_\theta$. Additionally, sensors extract probabilistic symbolic information from the same observation, *e.g.* there is lava to the right of the agent with 0.9 probability. The shield is composed by a set of (probabilistic) logic rules that define what is safe and what is not. It uses the probabilities from $\pi_\theta$, together with the sensor values, to renormalise $\pi_\theta$ and thus produces a safe, *shielded policy* $\pi_\theta^+$. In the following sections, we will assume that the policy $\pi_\theta$ is a neural predicate parametrised by $\theta$, and we omit the subscript $\theta$ for notational convenience.

### 5.2.1 Probabilistic Shielding

Following W.-C. Yang et al. (2023), we assume the existence of a probabilistic safety model $p(\texttt{safe} \mid s, a)$ that estimates the likelihood that an action $a$ is safe in state $s$. This model is not derived from the full MDP but rather represents a domain-specific abstraction of safety-relevant dynamics.

Given a base policy $\pi(a \mid s)$, the overall safety of the policy in state $s$ is computed as

$$p_\pi(\texttt{safe} \mid s) = \sum_{a \in \mathcal{A}} p(\texttt{safe} \mid s, a)\pi(a \mid s), \tag{5.1}$$

that captures the expected safety of executing the policy in that state.

**Definition 5.2.1** (Shielded Policy)**.** Given a base policy $\pi(a \mid s)$ and a probabilistic safety model $p(\texttt{safe} \mid s, a)$, we define the shielded policy $\pi^+$ as

$$\pi^+(a \mid s) = p_\pi(a \mid s, \texttt{safe}) = \frac{p(\texttt{safe} \mid s, a)}{p_\pi(\texttt{safe} \mid s)}\pi(a \mid s) \tag{5.2}$$

The shielded policy is therefore defined as a Bayesian renormalization of the base policy $\pi$ conditioned on the safety. This ensures that actions are selected proportionally to their safety likelihood, while still respecting the original policy's distribution. This formulation guarantees that $\pi^+$ is safer than $\pi$ in expectation, meaning that for all $s$ and $\pi$, $p_{\pi^+}(\texttt{safe}|s) \geq p_\pi(\texttt{safe}|s)$ (ibid.).

### 5.2.2 Probabilistic Logic Shielding

We will focus on probabilistic shields implemented via probabilistic logic programs (PLPs). Specifically, we define a program $\mathcal{T}(s)$, constructed from three components:

- $\Pi_s$: a categorical distribution representing the policy $\pi(a \mid s)$;
- $H_s$: a set of probabilistic facts representing symbolic abstractions of state $s$, *e.g.* `lava(right)` with 0.9 probability;
- $\mathcal{BK}$: background knowledge encoding logical safety constraints, such as `unsafe :- lava(right), act(right)`

The full program $\mathcal{T}(s) = \Pi_s \cup H_s \cup \mathcal{BK}$ defines a probability distribution $p_{\mathcal{T}}$ over possible outcomes. The shielded policy and associated safety terms are computed via probabilistic inference:

$$p(\texttt{safe} \mid s, a) = p_{\mathcal{T}}(\texttt{safe} \mid a), \tag{5.3}$$

$$p_\pi(\texttt{safe} \mid s) = p_{\mathcal{T}}(\texttt{safe}), \tag{5.4}$$

$$\pi^+(a \mid s) = p_{\mathcal{T}}(a \mid \texttt{safe}). \tag{5.5}$$

By employing probabilistic logic, the shield helps the agent handle uncertainty about state and action safety, maximizing safety despite incomplete or noisy information. This flexibility accommodates probabilistic safety rules, noisy sensors, or both, making it suitable for real-world environments[1].

### 5.2.3 Probabilistic Logic Policy Gradient

The shielded policy $\pi^+$ adheres to the safety constraints while maintaining the flexibility of the neural network's original policy. Importantly, this shielding technique is end-to-end differentiable, allowing seamless integration with neural networks and RL algorithms, such as PPO (Schulman, Wolski, et al. 2017).

**Definition 5.2.2** (Probabilistic Logic Policy Gradient)**.** Given a shielded policy $\pi_\theta^+$ and a safety model $P_{\pi_\theta^+}(\texttt{safe} \mid s)$, the probabilistic logic policy gradient (PLPG) is defined as

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta^+}\left[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log \pi_\theta^+(a_t \mid s_t) + \alpha \nabla_\theta \log P_{\pi_\theta^+}(\texttt{safe} \mid s_t)\right], \tag{5.6}$$

[1] In this work, we always consider perfect sensors, as MiniHack facilitates this by providing symbolic observations.

where $\alpha$ is a hyperparameter that balances the trade-off between maximising expected return and ensuring safety.

This gradient combines two terms. The first is the policy gradient of the shielded policy, directly obtained from the standard policy gradient (Equation 2.13) by replacing $\pi$ with $\pi^+$. Notice that this cannot be achieved with non-differentiable shields, such as rejection-based shields (Garcıa and Fernández 2015; Hunt et al. 2021; Jansen et al. 2020). The second term is the *safety gradient*, which penalises unsafe actions by computing the gradient of the safety model with respect to the policy parameters. This term encourages the agent to avoid unsafe actions, effectively steering it towards safer behaviours. Practically, the safety gradient backpropagates the safety signal through the neural policy, effectively guiding the learning process to prioritise safety while still optimising for long-term rewards.

### 5.2.4 On The Role of $\alpha$ in PLPG

Recall that the shielded policy is defined as

$$\pi_\theta^+(a_t \mid s_t) = p_\pi(a_t \mid s_t, \texttt{safe}),$$

from Equation 5.2. That is, the original (potentially unsafe) policy conditioned on the safety constraints. This conditional formulation has important implications for learning. In fact, the optimization process assumes that safety is externally guaranteed and proceeds to maximise performance under this assumption. As a result, the policy search is confined to the subset of actions deemed safe by the shield, but the training dynamics do not actively enforce or preserve this constraint. In the absence of the external enforcement mechanism provided by the shield, the renormalisation process can unintentionally increase the probability mass of unsafe actions, simply because the policy must remain a valid probability distribution.

The safety gradient term introduced in the probabilistic logic policy gradient (PLPG) addresses this issue by counterbalancing the side effects of conditioning. Given the subtlety of this effect, we now turn to the mathematical details to clarify the role of $\alpha$ in the PLPG.

Let us consider the PLPG without the safety gradient term

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta^+}\left[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log \pi_\theta^+(a_t \mid s_t)\right]. \tag{5.7}$$

We can re-write $\nabla_\theta \log \pi^+_\theta(a_t \mid s_t)$ in terms of the base policy $\pi_\theta$ as

$$\nabla_\theta \log \frac{p(\texttt{safe} \mid s_t, a_t)}{p_\pi(\texttt{safe} \mid s_t)} \pi_\theta(a_t \mid s_t) \tag{5.8}$$

$$= \underbrace{\nabla_\theta \log p(\texttt{safe} \mid s_t, a_t)}_{=0} + \nabla_\theta \log \pi_\theta(a_t \mid s_t) - \nabla_\theta \log p_{\pi_\theta}(\texttt{safe} \mid s_t) \tag{5.9}$$

$$= \nabla_\theta \log \pi_\theta(a_t \mid s_t) - \nabla_\theta \log p_{\pi_\theta}(\texttt{safe} \mid s_t). \tag{5.10}$$

Here, Equation 5.8 follows from Definition 5.2.1; Equation 5.9 applies basic rules of logarithms; and in Equation 5.10, the gradient of the safety model $p(\texttt{safe} \mid s_t, a_t)$ vanishes since it is independent of the policy parameters $\theta$.

Then, substituting Equation 5.10 back into Equation 5.7, we obtain

$$\mathbb{E}_{\pi^+_\theta}\left[\sum_{t=0}^{\infty} \Psi_t \left(\nabla_\theta \log \pi_\theta(a_t \mid s_t) - \Psi_t \nabla_\theta \log p_{\pi_\theta}(\texttt{safe} \mid s_t)\right)\right]. \tag{5.11}$$

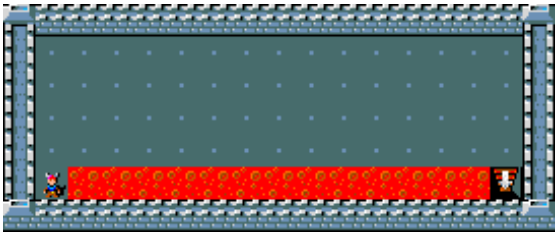
This decomposition confirms the earlier intuition: the second term, introduced by the renormalisation of the shielded policy, can act as a penalty on safety. This may reduce the probability assigned to safe actions, ultimately undermining the objective of learning a safe policy.

Finally, we can reintroduce the safety gradient term

$$\mathbb{E}_{\pi^+_\theta}\Big[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log \pi_\theta(a_t \mid s_t) - \Psi_t \nabla_\theta \log p_{\pi_\theta}(\texttt{safe} \mid s_t) + \alpha \nabla_\theta \log p_{\pi^+_\theta}(\texttt{safe} \mid s_t)\Big]. \tag{5.12}$$

Here, the safety gradient explicitly reinforces safe behaviour by counterbalancing the negative effect introduced by the renormalisation term. However, it is important to note that the renormalisation term is scaled by the expected return $\Psi_t$, while the safety gradient is scaled by the constant $\alpha$. If $\alpha$ is too small, the safety signal will be overwhelmed, and the agent may still learn unsafe behaviours. Conversely, if $\alpha$ is too large, the agent will over-prioritise safety at the expense of performance.

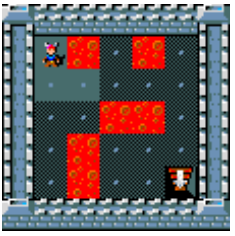
Thus, $\alpha$ must be carefully tuned to balance these competing objectives. The key insight is that $\alpha$ should be selected in proportion to the scale of the reward signal, ensuring that safety and performance are appropriately balanced throughout the learning process.



(a) Cliff

(b) Key Room

(c) Monster Room



(d) Lava Lake

Figure 5.2: Custom MiniHack environments. The agent's objective is to reach the goal (the staircase). Challenges include avoiding a lava cliff (Cliff), obtaining a key to unlock a door (Key Room), evading a chasing monster (Monster Room), and navigating a slippery floor with randomized lava (Lava Lake).

## 5.3 Neurosymbolic RL MiniHack Agents

To showcase this neurosymbolic integration, we created multiple custom MiniHack environments (see Figure 5.2), and we compare the shielded agent with an agent that uses reward shaping for encoding safety in these environments. Cliff (Figure 5.2a), firstly introduced in Sutton and Barto (2018) (Example 6.6), shows that shielding enables faster training convergence, as the shielding does not only prevent the agent from exploring dangerous states but also guides the learning process towards more productive areas of the state space. Key Room (Figure 5.2b), adapted from MiniHack environment zoo, is a randomised environment where the locations of every object (door, key, start, goal) change at each episode. In this environment, we show that perfect sensors can be used in probabilistic logic rules to provide a high-level policy that minimises the exhaustion of the agent by suggesting the preferred order, *i.e.* the key must be collected before opening the door. Monster Room (Figure 5.2c), is a dynamic and randomised environment where the start and goal locations are fixed, but monsters are spawned randomly, and they chase the agent. This demonstrates

how we can deal with moving objects, and that the shield helps the agent reaching the goal while avoiding the monsters. Lava Lake (Figure 5.2d), is inspired by Frozen Lake (Brockman et al. 2016). The agent has to reach the goal while avoiding holes with lava, considering that the floor is slippery. This setting shows that we can deal with stochastic transition functions and still maximise safety in these settings.

In order to define a shield, one has just to take care of describing the unsafe behaviour. For example, in Cliff, it can be easily defined by the rule

$$\forall a \quad unsafe \leftarrow lava(a) \wedge act(a),$$

where $a \in \{left, right, up, down\}$. Then, the probability of lava being in a certain direction is given by the sensor function, and the probability of taking an action $a$ is provided by the base policy $\pi$, as shown in Figure 5.1. The complete shields used in the demo for these environments can be found in Appendix B.1.

## 5.4 Implementation

Our implementation makes use of the *Stable Baselines3* library (Raffin et al. 2021), which implements state-of-the-art RL algorithms. We use the MiniHack framework to define Gym environments (Brockman et al. 2016), which are a standard in RL research. Shields are implemented in ProbLog (De Raedt, Kimmig, and Toivonen 2007). However, we exploit current advances in GPU-accelerated probabilistic inference, by using Klay (Maene et al. n.d.) as a backend for ProbLog. This allows us to perform efficient batched probabilistic inference on the GPU, which is crucial for real-time applications in RL.

**Extensibility and usability.** The structure of the codebase makes it easy to add new environments, change the RL algorithm or add new shields. For example, a new shield can be added by creating (1) a ProbLog file that defines the safety rules and (2) a function that maps an observation to sensor values used by the shield. Then, training an agent is as simple as running the code in Algorithm 6. We also provide a Jupyter notebook that helps users quickly get started with the core functionalities.

**Algorithm 6** Training a shielded reinforcement learning agent.

```
shield = Shield(problog_file, sensor_func)
agent = PPO(MlpPolicy, env, shield)
agent.learn(total_timesteps=10_000)
```

**Interactive website.** We have incorporated an interactive demo on our website that showcases the safety, training convergence, and generalization advantages of our approach. Users can load pretrained shielded and non-shielded agents (at multiple training checkpoints) in different environments to compare their behaviour, or manually control the agents in the environments. Additionally, the demo allows users to make small changes to the environment (*e.g.* add lava tiles) and test whether the pretrained agents still perform well. The users can also add a shield to an agent that was trained without one, or change the shield of a shielded agent.

## 5.5 Empirical Benefits of Shielding Agents

While this work was originally developed as a demonstration of the approach proposed by W.-C. Yang et al. (2023), it nonetheless offers valuable insights into the practical benefits of incorporating probabilistic logic shields. All agents are trained using the state-of-the-art PPO (Schulman, Wolski, et al. 2017) algorithm. We use the standard PPO implementation as a baseline that we will call "neural agent", while shielded agents are trained with the same algorithm, with the shielding mechanism integrated into the policy. Further implementation details are provided in Appendix B and in the open-source codebase.

**Faster convergence.** Table 5.1 shows the number of training steps required to reach the optimal policy in the four different MiniHack environments from Figure 5.2. In all cases, the shielded agent converges significantly faster than the baseline neural agent. This is particularly striking in environments like *Cliff* and *Lava Lake*, where safety constraints severely limit the effective state space: shielding helps the agent avoid unproductive or dangerous trajectories early in training.

**Increased safety.** Figure 5.4 reports the cumulative number of agent deaths during training. In both *Cliff* and *Monster Room*, the shielded agent avoids all unsafe actions from the start. Thanks to the perfect sensors, and the deterministic transition function, it is indeed possible for the agent to completely avoid death. In *Lava Lake*, which instead features stochastic transitions, the shield reduces the number of deaths substantially (one to two orders of magnitude) but cannot eliminate them entirely. This highlights the advantage of incorporating probabilistic logic: safety can still be maintained in the presence of uncertainty.

