NON-AXIOMATIC REASONING AND CAUSAL EXPLORATIVE SYSTEM (NARCES), A HYBRID SYMBOLIC REASONING WITH SUB-SYMBOLIC LEARNING FOR AUTONOMOUS INTELLIGENT AGENTS

A Dissertation
Submitted to
the Temple University Graduate Board

In Partial Fulfillment
of the Requirements for the Degree
DOCTOR OF PHILOSOPHY

---

by
Peter Isaev
August 2025

Examining Committee Members:

Justin Shi, Advisory Chair, Computer and Information Sciences
Pei Wang, Computer and Information Sciences
Slobodan Vucetic, Computer and Information Sciences
Antonio Chella, External Reader, Universtiy of Palermo

ABSTRACT

Non-Axiomatic Reasoning and Causal Explorative System (NARCES), a Hybrid Symbolic Reasoning with Sub-symbolic Learning for Autonomous Intelligent Agents

by

Peter Isaev

In Artificial Intelligence, reasoning can be distinguished into symbolic and sub-symbolic. Symbolic reasoning makes use of formal language for semantic knowledge representations, as well as utilizes logic for inference, and system control for processing input and derived knowledge. The paradigm examples of sub-symbolic reasoning are biological and artificial neural networks. Given the signals from sensory neurons, our nervous system identifies whether a particular flying object is a dangerous wasp or just a harmless fly. However, nowhere during the process 'symbols' do occur, instead, there are only neurons, synapses, electrical signals, etc. Symbols such as 'black', 'wasp', or 'dangerous' emerge as high-level descriptions of the process at best.

Recently the focus has been shifted toward combining the two approaches to provide hybrid solutions for practical real-time reasoning, thereby we present Non-Axiomatic Reasoning and Causal Explorative System (NARCES), a novel integrated multi-component system for autonomous intelligent agents, which consists of two main parts: MeTTa-NARS and NACE. MeTTa-NARS is the advanced stand-alone symbolic reasoning system that follows NARS model of intelligence. MeTTa-NARS employs a concept-centered semantic knowledge representation, Non-Axiomatic Logic with Narsese formal language and sophisticated control, all of which make it capable of executing various forms of reasoning including deductive, inductive, and abductive reasoning. It also handles uncertainty, revises its knowledge based on new evidence, and learns cause-effect relations from observation in real-time. The second component, Non-Axiomatic Causal Explorer (NACE), is an example of experiential learning,

which is a novel sub-symbolic system that showcases high data efficiency in learning and sequential decision-making tasks in grid world environments. NACE is developed with uncertainty in mind and thereby is driven by curiosity. It is capable of capturing logical causal relation induction from observations in real-time using some aspects of NAL as well as causal relation of transitions between parts of states. It iteratively learns an emerging behavior creating structures to allow more robust and efficient decision-making and planning and does not feature encodings of high-level semantic knowledge. Our comprehensive experiments show that NACE performs considerably more data efficient while reaching respectably high mean reward than most state-of-the-art Reinforcement Learning models in Minigrid environments.

Bringing two of the components into NARCES prototype, we chose to work toward a unified foundation of symbolic and sub-symbolic reasoning. It is clear, that we need semantics that add meaning to NACE's actions and cell values and at the same time MeTTa-NARS will benefit from efficient and robust sub-symbolic processing. Thereby with this framework in mind, we try to resolve a fundamentally important question: how can we obtain a combined system where semantics as in NARS and its high-level reasoning can be effectively interfaced with an array-based world representation for effective sensorimotor processing? This entails the use of a 2D array with values such as those obtained from a grid world, while the NAL representations will allow for human-provided background and derived knowledge to be incorporated and high-level semantic objectives to be pursued.

Furthermore, we showcase the Explorer, a robotic agent that leverages NARCES and operates within the complete ROS2 framework, incorporating simulated environments provided by the Gazebo robotics simulator. By employing industry-standard simulation tools and the ROS2 framework, the Explorer emerges as a promising system that is prepared for testing on actual robots and for deployment in real-world settings.

To those thinking about intelligence and those trying to create intelligent thinking agents, which are the greatest forms of the beautiful...

ACKNOWLEDGEMENTS

I am grateful to be part of the thrilling and compelling world of Artificial Intelligence. This unique field brings experts from diverse areas including computer science, mathematics, neuroscience, psychology, and philosophy, all working together having the central goal of developing a system capable of emulating human intelligence and performing tasks that typically require human cognitive functions. I feel deeply grateful to all the individuals who have inspired me, encouraged me to chase my goals, and supported my journey toward success. Their support has been invaluable as I navigated through a challenging, uncertain, and seemingly endless path toward conquering my doctorate degree in Artificial Intelligence.

I would like to express my deepest appreciation to Professor Pei Wang for including me in his AGI team and for his substantial contributions to the field of Artificial Intelligence. His remarkable NARS project, to which he has devoted much of his career, has been particularly inspiring. Pei's profound insights and understanding of current AI challenges have motivated me to advance significantly in the AI field. His guidance has been instrumental in my success during my PhD program and has helped me become a strong AI professional, actively shaping the future of AI in our communities.

I am profoundly thankful to the Artificial General Intelligence community, particularly to my friend Patrick Hammer and other key contributors like Tony Lofthouse, Nill Geisweiller, and Ben Goetzel, for their invaluable influence. Their unwavering motivation, innovative ideas for advancing AGI, and support for the NARS project have been crucial. Their influence has enabled me to overcome numerous challenges throughout this arduous journey, helping me achieve this significant academic milestone.

Furthermore, I extend my heartfelt thanks to Dr. Slobodan Vucetic, an exceptionally insightful professor at Temple University. His prolonged ability to follow NARS project and openness to fresh ideas and innovative systems beyond the traditional Machine Learning and Reinforcement Learning communities have been invaluable. Lastly, I am profoundly appreciate my immediate family, whose total support allowed me to allocate the time and resources necessary to reach this significant milestone in my career.

TABLE OF CONTENTS

Page

ABSTRACT ii

DEDICATION iv

ACKNOWLEDGEMENTS v

LIST OF FIGURES xi

LIST OF TABLES xiii

CHAPTER

1. INTRODUCTION 1

1.1 Motivation 1
1.2 Intelligence and Autonomy 1
1.3 Research Question 5

2. MeTTa-NARS: HIGH-LEVEL PRACTICAL REASONER 7

2.1 Introduction 7
2.2 Theoretical Basis 12
2.2.1 Truth Value and Expectation 12
2.2.2 Non-Axiomatic Logic 13
2.2.3 Inference Rules 15
2.2.4 Control Criteria and Attention Allocation Principles 16
2.2.5 Data Items 17
2.3 MeTTa-NARS Deign and Implementation 19
2.3.1 Logic Implementation in MeTTa-NARS 19
2.3.2 Overall Architecture 20
2.3.3 Main Memory 21
2.3.4 Memory Types 22
2.3.5 Data Structures and Corresponding Spaces 24
2.3.6 Inference Control 26
2.3.7 Declarative Inference 27
2.3.8 Temporal Inference 28
2.3.9 Procedural Inference 32
2.3.10 Attentional Control 37
2.4 Related Works 41

2.5 MeTTa-NARS Code Structure and Usage 46

3. COMPARISON WITH EXISTING NARS IMPLEMENTATIONS . . 51

3.1 OpenNARS v3 51
3.1.1 Data Structures 51
3.1.2 Data Items and Inference Cycle 52
3.1.3 Attentional Control 54
3.1.4 Goal Handling and Subgoaling 55
3.2 OpenNARS for Applications 55
3.2.1 ONA and OpenNARS 56
3.2.2 ONA and MeTTa-NARS 57
3.3 MeTTa-NARS Limitations 58

4. NON-AXIOMATIC CAUSAL EXPLORER: SUB-SYMBOLIC GRID-WORLD AGENT 61

4.1 Introduction 61
4.2 Related Work 63
4.2.1 Selected RL Techniques in Grid Worlds 64
4.3 Non-Axiomatic Causal Explorer 66
4.3.1 States and Rule Representation 66
4.3.2 Inductive Biases 67
4.3.3 Curiosity Model 68
4.3.4 NACE Architecture 70
4.4 Experiments in Minigrid 74
4.4.1 Stationary Environments 75
4.4.2 Dynamic Environments 77
4.4.3 Dynamic Environments with Sequential Dependencies 79
4.5 Discussions 80
4.6 Conclusion 84

5. TOWARD A UNIFIED FOUNDATION OF SYMBOLIC AND SUB-SYMBOLIC REASONING 85

5.1 Introduction 85
5.2 Related Works 86
5.3 Interlude: Symbolic vs Sub-symbolic 90
5.4 A Hybrid Approach to Reasoning 93
5.5 Architecture and Design Principles 97
5.5.1 Design Principles 97
5.5.2 Planning Objective for NACE 99
5.5.3 Bi-Directional Bridge 101
5.5.4 MeTTa-NARS to NACE 102
5.5.5 NACE to MeTTa-NARS 104

5.5.6 NACE Causal Rules and NARS Procedural Implications . . . 107
5.6 Capabilities and Limitations . . . 108
5.7 Explorer: Generalized Robotics Application . . . 111
5.7.1 Explorer Framework and Structure . . . 111
5.7.2 Semantic SLAM . . . 116
5.7.3 Hybrid Reasoning in Open World . . . 119
5.7.4 Future Works . . . 122

6. SELECT APPLICATIONS AND EXPERIMENTS . . . 125

6.1 Introduction . . . 125
6.2 Non-Axiomatic Reasoning for an Autonomous Mobile Robot . . . 126
6.2.1 Related Works . . . 128
6.2.2 ROS Integration . . . 129
6.2.3 Case Study: Garbage Collection . . . 131
6.2.4 Integration on Transbot . . . 132
6.2.5 Testing Conditions and Observations . . . 133
6.2.6 Final Test Setup and Results . . . 135
6.2.7 Comparison with BDI Model . . . 137
6.3 NARS-GPT: An integrated system for natural language interactions . . . 139
6.3.1 Related Works . . . 140
6.3.2 NARS-GPT Architecture . . . 141
6.3.3 Sentence Processing . . . 142
6.3.4 Question Processing . . . 143
6.3.5 Translation Process . . . 144
6.3.6 Answer Generation . . . 146
6.3.7 Real-time Reasoning Capabilities . . . 148
6.3.8 Belief Revision Mechanism . . . 151
6.3.9 Experiments and Evaluation . . . 153
6.3.10 Grounding Language in Event Streams . . . 154
6.3.11 Future Work . . . 156
6.4 Conclusion . . . 156

7. CONCLUSION AND SUMMARY . . . 158

7.1 Revisiting the Research Problem . . . 158
7.2 A Hybrid Solution: The NARCES Framework . . . 159
7.3 Contributions of this Research . . . 160
7.4 Research Outcomes and Future Prospects . . . 164