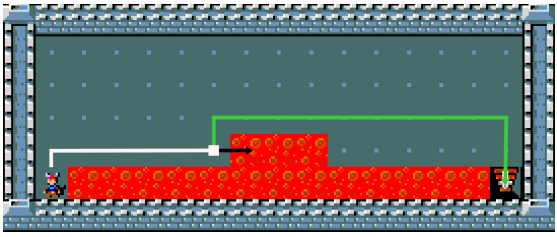


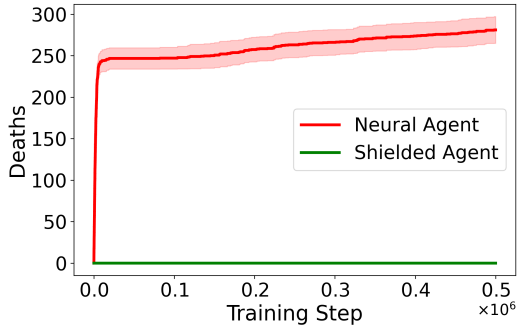
Figure 5.3: Example of test-time intervention on the Cliff environment (Figure 5.2a), where new lava tiles are added. At the beginning both agents follow the learned optimal policy (white path). However, when reaching the altered section (white box), the shielded agent (green path) adapts its behaviour to avoid unsafe actions, while the non-shielded agent (black path) continues to follow its original policy, leading to death.

Table 5.1: Training steps required to reach the optimal policy in different MiniHack environments. For Lava Lake the neural agent did not converge within the training budget of 3 million steps.

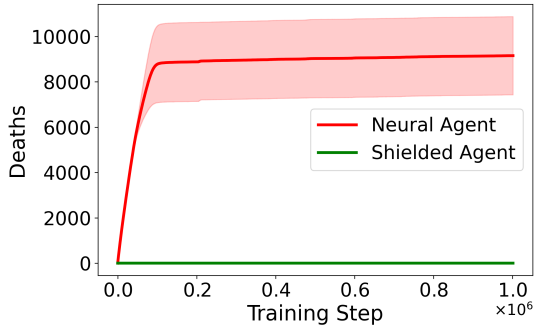
| **Environment** | **Neural Agent** | **Shielded Agent** |
|---|---|---|
| Cliff | 500 000 | 10 000 |
| Monster Room | 1 000 000 | 100 000 |
| Key Room | 600 000 | 200 000 |
| Lava Lake | >3 000 000 | 100 000 |

**Adaptability to out-of-distribution changes.** An important feature of our framework is its resilience to test-time modifications, also referred to as zero-shot learning. Thanks to its symbolic nature, the shielded agent can generalise to new configurations—such as additional monsters or lava tiles without retraining. This is illustrated in Figure 5.3, where the shielded pretrained policy continues to behave safely after the environment has been altered. On the demo website, users can interactively change the environments and test the agents' behaviour.

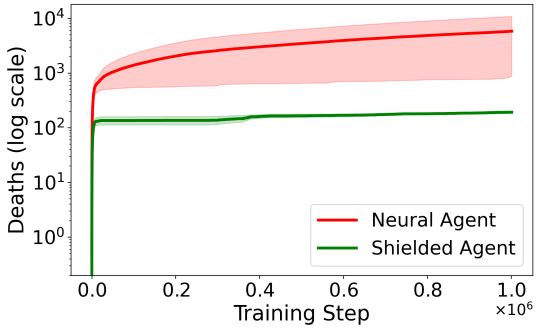
These results, although limited in scope, demonstrate the key advantages of probabilistic logic shielding: improved safety, faster training, and increased robustness to environmental variability.




(a) Cliff




(b) Monster Room




(c) Lava Lake

Figure 5.4: Number of cumulative deaths during training in different MiniHack environments. The shielded agent is able to avoid unsafe actions, while the neural agent dies frequently. The Key Room is not included in this figure, as it is impossible to die in that environment. In the Lava Lake environment, the shielded agent can still die, because of the stochastic nature of the environment.

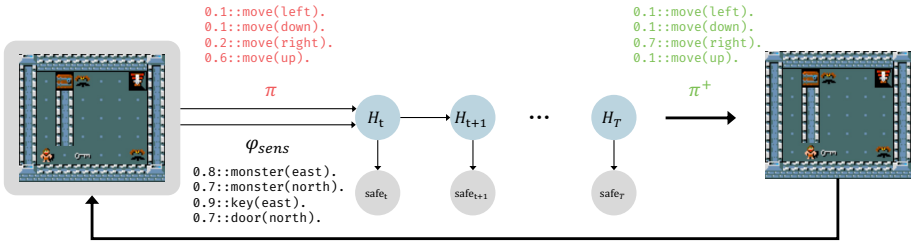



Figure 5.5: NeSy-MMs used as neurosymbolic policies that provide safety guarantees. As in a classic RL algorithms, ($\rightarrow$) executes an action in the environment, and ($\leftarrow$) provides a new observation to the policy. The agent (bottom left) has to reach the staircases (top right). Each NeSy state $H_i$ can contain raw data, or relational symbols. The transition from $H_i$ to $H_{i+1}$ can be fully logical, neural, or a mixture of both. Each state is also conditioned on a safety property, such that the agent is not killed by the monsters.

## 5.6 Conclusion and Future Work

This demonstration has illustrated the benefits of integrating deep reinforcement learning with probabilistic logic shields. Compared to standard neural agents trained via reward shaping, our shielded agents demonstrate improved safety, faster convergence, and greater robustness to test-time interventions. The symbolic nature of the shields allows for modular safety specifications and opens the door to greater interpretability and reusability.

While the current work focused on single-step safety guarantees within the MiniHack domain, ongoing research aims to extend this framework in two important directions: towards formal sequential guarantees, and towards real-world applications in robotics and autonomous driving.

### 5.6.1 Towards Agents with Sequential Guarantees

Chapter 4 introduced NeSy-MMs as a class of probabilistic relational models capable of integrating neural perception with symbolic sequential reasoning. In this section, we outline how NeSy-MMs can also be used to define reinforcement learning policies that offer safety guarantees over multiple time steps.

In this chapter we have shown how to enforce single-step safety using shielding techniques via probabilistic logic programs. While effective, this approach does not scale to multistep guarantees because of the #P-hardness of its inference

procedure. NeSy-MMs resolve this problem by using unbiased approximate inference techniques instead.

Consider again a MiniHack level where the agent must navigate past multiple threats to reach a goal (Figure 5.5). In such settings, safety cannot be assessed from a single state-action pair: the agent must consider the long-term consequences of its choices. For instance, the safest strategy may involve luring monsters away before proceeding, which requires multistep planning and reasoning about future safety.

Concretely, if the agent is governed by a policy $\pi$ and a sensor $\varphi_{sens}$ gives an estimate of the current state of the game, then these will form the input to a NeSy-MM. The NeSy-MM then updates the policy to $\pi^{+}(a \mid$  $) = \pi(a \mid \texttt{safe}_{t:T},$  $)$ that incorporates the safety constraints via approximate Bayesian inference. Finally, we want to obtain a policy such that,

$$p_{\pi^{+}}(\texttt{safe}_{t:T} \mid \text{  }) \geq p_{\pi^{+}}(\texttt{safe}_{t} \mid \text{  }) \tag{5.13}$$

$$\geq p_{\pi}(\texttt{safe}_{t:T} \mid \text{  }) \tag{5.14}$$

This means that our NeSy policy provides stronger safety guarantees than the single-step shielded policy (5.13) proposed by W.-C. Yang et al. (2023), which itself is safer than the unshielded baseline (5.14), and this holds over any time horizon. In future work, we plan to empirically validate this hypothesis and further integrate NeSy-MMs into the RL framework by analysing how sequential reasoning affects both safety and expected return.

More broadly, this line of research brings us closer to the central question of this thesis: how to design agents that can operate effectively in uncertain environments, while perceiving the world through subsymbolic input, reasoning over symbolic representations, planning ahead, and learning missing knowledge from experience.

### 5.6.2 Towards Real-World Applications

While our experiments in MiniHack assume access to perfect symbolic abstractions, ongoing work is exploring how to relax this assumption in more realistic settings. In particular, we have trained a YOLO-based perception module to extract symbolic predicates, such as the presence of lava or monsters, from raw pixel observations. These extracted symbols are then passed to the logic-based shield, allowing the system to operate without privileged access to the environment state.


Figure 5.6: Screenshot from the CARLA simulator. A shielded PPO agent drives down a lane while respecting high-level safety constraints encoded in a probabilistic logic program. The agent must stay within the lane and avoid unsafe actions, such as steering left while already deviating left. The shielded policy is trained to respect these constraints without requiring reward shaping.

Beyond MiniHack, we have also extended our approach to autonomous driving using the CARLA simulator (Dosovitskiy et al. 2017). In this setting, the agent learns to navigate a simple road while adhering to safety constraints, such as staying within lane boundaries or maintaining safe speed. Sensor readings are abstracted into coarse symbolic predicates (*e.g.* lane deviation, over speed), which are used by the logic shield to suppress unsafe actions during training. The shielded agent learns a safe policy more efficiently than with reward shaping alone. Figure 5.6 shows a screenshot of the agent training in action.

Finally, we are investigating the use of shielded reinforcement learning in robotics domains. Here, the goal is not only to guarantee safety, but also to constrain the large action and state spaces typical of real-world manipulation and navigation tasks. By leveraging symbolic abstractions and logic-based rules, we aim to improve data efficiency and robustness in physical systems.

# Chapter 6

# Can Large Language Models Be Intelligent Agents?

This short chapter is based on the following peer-reviewed conference publication.

> R. Hazra, G. Venturato, P. Z. D. Martires, and L. De Raedt (2025b). "Have Large Language Models Learned to Reason? A Characterization via 3-SAT". in: *Second Conference on Language Modeling, COLM 2025*

The idea of characterizing LLMs via the phase transition behaviour of 3-SAT was conceived by L. De Raedt. The methodology was developed collaboratively by R. Hazra, P. Zuidberg Dos Martires, L. De Raedt, and me, combining R. Hazra's expertise in LLMs with my background in computational complexity and satisfiability problems. R. Hazra and I conducted the experimental investigation. P. Zuidberg Dos Martires and L. De Raedt provided active supervision and helped shape the experimental design and the conceptual direction. The paper was written mainly by R. Hazra, with contributions and feedback from all co-authors.

In this chapter, we take a step back and ask whether recent advances in Large Language and Reasoning Models (LLMs and LRMs) could tackle some of the settings explored in this thesis. Specifically, we investigate whether LLMs can be used to build agents that perceive the world **(i)**, reason over time **(ii)**, and

operate under uncertainty in complex tasks such as planning and decision-making (Figure 1.1).

LLMs already demonstrate impressive capabilities across a variety of domains. They can interpret raw textual or visual input, and exhibit emergent behaviours such as planning (Ahn et al. 2022; Hazra, Zuidberg Dos Martires, et al. 2024; W. Huang et al. 2022), theorem proving (Jiang et al. 2023; Welleck et al. 2022; K. Yang et al. 2022), search and optimisation (Hazra, Sygkounas, et al. 2024; Romera-Paredes et al. 2024; C. Yang et al. 2024), self-reflection (Madaan et al. 2023; Yao, Yu, et al. 2023), and tool use via APIs (Schick et al. 2023). When combined with prompting strategies such as chain-of-thought reasoning (Wei et al. 2022; Yao, Zhao, et al. 2023), these models appear increasingly adept at solving complex problems in zero-shot or few-shot settings.

However, closer examination reveals substantial limitations. Despite their empirical successes, it remains unclear whether LLMs genuinely internalise structured reasoning processes or merely exploit statistical regularities. This concern is particularly acute in sequential decision-making tasks, such as those studied in this thesis, where agents must reason under uncertainty, maintain and update beliefs over hidden states, and plan over long horizons. Crucially, probabilistic reasoning, which is central to the models developed in this dissertation, remains implicit and unreliable in LLMs, and difficult to elicit consistently through prompting alone.

The sequential decision problems we consider span from NP-complete complexity in classical planning up to PSPACE-complete complexity in partially observable settings (Papadimitriou and Tsitsiklis 1987), often involving relational structure, symbolic constraints, and probabilistic inference (#P-hard). While LLMs offer exciting capabilities, they do not yet provide the grounded reasoning, sequential depth, and probabilistic semantics needed to meet the demands of truly intelligent agents as envisioned in this work.

From a theoretical standpoint, LLMs face inherent computational limits. Multi-layer transformers lie within low-complexity classes such as log-uniform $\mathrm{TC}^0$ (Merrill and Sabharwal 2023) and cannot solve problems like Derivability, 2-SAT, Horn-SAT, or Circuit Evaluation unless $\mathrm{L} = \mathrm{NL}$ (Peng et al. 2024). These bounds suggest structural constraints on their reasoning capacity. Yet this ceiling can be raised with additional computation. Recent work shows that transformers augmented with $T$ chain-of-thought (CoT) steps can simulate Boolean circuits of size $T$, expanding their power to $\mathrm{P}/\mathrm{poly}$, a superclass of P (Z. Li et al. 2024). This implies that such models can, in principle, solve problems in P, including 2-SAT (NL-complete) (Bollobás et al. 2001) and Horn-SAT (P-complete) (Demopoulos and Vardi 2006), and even tackle some tractable instances of NP-complete problems like 3-SAT, but only when heavily scaffolded

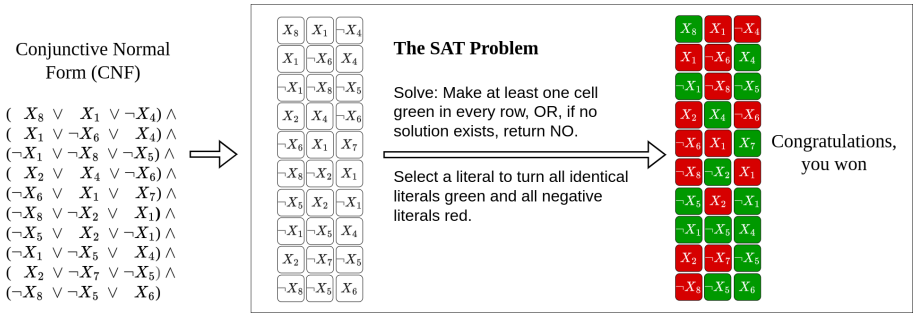



Figure 6.1: **The 3-SAT problem**, visualized using a variant of the SAT game (Roussel n.d.). In SAT, the goal is to return a truth assignment to Boolean variables that satisfies a Boolean formula in conjunctive normal form (CNF), or return unSAT if none exists. In the visualization, each row represents a clause – a disjunction (logical OR, $\vee$) of literals, where each literal is either positive ($X_1$) or negative ($\neg X_1$). A clause is satisfied if at least one of its literals is assigned true. Clauses are joined by logical AND ($\wedge$), so all must be satisfied for the formula to hold. If no such assignment exists, the formula is unsatisfiable.

and given ample compute resources (Lightman et al. 2024; Snell et al. 2025).

Still, theoretical capacity does not imply reliable reasoning. Increasing evidence suggests that LLMs do not learn to reason in a way that generalises. Even frontier LRMs often fail beyond certain complexity thresholds, casting doubt on their viability as general-purpose reasoning agents (Shojaee et al. 2025; H. Zhang, L. H. Li, et al. 2023). Recent work further suggests that gradient-based training tends to discover algorithms that fall within parallelisable complexity classes such as NC, and struggles with problems that require inherently sequential reasoning, hypothesised to lie in the class IS = P\NC (Saldyt and Kambhampati 2025). Static transformer architectures have also been shown to lack the ability to simulate such sequential computation unless explicitly augmented with control flow (Shaw et al. 2024). These results align with the motivation of this thesis: sequential reasoning, especially under uncertainty, requires explicit inference and learning mechanisms beyond those currently captured by standard LLMs.

To make these limitations more concrete, we narrow our focus in this chapter to a simpler class of problems: random instances of 3-SAT (Figure 6.1), a canonical NP-complete problem that underlies many core AI tasks such as planning and constraint satisfaction (Garey and Johnson 1990). We deliberately consider small instances with few variables and constraints, settings that are well within the capabilities of exact solvers. Nonetheless, we show that even in this

restricted regime, current state-of-the-art LLMs struggle to solve these problems reliably. This raises serious doubts about their present ability to support the kind of structured, multistep reasoning required for sequential decision making.

## 6.1 3-SAT Phase Transition

We follow the standard random 3-SAT generation model of Selman et al. (1996), where each clause is sampled independently with replacement. This ensures a distribution free of domain-specific biases and allows precise control over formula complexity through the clause-to-variable ratio $\alpha = {}^{m}/_{n}$. A key empirical observation is the *phase transition* phenomenon (Cheeseman et al. 1991): as $\alpha$ increases, the probability that a formula is satisfiable drops sharply around a critical threshold $\alpha_c \approx 4.267$ (Ding et al. 2015; Mertens et al. 2006). This defines three distinct regimes: an under-constrained (easy) region for $\alpha < \alpha_c$, a critical (hard) region near $\alpha_c$, and an over-constrained (easy) region for $\alpha > \alpha_c$ (see Figure 6.2).

SAT solvers, such as those based on the DPLL algorithm (Davis et al. 1962) and its modern CDCL variant (Silva and Sakallah 1996), exhibit peak runtimes near the phase transition due to increased search complexity. We adopt this framework to analogously assess LLM reasoning: by analysing model performance across different values of $\alpha$, we can investigate whether models rely on superficial patterns or exhibit genuine reasoning abilities.

## 6.2 LLMs Phase Transition Behaviour

We evaluate the performance of LLMs by measuring their accuracy in solving SAT Search (*i.e.* returning a satisfying assignment or "no") across formulas with varying $\alpha$. As shown in Figure 6.3 [Left], we find that all LLMs exhibit *inverted* phase transitions (Easy-Hard-Easy pattern). Their performance is high in the easy regions, while it significantly drops to $\approx 10\%$ in the hard region. In Figure 6.3 [Right], we plot the performance of LLMs against the *satisfiability ratio*, defined as $\frac{\text{model count}}{2^n}$, where the model count is the number of satisfying assignments and $n$ is the number of variables. This denotes the probability that a randomly selected variable assignment satisfies the given 3-SAT theory[1]. We can observe a clear dependence between the accuracy and satisfiability ratio:

[1]Note, this is different from the probability that at least one satisfying assignment exists. Two formulas can both be satisfiable but have different model counts, hence different satisfiability ratios.

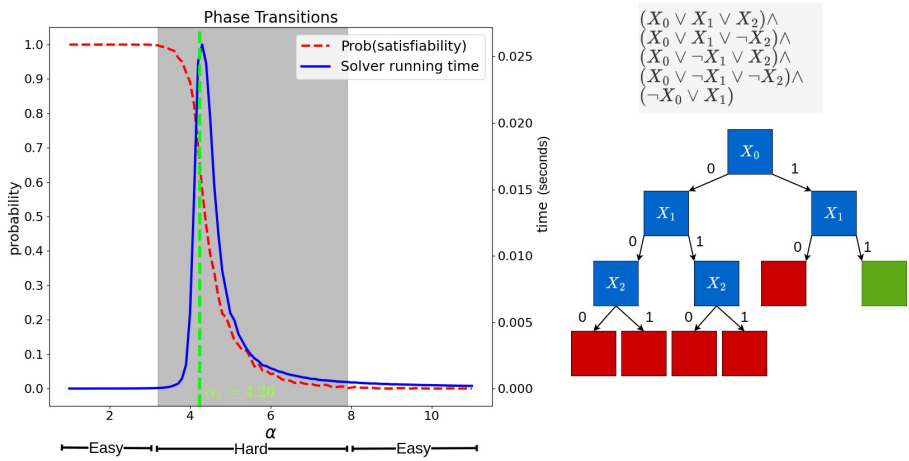



Figure 6.2: [Left]: **Random 3-SAT Phase Transitions** (Cheeseman et al. 1991). Plotted in red is the probability of a randomly sampled 3-SAT formula being satisfied against the hardness $\alpha$ of the formula. We can observe a clear phase transition occurring at $\alpha_c \approx 4.267$ (marked by a green - -). We identify two Easy regions, one on either side of $\alpha_c$. The grey area in the middle denotes the Hard region. The boundaries of the hard region are defined where the probability of the formula being satisfied ceases to be deterministically 1 (left) or 0 (right). The solid blue line shows the mean time taken by the MiniSAT solver to solve 3-SAT instances. Notably, there is a spike in the solver's runtime near $\alpha_c$. This is due to the absence of useful heuristics in this region, forcing the solver to resort to exhaustive searches. [Right]: **DPLL Search Trace** from a SAT Solver for the given formula. Red boxes denote unsatisfiable assignments; the green box highlights a satisfying one. 0, 1 are truth assignments.

formulas with more satisfying assignments tend to be easier for LLMs, and this holds across both easy and hard regions.

## 6.3 LLMs Search Traces

We qualitatively analysed the search traces generated by R1 and other LLMs on SAT-CNF, where the input to the LLM is given in CNF (Figure 6.4), and SAT-Menu, where the input is translated into a natural language formulation (Figure 6.5). Specifically, we annotated the "thinking" traces and attempted to map them to known symbolic search algorithms to better understand the type and depth of search strategies employed, particularly by R1.

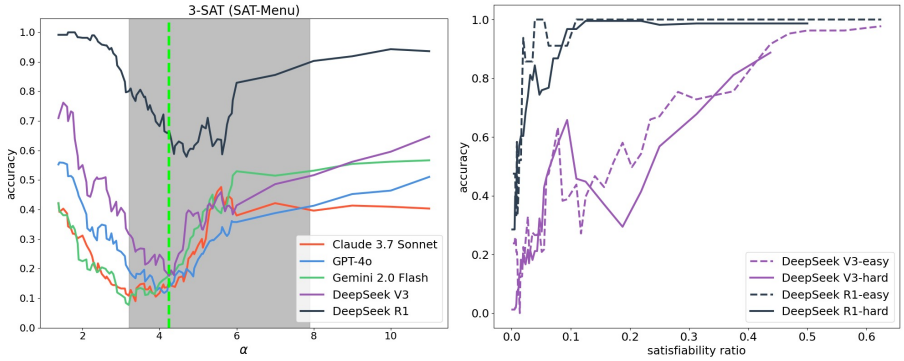



Figure 6.3: [Left] 3-SAT performance comparison for the **search** version of SAT-Menu. [Right] Accuracy vs. satisfiability ratio on the search version of SAT-Menu. We only include satisfiable instances and analyse hard (solid line) and easy regions (dashed line) separately.

In Figure 6.4, we highlight a failure case where both R1 and GPT-4o incorrectly conclude that a satisfiable formula is unsatisfiable. In Figure 6.5 we observe that the search process is noticeably more unstructured compared to SAT-CNF. We see that R1 struggles to map SAT-Menu inputs into structured CNF-like representations, which it could potentially handle more effectively. This limitation suggests that R1's reasoning ability is still closely tied to familiar input formats, and its generalisation to more natural or abstract representations remains a challenge.

## 6.4 Conclusion

This chapter investigated whether recent LLMs can reliably perform structured reasoning in a controlled, non-sequential setting. By analysing their behaviour on small random instances of 3-SAT, we found that even the most advanced models struggle to generalise, especially near the phase transition, where reasoning becomes more combinatorially demanding.

Despite their apparent capabilities in perception and language, LLMs fail to match even classical solvers on tractable instances of canonical NP-complete problems, which lie at the core of many symbolic AI tasks. Their reasoning is neither sound (*i.e.* produces incorrect conclusions) nor complete (*i.e.* it cannot guarantee finding a solution), and they exhibit strong sensitivity to input format and superficial features, rather than displaying robust, algorithmic behaviour.

This has direct implications for the broader goal of building intelligent agents. If LLMs cannot reliably solve even small, isolated satisfiability problems, it is unlikely that they can yet support the kind of sequential probabilistic reasoning required for planning, acting under uncertainty, or maintaining beliefs in dynamic environments. These findings underscore the limitations of current models, but they do not preclude future progress. On the contrary, they highlight the opportunity to develop hybrid systems that combine the strengths of LLMs, like their flexibility, scalability, and language understanding, with structured reasoning frameworks like those presented in this thesis. By grounding learning and inference in formal semantics, such approaches can compensate for the shortcomings of purely data-driven methods and offer the kinds of guarantees, such as soundness, completeness, and safety, that are essential for deploying intelligent agents in real-world, safety-critical applications.

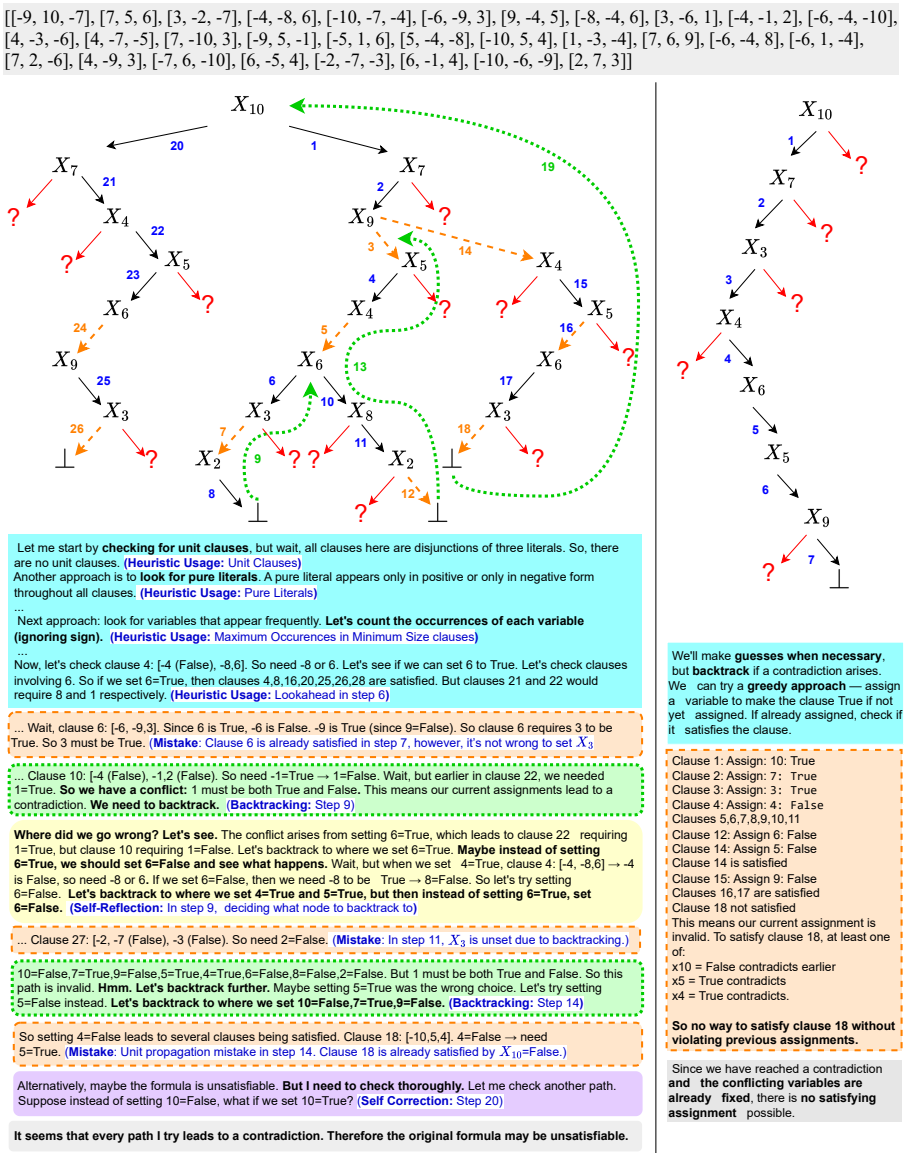



Figure 6.4: **Failure Cases**: SAT-CNF traces for DeepSeek-R1 and GPT-4o. Although the input formula is satisfiable, both models incorrectly predict it as unsatisfiable. Coloured boxes indicate model behaviours: cyan for heuristic variable selection, orange - - - for mistakes, green . . . for backtracking, yellow for self-reflection, and violet for self-correction. Left branch always represents a *True* assignment. ⊥ marks unsatisfiability, and ? indicates an unexplored subtree. Integers are denoted as variables ($10 \rightarrow X_{10}$). Numbers show the order of steps. The input formula is in CNF, where each list of integers represents a clause (e.g., [-9, 10, -7] $\mapsto (\neg X_9 \vee X_{10} \vee \neg X_7)$), and the full formula is a conjunction ($\wedge$) of these clauses.

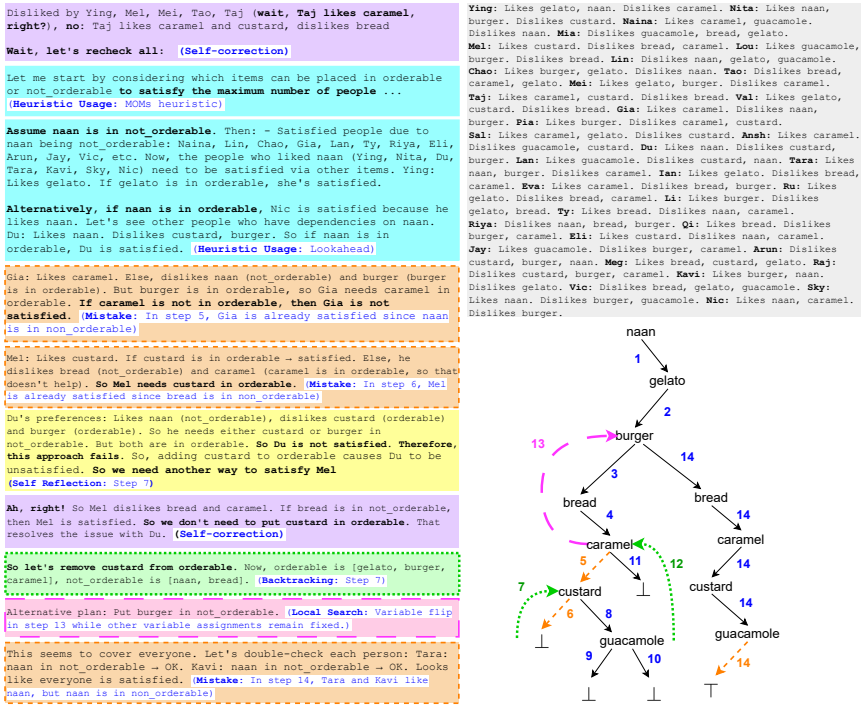



Figure 6.5: **Failure Cases**: SAT-Menu traces for DeepSeek-R1. Although the input formula is unsatisfiable, R1 incorrectly predicts it as satisfiable. Coloured boxes indicate model behaviours: cyan for heuristic variable selection, orange - - - for mistakes, green . . . for backtracking, yellow for self-reflection, violet for self-correction, and magenta for local search. Left branch always represents an assignment to the *orderable* list and vice versa. ⊥ marks unsatisfiability. Numbers show the order of steps.