BIBLIOGRAPHY . . . 166

APPENDICES

A. MeTTa-NARS INFERENCE RULES 177

B. MeTTa-NARS TRUTH FUNCTIONS 181

C. MeTTa-NARS TERM-REDUCTION RULES 182

D. MeTTa-NARS REVISION AND CHOICE 183

E. ABLATION STUDY 184

F. NOTATION AND SYMBOLS IN CHAPTER 4 188

G. NACE PSEUDOCODE 189

H. HYPERPARAMETER DETAILS FOR DRL MODELS 190

I. HARDWARE AND RUNTIME 200

J. EXAMPLE WITH LEARNED RULES 201

LIST OF FIGURES

Figure		Page
2.1	Overview of MeTTa-NARS.	21
2.2	Concept Structure of MeTTa-NARS.	26
2.3	Priority Decay with Time of a Data Item.	39
2.4	Code Structure Flowchart.	47
2.5	MeTTa-NARS Prompt in Interactive Mode.	49
2.6	MeTTa-NARS Inputs and Derivations.	50
4.1	State Components.	66
4.2	Rule schema: Preconditions with an Action Followed by Consequence.	67
4.3	Flow Diagram of the System.	69
4.4	Learning Curves in *MiniGrid-Empty-16x16-v0*.	75
4.5	Learning Curves in *MiniGrid-DistShift2-v0*.	76
4.6	Learning Curves in *MiniGrid-LavaGapS7-v0*.	77
4.7	Learning Curves in *MiniGrid-SimpleCrossingS11N5-v0*.	78
4.8	Learning Curves in *MiniGrid-Unlock-v0*.	79
4.9	Learning Curves in *MiniGrid-DoorKey-8x8-v0*.	80
5.1	NARCES Architecture.	97
5.2	NACE with Modified Planner.	100
5.3	Explorer ROS2 Diagram.	112
5.4	Semantic Grid Shown Within RViz Suite.	117
6.1	Yahboom Transbot.	127
6.2	Multi-modal Integration Illustration.	129
6.3	Corresponding ROS Network for NARS.	129

6.4 Garbage Collection in Aarhus Robotics Testbed. 132
6.5 Visual Object Detection and Corner Mapping. 134
6.6 Checkpoint Placement, Trajectory and Garbage Items. 137
6.7 NARS-GPT Overview. 141
6.8 Overview of MeTTa-NARS, the NARS Implementation. 148
J.1 Illustration of Minigrid-Empty-8x8. 201

LIST OF TABLES

Table		Page
4.1	Final values for *MiniGrid-Empty-16x16-v0*	75
4.2	Final values for *MiniGrid-DistShift2-v0*	76
4.3	Final values for *MiniGrid-LavaGapS7-v0*	77
4.4	Final values for *MiniGrid-SimpleCrossingS11N5-v0*	78
4.5	Final values for *MiniGrid-Unlock-v0*	79
4.6	Final values for *MiniGrid-DoorKey-8x8-v0*	80
4.7	Performance of NACE and different algorithms	81
4.8	NACE Performance with different planning horizons	83
6.1	Object Detection Performance	134
6.2	Object Collection Performance	136
6.3	Translation from the various triplets to Narsese	145
6.4	Translation from Narsese to English	146
6.5	Performance comparison of NARS-GPT and GPT-4	154
H.1	DQN and A2C Training Parameters	193
H.2	TRPO and PPO Training Parameters	194
I.1	Runtime (in seconds) across *MiniGrid-v0* environments.	200

CHAPTER 1

INTRODUCTION

1.1 Motivation

Intelligence is a term with many meanings, it is conceptually complex and continues to be a topic of significant debate. In the Journal of Artificial General Intelligence's (JAGI) 10th volume, an article by Dr. Pei Wang titled "On Defining Artificial Intelligence" *Wang* (2019) presents a definition of intelligence that I wholeheartedly support: "The essence of intelligence is the principle of adapting to the environment while working with insufficient knowledge and resources". This article underwent an extensive review process that enlisted insights from over 100 top AI experts and made a notable effort to bridge the persistent gap in consensus regarding the definition of Artificial Intelligence (AI). My motivation and commitment to the field of AI grows directly from this definition, believing that the core of intelligence lies in the ability to adapt in real-time to new environments and unforeseen circumstances, engage in incremental and interactive learning from observations, integrate various types of knowledge, and still remain explainable and transparent.

1.2 Intelligence and Autonomy

In recent decades the field of AI research has introduces numerous systems and solutions that aim to rival human level intelligence with some systems even surpassing human performance in specific tasks. While some AI systems excel in certain single-domain tasks, such as image recognition in static environments, two critical

challenges persist: generality and real-time online learning. Despite substantial efforts and resources dedicated to addressing these issues, they remain largely unsolved or rely on temporary fixes. These challenges have emerged as central obstacles in developing truly intelligent autonomous agents capable of functioning in open-world environments.

There are a multitude of existing State-Of-The-Art (SOTA) "intelligent" systems developed by the Machine Learning and Reinforcement Learning communities, applied across domains like computer vision, speech recognition, language translation, among others. A notable advancement in recent AI development are the models based on Deep Neural Networks (DNN), which have given rise to various tools adept at handling natural language inputs *Min et al.* (2021) and being prompted for a wide array of tasks *Bubeck et al.* (2023). Nevertheless, these systems are generally optimized and trained for specific, narrow tasks, such as solely recognizing objects or translating languages, and they heavily rely on large-scale human-labeled training datasets alongside offline, inefficient optimization methods such as backpropagation, the most prevalent Deep Learning optimization technique to date. Generative AI models based on the Generative Pre-Trained Transformer (GPT) *Radford et al.* (2018) encounter additional limitations and constraints, in particular, they are constrained by a context size limit of a few thousand tokens and require augmentation with long-term memory to facilitate the retention of user input over extended periods. When the information is retained for longer, the need for updating the system's knowledge arises, as the newly encountered knowledge can either support or contradict previously acquired information. In such scenarios, it is advantageous for the system to be able to formulate and store its conclusions for long-term use and to reason in a dependable and explainable manner.

One of the significant challenges that AI has yet to convincingly address is transitioning from offline optimization to quick and reliable learning in real-time. This

capability is not only essential for intelligence in the nature, but it is also the fundamental goal of AI. Current state-of-the-art AI solutions are not equipped with the ability to learn at runtime after the training phase and especially after being deployed and therefore unable to adapt to unforeseen scenarios. This is attributed to several challenges, some of which are that the agent has to deal with non-stationary environments and the proper decision theory should be discovered, whereas for biological species it was achieved through evolution and not during a single agent's lifetime. Reinforcement Learning (RL) systems utilize a pre-defined reward function and are concerned with learning an optimal policy, namely a state-action response mapping, with the highest expected rewards. Such systems lack learning of general casual relations not defined by the policy but encountered in the environment resulting in a narrow, reward-centric view of the world. Furthermore, by design, RL systems are limited in transferring already learned knowledge because the learning happens using a utility function (which generates the rewards), and once the utility function changes, the agent needs to re-learn a proper policy and thus its current knowledge which might have captured some general casual relations will not be able to transfer to a new task.

This leads to the discussion about explainability and transparency of contemporary AI models. Today's practical ML/DL systems are truly immense in scale, often encompassing hundreds of billions of parameters needed to learn and a vast array of hyperparameters that require collective and systematic optimization *Wang et al.* (2023b). Such complexity often makes it extremely challenging to pinpoint the exact causes behind a particular detection, derivation or inference, or response generated by the model. In mission-critical domains, being able to explain and comprehend a model's reasoning in real-time is crucial and is often prioritized over mere "effective on average" performance. Our goal is to enhance our models with the ability to reveal the derivation process and clarify the reasoning behind their decisions. This includes

listing the fulfilled and involved preconditions and identifying the various pieces of knowledge employed during the derivation process.

Evaluating the intelligence of AI systems, particularly their general capabilities, poses yet another challenge. It is often viewed as assessing how well an agent performs on specific, narrow tasks or in certain domains, with the flexibility to adjust the system's hyperparameters and weights as it transitions between these domains. For instance, a computer vision classifier might effectively use a Convolutional Neural Network (CNN) to detect particular objects, but applying the same architecture to different tasks would necessitate retraining to learn new parameters and optimization of hyperparameters. This method yields various specialized performance metrics for different agents, which can be advantageous for specific applications. However, it does not provide an assessment of an agent's overall general intelligence.

Considering the mentioned above fundamental challenges the current AI systems face, it is clear that achieving generalization, maintaining effective performance in different domains, learning at run time, retaining important knowledge for longer periods, and being able to constantly update current knowledge requires a fusion of multiple approaches including neural, symbolic and sub-symbolic. This involves incorporating advanced reasoning capabilities through the utilization of sophisticated control mechanism, including attention allocation, complicated memory structure, some kind of logic as well as different types of knowledge representation. In some cases, it becomes essential to integrate multiple systems into a hybrid system to maximize their combined potential and reach a certain degree of generality.

Approaches that combine multiple components into a sophisticated system claiming to encompass general capabilities do however exist. The field of AI research created numerous complex architectures and artificial general intelligence systems that aim to explain a wide range of human behavior and mimic the capabilities of human cognition. These architectures are typically seen as a single, integrated system consist-

ing of multiple individual modules or components working together to imitate some behavior *Chong et al.* (2007). One prominent example is the LIDA (Learning Intelligent Distribution Agent) *Franklin et al.* (2013) system which which attempts to model human consciousness. The logical part of LIDA enforces Global Workspace Theory (GWT) (Baars 1988) and implements the cognition process in a serial way through the use of system cycles. LIDA's overall structure is notably complex, featuring both symbolic and sub-symbolic connectionist components, incorporating numerous independently structured modules, and hosting various memory types such as Perceptual Memory, Procedural Memory, Episodic Memory, and Local Workspace, each with different connectionist architectures. However, as complexity increases, maintaining the practicality of the system becomes a challenging task because the system should remain modular, sophisticated, and flexible to allow real-world practical applications to multiple domains.

In this thesis, we particularly focus on incremental and interactive learning at runtime, adaptation, and generality as key aspects of intelligence. We aim to create a flexible, multi-component autonomous intelligent system that integrates both symbolic and sub-symbolic components capable of performing a variety of tasks across a wide range of domains without altering the model or its structure. Furthermore, we touch upon important aspects of measurement and the explainability of systems that exhibit these general capabilities.

1.3 Research Question

We understand that it is difficult for a single monolithic system to provide a satisfactory level of generality and performance in a reliable and data-efficient way. Therefore, our focus has been shifted toward combining the multiple approaches to provide hybrid solutions for practical real-time reasoning, in particular, we created symbolic practical reasoning system MeTTa-NARS and sub-symbolic data-efficient

explorer NACE. MeTTa-NARS is the advanced stand-alone symbolic reasoning system that follows NARS model of intelligence, while NACE is an example of experiential learning, which is a novel sub-symbolic system that showcases high data efficiency in learning and sequential decision-making tasks in grid world environments.

Bringing two of the components into NARCES (Non-Axiomatic Reasoning and Causal Explorative System) prototype, we chose to work toward a unified foundation of symbolic and sub-symbolic reasoning. We recognize the need for semantics that provide meaning to NACE's actions and cell values within the grid world to express high-level knowledge and establish goals to be pursued, and at the same time, MeTTa-NARS will benefit from efficient and robust sub-symbolic processing. Thereby with this framework in mind, we try to resolve a fundamentally important research question namely how can we obtain a combined system where semantics as in NARS and its high-level reasoning can be effectively and efficiently interfaced with an array-based world representation for effective sensorimotor processing.