# Chapter 7

# Conclusion

Throughout this dissertation, we explored how to integrate symbolic and subsymbolic methods to build agents capable of expressive and efficient sequential decision making. The overarching goal was to advance neurosymbolic AI (NeSy) by enabling learning agents to perceive, reason, and act under uncertainty. This final chapter summarises the main findings in light of the research questions and outlines future directions toward a more general and complete neurosymbolic framework.

## 7.1 Summary

This dissertation pushes the boundaries of neurosymbolic AI towards expressive and efficient sequential decision making. The ultimate goal is to develop agents that can perceive the world **(i)**, reason and plan sequentially **(ii)**, act effectively **(iii)**, compensate for missing knowledge **(iv)**, and generalise across entities **(v)**. Each of the following research questions targeted a key facet of this overarching ambition.

**How can sequential symbolic reasoning incorporate subsymbolic perception in a principled and probabilistic way?** This question lies at the heart of sequential neurosymbolic AI, and is addressed most directly in Chapter 4. There, we introduced *relational neurosymbolic Markov models* (NeSy-MMs), which connect neural perception to symbolic reasoning through a probabilistic latent space. The model maps subsymbolic inputs (*e.g.* images) to interpretable

symbolic facts via neural predicates, which are then used in a logic-constrained probabilistic model that evolves over time. Crucially, the interface between perception and reasoning is both probabilistic and differentiable, enabling end-to-end training and uncertainty-aware inference. This addresses the perception-reasoning interface **(i)** - **(ii)**, and provides a principled solution to our first question, forming the core of the intelligent agent in Figure 1.1.

**How can probabilistic reasoning be extended to support sequential decision making and planning under uncertainty?** In Chapter 3, we proposed a novel framework for sequential probabilistic reasoning based on *dynamic decision circuits* (DDCs). These circuits compile dynamic decision networks into algebraic representations suitable for efficient and differentiable evaluation. Unlike classical planning methods, our approach embeds symbolic structure directly into the computation graph of the Bellman updates via knowledge compilation. This enables gradient-based learning of model parameters and exact inference in stochastic decision problems. Despite scalability limitations due to the use of exact methods, this constitutes a first step towards end-to-end differentiable symbolic planning under uncertainty **(ii)**, addressing our second research question.

**Can neurosymbolic methods be effectively applied in online learning settings?** Chapter 5 examined this question in the context of reinforcement learning. We demonstrated, with a novel and optimised code base, how integrating *probabilistic logic shields* with deep RL agents, enables safe and structured policy learning in online settings. These shields enforce symbolic safety constraints during both exploration and training, reducing unsafe behaviour and improving sample efficiency. Compatible with standard RL algorithms and operating in real time, our framework demonstrates that neurosymbolic reasoning can be applied effectively in interactive learning scenarios **(iii)**, completing the loop between the intelligent agent and the environment depicted in Figure 1.1, and affirmatively answering our third research question.

**Is it possible to compensate for partial domain knowledge through learning, while retaining coherent reasoning capabilities?** This challenge recurs throughout the thesis and is addressed from multiple angles. In Chapter 3, we demonstrated how parameters of symbolic decision models can be learned from data via gradient descent. In Chapter 4, we showed that even when symbolic knowledge is incomplete (*i.e.* missing rules), we can approximate it with learned neural predicates, and the amortised inference mechanism can still extract

useful structure. The probabilistic formulation of NeSy-MMs further allows the model to reason under uncertainty and maintain coherence with the provided constraints. Finally, Chapter 5 illustrated how constraints can be specified independently of the policy and still guide learning. Together, these insights confirm that coherent symbolic reasoning can be retained while compensating for partial knowledge through learning **(iv)**, addressing our fourth research question.

**How can relational rules be leveraged to enable generalisation across entities, tasks, or environments?** Generalisation is a key strength of symbolic approaches, and this thesis demonstrated how relational logic can be used to structure both models and constraints for this purpose. Chapter 4 showed that relational rules let agents adapt to different numbers or configurations of objects at test time, and even modify constraints on-the-fly. In Chapter 5, we illustrated that rules can encode abstract safety requirements which apply to unseen environments. Moreover, by formulating dynamics and constraints in terms of entity-agnostic predicates, all proposed models support zero-shot generalisation to new configurations, addressing the problem of our-of-distribution generalisation **(v)**, and answering our final research question.

**Can Large Language Models Be Intelligent Agents?** Although this question falls outside the main research line of this dissertation, it directly connects to the broader goal of building intelligent agents capable of perception, reasoning, and action. Chapter 6 explores this by formally analysing the reasoning abilities of Large Language and Reasoning Models (LLMs and LRMs) using phase transitions in 3-SAT. This approach moves beyond benchmark-based evaluation and provides principled insights grounded in computational complexity. Our findings reveal that, while most models rely on statistical shortcuts, certain LRMs like DeepSeek R1 exhibit structured reasoning behaviours resembling search-based methods, although sill with important limitations. Therefore, we believe there is a good opportunity to combine these emerging reasoning models with structured symbolic components to create hybrid agents that benefit from both flexibility and interpretability (Shindo et al. 2024).

## 7.2 Future Perspectives

This dissertation introduced new models, algorithms, and tools that together provide solid building blocks for neurosymbolic agents that can learn, reason, and act in complex, uncertain environments. While each contribution offers a

targeted solution to one or more research questions, a complete, unified system capable of robust open-world generalisation remains out of reach. In this section, we discuss what is missing and propose directions to move closer to that vision.

**From Inference to Control.** As discussed in Section 5.6.1, a natural next step is to extend NeSy-MMs to reinforcement learning settings, thereby closing the perception-reasoning-action loop. Early experiments show promise in using the evidence-constrained sequential inference as a way to re-normalise the agent's policy. However, RL introduces new challenges such as additional optimisation, exploration, and distant rewards. Incorporating symbolic structure may provide an avenue for more sample-efficient and safer exploration.

**Relational Perception.** While the current models we introduced support relational reasoning, perception remains bottlenecked by fixed-object encodings or limited predicate extractors. A key challenge is developing *relational detectors*, perception modules that can adapt to varying numbers of entities and their relationships. Existing work on object-centric learning (Fan et al. 2024; Locatello et al. 2020; Y. Zhang et al. 2023) provides early results, but with limited capabilities, or with focus on specific domains (Delfosse et al. 2023). At the other extreme, object detectors like Redmon et al. (2016), and its modern adaptations, offer excellent performance but break differentiability, complicating their use in end-to-end learning. Bridging this gap remains an open research direction.

**Beyond Filtering.** In partially observable domains, smoothing provides a means to incorporate future evidence into belief updates. Recent work on differentiable particle smoothers (Younis and E. Sudderth 2024) may allow agents to condition inference not only on past and current observations, but also on future goals or specifications, an idea already hinted at in Section 5.6.1. This could improve planning under constraints and long-horizon tasks.

**Hierarchical Reasoning and Policy Composition.** As the complexity of tasks grows, agents must learn to decompose problems into manageable sub-goals. Hierarchical reinforcement learning and policy compositionality are natural solutions. Symbolic logic can act as the glue connecting sub-policies, enabling modularity, reusability, and explainability. This connects to recent work on hierarchical planning and skill learning (Kokel et al. 2022; Wang et al. 2021).

**Applications in Robotics and Sports Analytics.** Several real-world domains stand to benefit from neurosymbolic methods. In sports analytics, symbolic priors can model strategic constraints or domain-specific knowledge (*e.g.* offside rules, teams formations, and similar strategic constraints), while learned components provide adaptability to handle missing data (Van Roy et al. 2023). In robotics, NeSy systems can help bridge the gap between high-level task specifications and low-level control policies.

**Temporal Logic and Non-Markovian Structure.** Finally, many domains exhibit dependencies that violate the Markov assumption. Temporal logics like LTL offer a principled way to express constraints over future sequences, and could be integrated with symbolic planning or RL. Recent works explore LTL-guided reinforcement learning (Umili et al. 2024) and open up the possibility for NeSy agents to handle more complex temporal tasks.

In conclusion, this dissertation contributes foundational work for a class of AI agents that can reason and act in structured, uncertain environments using a principled fusion of logic and learning. While each piece is promising on its own, the path forward lies in making these components more integrated, adaptive, and scalable, ultimately moving closer to the long-standing vision of intelligent behaviour in the real world.

# Appendix A

# A Model To Rule Them All

## A.1 Architectures

**Generative Task.** The variational transformer architectures consist of a separate initial convolutional VAE architecture coupled with a transformer decoder that autoregressively generated the next images in the sequence. This transformer has a causal self-attention layer with 8 heads and 64 keys, followed by a cross attention layer with the same parameters. After these layers, the decoder portion of the VAE is used to generate the images. The NeSy-MM model used multiple smaller networks as it is naturally decomposed. Concretely, there is a small convolutional neural network that classifies the initial location of the agent from the first image into either $(5+2)^2 = 49$ or $(10+2)^2 = 144$ classes, depending on the grid size. Additionally, a small convolutional encoder network encodes the same initial image into a two-dimensional Gaussian distribution. The NeSy-MM moves the location of the agent according to the rules of MiniHack for a given set of actions, resulting in an estimated distribution for the agent at every subsequent time step. A final convolutional decoder, the same as used by the transformer architecture, then generates an image from the encoding of the initial image together with the planned location of the agent for every time step. The deep Markov model (Deep-HMM) uses the exact same setup, only replacing the logical transitions by small neural networks with two hidden layers of size 64 and 32.

**Discriminative Task.** Our transformer architecture follows the usual pattern of an encoder-free transformer. That is, it has a decoder component that operates

on a sequence of embedding vectors of size 32. At the start, there is only one embedding containing the two-dimensional starting location of the agent as the first two components, followed by a series of zeroes. The decoder then autoregressively computes the next embedding from all previous embeddings by applying, in sequence, a dropout operation with probability 0.1, a causal self-attention layer with 8 heads and key dimension 64, and finally a cross-attention layer with as context the incoming action and observation on whether the agent was hit or not. The cross-attention layer again has 8 heads and key dimension 64 and both attention layers also use dropout internally with a probability of 0.1. To provide the final sequence classification, a simple MLP network with 2 hidden layers of sizes 64 and 32 with ReLU activations is used. It takes the final embedding vector as input such that it can deal with sequences of varying length, *i.e.* it is relational in time. The dense output layer is of dimension 1 and uses a sigmoid activation, predicting the probability that the agent dies in the trajectory or not.

Our NeSy-MM model and Deep-HMM are again close in terms of architecture, but differ in their use of neural networks. Both models take the initial location of the agent as input and estimate the location of all enemies by a uniform prior. They transition these locations to future time steps using actions. For the enemies, the actions are not given and this is where the NeSy-MM uses a simple MLP with two hidden ReLU layers of size 64 and 32 and an output log softmax layer of size 8, as the enemies can move vertically, horizontally, and diagonally. In short, the NeSy-MM transitions the agent from its previous location using the given action, while the enemies are transitioned from their sampled previous locations and *predicted* actions. Deep-HMMs use a neural network to immediately predict the next location for both the agent and the enemy. In case of the agent, the neural net takes both location and given action as input. The architecture is the same as the action network of the NeSy-MM, albeit with an output of size $(N + 2)^2$ where $N$ is the grid size, predicting a distribution over grid cells. Here we can also see why the NeSy-MM can tackle larger grid sizes while the Deep-HMM can not. Our NeSy-MM uses a relational logic transition to move entities in the world (Appendix A.2) while the Deep-HMM instead uses a neural network that necessarily has a fixed output dimension in terms of the grid size. Next, both models *exactly* condition, in the probabilistic sense, the predicted distribution of future locations on the incoming evidence whether the agent was hit or not. For the NeSy-MM, the observation function is again logical, meaning it computes the probability that the agent is hit, given the agent location and all enemy locations. This involves knowing the probability that an attack from an enemy succeeds. Since this information can be seen as part of the behaviour of the monster, we do not give it as input to our model and instead replace it by a learnable parameter. For the Deep-HMM, the observation function is completely replaced by a neural

network with two hidden ReLU layers of size 64 and 32 followed by a dense log softmax output layer modelling the log probability. That is, computing the probability that an enemy hits the agent from their locations is computed by a neural net for each enemy separately and then combined into the total probability of hitting following the correct expression for the probability of the union of multiple events. Finally, both models explicitly model the health of the agent and subtract an estimate of the damage based on the computed probability of hitting. The final probability that the agent is dead predicted by the models is then given by the frequency of sampled trajectories where the agent dies.

## A.2 Rules of NetHack

Our NeSy-MM relies on relational logical knowledge. In this section, we describe in detail the knowledge that we included in our model.

**Generative Task.** The knowledge necessary in this case is very simple and can be summarised by the logic program in Algorithm 7.

**Algorithm 7** Logic programming encoding of the knowledge required for the generative task in Chapter 4.

```
agent(X, Y, T) ~ detector(Image,T).

action(A, T) ~ categorical(
    [0.25, 0.25, 0.25, 0.25],
    [up, down, left, right]
).

agent(X,Y+1,T) :- action(up,T-1), agent(X,Y,T-1).
agent(X,Y-1,T) :- action(down,T-1), agent(X,Y,T-1).
agent(X+1,Y,T) :- action(right,T-1), agent(X,Y,T-1).
agent(X-1,Y,T) :- action(left,T-1), agent(X,Y,T-1).
```

Where, on the first line, we say that the agent location at time `T` is given by a neural detector. Then, we just describe that there are four possible mutually exclusive actions (`up`, `down`, `left`, and `right`). Finally, we describe the effect of each action on the agent's location.

**Discriminative Task.** In this experiment, we do not get the agent's location by applying a neural detector to an image. Instead, the exact and deterministic

initial symbolic location is given. The remainder of the logic for how the agent transitions is the same as in the generative task.

For the enemies, the setup is analogous, at least in terms of how they transition. The difference being that the enemies' location is initially completely uncertain, modelled by a uniform categorical over the entire grid. Additionally, the actions are not uniformly sampled, but are predicted by the neural network that we are trying to optimise.

**Algorithm 8** Logic programming encoding of the knowledge required for modelling the health of the agent in the discriminative task in Chapter 4.

```
damage(T, Damage) ~ categorical(
    [0.25, 0.25, 0.25, 0.25],
    [1, 2, 3, 4]
)

agent_hp(0, 12).
agent_hp(T, HP) :-
    agent_hp(T-1, HP),
    not hit(T).
agent_hp(T, HP - Damage) :-
    agent_hp(T-1, HP),
    damage(T, Damage),
    hit(T).
```

To eventually deduce whether the agent has died or not, we need to keep track of the agent's health and model the damage the agent takes. This is described in Algorithm 8. At the beginning when `T` is 0, the agent has 12 hitpoints (HP) that decreases by the amount `Damage` if there is a `hit`. The damage value is dependent on the enemy type. In our case, we used the NetHack `imp` that has a claw attack with 1d4 damage. Hence, the damage is modelled by a uniform categorical variable over the domain `[1, 2, 3, 4]`.

**Algorithm 9** Logic programming encoding of the knowledge required for modelling the hit condition in the discriminative task in Chapter 4.

```
hit(T) ~ bernoulli(t(_)) :-
    agent(Xa, Ya, T), A = [Xa, Ya],
    enemy(Xe, Ye, T), E = [Xe, Ye],
    distance(A, E, D), D = 1.
```

Furthermore, a `hit` can occur only if the enemy is in one of the 8 cells around the agent, because the claw attack is a melee attack (Algorithm 9). Notice that the `hit` variable is Bernoulli distributed and the parameters are learned as indicated by the predicate `t(_)`. This predicate represents a learnable variable.

In this case the probability that a hit succeeds, which we do not assume to know as we consider it as part of the unknown behaviour of the enemy. Therefore, we do not know when the hit is successful, but we observe the `hit` variable during training. Finally, we need a rule to understand when the agent is dead (Algorithm 10).

**Algorithm 10** Logic programming encoding of the knowledge required for modelling the dead condition in the discriminative task in Chapter 4.

```
agent_dead(T) :-
    agent_hp(T, HP),
    HP =< 0.
```

## A.3 Setup and hyperparameters

We ran the experiments on an NVIDIA P100 SXM2@1.3 GHz (16 GB HBM2) GPU, coupled with an Intel Xeon Gold 6140 CPU@2.3 GHz (Skylake) and 192 GB of RAM. From the software perspective, the machine we used runs the Rocky Linux 8.9 (Green Obsidian) operating system. Moreover, we ran all the experiments in a Python 3.10.4 environment with the packages listed in the `requirements.txt` file available in the codebase. We report in Table A.1 the exact versions used to produce the results in the paper. All experiments were repeated 5 times on this setup for our method and all baselines. Results are reported using averages and standard errors. For the generative experiments on grid size 10, we downscale the images by a factor of 2 to use 8 pixels per cell instead of the standard 16 pixels per cell. This downscaling is due to the memory limitation of the GPU we had available and was applied for all the models. The hyperparameters were obtained via a separate grid search using a held-out validation set. Table A.3 and Table A.4 report the tested values, together with the optimal ones used to produce the results in the paper. Adam (Kingma and Ba 2015) was used as the optimiser for all methods. We used seeds to remove as much randomness as possible from the training process. The five runs for the generative experiment were obtained all with seed 42, while the ones for the discriminative experiment have been obtained with seeds from 0 to 4. The reason for this difference is merely technical: in the discriminative case, the evaluation is done in a second phase because of the several out-of-distribution settings, so we needed to easily distinguish the 5 runs and the seed number was the easiest choice.

The datasets used in the paper are generated with the `generator.py` scripts present in the `data/` folder of each experiment. The seed used for the generation

| Package | Version |
|---|---|
| tensorflow[and-cuda] | `2.13.1` |
| tensorflowprobability | `0.19.0` |
| gym | `0.23.0` |
| minihack | `0.1.6` |
| nle | `0.9.0` |
| einops | `0.8.0` |
| wandb | `0.13.5` |
| matplotlib | `3.8.0` |

Table A.1: Required packages and their versions.

| | Methods | | |
|---|---|---|---|
| $N$ | VT | Deep-HMM | NeSy-MM |
| 5 | 385.48 ± 2.79 | 1392.48 ± 6.83 | 726.17 ± 3.40 |
| 10 | 409.32 ± 1.97 | 410.15 ± 0.31 | 399.61 ± 4.83 |

Table A.2: Total training time (s) for the generative experiment, for grid sizes $N \times N$ and sequence length 10.

is 0. All the other parameters are described in Section 4.4. Running the training scripts with the default parameters automatically creates the datasets used to produce the results in the paper. With this information, the generation of the datasets is fully deterministic and perfectly reproducible. Train, test, and validation sets contain, respectively 5000, 1000, and 500 trajectories.

## A.4 Generation

In this section, we showcase the generative quality of NeSy-MMs (Figure A.1) and variational transformers (Figure A.2), compared to the NeSy-MM generation from Figure 4.5. Both the transformer and Deep-HMM might be able to provide reasonable test metrics, but they prove to be rather poor generative models. The Deep-HMM provides clean generations of the background, but it fails to capture the exact location of the agent and often generates the agent in a superposition of different locations. Moreover, the agent can jump around the world because the transitions are not guaranteed to follow the rules of the game. The transformer also provides, most of the time, crisp generations of the background, but it also fails to properly learn the correct transition function and sometimes shows artefacts. In many cases, even if the initial generation is clear, the transformer is unable to move the agent according to the required actions.

| | | Optimal Values | | |
|---|---|---|---|---|
| **Hyperparameter** | **Tested Values** | VT | Deep-HMM | NeSy-MM |
| `n_samples` | 5, 10, 20 | - | 10 | 5 |
| `latent_dim` | 2, 4, 8 | - | 2 | 2 |
| `dropout` | 0.1, 0.2, 0.4, 0.5 | 0.1 | 0.2 | 0.2 |
| `batch_size` | - | 10 | 10 | 10 |
| `n_epochs` | 15, 30, 50, 100, 200 | 100 | 30 | 30 |
| `beta` | 1, 10, 50, 100, 150, 200, 300 | 1 | 50 | 300 |
| `learning_rate` | 0.1, 0.03, 0.01, 0.001, 1e-4 | 1e-4 | 0.001 | 0.001 |

Table A.3: Hyperparameters for the generative experiment. Tested values, and their optimal values for Transformer, Deep-HMM, and NeSy-MM. For the experiment with grid size 10, we changed: for NeSy-MM and Deep-HMM, `n_samples` to 20, `batch_size` to 5, and `n_epochs` to 15; for the transformer `beta` to 50.

| | | Optimal Values | | |
|---|---|---|---|---|
| **Hyperparameter** | **Tested Values** | T | Deep-HMM | NeSy-MM |
| `n_samples` | 100, 1000 | - | 100 | 1000 |
| `dropout` | 0.1, 0.2, 0.4 | 0.1 | - | - |
| `batch_size` | - | 50 | 10 | 50 |
| `n_epochs` | 15, 30, 50, 100, 200 | 50 | 20 | 100 |
| `learning_rate` | 0.01, 0.001, 1e-4 | 1e-3 | 1e-3 | 1e-3 |

Table A.4: Hyperparameters for the discriminative experiment. Tested values, and their optimal values for Transformer (T), Deep-HMM, and NeSy-MM.

In contrast, NeSy-MMs provide clean generation that follow the required actions and adhere to the rules of Nethack. The reader can try to generate more images, to further confirm our findings, with the Jupyter notebook provided in the submitted code material.

### A.4.1 Training Time

Table A.2 shows the total training times for the generative experiment. For the discriminative experiment, where we train the model only on grid size $10\times10$, sequence length 10, and 1 enemy, the times are: $161.79 \pm 1.19$s for the

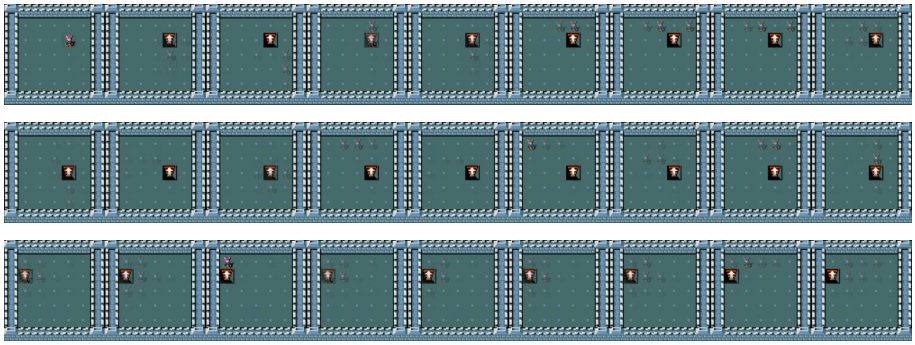


Figure A.1: Generated trajectory for actions: right, down, left, up, right, up, left, down using the Deep-HMM.

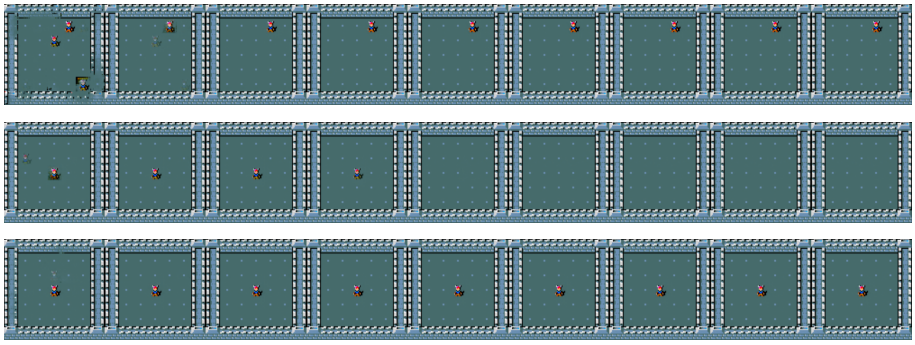


Figure A.2: Generated trajectory for actions: right, down, left, up, right, up, left, down using the variational transformer.

transformer, $805.34 \pm 2.62$s for the Deep-HMM, and $254.53 \pm 9.11$s for our NeSy-MM. Note how the Deep-HMM model takes considerably longer to train and delivers consistently lower performance. For the transformer baselines, we always trained until convergence or until reaching the hardware's limitations. Although these models generally train faster, they still underperform as demonstrated in the paper. The slightly longer training times for our NeSy-MMs can be attributed to the fact we perform sample-wise exact reasoning.

# Appendix B

# Fantastic Shields And Where To Find Them

## B.1 Shields

In this section, we provide the probabilistic logic shields used in the experiments of Chapter 5. The shields are written in Problog, where the query is to compute the probability of the `safe_next` predicate. The inputs to the program are represented by the `action(X)` and `sensor_value(X)` predicates. At runtime, these are substituted with probability values coming from, respectively, the policy and the sensors of the environment.

The shields are designed to prevent the agent from taking actions that would lead to unsafe states, such as falling into lava, bumping into walls, or being attacked by monsters. Algorithm 11 is the shield for the Cliff environment, where there are only two simple rules: do not bump into walls, and do not fall into lava. Algorithm 12 is for the Key Room environment, and it is slightly more complicated. We provide probabilistic rules that lead the agent to perform a meaningful high-level policy. Specifically, we incentivise the agent to first pick up the key, and only after unlocking the door. Algorithm 13 is for the Monster Room environment, where we prevent the agent from being attacked by monsters that are too close or can move towards the agent. In this case, we also handle the case where the monster is at distance 2 from the agent, which is necessary because if the monster and the agent both move in the same grid cell, at the same time, the monster can attack the agent. Notice that in this environment, the monster moves every other time step. Finally, Algorithm 14

is very similar to the one for Cliff, but in this case we model the probabilistic transition function. Notice that in this case, we do not prevent the agent from bumping into walls, as in some cases that is the best action to take to not fall into lava.

## B.2 Architectures and Hyperparameters

All agents trained in the experiments of Chapter 5 used a policy consisting of two identical networks: one for the actor and one for the critic. Each network is a simple MLP with two hidden layers of size 64 and ReLU activations. Training was performed using the Adam optimiser with a learning rate of $10^{-4}$.

We trained the policy using the Proximal Policy Optimization (PPO) algorithm with the following key hyperparameters. Each update was based on $n = 2048$ rollout steps. The collected trajectories were split into mini-batches of size 128 and optimized over 10 epochs. We used a discount factor $\gamma = 0.99$, Generalized Advantage Estimation with $\lambda = 0.95$, and a clipping range of 0.2 for the policy objective. Entropy regularization was applied with a coefficient of 0.01 to encourage exploration. Unless otherwise specified, the default values from Stable-Baselines3 were used for all other parameters.

For the shielded version of PPO, we used the same architecture and hyperparameters, with an additional shielding coefficient $\alpha = 0.1$ for all environments, except for Lava Lake, where we used $\alpha = 30$.

For the plots in Figure 5.4, we repeated each experiment three times with different random seeds (0, 1, and 2) and report the average results. Shaded areas represent the standard error across the three runs. All environments except Cliff, which has a fixed configuration, were initialized with random locations for the agent and the obstacles (or monsters). For Key Room and Monster Room, we used seeds 0, 1, and 2 to initialize the environments. For Lava Lake, we used seeds 1, 2, and 4, as seeds 0 and 3 did not yield instances with a viable safe path to the goal. Additionally, the Monster Room was initialized with one monster, and the Key Room had a room size of 5×5 with an inner locked room of size 2×2. In the Lava Lake environment, the transition function gives a probability of 0.9 to the action that moves the agent in the intended direction, and a probability of 0.05 to each of the two orthogonal actions.

The reward functions are defined as follows. For all the environments there is a $-0.01$ reward for each time step, to encourage the agent to reach the goal as quickly as possible. In the Cliff environment, the agent receives a reward of $+0.5$ for reaching the goal and $-1$ for falling into lava. In Key Room, the agent

receives a reward of +1 for opening the door. In Monster Room, the agent receives a reward of +0.5 for reaching the goal and −0.5 for dying. Finally, in Lava Lake, the agent receives a reward of +1 for reaching the goal and −0.5 for falling into lava.