CHAPTER 2

MeTTa-NARS: HIGH-LEVEL PRACTICAL REASONER

2.1 Introduction

MeTTa-NARS follows NARS (Non-Axiomatic Reasoning System) model of intelligence, it is a high-level symbolic practical reasoning system where knowledge is represented through formal language known as Narsese and is grounded using terms, relations between the terms, concepts, and statements. The reasoning abilities of MeTTa-NARS are accomplished by the repeated application of Logic Rules and are realized in the framework of a reasoning system using effective Inference Control. NARS is designed around the fundamental belief that intelligence entails adapting to the environment while working with the additional constraints to operate under the Assumption of Insufficient Knowledge and Resources (AIKR) *Wang* (2009a) which includes the following key requirements:

- **Finite**: The memory and information-processing capacity of the system is strictly constant-bounded during its lifetime in terms of both storage capacity and processing speed.
- **Open**: The reasoner should be able to accept new information of any content at any time while it is operating and also alow all knowledge to be updated by the new evidence.
- **Real-Time**: Learning from events as they enter the system in the real-time.

React to new knowledge and evolving scenarios in a timely manner. This often involves changing focus on more important tasks because reasoning steps can only take up to a maximum sub-second constant amount of time.

Being a symbolic reasoning system, MeTTa-NARS can be analyzed in four major parts namely language, logic, memory, and inference control *Wang* (1995, 2006, 2013a).

1. **The Language** used in NARS, known as Narsese, decides what can be expressed as tasks and knowledge. It is a formal rich and expressive language capable of representing pieces of knowledge or even capturing the semantics and structures of natural language. Narsese employs terms for words or atomic concepts and utilizes relations between these terms in statements formatted as $subject - copula - predicate$. Narsese allows for nesting of statements, providing it with full expressive capabilities. Due to its formal and structured nature, Narsese can be easily converted to natural language through a deterministic algorithm, which is explained in subsequent chapters, facilitating simple comprehension for humans.

2. **Logic** decides what new tasks and knowledge can be derived from the already available information in the form of input or derivation within the system's experience. Multiple logics have been proposed to support reasoning under uncertainty, such as: Markov Logic Networks *Richardson and Domingos* (2006), ProbLog *De Raedt et al.* (2007), Fuzzy Logic *Zadeh* (1988), Probabilistic Logic Networks *Goertzel et al.* (2008) and Non-Axiomatic Logic *Wang* (2013a), where they commonly extend truth value of propositions from boolean to a degree of belief to allow capturing knowledge that is neither true or false, but somewhere in-between. Of these logics, *De Raedt et al.* (2007), *Richardson and Domingos* (2006) and *Goertzel et al.* (2008) operate with probability values associated to

the prepositions. Additionally, *Goertzel et al.* (2008) and *Wang* (2013a) use a second value to address the size of the sample spaces, which intuitively speaking corresponds to the stability of the probability of a new evidence. Such approach allows to allocate a higher certainty for a 100/100 over a 10/10 coin flip case, while still converging to the same truth value in the limit of infinite samples. Therefore these two logics become extremely well-suited for cases where "degree of belief" has to be estimated from samples and justifiable conclusions should be drawn (or decisions being made) even when samples supporting a relevant hypothesis are low in count. In this case the ratio of confirming cases over total cases is not necessarily yet representative because the amount of samples also needs to be considered when comparing competing hypotheses to make them suitable for learning goal-directed behaviors and solving the Goal-Directed Procedure Learning Problem *Hammer and Lofthouse* (2018), which is concerned with learning behaviors to reach (potentially changing, and conflicting) goals. MeTTa-NARS utilizes Non-Axiomatic Logic since it incorporates goal reasoning and decision making, hence can be considered a Practical Reasoner able to learn from experience.

3. **The Memory** in cognitive architectures *Kotseruba and Tsotsos* (2020) as well as reasoning systems is being categorized in terms of its duration: short-term, long-term, and type: declarative, episodic, procedural, etc. Particular implementations of memory systems, however, might differ significantly and depend on the current goals of a designer, conceptual limitations and multiple engineering factors. In NARS, choice of memory system and clever integration within system's components plays a crucial role. Given that NARS operates under AIKR in the real-time, a new piece of knowledge can arrive at any given moment requiring the system to work under finite resource constraints and be always available for the new data. System should efficiently process various

types of knowledge (declarative, procedural, episodic etc.), and by doing so it decides on where to store the data, for how long it should be available, and how to efficiently retrieve it for further inference, question answering or goal processing. The memory in NARS follows a unique approach, we do not include well-defined separate components dedicated for different processing types featuring own memory, instead we follow an integrated approach, within which most types of memories are present and processed. To maintain memory within certain limits, important items are retained for longer durations, while less crucial information is discarded, or forgotten. The criteria for preserving specific pieces of knowledge are determined by the Attention Allocation mechanism, which varies depending on the implementation. We will explore Attention Allocation and its related considerations later in the chapter.

4. **Inference Control** specifies which premises to pick and which inference rules to apply at any moment on a cyclic basis operating in real-time under uncertainty, whereby the reasoning system is designed to favor selection of goal- and question-relevant knowledge for reasoning, which treats attention allocation as a resource allocation problem. In other words, inference control decides which possible derivations will be actually carried-out among all existing possibilities. The responsibility of Inference Control is also to efficiently process various types of knowledge and by doing so it decides on where to store the data and how long to retain it with guidance by an in-built Attention Allocation mechanism. Being a real-time system requires that an inference cycle is completed within a constant time insuring the system remains responsive to new input data. If this time constraint isn't met, the system will fail to operate effectively and responsively. Attention and Resource Allocation mechanism is essential in maintaining the system's effectiveness and responsiveness. As with the memory approach mentioned earlier, we use an integrated approach where we don't have separate

Inference Control types for handling various kinds of knowledge. NARS features a single, conceptually unified inference control with interconnected parts forming a cohesive, iterative architecture.

At the current stage, Language and Logic of NARS have become relatively stable, while the Inference Control and Memory still remains the topic of active research and development. There have been publications on the conceptual design of the control part *Wang* (1995, 2004, 2006, 2009b), as well as multiple open-source implementations of NARS model[1] *Hammer et al.* (2016); *Isaev and Hammer* (2020, 2023); *Hammer and Lofthouse* (2020). However, early NARS implementations struggled with practical applications due to various limitations, while the later versions became excessively complex and monolithic, making them difficult to understand, maintain, and support. The major motivation of creating MeTTa-NARS was to develop a lightweight, modular, and standalone reasoning system that is efficient and versatile enough for a wide range of applications and domains. Our intentions were to build a practical reasoning system using a high-level functional language without compromising on performance, thus ensuring that it remains easy to develop, support, and understand right from reading the source code itself.

This chapter provides a comprehensive description of the conceptual design of MeTTa-NARS with a focus on the inference control, architecture, attention allocation mechanism, as well as explores the design choices, innovations, and implementation details which are crucial for implementing a practical reasoning system in a functional language, along with relevant data structures and logic details.

[1] `http://opennars.org/`

2.2 Theoretical Basis

2.2.1 Truth Value and Expectation

Truth Value is an essential component of NAL, it represents the degree of believe in a particular piece of system's knowledge. It is based on positive evidence w^+ and negative evidence w^- which speaks for or against a statement, belief, or hypothesis, and the total evidence $w := w^+ + w^-$, each of which is zero or greater. In other words, truth value of a statement measures its evidential support, it indicates how close it is to the evidence captured by the system experience so far. Based on these evidence values, the NAL truth value is specified as the frequency and confidence tuple (f, c) where frequency defined as

$$f := \frac{w^+}{w} \in [0, 1]$$

and confidence

$$c := \frac{w}{w + 1} \in [0, 1)$$

Please note the similarity between frequency and probability, with the difference being that the limit $\lim_{w\to\infty} f$ is not taken, as it cannot be obtained from any finite amount of samples. Also, clearly for $w > 0$, the mapping $(w^+, w^-) \mapsto (f, c)$ is bijective, and statements with $w = 0$ do not need to be handled as they do not contribute any evidence.

Additionally, truth expectation is an important measure which is necessary for multiple purposes discussed later. Truth Expectation is defined as

$$expectation(f, c) := (c * (f - \frac{1}{2}) + \frac{1}{2})$$

This measure allows to summarize the two-valued truth value into a single value metrics with the extremes being 0 for $c = 1, f = 0$, and 1 for $c = 1, f = 1$, which both are approachable but unreachable, since $\forall w \in \mathbb{R} : c < 1$ while $\lim_{w\mapsto\infty} c = 1$.

2.2.2 Non-Axiomatic Logic

Non-Axiomatic Logic (NAL) *Wang* (2013a) specifies which information can be derived from combinations of at most two premises with support of uncertainty estimation. It incorporates inference rules for learning from event streams, goal reasoning and decision making, as well as truth functions to estimate the uncertainty of statements based on evidential support. Hence, a means-end reasoner can be built to learn from experience using NAL. NAL is defined using multiple layers (1-8), whereby logic becomes more advanced with each layer. Below we use Narsese formal language to introduce NAL copulas and describe its layers:

- NAL-1: The most basic level that introduces Inheritance relation. Inheritance $< A \rightarrow B >$ indicates A to be a special case B. For example, raven is bird can be encoded as an Inheritance statement $(raven \rightarrow bird)$.

- NAL-2: Adds Similarity and introduces Instances and Properties. Similarity functions as a bi-directional version of Inheritance that is $< A \leftrightarrow B >$. Instance and property terms are encoded as $\{instance\}$, $[property]$ respectively. For example $(\{Tweety\} \rightarrow [black])$ states that instance Tweety has the property of black.

- NAL-3: Compound Terms. This includes extensional sets $\{a_1, ..., a_n\}$ as well as intensional sets $[a_1, ..., a_n]$, which can include multiple instances/properties at once. In addition extensional and intensional intersection, $(a \,\&\, b)$ and $(a \mid b)$ are introduced, together with extensional and intensional difference, $(a - b)$ and $(a \sim b)$.

- NAL-4: Ordinary Relations. This includes products $(a*b)$ to express anonymous relationships like that cats eat mice, $< (cat * mouse) \rightarrow eat >$, as well as extensional and intensional images, $(R /1\, x)$ and $(R \backslash 1\, x)$ respectively. The

index "1" allows to distinguish "driving a car" ($drive\,/1\,car$) from "driven by car" ($drive\,/2\,car$), which conveys a different meaning.

- NAL-5: Higher Order Statements. It introduces higher order statements and establishes truth-relationships between them. This includes negation ($\neg\, a$), conjunction ($a \,\&\&\, b$), disjunction ($a \,||\, b$), implication $< a \Rightarrow b >$ and equivalence $< a \Leftrightarrow b >$.

- NAL-6: Variable Terms. While NAL distinguishes between independent and dependent variables syntactically, MeTTa-NARS relies entirely on MeTTa's pattern-matching mechanism rather than explicit variable declarations. In this framework, variables are implicitly handled by term unification, whereby the logical structure of a term determines the variable type:

  - In a conjunction, variables are treated as dependent
  - In an implication, if a variable appears on both sides, it functions as an independent variable
  - If the variable appears on only one side of the implication, it is treated as a dependent variable.

  This approach allows MeTTa-NARS to reason about variable roles based on structural patterns rather than syntactic annotations.