**Algorithm 11** Probabilistic logic shield for the Cliff environment (Figure 5.2a).

```
action(0)::action(up);
action(1)::action(right);
action(2)::action(down);
action(3)::action(left).

sensor_value(0)::lava(up).
sensor_value(1)::lava(left).
sensor_value(2)::lava(down).
sensor_value(3)::lava(right).
sensor_value(4)::wall(up).
sensor_value(5)::wall(left).
sensor_value(6)::wall(down).
sensor_value(7)::wall(right).

transition(left,left).
transition(right,right).
transition(up,up).
transition(down,down).

% prevent falling into lava
unsafe_next :- action(A), transition(A,NextPos), lava(NextPos).
% prevent bumping into walls
unsafe_next :- action(A), transition(A,NextPos), wall(NextPos).

safe_next :- \+ unsafe_next.
```

**Algorithm 12** Probabilistic logic shield for the Key Room environment (Figure 5.2b).

```
action(0)::action(up);
action(1)::action(right);
action(2)::action(down);
action(3)::action(left).


sensor_value(0)::wall(up).
sensor_value(1)::wall(left).
sensor_value(2)::wall(down).
sensor_value(3)::wall(right).

sensor_value(4)::key(up).
sensor_value(5)::key(left).
sensor_value(6)::key(down).
sensor_value(7)::key(right).

sensor_value(8)::door(up).
sensor_value(9)::door(left).
sensor_value(10)::door(down).
sensor_value(11)::door(right).

sensor_value(12)::has_key.
sensor_value(13)::closed_door.


transition(left,left).
transition(right,right).
transition(up,up).
transition(down,down).

% prevent bumping into walls
unsafe_next :- action(A), transition(A,NextPos), wall(NextPos).

% if the agent doesn't have a key, penalise actions that don't end
 ↪  up in obtaining the key
0.9::unsafe_next :- \+ has_key, closed_door, action(A),
 ↪  transition(A,NextPos), \+ key(NextPos).

% if the agent has the key, penalise actions that don't end up in
 ↪  opening the door
0.9::unsafe_next :- has_key, closed_door, action(A),
 ↪  transition(A,NextPos), \+ door(NextPos).

safe_next :- \+ unsafe_next.
```

**Algorithm 13** Probabilistic logic shield for the Monster Room environment (Figure 5.2c).

```
action(0)::action(up);
action(1)::action(right);
action(2)::action(down);
action(3)::action(left).

sensor_value(0)::wall(up).
sensor_value(1)::wall(left).
sensor_value(2)::wall(down).
sensor_value(3)::wall(right).
sensor_value(4)::monster(up).
sensor_value(5)::monster(left).
sensor_value(6)::monster(down).
sensor_value(7)::monster(right).
% to handle diagonal positions
sensor_value(8)::monster_near(up,left).
sensor_value(9)::monster_near(left,down).
sensor_value(10)::monster_near(down,right).
sensor_value(11)::monster_near(right,up).
% if the monster is at distance 2 from the agent
sensor_value(12)::monster_near(up).
sensor_value(13)::monster_near(left).
sensor_value(14)::monster_near(down).
sensor_value(15)::monster_near(right).

sensor_value(16)::monster_can_move.

monster_near(X,Y) :- monster_near(Y,X).

transition(left,left).
transition(right,right).
transition(up,up).
transition(down,down).

% if a monster is at distance 1
unsafe_next :- action(A), monster(A).

% if a monster can move and is at distance 2
unsafe_next :- action(A), monster_near(A), monster_can_move.
unsafe_next :- action(A), monster_near(A,_), monster_can_move.

% monster at distance 1 and can move, bumping into walls is unsafe
unsafe_next :- action(A), monster(_), wall(A), monster_can_move.

safe_next :- \+ unsafe_next.
```

**Algorithm 14** Probabilistic logic shield for the Lava Lake environment (Figure 5.2d).

```
action(0)::action(up);
action(1)::action(right);
action(2)::action(down);
action(3)::action(left).

sensor_value(0)::lava(up).
sensor_value(1)::lava(left).
sensor_value(2)::lava(down).
sensor_value(3)::lava(right).

% probabilistic transition function
0.9::transition(left,left);
0.05::transition(left,up);
0.05::transition(left,down).

0.9::transition(right,right);
0.05::transition(right,up);
0.05::transition(right,down).

0.9::transition(up,up);
0.05::transition(up,right);
0.05::transition(up,left).

0.9::transition(down,down);
0.05::transition(down,right);
0.05::transition(down,left).

% prevent falling into lava
unsafe_next :- action(A), transition(A,NextPos), lava(NextPos).

safe_next :- \+ unsafe_next.
```

# Bibliography


Ahn, M., A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. J. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, M. Yan, and A. Zeng (2022). "Do As I Can, Not As I Say: Grounding Language in Robotic Affordances". In: *6th Annual Conference on Robot Learning*. URL: https://openreview.net/forum?id=bdHkMjBJG_w.

St-Aubin, R., J. Hoey, and C. Boutilier (2000). "APRICODD: Approximate Policy Construction Using Decision Diagrams". In: *NeurIPS*. Vol. 13. MIT Press.

Austin, J., D. D. Johnson, J. Ho, D. Tarlow, and R. Van Den Berg (2021). "Structured denoising diffusion models in discrete state-spaces". In: *Advances in Neural Information Processing Systems* 34, pp. 17981–17993.

Badreddine, S., A. d. Garcez, L. Serafini, and M. Spranger (2022). "Logic tensor networks". In: *Artificial Intelligence*.

Bahar, R. I., E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi (1997). "Algebraic Decision Diagrams and Their Applications". In: *Formal Methods Syst. Des.* 10.2/3, pp. 171–206.

Baum, L. E. and T. Petrie (1966). "Statistical inference for probabilistic functions of finite state Markov chains". In: *The annals of mathematical statistics* 37.6, pp. 1554–1563.

Bellman, R. E. (1957). *Dynamic Programming*. Princeton University Press.

Bhattacharjya, D. and R. D. Shachter (2007). "Evaluating influence diagrams with decision circuits". In: *Proceedings of the Twenty-Third Conference on Uncertainty in Artificial Intelligence*, pp. 9–16.

– (2010). "Three new sensitivity analysis methods for influence diagrams". In: *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, pp. 56–64.

Bishop, C. M. (2006). "Pattern recognition and machine learning". In: *Springer google schola* 2, pp. 645–678.

Bollobás, B., C. Borgs, J. T. Chayes, J. H. Kim, and D. B. Wilson (2001). "The scaling window of the 2-SAT transition". In: *Random Structures & Algorithms* 18.3, pp. 201–256.

Boutilier, C., R. Dearden, and M. Goldszmidt (2000). "Stochastic dynamic programming with factored representations". In: *Artificial intelligence* 121.1-2, pp. 49–107.

Brockman, G., V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba (2016). *OpenAI Gym*. arXiv: 1606.01540.

Browne, C. B., E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton (2012). "A survey of monte carlo tree search methods". In: *IEEE Transactions on Computational Intelligence and AI in games* 4.1, pp. 1–43.

Buchanan, B., G. Sutherland, and E. A. Feigenbaum (1969). "Heuristic DENDRAL: A program for generating explanatory hypotheses". In: *Organic Chemistry* 30.

Chavira, M. and A. Darwiche (Apr. 2008). "On Probabilistic Inference by Weighted Model Counting". In: *Artificial Intelligence* 172.6, pp. 772–799. ISSN: 0004-3702. DOI: 10.1016/j.artint.2007.11.002.

Cheeseman, P., B. Kanefsky, and W. M. Taylor (1991). "Where the really hard problems are". In: *Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 1*. IJCAI'91. Morgan Kaufmann Publishers Inc., pp. 331–337.

Ciatto, G., F. Sabbatini, A. Agiollo, M. Magnini, and A. Omicini (2024). "Symbolic knowledge extraction and injection with sub-symbolic predictors: A systematic literature review". In: *ACM Computing Surveys* 56.6, pp. 1–35.

Corenflos, A., J. Thornton, G. Deligiannidis, and A. Doucet (2021). "Differentiable particle filtering via entropy-regularized optimal transport". In: *International Conference on Machine Learning*. PMLR, pp. 2100–2111.

Czech, J., P. Korus, and K. Kersting (2021). "Improving alphazero using monte-carlo graph search". In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 31, pp. 103–111.

Darwiche, A. (Apr. 2002). "A Logical Approach to Factoring Belief Networks". In: *Proceedings of the Eights International Conference on Principles of Knowledge Representation and Reasoning*, pp. 409–420.

– (2009). *Modeling and reasoning with Bayesian networks*. Cambridge university press.

– (2020). "An Advance on Variable Elimination with Applications to Tensor-Based Computation". In: *ECAI 2020*. IOS Press, pp. 2559–2568.

Darwiche, A., P. Marquis, D. Suciu, and S. Szeider (2018). "Recent trends in knowledge compilation (dagstuhl seminar 17381)". In: *Dagstuhl Reports*. Vol. 7.

Davis, M., G. Logemann, and D. Loveland (1962). "A machine program for theorem-proving". In: *Communications of the ACM* 5.7, pp. 394–397.

De Raedt, L., K. Kersting, S. Natarajan, and D. Poole (2016). "Statistical relational artificial intelligence: Logic, probability, and computation". In: *Synthesis lectures on artificial intelligence and machine learning*.

De Raedt, L. and A. Kimmig (2015). "Probabilistic (logic) programming concepts". In: *Machine Learning* 100, pp. 5–47.

De Raedt, L., A. Kimmig, and H. Toivonen (2007). "ProbLog: A Probabilistic Prolog and Its Application in Link Discovery." In: *IJCAI*. Hyderabad.

De Smet, L., E. Sansone, and P. Zuidberg Dos Martires (2023). "Differentiable Sampling of Categorical Distributions Using the CatLog-Derivative Trick". In: *NeurIPS*.

De Smet, L., P. Zuidberg Dos Martires, R. Manhaeve, G. Marra, A. Kimmig, and L. De Readt (2023). "Neural Probabilistic Logic Programming in Discrete-Continuous Domains". In: *UAI*.

De Smet*, L., G. Venturato*, L. De Raedt, and G. Marra (2024). *Neurosymbolic Markov Models*. Workshop Paper. Workshop on Structured Probabilistic Inference & Generative Modeling @ ICML 2024.

– (2025). "Relational neurosymbolic Markov models". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 39. 15, pp. 16181–16189.

De Smet*, L., G. Venturato*, G. Marra, and L. De Raedt (2024). *Neurosymbolic Reinforcement Learning With Sequential Guarantees*. Extended Abstract. Joint International Scientific Conferences on AI and Machine Learning (BNAIC/BeNeLearn 2024).

Dean, T. and K. Kanazawa (1989). "A Model for Reasoning about Persistence and Causation". In: *Computational Intelligence* 5.2, pp. 142–150. ISSN: 1467-8640.

Debot*, D., G. Venturato*, G. Marra, and L. De Raedt (2025). "Neurosymbolic Reinforcement Learning: Playing MiniHack With Probabilistic Logic Shields". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 39. 28, pp. 29631–29633.

Dechter, R. (1999). "Bucket elimination: A unifying framework for reasoning". In: *Artificial Intelligence* 113.1-2, pp. 41–85.

Delfosse, Q., J. Blüml, B. Gregori, S. Sztwiertnia, and K. Kersting (2023). "Ocatari: Object-centric atari 2600 reinforcement learning environments". In: *arXiv preprint arXiv:2306.08649*.

Demopoulos, D. D. and M. Y. Vardi (2006). "The phase transition in the random HornSAT problem". In: *Computational Complexity and Statistical Physics*. Oxford University Press.

Derkinderen, V. and L. De Raedt (2020). "Algebraic circuits for decision theoretic inference and learning". In: *ECAI 2020*. Vol. 325. IOS Press.

Ding, J., A. Sly, and N. Sun (2015). "Proof of the satisfiability conjecture for large k". In: *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pp. 59–68.

Dinh, L., J. Sohl-Dickstein, and S. Bengio (2016). "Density estimation using real nvp". In: *arXiv preprint arXiv:1605.08803*.

Dosovitskiy, A., G. Ros, F. Codevilla, A. Lopez, and V. Koltun (2017). "CARLA: An Open Urban Driving Simulator". In: *Proceedings of the 1st Annual Conference on Robot Learning*, pp. 1–16.

Dudek, J. M., A. A. Shrotri, and M. Y. Vardi (2022). "DPSampler: Exact Weighted Sampling Using Dynamic Programming". In: *Proc. 31st IJCAI*.

Eisner, J. (2002). "Parameter estimation for probabilistic finite-state transducers". In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pp. 1–8.

Fan, K., Z. Bai, T. Xiao, T. He, M. Horn, Y. Fu, F. Locatello, and Z. Zhang (2024). "Adaptive slot attention: Object discovery with dynamic slot number". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 23062–23071.

Feigenbaum, E., B. Buchanan, and J. Lederberg (Sept. 1970). "On generality and problem solving: A case study using the DENDRAL program". In: *Machine Intelligence* 6.

Fikes, R. E. and N. J. Nilsson (1971). "STRIPS: A new approach to the application of theorem proving to problem solving". In: *Artificial intelligence* 2.3-4, pp. 189–208.

Gallier, J. H. (1985). *Logic for computer science: foundations of automatic theorem proving*. Harper & Row Publishers, Inc. ISBN: 0060422254.

Garcez, A. d. and L. C. Lamb (2023). "Neurosymbolic AI: the 3rd wave". In: *Artificial Intelligence Review* 56.11, pp. 12387–12406. DOI: 10.1007/s10462-023-10448-w. URL: https://doi.org/10.1007/s10462-023-10448-w.

Garcıa, J. and F. Fernández (2015). "A comprehensive survey on safe reinforcement learning". In: *Journal of Machine Learning Research* 16.1, pp. 1437–1480.

Garey, M. R. and D. S. Johnson (1990). *Computers and Intractability; A Guide to the Theory of NP-Completeness*. USA: W. H. Freeman & Co. ISBN: 0716710455.

Geatti, L., N. Gigante, A. Montanari, and G. Venturato (2021). "Past matters: Supporting LTL+ Past in the BLACK satisfiability checker". In: *28th International Symposium on Temporal Representation and Reasoning (TIME 2021)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, pp. 8–1.

– (2024). “SAT meets tableaux for linear temporal logic satisfiability”. In: *Journal of Automated Reasoning* 68.2, p. 6.

Genesereth, M. R. and N. J. Nilsson (1987). *Logical foundations of artificial intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN: 0934613311.

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2020). “Generative adversarial networks”. In: *Communications of the ACM* 63.11, pp. 139–144.

Gordon, N. J., D. J. Salmond, and A. F. Smith (1993). “Novel approach to nonlinear/non-Gaussian Bayesian state estimation”. In: *IEE proceedings F (radar and signal processing)*. Vol. 140. 2. IET, pp. 107–113.

Guestrin, C., D. Koller, R. Parr, and S. Venkataraman (Oct. 2003). “Efficient Solution Algorithms for Factored MDPs”. In: *Journal of Artificial Intelligence Research* 19, pp. 399–468. ISSN: 1076-9757. DOI: 10.1613/jair.1000.

Gutmann, B., A. Kimmig, K. Kersting, and L. De Raedt (2008). “Parameter Learning in Probabilistic Databases: A Least Squares Approach”. In: *Machine Learning and Knowledge Discovery in Databases. ECML/PKDD. Lecture Notes in Computer Science*. Vol. 5211. Springer, pp. 473–488. DOI: 10.1007/978-3-540-87479-9_49.

Handschin, J. E. and D. Q. Mayne (1969). “Monte Carlo techniques to estimate the conditional expectation in multi-stage non-linear filtering”. In: *International journal of control* 9.5, pp. 547–559.

Hayes, L., P. Doshi, S. Pawar, and H. T. Tatavarti (2021). “State-Based Recurrent SPMNs for Decision-Theoretic Planning under Partial Observability”. In: *Proc. 30th IJCAI*.

Hazra, R., A. Sygkounas, A. Persson, A. Loutfi, and P. Z. D. Martires (2024). *REvolve: Reward Evolution with Large Language Models for Autonomous Driving*. arXiv: 2406.01309 [cs.NE]. URL: https://arxiv.org/abs/2406.01309.

Hazra, R., G. Venturato, P. Z. D. Martires, and L. De Raedt (2025a). *Can Large Language Models Reason? A Characterization Via 3-SAT*. Workshop Paper. Reasoning and Planning for LLMs @ ICLR 2025.

– (2025b). “Have Large Language Models Learned to Reason? A Characterization via 3-SAT”. In: *Second Conference on Language Modeling, COLM 2025*.

Hazra, R., P. Zuidberg Dos Martires, and L. De Raedt (2024). “SayCanPay: Heuristic Planning with Large Language Models using Learnable Domain Knowledge”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 38, pp. 20123–20133.

Heß, T., C. Sundermann, and T. Thüm (2021). “On the scalability of building binary decision diagrams for current feature models”. In: *SPLC (A)*.

Ho, J., A. Jain, and P. Abbeel (2020). "Denoising diffusion probabilistic models". In: *Advances in neural information processing systems* 33, pp. 6840–6851.

Hoey, J., R. St-Aubin, A. Hu, and C. Boutilier (July 1999). "SPUDD: Stochastic Planning Using Decision Diagrams". In: *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pp. 279–288.

Holtzen, S., G. Van den Broeck, and T. Millstein (2020). "Scaling exact inference for discrete probabilistic programs". In: *Proceedings of the ACM on Programming Languages* 4.OOPSLA, pp. 1–31.

Howard, R. A. and J. E. Matheson (1984). "Influence Diagrams". In: *Readings on Principles and Applications of Decision Analysis*, pp. 721–762.

Huang, J., Z. Li, B. Chen, K. Samel, M. Naik, L. Song, and X. Si (2021). "Scallop: From probabilistic deductive databases to scalable differentiable reasoning". In: *NeurIPS*.

Huang, W., P. Abbeel, D. Pathak, and I. Mordatch (2022). "Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents". In: *Proceedings of the 39th International Conference on Machine Learning*. Vol. 162, pp. 9118–9147.

Hummel, J. E. and K. J. Holyoak (2003). "A symbolic-connectionist theory of relational inference and generalization." In: *Psychological review* 110.2, p. 220.

Hunt, N., N. Fulton, S. Magliacane, T. N. Hoang, S. Das, and A. Solar-Lezama (2021). "Verifiably safe exploration for end-to-end reinforcement learning". In: *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control*, pp. 1–11.

Jansen, N., B. Könighofer, S. Junges, A. Serban, and R. Bloem (2020). "Safe Reinforcement Learning Using Probabilistic Shields". In: *31st International Conference on Concurrency Theory (CONCUR 2020)*.

Jiang, A. Q., S. Welleck, J. P. Zhou, T. Lacroix, J. Liu, W. Li, M. Jamnik, G. Lample, and Y. Wu (2023). "Draft, Sketch, and Prove: Guiding Formal Theorem Provers with Informal Proofs". In: *The Eleventh International Conference on Learning Representations*. URL: https://openreview.net/forum?id=SMa9EAovKMC.

Juang, B. H. and L. R. Rabiner (1991). "Hidden Markov models for speech recognition". In: *Technometrics* 33.3, pp. 251–272.

Jurafsky, D. and J. H. Martin (2025). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics,and Speech Recognition with Language Models*. 3rd. Online manuscript released January 12, 2025. URL: https://web.stanford.edu/~jurafsky/slp3/.

Kaelbling, L. P., M. L. Littman, and A. R. Cassandra (1998). "Planning and acting in partially observable stochastic domains". In: *Artificial intelligence* 101.1-2, pp. 99–134.

Kanazawa, K. and T. Dean (Aug. 1989). "A Model for Projection and Action". In: *Proc. 11th IJCAI*, pp. 985–990.

Khiatani, D. and U. Ghose (2017). "Weather forecasting using hidden Markov model". In: *2017 International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN)*. IEEE, pp. 220–225.

Kimmig, A., G. Van den Broeck, and L. De Raedt (July 2017). "Algebraic Model Counting". In: *Journal of Applied Logic*. SI:Uncertain Reasoning 22, pp. 46–62. ISSN: 1570-8683. DOI: 10.1016/j.jal.2016.11.031.

Kingma, D. P. and M. Welling (2013). "Auto-encoding variational bayes". In: *arXiv preprint arXiv:1312.6114*.

Kingma, D. P. and J. Ba (2015). "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR*.

Kiran, B. R., I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez (2021). "Deep reinforcement learning for autonomous driving: A survey". In: *IEEE Transactions on Intelligent Transportation Systems* 23.6, pp. 4909–4926.

Kisa, D., G. Van den Broeck, A. Choi, and A. Darwiche (2014). "Probabilistic sentential decision diagrams". In: *Fourteenth International Conference on the Principles of Knowledge Representation and Reasoning*.

Kokel, H., N. Prabhakar, B. Ravindran, E. Blasch, P. Tadepalli, and S. Natarajan (2022). "Hybrid deep reprel: Integrating relational planning and reinforcement learning for information fusion". In: *2022 25th International Conference on Information Fusion (FUSION)*. IEEE, pp. 1–8.

Koller, D. and N. Friedman (2009). *Probabilistic graphical models: principles and techniques*. MIT press.

Kool, W., H. van Hoof, and M. Welling (2019). "Buy 4 reinforce samples, get a baseline for free!" In: *ICLR Deep RL Meets Structured Prediction Workshop*.

Krieken, E. van, T. Thanapalasingam, J. Tomczak, F. Van Harmelen, and A. Ten Teije (2024). "A-nesi: A scalable approximate method for probabilistic neurosymbolic inference". In: *Advances in Neural Information Processing Systems* 36.

Krishnan, R., U. Shalit, and D. Sontag (2017). "Structured inference networks for nonlinear state space models". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 31.1.

Küttler, H., N. Nardelli, A. Miller, R. Raileanu, M. Selvatici, E. Grefenstette, and T. Rocktäschel (2020). "The nethack learning environment". In: *Advances in Neural Information Processing Systems* 33, pp. 7671–7684.

Lam, S. K., A. Pitrou, and S. Seibert (2015). "Numba: a LLVM-based Python JIT compiler". In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pp. 1–6.

Li, Z., H. Liu, D. Zhou, and T. Ma (2024). "Chain of Thought Empowers Transformers to Solve Inherently Serial Problems". In: *The Twelfth*

*International Conference on Learning Representations*. URL: https://openreview.net/forum?id=3EWTEy9MTM.

Lightman, H., V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever, and K. Cobbe (2024). "Let's Verify Step by Step". In: *The Twelfth International Conference on Learning Representations*. URL: https://openreview.net/forum?id=v8L0pN6EOi.

Liu, R., J. Regier, N. Tripuraneni, M. Jordan, and J. Mcauliffe (2019). "Rao-Blackwellized stochastic gradients for discrete distributions". In: *ICML*.

Lloyd, J. W. (1984). *Foundations of logic programming*. Berlin, Heidelberg: Springer-Verlag. ISBN: 0387132996.

Locatello, F., D. Weissenborn, T. Unterthiner, A. Mahendran, G. Heigold, J. Uszkoreit, A. Dosovitskiy, and T. Kipf (2020). "Object-centric learning with slot attention". In: *Advances in neural information processing systems* 33, pp. 11525–11538.

Madaan, A., N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang, S. Gupta, B. P. Majumder, K. Hermann, S. Welleck, A. Yazdanbakhsh, and P. Clark (2023). "Self-Refine: Iterative Refinement with Self-Feedback". In: *Advances in Neural Information Processing Systems*. Vol. 36, pp. 46534–46594. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/91edff07232fb1b55a505a9e9f6c0ff3-Paper-Conference.pdf.

Maene, J., V. Derkinderen, and P. Z. Dos Martires (n.d.). "Klay: Accelerating Arithmetic Circuits for Neurosymbolic AI". In: *The Thirteenth International Conference on Learning Representations*.

Manhaeve, R., S. Dumancic, A. Kimmig, T. Demeester, and L. De Raedt (2018). "DeepProbLog: Neural probabilistic logic programming". In: *NeurIPS* 31.

Manhaeve, R., S. Dumančić, A. Kimmig, T. Demeester, and L. De Raedt (2021). "Neural probabilistic logic programming in DeepProbLog". In: *Artificial Intelligence*.

Marra, G., S. Dumančić, R. Manhaeve, and L. De Raedt (2024). "From Statistical Relational to Neurosymbolic Artificial Intelligence: a Survey". In: *Artificial Intelligence*, p. 104062.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng (2015). *TensorFlow: Large-Scale Machine*

*Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: https://www.tensorflow.org/.

Merrill, W. and A. Sabharwal (2023). "The Parallelism Tradeoff: Limitations of Log-Precision Transformers". In: *Transactions of the Association for Computational Linguistics* 11, pp. 531–545. DOI: 10.1162/tacl_a_00562. URL: https://aclanthology.org/2023.tacl-1.31.

Mertens, S., M. Mézard, and R. Zecchina (2006). "Threshold values of random K-SAT from the cavity method". In: *Random Structures & Algorithms* 28.3, pp. 340–373.

Misino, E., G. Marra, and E. Sansone (2022). "VAEL: Bridging Variational Autoencoders and Probabilistic Logic Programming". In: *NeurIPS*.

Mnih, V., A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu (2016). "Asynchronous methods for deep reinforcement learning". In: *International conference on machine learning*. PmLR, pp. 1928–1937.

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. (2015). "Human-level control through deep reinforcement learning". In: *nature* 518.7540, pp. 529–533.

Mor, B., S. Garhwal, and A. Kumar (2020). "A Systematic Review of Hidden Markov Models and Their Applications". In: *Archives of Computational Methods in Engineering* 28, pp. 1429–1448.

Moreira, D. A. M., K. V. Delgado, L. N. de Barros, and D. D. Mauá (2021). "Efficient algorithms for Risk-Sensitive Markov Decision Processes with limited budget". In: *Int. J. Approx. Reason.* 139.

Morettin, P., P. Zuidberg Dos Martires, S. Kolb, and A. Passerini (2021). "Hybrid Probabilistic Inference with Logical and Algebraic Constraints: a Survey." In: *IJCAI*.

Mundhenk, M., J. Goldsmith, C. Lusena, and E. Allender (2000). "Complexity of finite-horizon Markov decision process problems". In: *Journal of the ACM (JACM)* 47.4, pp. 681–720.

Murphy, K. and S. Russell (2001). "Rao-Blackwellised particle filtering for dynamic Bayesian networks". In: *Sequential Monte Carlo methods in practice*. Springer, pp. 499–515.

Murphy, K. P. (2002). "Dynamic Bayesian Networks: Representation, Inference and Learning". PhD thesis. University of California, Berkeley.

Ng, A. Y., D. Harada, and S. J. Russell (1999). "Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping". In: *Proceedings of the Sixteenth International Conference on Machine Learning*. ICML '99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., pp. 278–287. ISBN: 1558606122.

Nitti, D., T. De Laet, and L. De Raedt (2016). "Probabilistic logic programming for hybrid relational domains". In: *Machine Learning*.

Papadimitriou, C. H. and J. N. Tsitsiklis (1987). "The complexity of Markov decision processes". In: *Mathematics of operations research* 12.3, pp. 441–450.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan kaufmann.

Peng, B., S. Narayanan, and C. Papadimitriou (2024). "On Limitations of the Transformer Architecture". In: *arXiv* 2402.08164. URL: https://api.semanticscholar.org/CorpusID:267636545.

Penning, L. de, A. Garcez, L. C. Lamb, and J. Meyer (2011). "A neural-symbolic cognitive agent for online learning and reasoning". In: *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence*. Vol. 2. International Joint Conferences on Artificial Intelligence, pp. 1653–1658.

Pineau, J., G. Gordon, S. Thrun, et al. (2003). "Point-based value iteration: An anytime algorithm for POMDPs". In: *IJCAI*. Vol. 3, pp. 1025–1032.

Pineau, J., G. Gordon, and S. Thrun (2006). "Anytime point-based approximations for large POMDPs". In: *Journal of Artificial Intelligence Research* 27, pp. 335–380.

Poole, D. (1993). "Probabilistic Horn abduction and Bayesian networks". In: *Artificial intelligence* 64.1, pp. 81–129.

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. 1st. USA: John Wiley & Sons, Inc. ISBN: 0471619779.

Raffin, A., A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann (2021). "Stable-Baselines3: Reliable Reinforcement Learning Implementations". In: *Journal of Machine Learning Research* 22.268, pp. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

Redmon, J., S. Divvala, R. Girshick, and A. Farhadi (2016). "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788.

Reichstein, M., G. Camps-Valls, B. Stevens, M. Jung, J. Denzler, N. Carvalhais, and F. Prabhat (2019). "Deep learning and process understanding for data-driven Earth system science". In: *Nature* 566.7743, pp. 195–204.

Romera-Paredes, B., M. Barekatain, A. Novikov, M. Balog, M. P. Kumar, E. Dupont, F. J. R. Ruiz, J. S. Ellenberg, P. Wang, O. Fawzi, P. Kohli, and A. Fawzi (Jan. 2024). "Mathematical discoveries from program search with large language models". In: *Nature* 625.7995, pp. 468–475. ISSN: 1476-4687. DOI: 10.1038/s41586-023-06924-6. URL: https://doi.org/10.1038/s41586-023-06924-6.

Roth, D. (1996). "On the hardness of approximate reasoning". In: *Artificial Intelligence* 82.1-2, pp. 273–302.

Roussel, O. (n.d.). *The SAT Game*. https://www.cril.univ-artois.fr/en/software/satgame/.

Russell, S. and P. Norvig (2020). *Artificial Intelligence: A Modern Approach*. 4th edition. Hoboken: Pearson. ISBN: 978-0-13-461099-3.

Saffidine, A., T. Cazenave, and J. Méhat (2012). "UCD: Upper Confidence bound for rooted Directed acyclic graphs". In: *Knowledge-Based Systems* 34, pp. 26–33.

Saldyt, L. and S. Kambhampati (2025). "Mind The Gap: Deep Learning Doesn't Learn Deeply". In: *arXiv preprint arXiv:2505.18623*.

Samvelyan, M., R. Kirk, V. Kurin, J. Parker-Holder, M. Jiang, E. Hambro, F. Petroni, H. Kuttler, E. Grefenstette, and T. Rocktäschel (2021). "MiniHack the Planet: A Sandbox for Open-Ended Reinforcement Learning Research". In: *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.

Schick, T., J. Dwivedi-Yu, R. Dessi, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom (2023). "Toolformer: Language Models Can Teach Themselves to Use Tools". In: *Advances in Neural Information Processing Systems*. Vol. 36, pp. 68539–68551. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/d842425e4bf79ba039352da0f658a906-Paper-Conference.pdf.

Schrittwieser, J., I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, et al. (2020). "Mastering atari, go, chess and shogi by planning with a learned model". In: *Nature* 588.7839, pp. 604–609.