- NAL-7: Temporal Reasoning. This layer describes events which are statements with time-dependent truth value. In general, truth value of an event depends on how close its occurrence time is to the system's current time which is determined by Projection formula (discussed later) whose role is to decrease the event's confidence value c as time distance Δt increases to their occurrence time: $c^* \mapsto c * \lambda^{\Delta t}$. We also obtain a time-independent generalization of the event statement using eternalization formula as $c^* \mapsto \frac{c}{c+1}$ which is a form of

induction. In addition, NAL-7 describes temporal compositions, this includes temporal variants of NAL-5 copulas, including sequential conjunction ($a \,\&/\, b$), parallel conjunction, ($a \,\&|\, b$), sequential implication (a `=/>` b) and parallel implication (a `=|>` b).

- NAL-8: Introduces Procedural Reasoning which includes operator terms $\hat{}op$ and involves fulfilling the goals and executing operations. The system is capable of deriving such events to fulfill the stated goals. Goals are the events with desire value which resembles the truth value however is conceptually different. That is, the desire value of a goal $G!$ is formalized as the truth value of $(G \Rightarrow D)$ where D stands for an implicit or virtual "desired state". This way reasoning on goals is able to utilize the belief-related inference rules from lower NAL layers without needing a special treatment.

Now we can observe that in NARS, the meaning of a term is shaped by the relations it participates in, which forms the basis of Experience-Grounded Semantics *Wang* (2005). This differs from other symbolic systems like expert systems *Lucas and Van Der Gaag* (1991), where symbols represent objects and relationships in an external world according to a predetermined mathematical model (Model-theoretic semantics). Ambiguities in meaning, and contradictions in truth value are both captured by Non-Axiomatic Logic.

In this chapter, our primary emphasis will be on layers 7 and 8 (Temporal and Procedural), as they are the most critical layers that affect implementation and dictate specific design decisions.

2.2.3 Inference Rules

NARS's expressive and rich inferential abilities are achieved through repeated application of inference rules. With the previously defined truth value in mind, *Wang* (2013a), the following essential inference rules can be articulated:

Deduction: $\{A \longrightarrow B, B \longrightarrow C\} \vdash A \longrightarrow C, f_{ded}$

Abduction: $\{A \longrightarrow C, B \longrightarrow C\} \vdash A \longrightarrow B, f_{abd}$

Induction: $\{C \longrightarrow A, C \longrightarrow B\} \vdash A \longrightarrow B, f_{ind}$

Each inference rule is paired with a specific truth value function designed to estimate the truth value for the derivation given the two premises. The corresponding truth value functions defined as follows:

$$f_{ded}((f_1, c_1), (f_2, c_2)) = (f_1 * f_2, f_1 * f_2 * c_1 * c_2)$$

$$f_{abd}((f_1, c_1), (f_2, c_2)) = (f_2, \frac{f_1 * c_1 * c_2}{f_1 * c_1 * c_2 + 1})$$

$$f_{ind}(T_1, T_2) = f_{abd}(T_2, T_1)$$

This selected collection of NAL inference rules captures the most relevant inferences. The complete set of inference rules as well as accompanying truth value functions utilized in MeTTa-NARS can be found in Appendix A and B.

2.2.4 Control Criteria and Attention Allocation Principles

When designing a practical reasoning system, there are several critical factors to consider. Given that the system must work under AIKR, all data structures should be bound in size, and eviction strategies to maintain this constraint need to be in place. Additionally, the system's reasoning cycle needs to finish roughly in a constant time, meaning there will not be enough time to exploit the entire derivation space or even exhaustively check the memory for the best derivation options. Furthermore, a single inference step cannot be interrupted by more important tasks, however, an important task should be able to interrupt all the work carried out over multiple cycles, which captures all the problem-solving activities of the system. To facilitate this, the system should stay responsive to new inputs and derivations.

Ideally, only the most contextually relevant and goal-related inferences should be made. This is achieved through relative forgetting, which makes sure that only contextually relevant items are active at any moment in time. This strategy, however, raises a critical question: which data items should be retained, and which should be discarded to make room for new information? A key here is to see that the criteria for selection and forgetting are quite different. When a selection is made, the context reflected by the *priority* of a data item usually (but not always) matters more than its historic value. For forgetting, on the other hand, it will be a *usefulness* of the item that is of interest, though a new data item needs to have a chance to prove its importance. Item's usefulness is based on use count, how many times the item has been used, and last used time, when the item was used last. Priority estimation as well as Attention Allocation principles are discussed later in the chapter.

MeTTa-NARS effectively addresses all these considerations through its sophisticated Attentional Control and a versatile memory structure that includes short- and long-term memory. This setup enables efficient goal-reasoning and decision-making by leveraging learned temporal and procedural structures. Many of above considerations have been discussed in *Helgason* (2013) as well, and the need to take them into account altogether is what makes designing the inference control a difficult task.

2.2.5 Data Items

MeTTa-NARS involves various types of data items that participate in inference and allow for efficient implementation. Here, we will focus on the most crucial ones: *terms*, *events*, and *implications*, while *concepts* will be discussed in the subsequent sections.

Term, Statement and Sentence: All knowledge within the system is represented using terms and sentences. Term in NARS can be seen as a fragment of experience, it is a data item that might refer to an object, property or an operator.

This can be viewed as a single "word" such as *bird*, *black* or *open*. A statement is a syntactic construct, which is built using terms and copulas such as ($raven \rightarrow bird$). In other words statement can be seen as a compound term. Sentence, however, is a statement paired with a truth value.

Event in MeTTa-NARS encompasses a statement paired with a truth value, an evidential base, an occurrence time, and a priority value. The evidential base is consisting of IDs events which either contributed to the current event or an initial ID for the new event. Evidential base is essential for assessing the statistical independence of premises, allowing derivations only when there's no evidential overlap between premises. Each event can be categorized as a *judgment* (declarative piece of knowledge) or a *goal* (an event systems desires to be re-created). For inputs, a *question* type of events is permissible, functioning as a memory search and returning the answer with the highest truth expectation among concepts that exactly match the question.

Eternal belief is the declarative or generalized version of an event, which is sometimes called an eternalized event. That is an event with the occurrence time removed and confidence c being updated as $c^* \mapsto \frac{c}{c+1}$.

Implications can take on different forms. Declarative implications are higher-order statements built using NAL-5 layer, as discussed previously. In MeTTa-NARS, declarative implications represent the eternalized form of their temporal equivalent. The primary focus, however, is placed on temporal implications ($A \Rightarrow B$) and procedural implications of the form $((A, Op) \Rightarrow B)$. The former denotes that B will happen after A, and the latter that B will happen when Op is executed right after A has happened. To calculate the truth values of these correlative implications *Wang and Hammer* (2015), the evidences of w^+ and w^- are needed. If events would have binary truth, for $(A \Rightarrow B)$, w^+ would be the amount of cases in which A happened and B happened after it, and w^- would be the amount of cases where A happened

but B did not happen thereafter. Slightly more complex but following the same idea, given $((A, Op) \Rightarrow B)$, w^+ would be the amount of cases in which A happened, *op* was executed and B happened after it, while w^- would be the amount of cases where A happened and *op* was executed but B did not happen thereafter. Differently than the schemas in *Kansky et al.* (2017), implications can be supported to various degree instead of having to match all the data the agent has seen so far.

Now, using w^+ and w^-, the truth value (f, c) of the implication statements would be fully determined. While this captures the main idea, we need to make the temporal reasoning more robust in regards to timing variations which will be discuss in subsequent section.

2.3 MeTTa-NARS Deign and Implementation

This section provides technical aspects and design principles of MeTTa-NARS as well as implementation details of its inference control and architecture.

2.3.1 Logic Implementation in MeTTa-NARS

Being a high-level functional language, MeTTa provides the ways to elegantly implement NAL including its inference rules along with truth value functions in a descriptive manner. Below there is an example of select inference rules being explicitly defined as part of the Logic component of MeTTa-NARS. The remaining inference rules as well as structural term reduction rules are specified in Appendix A and C.

NAL-1 Inference Rules for Inheritance:

```
(= (|- (($a --> $b) $T1) (($b --> $c) $T2)) (($a --> $c) (Truth_Deduction $T1 $T2)))
(= (|- (($a --> $b) $T1) (($a --> $c) $T2)) (($c --> $b) (Truth_Induction $T1 $T2)))
(= (|- (($a --> $c) $T1) (($b --> $c) $T2)) (($b --> $a) (Truth_Abduction $T1 $T2)))
```

NAL-5 Inference Rules for Implications:

```
(= (|- (($A ==> $B) $T1) (($B ==> $C) $T2)) (($A ==> $C) (Truth_Deduction $T1 $T2)))
(= (|- (($A ==> $B) $T1) (($A ==> $C) $T2)) (($C ==> $B) (Truth_Induction $T1 $T2)))
(= (|- (($A ==> $C) $T1) (($B ==> $C) $T2)) (($B ==> $A) (Truth_Abduction $T1 $T2)))
```

Selected Truth Functions:

```
(= (Truth_Deduction ($f1 $c1) ($f2 $c2)) ((* $f1 $f2) (* (* $f1 $f2) (* $c1 $c2))))
(= (Truth_Abduction ($f1 $c1) ($f2 $c2)) ($f2 (Truth_w2c (* (* $f1 $c1) $c2))))
(= (Truth_Induction $T1 $T2) (Truth_Abduction $T2 $T1))
```

Each rule is expressed in prefix notation using the entails operator, comprising two premises each associated with a truth value, and a derivation whose truth value is generated by applying the relevant truth function. Throughout the inference cycle, all rules are applied to the premises using MeTTa's built-in pattern matcher, resulting in derivations produced only by the matching rules only.

2.3.2 Overall Architecture

The MeTTa implementation of NARS follows design paradigms borrowed from multiple previous NARS versions (*Ivanović et al.* (2017), *Lofthouse* (2019), *Wang et al.* (2020), *Hammer and Lofthouse* (2020)) as well as introduces its own distinctive structure and design. MeTTa-NARS utilizes concept-centric semantic memory *Isaev and Hammer* (2023) ensuring premises in inference will be semantically related. This property, along with the priority based selection and choice of data structures, helps to avoid combinatorial explosion and reason in real-time. The overall architecture is shown in Figure 2.1, with the *Event Provider* functioning as the entry point to the system. The reasoner is designed to process inputs from various modalities, expressed as Narsese statements. It can receive inputs from one or more sensory channels, and it must reason on this information and produce meaningful derivations. While simple input is often sufficient, system can accept more complex inputs such as Multi-Class

Multi-Object Tracker for the detection and tracking of instances and their types *Hammer et al.* (2021).

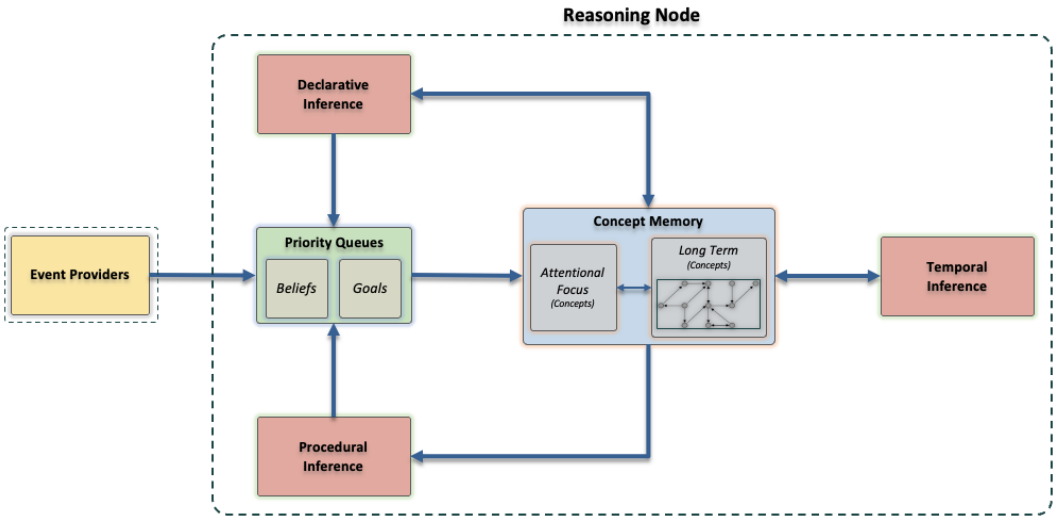


Figure 2.1: Overview of MeTTa-NARS.

The inference control mechanism of MeTTa-NARS features declarative, temporal, and procedural types of reasoning enabling rich expressive practical applications. It is mainly concerned with premise selection and NAL inference rules applications resulting in knowledge derivation, triggering of a decision, leading to the derivations of subgoals, or executing operations. Providing an in-depth explanation of the functionality and structure of the control components is outside the scope of this thesis. However, we will discuss the most pertinent definitions of these components in the relevant sections to help readers grasp their fundamental functionalities.

2.3.3 Main Memory

In systems based on Non-Axiomatic Logic *Wang* (2013a), choice of memory system and clever integration within system's components plays a crucial role because a new piece of knowledge can arrive at any given moment requiring the system to work under finite resource constraints and be always available for the new data. Control mechanism of such system should efficiently process various types of knowledge

(declarative, procedural, episodic etc.), and by doing so it decides on where to store the data, for how long it should be available, and how to efficiently retrieve it for further inference, question answering or goal processing. Based on these considerations, the memory system of MeTTa-NARS follows a unique approach. As shown in Fig. 2.1, system does not include separate components dedicated for different processing types featuring own memory and follows an integrated approach, within which different types of memory are present and processed.

In MeTTa-NARS, memory follows a concept-centric semantic memory structure in accordance with the NAL term logic the system uses. It can be viewed as a graph where concepts are represented as nodes and links designate relationships among them as seen within the memory component in Fig. 2.1. Technically, main memory is a collection of concepts representing a conceptual network with prioritized nodes. A *concept* is a major entity, an identifiable unit of system's experience that has grounded meaning. It is also a data structure to hold various components of knowledge summarizing a fragment or pattern of the system's experience identified by a term *Isaev and Hammer* (2023) in Narsese formal language.

2.3.4 Memory Types

MeTTa-NARS memory can be categorized in terms of its type, duration and content. We will proceed by discussing different types present within main memory, namely Declarative, Episodic and Procedural.

- **Declarative:** Most content of the main memory, that is general pieces of knowledge processed to become concept nodes without episodic or procedural information, functions as declarative memory of the system.
- **Episodic** memory can be viewed as events or compound events, which have been perceived by the system, that is reached the selection to the main memory. In general, an event is a piece of declarative knowledge accompanied with

temporal information. Given the nature of attentional control, not all events are being considered for the selection to the main memory. Upon selection of an event, its episodic information is stored local at the concept level.

- **Procedural memory** is a special kind of knowledge, namely procedural hypotheses (discussed later), that is a episodic knowledge including operation or compound operations within its description. Procedural information is stored local at the concept level, however, derived procedural knowledge can become a concept node by itself.

Categorizing memory by duration involves classification in terms of *long-term* and *short-term* memories. In context of NARS, time is a fluid relative conception that complicates the definition of long/short-term memory. Conceptually speaking, declarative knowledge, i.e., knowledge without episodic information, can be considered a kind of long-term memory. However, being a real-time reasoning system, MeTTa-NARS requires resource allocation mechanism, in particular forgetting techniques to be in place. For example, a piece of knowledge can be removed from the memory given that memory capacity is full and its priority decreases to the certain level reaching the lowest usefulness for the system at a given moment in time. In this case, forgetting process, can be seen as a decisive factor for considering pieces of knowledge falling into long or short-term category. Overall, we advance that long/short-term memory discussion might not be meaningful in the context of NARS.

It is also important to discuss memory in terms of content to be processed such as *working memory* and *attentional focus*. Working memory is a priority-based selected knowledge during the inference cycle. It is not only limited to the selection of concepts within the main memory, but also consists of selections from priority queues as seen in Fig. 2.1. In other words, working memory is where the processing currently happens or a storage under current attention. Alternatively, attentional focus can be defined as the distribution of concepts with the highest priority, such as the knowledge most

likely to be processed in the future unless a derived or a new input knowledge has even higher priority.

2.3.5 Data Structures and Corresponding Spaces

At the time of creation of MeTTa-NARS, MeTTa language did not include various types of predefined data structures. As a result, inbuilt MeTTa *atom spaces* are used for storing different kinds of knowledge and data items. There are four (4) atom spaces used in MeTTa-NARS as seen in Fig. 2.1 for *belief_events*, *goal_events*, *attentional_focus* and *long-term memory*. Attentional focus and long-term memory both hold concepts and collectively forming the main memory of the system. All of the spaces contribute to Attentional Control of the system and are maintained on the basis of *priority* of items which can be seen as an importance of the item to the current context of the system. Spaces are summarized as following:

- *belief_events*: a collection to hold input and derived events
- *goal_events*: a collection that holds input goals and derived sub-goals
- *attentional_focus*: a collection holding highest priority concepts
- *long-term memory*: a collection that holds all the concepts of the system

All of the above spaces are bound in size by system's hyper-parameter defined in system configuration. *Belief_events* and *goal_events* serve the purpose of the priority queues where items should be sorted according to their priority value and only the highest priority event or goal to be selected during the cycle. These spaces features threshold based selection always referencing k highest priority items what enables for efficient selection taking constant time (O(1)). Maintaining the space constraint happens with the help of a recursive procedure using dynamic threshold. The role of *attentional_focus* is to hold k highest priority concepts which are the most relevant to the current context or experience of the system. When the number of these

concepts surpasses the attentional focus' capacity, the lowest priority concepts aren't simply discarded or irretrievably forgotten. Instead, they are moved to the system's long-term memory, where the actual process of forgetting occurs. On the other hand, hard forgetting is applicable to *belief_events* and *goal_events*. In these cases, when the number of events exceeds the available capacity, the items with the lowest priority are removed from the queues. The capacity of *belief_events*, *goal_events* and *attentional_focus* is limited to 100 elements each, which we found to be a good balance for responsiveness of the system and is a system hyper-parameter. These three spaces serve as the immediate storage within the system's current focus, and therefore directly contribute to the working memory of the system as events, goals, and concepts are being selected for processing during the inference cycle strictly from these data structures.

Long-term memory also utilizes MeTTa's generic atom space and stores the entire system experience with the exception of concepts present in the attentional focus. It functions as a long term memory and is limited to 10000 concepts (also a hyper-parameter). MeTTa's built-in pattern matcher facilitates the selection of concepts in constant time because concepts are hashed based on the event's statements for which they were created. When a concept is being created for an event, either an input or a derivation, it is being added to *attentional_focus* of the system. If the concept is already present in the long-term memory, it is being queried, revised and moved to the attentional focus. It is important to note that a concept can only exist within the attentional focus or the long-term memory of the system but not in both. Therefore if a concept has lost the competition for resources, after some time it is being moved to the long-term memory and given its usefulness decreases because of not being participated during the inference, the concept is being forgotten i.e. removed from the long term storage. A concept by itself consists of multiple components as can be seen in Fig. 2.2, all of which are stored as components of a generic MeTTa tuple.

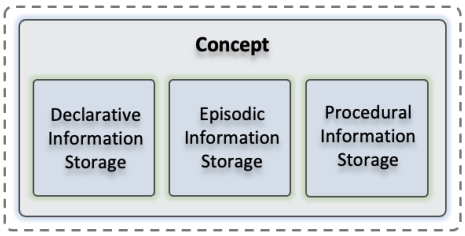


Figure 2.2: Concept Structure of MeTTa-NARS.

That includes a parent event (statement, truth value, occurrence time), evidential base, belief event (eternalized event), priority as well as procedural information if available.

2.3.6 Inference Control

The reasoning abilities of MeTTa-NARS are accomplished by the repeated application of inference rules of Non-Axiomatic Logic. This is realized in the framework of a reasoning system using effective inference control with the additional constraint to operate under Insufficient Knowledge and Resources (AIKR). Inference control specifies which premises to pick and which inference rules to apply at any moment on a cyclic basis, whereby the reasoning system is designed to favor selection of goal- and question-relevant knowledge for reasoning, which treats attention allocation as a resource allocation problem. In other words, inference control decides which possible derivations will be actually carried-out among all existing possibilities.

In MeTTa-NARS, the control feature three major components as illustrated in Fig. 2.1, each conforming to the type of reasoning:

- **Declarative:** deriving declarative knowledge from events.
- **Temporal:** temporal reasoning, sequence and temporal implication formation (temporal compounding), anticipation.
- **Procedural:** decision making with subgoaling and planning.

All these components operate simultaneously during the cycling reasoning process. We will delve into a detailed discussion of each component in the next sections.

2.3.7 Declarative Inference

The declarative inference component is concerned with the selection of an event from *belief_events* space and a concept from the *attentional focus*, and performing reasoning on the basis of a common term between the selections. All declarative reasoning using NAL layers 1-5 takes place here. Because of the descriptive capability of NAL and its experience-based semantics, inheritance relation serves as a means to define objects and the associated inference enables the reasoner to categorize events and refine these categories with additional experience. This allows the reasoner to learn and apply arbitrary relationships, thereby interpreting situations in more nuanced ways and identifying key commonalities and differences across various knowledge domains. This approach also accommodates instances where the meanings of certain terms are initially unclear and require further clarification to become functional.

Declarative reasoning is achieved on a cyclic basis which is described as following:

- **Event selection:** the single highest priority event is being selected from *belief_events* priority queue.
- **Concept selection:** k highest priority concepts are selected from attentional focus of the system.
- **Reasoning:** The selected event is to be taken as the first premise, and the concept's eternal belief or an event as the second premise. Now, for each concept, reasoning is performed by applying all matching NAL inference rules between each concept's event or belief and the selected event to produce the derivations.
- **Priority estimation:** For each derived event, priority is being evaluated based on the priorities of both the selected event and the concept (discussed later).

- **Derivation processing:** Newly derived events are being processed by adding to the *belief_events* priority queue as well as to the concept memory to create new concepts or update eternal beliefs within the existing concepts.

- **Space maintenance:** Predefined size of participated spaces i.e. *belief_events* priority queue and concept memory is being maintained, meaning the lowest priority events or least useful concepts are being forgotten given the size constraints are violated.