Schulman, J., S. Levine, P. Abbeel, M. Jordan, and P. Moritz (July 2015). "Trust Region Policy Optimization". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by F. Bach and D. Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, pp. 1889–1897. URL: https://proceedings.mlr.press/v37/schulman15.html.

Schulman, J., P. Moritz, S. Levine, M. Jordan, and P. Abbeel (2018). "High-dimensional continuous control using generalized advantage estimation". In:

Schulman, J., F. Wolski, P. Dhariwal, A. Radford, and O. Klimov (2017). "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347*.

Ścibior, A., V. Masrani, and F. Wood (2021). "Differentiable Particle Filtering without Modifying the Forward Pass". In: *International Conference on Probabilistic Programming (PROBPROG)*. arXiv: 2106.10314.

Selman, B., D. G. Mitchell, and H. J. Levesque (1996). "Generating hard satisfiability problems". In: *Artificial intelligence* 81.1-2, pp. 17–29.

Shaw, P., J. Cohan, J. Eisenstein, K. Lee, J. Berant, and K. Toutanova (2024). "ALTA: Compiler-Based Analysis of Transformers". In: *arXiv preprint arXiv:2410.18077*.

Shindo, H., Q. Delfosse, D. S. Dhami, and K. Kersting (2024). "BlendRL: A Framework for Merging Symbolic and Neural Policy Learning". In: *arXiv preprint arXiv:2410.11689*.

Shojaee, P., I. Mirzadeh, K. Alizadeh, M. Horton, S. Bengio, and M. Farajtabar (2025). *The Illusion of Thinking: Understanding the Strengths and Limitations of Reasoning Models via the Lens of Problem Complexity*. URL: https://ml-site.cdn-apple.com/papers/the-illusion-of-thinking.pdf.

Silva, J. M. and K. A. Sakallah (1996). "GRASP-a new search algorithm for satisfiability". In: *Proceedings of International Conference on Computer Aided Design*. IEEE, pp. 220–227.

Silver, D. and J. Veness (2010). "Monte-Carlo planning in large POMDPs". In: *Advances in neural information processing systems* 23.

Smallwood, J. E. and E. J. Sondik (1973). "The optimal control of partially observable Markov processes over a finite horizon". In: *Operations Research* 21.5, pp. 1071–1088.

Smith, T. and R. Simmons (2012). "Heuristic search value iteration for POMDPs". In: *arXiv preprint arXiv:1207.4166*.

Snell, C. V., J. Lee, K. Xu, and A. Kumar (2025). "Scaling LLM Test-Time Compute Optimally Can be More Effective than Scaling Parameters for Reasoning". In: *The Thirteenth International Conference on Learning Representations*. URL: https://openreview.net/forum?id=4FWAwZtd2n.

Sterling, L. and E. Shapiro (1986). *The Art of Prolog*. Cambridge, MA, USA: MIT Press. ISBN: 0262192500.

Sutton, R. S. (1988). "Learning to predict by the methods of temporal differences". In: *Machine learning* 3, pp. 9–44.

Sutton, R. S. and A. G. Barto (Nov. 2018). *Reinforcement Learning: An Introduction*. 2nd ed. Adaptive Computation and Machine Learning Series. Cambridge, MA, USA: A Bradford Book. ISBN: 978-0-262-03924-6.

Svoreňová, M., M. Chmelík, K. Leahy, H. F. Eniser, K. Chatterjee, I. Černá, and C. Belta (2015). "Temporal logic motion planning using POMDPs with parity objectives: Case study paper". In: *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, pp. 233–238.

Szeliski, R. (2022). *Computer vision: algorithms and applications*. Springer Nature.

Tan, A. H. and G. Nejat (2022). "Enhancing Robot Task Completion Through Environment and Task Inference: A Survey from the Mobile Robot Perspective". In: *J. Intell. Robotic Syst.* 106.4.

Tatavarti, H., P. Doshi, and L. Hayes (May 2021). "Data-Driven Decision-Theoretic Planning Using Recurrent Sum-Product-Max Networks". In: *Proceedings of the International Conference on Automated Planning and Scheduling* 31. ISSN: 2334-0843.

Theeuwes, E., G. Venturato, and G. Rens (2025). "Belief Re-Use in Partially Observable Monte Carlo Tree Search". In: *Proceedings of the 17th International Conference on Agents and Artificial Intelligence, ICAART 2025 - Volume 2, Porto, Portugal, February 23-25, 2025*. Ed. by A. P. Rocha, L. Steels, and H. J. van den Herik. SCITEPRESS, pp. 634–645.

Towell, G. G. and J. W. Shavlik (1994). "Knowledge-based artificial neural networks". In: *Artificial intelligence* 70.1-2, pp. 119–165.

Tran, S. N. and A. d. Garcez (2023). "Neurosymbolic reasoning and learning with restricted boltzmann machines". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 37, pp. 6558–6565.

Tran, S. N. (2021). "Compositional Neural Logic Programming". In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence, pp. 3059–3066.

Tsamoura, E., D. Carral, E. Malizia, and J. Urbani (2021). "Materializing knowledge bases via trigger graphs". In: *Proceedings of the VLDB Endowment* 14.6, pp. 943–956.

Umili, E., F. Argenziano, and R. Capobianco (2024). "Neural Reward Machines". In: *ECAI 2024*. Ios Press, pp. 3055–3062.

Valiant, L. G. (1979). "The Complexity of Enumeration and Reliability Problems". In: *SIAM Journal on Computing* 8.3. DOI: 10.1137/0208032.

Van Roy, M., P. Robberechts, W.-C. Yang, L. De Raedt, and J. Davis (2023). "A Markov framework for learning and reasoning about strategies in professional soccer". In: *Journal of Artificial Intelligence Research* 77, pp. 517–562.

Venturato, G., V. Derkinderen, P. Zuidberg Dos Martires, and L. De Raedt (2022). *Towards Tractable Dynamic Decision Making With Circuits*. Workshop Paper. 5th Workshop on Tractable Probabilistic Modeling @ UAI 2022.

– (2024). "Inference and learning in dynamic decision networks using knowledge compilation". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 38. 18, pp. 20567–20576.

Vlasselaer, J., W. Meert, G. Van den Broeck, and L. De Raedt (Mar. 2016). "Exploiting Local and Repeated Structure in Dynamic Bayesian Networks". In: *Artificial Intelligence* 232. ISSN: 0004-3702. DOI: 10.1016/j.artint.2015.12.001.

Wang, Z., C. R. Garrett, L. P. Kaelbling, and T. Lozano-Pérez (2021). "Learning compositional models of robot skills for task and motion planning". In: *The International Journal of Robotics Research* 40.6-7, pp. 866–894.

Watkins, C. J. and P. Dayan (1992). "Q-learning". In: *Machine learning* 8, pp. 279–292.

Wei, J., X. Wang, D. Schuurmans, M. Bosma, b. ichter brian, F. Xia, E. Chi, Q. V. Le, and D. Zhou (2022). "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models". In: *Advances in Neural Information Processing Systems*. Vol. 35, pp. 24824–24837. URL: https:

//proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf.

Welleck, S., J. Liu, X. Lu, H. Hajishirzi, and Y. Choi (2022). "NaturalProver: Grounded Mathematical Proof Generation with Language Models". In: *Advances in Neural Information Processing Systems*. Vol. 35, pp. 4913–4927. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/1fc548a8243ad06616eee731e0572927-Paper-Conference.pdf.

Williams, R. J. (1992). "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning*.

Yang, C., X. Wang, Y. Lu, H. Liu, Q. V. Le, D. Zhou, and X. Chen (2024). "Large Language Models as Optimizers". In: *The Twelfth International Conference on Learning Representations*. URL: https://openreview.net/forum?id=Bb4VGOWELI.

Yang, F., Z. Yang, and W. W. Cohen (2017). "Differentiable learning of logical rules for knowledge base reasoning". In: *Advances in neural information processing systems* 30.

Yang, K., J. Deng, and D. Chen (Dec. 2022). "Generating Natural Language Proofs with Verifier-Guided Search". In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pp. 89–105. DOI: 10.18653/v1/2022.emnlp-main.7. URL: https://aclanthology.org/2022.emnlp-main.7.

Yang, W.-C., G. Marra, G. Rens, and L. De Raedt (Aug. 2023). "Safe Reinforcement Learning via Probabilistic Logic Shields". In: *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*. Ed. by E. Elkind. Main Track. International Joint Conferences on Artificial Intelligence Organization, pp. 5739–5749. DOI: 10.24963/ijcai.2023/637.

Yang, Z., A. Ishay, and J. Lee (2020). "Neurasp: Embracing neural networks into answer set programming". In: *IJCAI*.

Yao, S., D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, and K. Narasimhan (2023). "Tree of Thoughts: Deliberate Problem Solving with Large Language Models". In: *Advances in Neural Information Processing Systems*. Vol. 36, pp. 11809–11822. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/271db9922b8d1f4dd7aaef84ed5ac703-Paper-Conference.pdf.

Yao, S., J. Zhao, D. Yu, N. Du, I. Shafran, K. R. Narasimhan, and Y. Cao (2023). "ReAct: Synergizing Reasoning and Acting in Language Models". In: *The Eleventh International Conference on Learning Representations*. URL: https://openreview.net/forum?id=WE_vluYUL-X.

Younis, A. and E. Sudderth (2024). "Learning to be Smooth: An End-to-End Differentiable Particle Smoother". In: *Advances in Neural Information Processing Systems* 37, pp. 7125–7155.

Younis, A. and E. B. Sudderth (2023). "Differentiable and Stable Long-Range Tracking of Multiple Posterior Modes". In: *Thirty-seventh Conference on Neural Information Processing Systems*.

Zhang, H., M. Dang, N. Peng, and G. Van Den Broeck (July 2023). "Tractable Control for Autoregressive Language Generation". In: *Proceedings of the 40th International Conference on Machine Learning*. Ed. by A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett. Vol. 202. Proceedings of Machine Learning Research. PMLR, pp. 40932–40945.

Zhang, H., L. H. Li, T. Meng, K.-W. Chang, and G. Van den Broeck (Aug. 2023). "On the Paradox of Learning to Reason from Data". In: *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pp. 3365–3373. DOI: 10.24963/ijcai.2023/375. URL: https://doi.org/10.24963/ijcai.2023/375.

Zhang, Y., D. W. Zhang, S. Lacoste-Julien, G. J. Burghouts, and C. G. Snoek (2023). "Unlocking slot attention by changing optimal transport costs". In: *International Conference on Machine Learning*. PMLR, pp. 41931–41951.

# Statement on the use of Generative AI

I did use the generative AI assistance tools OpenAI ChatGPT and MS Copilot. Generative AI was used for generating programming code. Specifically, I used MS Copilot to assist me in writing code for the implementation of the algorithms and models described in this thesis. Generative AI was used to generate visuals. Specifically, I used OpenAI ChatGPT to create the robot illustration on the (front and back) cover page of this thesis, as well as the robot and the driving car perspective image of Figure 1.1.
The text, code, and images in this thesis are my own (unless otherwise specified). Generative AI has only been used in accordance with the KU Leuven guidelines and appropriate references have been added. I have reviewed and edited the content as needed and I take full responsibility for the content of the thesis.

# Curriculum Vitae

Gabriele Venturato was born in Venice, Italy, on September 15th, 1994. He obtained a Bachelor's degree in Computer Science in 2017 from the University of Udine, where he was awarded the distinction of *Best Computer Science Bachelor Student Graduated in 2016–2017*. In 2020, he completed a Master's degree in Computer Science at the same university, with a specialisation in automated reasoning. His thesis focused on temporal logic and formal methods. During his studies, in 2018, he completed an internship at the NATO Communications and Information Agency, where he investigated AI-driven methods for semantic similarity across interoperability standards.

In 2021, he began a PhD in Artificial Intelligence at KU Leuven under the supervision of Prof. Luc De Raedt, within the DTAI (Declarative Languages and Artificial Intelligence) research group. His research spans neurosymbolic AI, probabilistic reasoning, and reinforcement learning, with an emphasis on safety and sequential decision-making. His work has been presented at leading AI conferences and was recognized with the *Best Technical Demonstration Award* at AAAI 2025 for the demo *"Neurosymbolic Reinforcement Learning: Playing MiniHack With Probabilistic Logic Shields"*.

# List of publications

## Jorunal Articles

L. Geatti, N. Gigante, A. Montanari, and G. Venturato (2024). "SAT meets tableaux for linear temporal logic satisfiability". In: *Journal of Automated Reasoning* 68.2, p. 6

## Conference Proceedings

R. Hazra, G. Venturato, P. Z. D. Martires, and L. De Raedt (2025b). "Have Large Language Models Learned to Reason? A Characterization via 3-SAT". in: *Second Conference on Language Modeling, COLM 2025*

D. Debot*, G. Venturato*, G. Marra, and L. De Raedt (2025). "Neurosymbolic Reinforcement Learning: Playing MiniHack With Probabilistic Logic Shields". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 39. 28, pp. 29631–29633

L. De Smet*, G. Venturato*, L. De Raedt, and G. Marra (2025). "Relational neurosymbolic Markov models". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 39. 15, pp. 16181–16189

E. Theeuwes, G. Venturato, and G. Rens (2025). "Belief Re-Use in Partially Observable Monte Carlo Tree Search". In: *Proceedings of the 17th International Conference on Agents and Artificial Intelligence, ICAART 2025 - Volume 2, Porto, Portugal, February 23-25, 2025*. Ed. by A. P. Rocha, L. Steels, and H. J. van den Herik. SCITEPRESS, pp. 634–645

G. Venturato, V. Derkinderen, P. Zuidberg Dos Martires, and L. De Raedt (2024). "Inference and learning in dynamic decision networks using knowledge

compilation". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 38. 18, pp. 20567–20576

L. Geatti, N. Gigante, A. Montanari, and G. Venturato (2021). "Past matters: Supporting LTL+ Past in the BLACK satisfiability checker". In: *28th International Symposium on Temporal Representation and Reasoning (TIME 2021)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, pp. 8–1

## Workshop Papers and Extended Abstracts

R. Hazra, G. Venturato, P. Z. D. Martires, and L. De Raedt (2025a). *Can Large Language Models Reason? A Characterization Via 3-SAT*. Workshop Paper. Reasoning and Planning for LLMs @ ICLR 2025

L. De Smet*, G. Venturato*, L. De Raedt, and G. Marra (2024). *Neurosymbolic Markov Models*. Workshop Paper. Workshop on Structured Probabilistic Inference & Generative Modeling @ ICML 2024

L. De Smet*, G. Venturato*, G. Marra, and L. De Raedt (2024). *Neurosymbolic Reinforcement Learning With Sequential Guarantees*. Extended Abstract. Joint International Scientific Conferences on AI and Machine Learning (BNAIC/BeNeLearn 2024)

G. Venturato, V. Derkinderen, P. Zuidberg Dos Martires, and L. De Raedt (2022). *Towards Tractable Dynamic Decision Making With Circuits*. Workshop Paper. 5th Workshop on Tractable Probabilistic Modeling @ UAI 2022

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
Celestijnenlaan 200A box 2402
3001 Leuven
gabriele.venturato@kuleuven.be
https://gabventurato.github.io