When a concept for a specific derivation is already present in the concept memory, the Revision rule is used to update the concept's belief event with the newly derived evidence, provided there is no overlap in their evidential bases — in other words, the knowledge comes from distinct evidential sources. If the derivation and the belief event have one or more overlapping components in their evidential bases, the system will make a choice to favor the one with higher confidence. Please see Appendix D for the Revision and Choice rules pseudocode. Additionally, it is important to note that while the Declarative reasoning component predominantly deals with declarative reasoning, it does include some temporal aspects. Specifically, after selecting a concept from the attentional focus, its eternal belief (or eternalized event) is utilized only if it falls outside a designated temporal window in relation to the selected event from *belief_events*. Otherwise, the concept's event is used directly but is temporarily aligned with the selected event from *belief_events* using a projection rule (explained later) to better fit to the current context of the system.

2.3.8 Temporal Inference

MeTTa-NARS is equipped with strong temporal reasoning capabilities with the help of the Temporal Inference control which focuses on reasoning that involves temporal relations between premises rather than shared terms. This includes temporal

induction, projection, and anticipation mechanisms, which handle predictions and assigns negative evidence for false predictions. Events are not considered "true" only at a specific moment, instead, they are assigned an occurrence time (timestamp) alongside their truth value. The management of time is then executed through inference cycles, with each cycle incrementing the time by 1 unit.

A set of statements can typically provide only a partial description of an environment. An entire experience of the system occasionally is able to capture the complete state for a limited practical case, however, in general, the system must be expected to work in non-Markovian environments where the Markov assumption does not hold. This is realized by temporally relating two or more events and by forming compound events such as temporal sequences and temporal implications. MeTTa-NARS employs Temporal Induction, a special type of induction for temporal compounding to create a predictive temporal implication of the form $(A \Rightarrow B)$ such as

$$\{A, B\} \vdash (A \Rightarrow B)$$

$$truth((A \Rightarrow B)) = f_{ind}((f_A, c_A), (f_B, c_B)) = (f_B, \frac{f_A * c_A * c_B}{f_A * c_A * c_B + 1})$$

where A and B are events meaning A has happened before B. When the same implication is derived multiple times, their truth values are revised by simply adding up positive evidence w^+ and negative evidence w^- of the premises: $w^+ = w_1^+ + w_2^+$, $w^- = w_1^- + w_2^-$. This makes sure that the implication receives increasing amounts of evidence when the supporting events do occur (the antecedent and consequent).

In the simplest case, both events in a temporal implication are simply Inheritance statements, however they also can be temporal sequences by themselves allowing for complex temporal compounding. For example, because operations are also considered as events, system initially forms a sequence of (A, Op) and later transforms into a temporal implication such as $(A, Op) \Rightarrow B$ which serves as a procedural hypothesis

discussed later.This exemplifies the central concept of procedural learning: when event A has occurred and operation Op is executed afterward, event B is likely to follow.

Here, proper time estimation is important as it allows the system to estimate the expected arrival of the predicted or derived event. Given that events have occurrence time, the time internal is always measured when temporal implication is formed. Consider a scenario where the system induces two hypotheses of the form $(A \Rightarrow B)$, with each having a distinct time interval between events A and B. Essentially, the system is experiencing the same pair of events occurring one after another at varying time intervals. The key idea is to be able to revise such hypotheses, i.e. enable the revision of temporal implications with different time intervals between the occurring events. This is accomplished with the Projection being applied right before the Induction rule, in a way that the first event is being projected to the second one. That is the confidence decreases with increasing time distance to the second premise. This is realized when two premises are used in inference, the confidence of the first premise is discounted by the projection rule as:

$$c_A = c_A * min(1,\ \frac{1}{\Delta t})$$

with $\Delta t = |time(B) - time(A)|$. As a last detail, the Δt is also updated in revision, by taking a weighted average between the time deltas of the premises, weighted by the confidence of the premises. The Δt is being stored as metadata of a temporal implication because we need this in order to decide the occurrence time of derived events.

Projection along with Induction and followed by Revision are the fundamental parts of Temporal Inference making the temporal credit assignment succeed. On the same matter, Projection that discounts evidence of the induction results based on

temporal distance between the premises plays the same role as Eligibility Traces do for Reinforcement Learners *Sutton* (1988). Unlike Reinforcement Learning, temporal credit assignment is done for all input events and temporal compositions within the system, enabling it to learn from real-time observations and adapt to unforeseen circumstances.

The unpredictable nature of input makes it difficult to foresee what will be important to the system's experience, often leading to the development of temporal implications with little relevance. Additionally, resource constraints limit the number of implications that can be retained. Therefore, it is essential to track the success rate of these hypotheses to retain the most effective ones and eliminate those that are less viable. Finding positive evidence is achieved using temporal induction while finding negative evidence is the function of Anticipation. Given a predictive temporal implication of the form $(A \Rightarrow B)$, we define Anticipation as the expectation that the antecedent A will lead to the consequent B being observed as predicted. By leveraging anticipation, a system can identify negative evidence for previously learned temporal implications that led to incorrect predictions.

In MeTTa-NARS, the technical process of forming temporal and procedural implications from events as well as calculation of their evidence involves a sliding-window (a first-in-first-out buffer), a common approach in Data Stream Mining *Pramod and Vyas* (2012). It is implemented using a FIFO queue that only holds the latest k events where k is a hyperparameter. We have chosen the FIFO size of 128 elements to be a good trade-off for performing efficient yet reliable temporal reasoning. This way evidence for implication $(A \Rightarrow B)$ is only attributed when both the antecedent A and consequent B of the implication exist within the sliding window. Please note that A can as well be a sequence here, such as (A, Op) discussed earlier. In principle sequences can contain more than just two elements allowing MeTTa-NARS to learn temporal patterns which span a larger time distance (up to the sliding window size).

Collecting negative evidence is achieved with the Anticipation procedure. The idea is to discount evidence for the hypotheses that have been learned in the past and provide incorrect predictions. To achieve this, whenever a temporal implication is formed in the form of ($antecedent \Rightarrow consequent$), the entire system's concept memory is being queried using MeTTa's pattern matcher which returns all hypotheses with the same antecedent. Now, for the hypotheses having a consequent that is different from the one in just created temporal implication the system decreases their frequency, thus lowering the truth expectation and as a result limiting their selection in the future and expediting forgetting. Overall, the accumulation of positive and negative evidence leads to frequency values which encode the hypotheses (the implications) proficiency to predict successfully. Therefore, truth expectation can be seen as the expected frequency, as it takes into account how many samples have been seen about a certain implication, eliminating initially lucky hypotheses to be preferred over consistently competently predicting ones.

2.3.9 Procedural Inference

Procedural knowledge represents the preconditions and consequences of actions. In traditional planning *Russell and Norvig* (2002), procedural knowledge is provided as a prior knowledge, enabling the system to search for the plan leading to the achievement of a desired state. More sophisticated attempts incorporated probabilities of states and the system was asked to find a plan leading to the highest probability state. In contrast, Reinforcement Learning (RL) does not search for any plans, instead it treats the procedural learning as the selection of the action with the maximum expected utility value *Sutton and Barto* (2018). RL's decision-making focuses solely on the currently observed state, operating under the assumption that it fully represents the current environment. The limitations of the classical approach include its inability to adapt to the changes in the environment during the system operation.

Conversely, RL, having a limited and sparse view of the environment represented by the utility function, struggle to efficiently handle the novel goals.

In our view, AI system must be adaptable as it is not feasible to provide comprehensive preemptive descriptions of environments. Furthermore, a change of goals or conditions should not require re-learning of situation-action associations, instead the system should learn the consequences of its actions such that the new actions can be derived. We further assert that procedural knowledge should be represented through modular structures akin to the procedural implications found in MeTTa-NARS, which can be independently learned and updated to suggest a particular operation in a certain context to realize a stated goal or a derived subgoal.

In MeTTa-NARS, procedural inference pertains to the formation and selection of procedural hypotheses to allow adaptable decision-making and planning. The process relies on temporal patterns of events, represented as a sequence of n consecutive events and operations occurring at a certain time. These events and operations are utilized for constructing procedural hypothesis, designed to capture useful regularities within the system's experience. A procedural hypothesis is formulated as $(A, Op) \Rightarrow B$, where A represents the antecedent serving as preconditions, Op denotes the operation whose execution leads to the consequent B. The operation can be interpreted as an explicitly invoked procedure, and the consequent often represents a goal to be achieved. Each hypothesis has a truth value, which quantifies the degree of belief corresponding to the likelihood of the prediction being confirmed when its precondition is met. Thus, MeTTa-NARS decision-making is focused on achieving a goal by executing the operation that is most likely and sufficiently likely leads to its fulfillment.

Given a goal $G!$, the system realizes it using forward inference, which is hypotheses formation from existing knowledge, and backward inference, which is a derivation of a subgoal from $G!$ and an existing knowledge *Wang* (2013b). When a relevant hypotheses already exists, a single backward chaining step is performed to derive

the operation as a subgoal. However, the relevant hypothesis typically needs to be learned by first deriving the preconditions as subgoals under which $G!$ can be fulfilled. Because of relevant hypothesis are seldom found within system's experience, the decision-making employs multi-step reasoning with an interchange between forward and backward inference deriving subgoals and realizing these subgoals before fulfilling the main goal.

Technically, a goal event $G!$ is represented as a temporal implication $(G \Rightarrow D)$ where D is implicitly present and stands for "desired state", and the desire value of $G!$ is defined as the truth value of this implication. In each system cycle, a goal (if existent) is selected to either trigger a decision (execution of an operation) or lead to the derivations of subgoals. For this purpose, the existing procedural implications of the form $((A_i, Op_j) \Rightarrow G)$ are checked. If the implication has a sufficiently high truth value and event A_i recently happened, it will generate a high desire value for the reasoner to execute Op_j. The truth expectations of the implications with G as a consequent are compared, and the operation from the candidate which leads to the highest desire value expectation will be executed if above the decision threshold (a hyperparameter chosen to be 0.5). Alternatively, all the preconditions (A_i) of the considered implications will be derived as subgoals, competing for attention and processing in the bounded *goal_events* priority queue discussed earlier.

Here, the desire value of the subgoal is evaluated using deduction, between the implication and the goal. To determine desire value of an operation (Op_j), one additional deduction step is required taking the truth value of a precondition (A_i) into consideration. This corresponds to the following inference rule:

$$\{(X \Rightarrow G), (G \Rightarrow D)\} \vdash (X \Rightarrow D) = \{(X \Rightarrow G), G!\} \vdash X!$$

in case X being of the form (A, Op) the rule becomes:

$$\{((A, Op) \Rightarrow D), A\} \vdash (Op \Rightarrow D) = \{(A, Op)!, A\} \vdash Op!$$

which means when this implication is present, system desires to execute Op only after A happens.

The desire values of conclusion goals which are operational precondition (A_i, Op_j) and operation itself (Op_j) are determined as following:

$$desire(A_i, Op_j) = f_{ded}(desire(G), truth(((A_i, Op_j) \Rightarrow G))$$

for the subgoal which corresponds to the antecedent of the implication, and

$$desire(Op_j) = f_{ded}(desire((A_i, Op_j)), truth(A_i))$$

for the operation subgoal to potentially execute if A_i has happened with f_{ded} being as in *Wang* (2013a):

$$f_{ded}((f_1, c_1), (f_2, c_2)) = (f_1 * f_2,\ \ f_1 * f_2 * c_1 * c_2)$$

When there is no such operation to accomplish in a single step, preconditions (A_i) are derived as subgoals from which a candidate might fulfill the requirement or again lead to further subgoaling. The desire value for these subgoals is defined as:

$$desire(A_i) = f_{struct_ded}(desire((A_i, Op_j)))$$

where f_{struct_ded} defined as:

$$f_{struct_ded}(f, c) = (f, 0.9 * f * c)$$

Such approach is similar to backward planning from a goal to current circumstances, while preferring to process more attainable goals by taking uncertainties of events and implications into consideration. Hereby, uncertainties are not assumed-to-be known probabilities as in Probabilistic STRIPS extensions *Teichteil-Königsbuch et al.* (2011) and systems which utilize Probabilistic Planning Domain Definition Language (PPDDL *Younes and Littman* (2004)), but include confidence values for empirical frequency values estimated from data. Additionally, different from *Chaslot et al.* (2008) *Browne et al.* (2012) *Wang and Sebag* (2012), this process derives goals backwards and random rollouts are not assumed to be possible, which also allows for usage in open-ended environments. MeTTa-NARS decision-making and subgoaling process is summarized in Alg. 1.

Algorithm 1: Decision and subgoaling

Input: Goal G **Result:** Execution of Op, or subgoaling
subgoals = {}, bestDesire = 0.0
forall $((X, Op) \Rightarrow G) \in memory$ **do**
  $subgoals = subgoals \cup \{X\}$
  if $desire(Op) > bestDesire$ **then**
    bestDesire = $desire(Op)$, bestOp = Op
  end
end
if $bestDesire > desicion_threshold$ **then**
  execute(bestOp)
else
  forall $s \in subgoals$ **do**
    derive s (for potential selection in next inference step)
  end
end

To allow effective processing, the procedural implications should be indexed by their consequent, where only a constant amount of implications is allowed for each consequent. One way of achieving is by ranking them according to their truth expectation, such that too weak and wrongly predicting implications are removed while those predicting successfully are kept, thus keeping the resource demands bounded.

In MeTTa-NARS, inbuilt pattern matcher is utilized to select all implications with corresponding consequent from both attention focus and concept memory of the system.

Sometimes the operation to execute is ignored and a random one is executed instead, which can be considered a form of exploration through motor babbling. It is necessary in the beginning, when no procedural implications exist, and hence no decision can be derived leading to the desired outcome. Yet sometimes an action should be tried such that the first implications will be formed, and "informed decision" can increasingly replace random trial, e.g. exploitation replacing exploration.

2.3.10 Attentional Control

Selecting "relevant" premises during the inference cycle unequivocally benefits future reasoning outcomes, facilitates effective learning as well as improves the system's overall performance and efficiency. The role of Attentional Control is, therefore, to decide which particular premises are relevant and which data items are important to the system at a given time and context, ensuring they are selected for inference. Additionally, it involves deciding on particular data items to be retained or removed from the system's storage in real-time. Because the objective is to use limited resources to process all pieces of knowledge, and each of them usually requires multiple steps, a selection problem becomes a distribution problem, that is, instead of processing the data items sequentially, each of them gets a "priority" indicating its relative share in the system's overall resource supply in the near future. In MeTTa-NARS, Attentional Control is achieved both implicitly through the implementation of previously discussed data structures and explicitly through methods for estimating *priority* for events, goals, and concepts, as well as *usefulness* for the concepts.

The *priority* and *usefulness* are determined by summarizing all the factors under consideration, including the (relative) urgency, salience, potential, etc., as well as its

relevance to the context. For example, a single inference step cannot be interrupted by more important tasks, however an important task is able to interrupt all the work carried out over multiple cycles, which captures all the problem solving activities of the system. This is facilitated by the *priority* of that task which signifies the short-term importance of the items involved in inference to the current system's context. Additionally, historical factors are important, whether a certain inference path was fruitful in the past in a similar context, which is mostly captured through the *usefulness* of a data item. A key here is to see that the criteria for selection and forgetting are quite different. When a selection is made, the context reflected by the *priority* of a data item usually (but not always) matters more than its historic value. For forgetting, on the other hand, the long-term *usefulness* of the item is of interest, though new data item needs to have a chance to prove its relevance. These considerations, and the need to integrate them effectively, is what makes the design of attentional control for MeTTa-NARS particularly complex. Below we will provide key insights on how *priority* is estimated and updated, as well as how *usefulness* affects the system's forgetting process.

Priority Estimation: Priority value p measures short-term importance of an item to the current context and experience of the system ensuring the item will receive more attention and expedited processing as a result. A timestamp t, indicating when the priority value has been assigned, accompanies the priority value allowing to keep track of time, and enabling continuous decay of p as time progresses. Priority value ranges from 0 to 1 with the highest priority of 1 reserved only for events and goals entering the system. Finally, priority value is being assigned to events, derivations, goals and sub-goals within the priority queues as well as concepts in the concept memory in real-time during inference cycle.

Because there is no efficient method to adjust the priority value of each data item within a data structure after every inference step, during the selection of a data item,

its priority value is being discounted by the time difference to the current system's time. This approach ensures the time relevance of selected item is accounted for, and that the priority decay rate is proportional to this time distance as:

$$\frac{p}{1 + (T - t)}$$

where T is the current system time and t is the priority value timestamp. This also can be seen in the Fig 2.3.

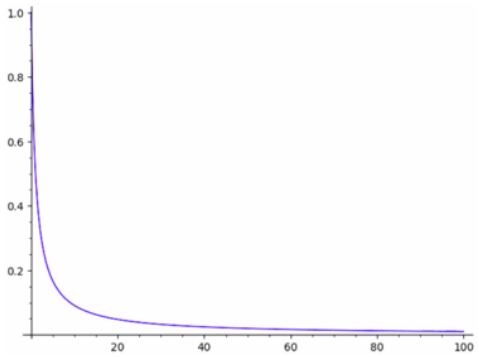


Figure 2.3: Priority Decay with Time of a Data Item.

The estimation of priority values occurs for derived events during the declarative inference cycle and for sub-goals during procedural inference, particularly when a sub-goal is deduced from a given goal. For derived events, the priority is calculated by multiplying the parent event's priority (from the first premise) by the priority of the belief concept (associated with the second premise) and the truth expectation of the conclusion event. For sub-goals, the calculation is similar, but without the involvement of a belief concept, hence that factor is omitted. This approach guarantees that all elements contributing to the derivation are factored into the priority value of the conclusion. A concept's priority initially inherits the event's priority when the concept is created. During belief revision process, the concept's priority updates to the higher value between its current priority and the event's priority that triggers the revision.

Usefulness estimation: In addition to the priority, the usefulness value is assigned to concepts within the system's concept memory. It reflects the long-term importance of a concept and plays a crucial role in the system's forgetting process. Essentially, it determines the duration a concept should remain in the concept memory before it is removed. In MeTTa-NARS, forgetting involves removing concepts with the lowest usefulness from long-term memory during each system's cycle, provided the number of concepts exceeds the storage capacity.

Usefulness is determined by two critical factors: how frequently a concept is used and how recently it was last used. This involves tracking both the concept's use count and its recency. The use count is straightforwardly increased each time the concept is selected for inference, while recency measures the interval between the current system time and when the concept was last utilized. Having this in mind the usefulness is defined as:

$$usefulness = \frac{usefulnessRaw}{usefulnessRaw + 1}$$

where $usefulnessRaw$ is

$$usefulnessRaw = \frac{use_count}{recency + 1}$$

and

$$recency = current_time - last_used$$

In conclusion, the integration of *priority* and *usefulness* within the system's Attentional Control is essential for managing the selection, retention and removal of data items based on their short- and long-term importance. By balancing priority and usefulness values, the system effectively manages the process of selection and forgetting, ensuring that only the most relevant knowledge is utilized during inference in real-time.

2.4 Related Works

In this section, we will provide a brief overview of several notable cognitive architectures and reasoning systems, such as SOAR *Laird* (2012), ACT-R *Anderson et al.* (2004), LIDA *Franklin et al.* (2016), and a few others. These systems are among the most widely used and exhibit structural properties that are relevant to the scope of this thesis.

SOAR is one of the oldest cognitive systems used by numerous researchers during the last decades. SOAR exhibits complex multi-component design encompassing different architectures and consisting of various memory structures, learning mechanisms and a decision cycle that links the perception (inputs from the environment) to actions (system output). The memory types in Soar are separated into long-term and short-term (working) memory components *Chong et al.* (2007). The information from the environment is available in the working memory through use of perception components with dedicated perceptual memory, while external environment is influenced through implementations of system selected actions. The representation of knowledge is entirely symbolic with pattern matching mechanism employed to retrieve relevant knowledge elements from the long-term declarative memory. SOAR's long-term memory can be classified into semantic, procedural and episodic types with semantic memory being considered as declarative. Procedural and semantic memories are universally applicable during the reasoning process, where semantic memory stores general experience and description of the environment, and procedural memory provides knowledge for performing actions. Episodic memory is used for knowledge, which is specific to a certain content, conceptually speaking, it contains information about specific events. When procedural memory is incomplete for solving a particular task, knowledge is being sourced from semantic and episodic storage to assist with reasoning. Content of the short-term memory, also known as working memory elements, describe all the knowledge that is relevant to the current context, in partic-

ular, it contains system states, goals, perceptions and operators. If working memory content is insufficient, working memory elements often retrieve relevant knowledge from different types of long-term memory.

ACT-R is a cognitive architecture explicitly inspired by theories of human cognition and empirical data from experiments in cognitive psychology and brain imaging. ACT-R incorporates a unique architecture consisting of four modules: visual, manual, declarative and goal. The visual module serves two purposes for recognizing an object and identifying its location within the visual field. The manual module is used for the control of actuators. Memory retrieval procedures are accomplished within the declarative module, and the goal module monitors the agent's current goals and enables the maintenance of the agent's thoughts in the absence of supporting external stimuli *Anderson et al.* (2004). For each of these modules a dedicated buffer is utilized. Hence, the goal buffer helps to keep track of internal states during problem solving; the retrieval buffer stores the chunks of declarative memory retrieved from the long-term memory; the manual buffer is used for controlling the agent's hands and associated with the motor and somatosensory cortical areas *Chong et al.* (2007); and the visual buffer include both the dorsal 'where' visual pathway system and the ventral 'what' system which are essential for locating, identifying and tracking objects. ACT-R features a central production system coordinating all the modules using implemented set of IF-ELSE production rules and functioning as a basal ganglia in human brain. The communication between the central production system and the modules happens through the information present in the buffers, thus limiting the response of the central production system to the amount of information available within the buffers of various modules. In terms of memory types, ACT-R distinguishes between two types of knowledge: declarative and procedural. Knowledge within declarative memory is represented as chunks and describe explicit facts known to the system while knowledge in procedural memory encodes production rules for processing declarative knowledge.

Declarative memory is considered system's long-term memory while short-term memory is available through the use of limited information within the buffers. ACT-R does not include a dedicated working memory where different types of knowledge are processed, instead it uses a distributed memory system, wherein the goals, beliefs, sensory, and motor signals are situated in distinct buffers.

Learning Intelligent Distribution Agent (LIDA) is a notable example of multi-component cognitive architecture that aims to model human consciousness *Franklin et al.* (2016). Logical part of LIDA enforces Global Workspace Theory (GWT) *Baars et al.* (2007)) and implements cognition process in a serial way through use of system cycles. LIDA's control mechanism is immensely complex, it incorporates large number of components with independent architectures which can be symbolic or connectionist. A cycle in LIDA models human cognition and mainly consists of three phases: sense, attend and action selection *Helgason* (2013), where "attend" phase is implemented through use of attention *codelets* each looking for "interesting" situation and attempting to bring it to "consciousness" modeled by Global Workspace. Attention *codelets* similarly to daemons look into Workspace and Episodic memory, form coalition of "appropriate" data and bring it into the Global Workspace. Attention is implemented as a filtering process which allows the system to handle information overload situations. LIDA utilizes numerous memory modules each with different structures: Sensory, Sensory-Motor, Spatial, Perceptual Associative, Transient Episodic, Declarative, and Procedural. Since examining every module is out of scope of this section, only few are highlighted here. Perceptual Associative Memory (PAM) is a module that senses the incoming sensory information. It contains feature detector processes for feature detection within Sensory Memory. PAM uses a semantic net implementation with activation passing. There are two episodic memories in LIDA, Transient Episodic Memory and Declarative Memory, which are both implemented using a sparse distributed memory *Franklin et al.* (2016). In Transient

Episodic Memory knowledge decays after a few hours or up to a day and Declarative memories are formed offline from transient episodic memories by a consolidation process. Procedural Memory stores schemes which can be activated and sent for action selection. Working memory can be found within Workspace module which receives and stores content from several components including precepts from PAM, recent local associations from Episodic Memory, and the recent contents of consciousness from the Global Workspace. Multiple LIDA's memory modules fall into long-term and short-term memory categories. In particular, PAM, Spatial, Declarative, and Procedure modules are viewed as long-term memory, while Sensory and Transient Episodic memories fall under short-term category.

One of the earlier systems which share attentional control ideas with NARS is the Copycat project *Hofstadter and Mitchell* (1995). While Copycat can be applied to very narrow domain it features sophisticated attentional and resource allocation mechanism incorporating ideas similar to ones found in more complex systems. Copycat is a system that tries to discover and build analogies in psychological realistic way, its main idea is to allow mental fluidity achieved by concept slippage which results from building pressure during real-time processing *Rehling and Hofstadter* (1997). Copycat control system has three main sub-components: Slipnet, Workspace and Coderack. Slipnet is system's main memory and can be considered as a long-term memory represented by graph where concepts are nodes connected by edges as numerical distances between them. Concepts are used in building bigger structures in the Workspace, and the Coderack is a place where *codelets*, small working agents, are being created and chosen to complete the work in the Workspace. Copycat uses Parallel Terraced Scan *Rehling and Hofstadter* (1997) as a resource allocation mechanism in order to proceed and discover promising structures in the Workspace to be further developed. Function of attentional control is to decide which structure to pick from the Workspace and which *codelets* should be allowed to work. Each struc-

ture in Workspace is assigned with *salience*, a dynamic quantity, that determines the probability of acquiring attention from *codelets*. A *codelet*, when created, is placed in a pool with *urgency* value that determines probability of being selected as the next *codelet* to run. *Urgency* estimates an importance of *codelet*'s action which in turn reflects the current state of the system. *Urgency* of a *codelet* is not its priority but rather a relative speed at which the pressures represented by this *codelet* should be attended to. Calculation of *urgency* and *salience* depends on many system's factors and impacts overall dynamics and performance.

Auto-catalytic Endogenous Reflective Architecture (AERA) is another interesting AGI system that has been designed as a part of HUMANOBS project: a system able to learn socio-communicative skills by observing people *Nivel et al.* (2013). AERA incorporates an unusual model-based architecture with unique approach to attentional control. In AERA, knowledge is represented as models which are of two types: *forward models* that predict the behavior of entities and *inverse models* that prescribe actions to be taken. Models themselves are executable and its execution is in turn controlled by other models, making the architecture model-driven. Models and other components in AERA are built using low-level building blocks which are objects defined in a specially designed programming language called *Replicode*. There are four fundamental high-level processes which are being continuously performed in a concurrent fashion: Model Acquisition, Reaction, Model Revision and Compaction. Attentional control in AERA can be seen as a part of the Reaction process. In general, the Reaction activities determine the course of an action to pursue goals and are carried out by Attentional Control whose function is to continuously control input saliency within the system' specified short time horizon. This approach results in reduction of inputs to the models and more importantly input saliency control allows salience spreading across other objects in the memory including goals and execution traces.

At present time OpenCog and its implementation OpenCogPrime (OCP) is an advanced and complex integrative AGI system. OCP uses Economic Attention Networks (ECANs), a sophisticated way for attentional control and resource allocation *Ikle et al.* (2009). ECANs share similarities in architecture with connectionist systems but the spread of activation within the system uses equations based on ideas borrowed from economics rather than neural modeling. ECANs are represented as a graph that consists of nodes, links and also HebbianLink and InverseHebbianLink. Each item in a graph, a node or a link, is referred as an *Atom* and assigned two values: Short-Term Importance (STI) and Long-Term Importance (LTI). STI of an *Atom* indicates its immediate urgency at a point in time while LTI represents the amount of value in keeping an *Atom* into the memory. ECANs also integrate a "forgetting" mechanism that removes certain percentage of *Atoms* with lowest LTI values. Each HebbianLink or InverseHebbianLink designates probability value, that is, given a HebbianLink from A to B it shows the probability if A being in attentional focus so is B, and given InverseHebbianLink from A to B it shows the probability if A being in attentional focus so B is not *Ikle et al.* (2009). ECANs use defined "economics" equations to update system's values dynamically over time as well as Attentional Focus that treats *Atoms* differently with STI values above certain threshold. Additionally *Atoms* are able to spread its LTI and STI between other *Atoms* connected with HebbianLink or InverseHebbianLink.

2.5 MeTTa-NARS Code Structure and Usage

When implementing a high-level symbolic reasoning system, the codebase often grows very complex, with many interconnected components. However, one of our key objectives was to create a practical reasoning system with a modular structure in a compact package. The choice of programming language is essential to achieving this, therefore we selected novel MeTTa functional language. This choice allowed

us to elegantly build a fully functional, robust prototype that is both modular and straightforward to understand.

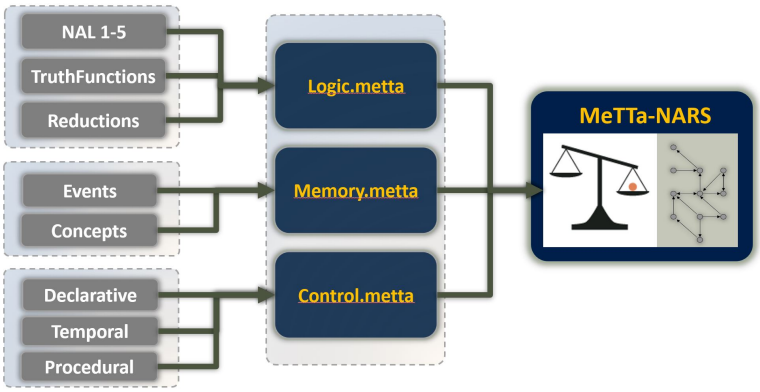


Figure 2.4: Code Structure Flowchart.

The overall structure of the codebase is illustrated in Fig. 2.4. At the root of the directory, the build script *build.sh* handles the code building process. Due to the MeTTa language lacking a reliable code importing feature—at least at the time this thesis was written—the build script functions by simply concatenating smaller source files. The build process involves two intermediate steps: first, consolidating the code into three larger components, and finally merging these into a single reasoner. Once the build is complete, the main source file *NARS.metta*, featuring the entire reasoner's MeTTa code, becomes available. MeTTa-NARS consists of three main components, each represented by a MeTTa source file correspondingly named during an intermediate build step such as:

- **Logic**: logic.metta
- **Memory**: memory.metta
- **Control**: control.metta

Each of these components is assembled from several smaller source files. Specifically, the logic component is constructed using sources for NAL inference rules, NAL

Truth Functions, and Term Reductions. The system's memory is divided into Events and Concepts. The control component is comprised of several sub-parts, each corresponding to a different type of reasoning as:

- Declarative: declarative cycle with derivation of events from events and concept's beliefs.
- Temporal: temporal compounding (sequence and temporal implication formation), anticipation and projection .
- Procedural: decision making with goal processing, sub-goal derivation and planning.

As noted, MeTTa-NARS has a highly modular structure, which facilitates easy modifications, debugging, and comprehension. This design is intended to allow for the swapping of one module with another. For instance, should a new version of procedural control be developed in the future, this structure enables the simple replacement of the current procedural control with the new version by merely substituting one smaller MeTTa file for another.

To run MeTTa-NARS, you will need to have either the full MeTTa interpreter or a translator installed. Please follow the installation instructions for the MeTTa interpreter available on the official MeTTa website[2]. Once the interpreter is installed, download the source code and run the system interactively using the following commands:

```
git clone https://github.com/pisaev1/metta-nars
cd metta-nars/
sh run_metta_repl.sh
```

[2] https://metta-lang.dev/

Executing the last command will start the system in interactive mode, where you'll be prompted to enter inputs, as illustrated in Fig. 2.5.

For more efficient and reliable performance, we recommend using our MeTTa-Morph, a macro-based MeTTa to (Chicken) Scheme translator. To run MeTTa-NARS with MeTTa-Morph, use the following commands:

```
git clone https://github.com/trueagi-io/metta-morph
cd metta-morph/
sh install_nars.sh
cd metta-nars/
sh run_scheme_repl_interpreted.sh
```

Executing the last command will start the system in interactive mode, but with the MeTTa to Scheme layer. Similar to running the full MeTTa interpreter, you'll be prompted to provide input to the system, as seen in Fig. 2.5.

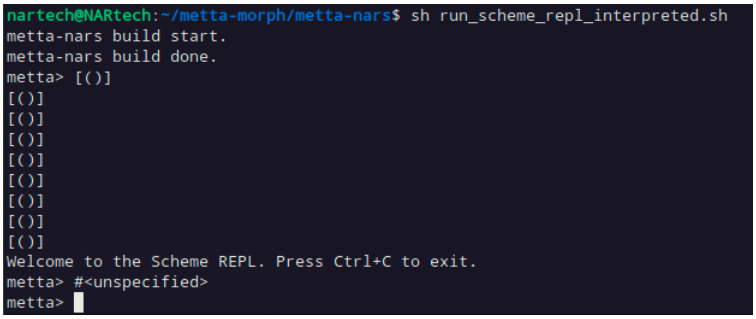


Figure 2.5: MeTTa-NARS Prompt in Interactive Mode.

Entering inputs into the system is done by typing *Narsese* sentences, as indicated in Fig. 2.6.

When entering inputs, use `AddBeliefEvent` for input events and `EternalQuestion` for questions on eternal events (beliefs) as illustrated in in Fig. 2.6. Also the last line in Fig. 2.6 shows the system's answer to the user's question, which includes the system's derivation. The package also includes a set of tests to thoroughly verify the system's functionality:

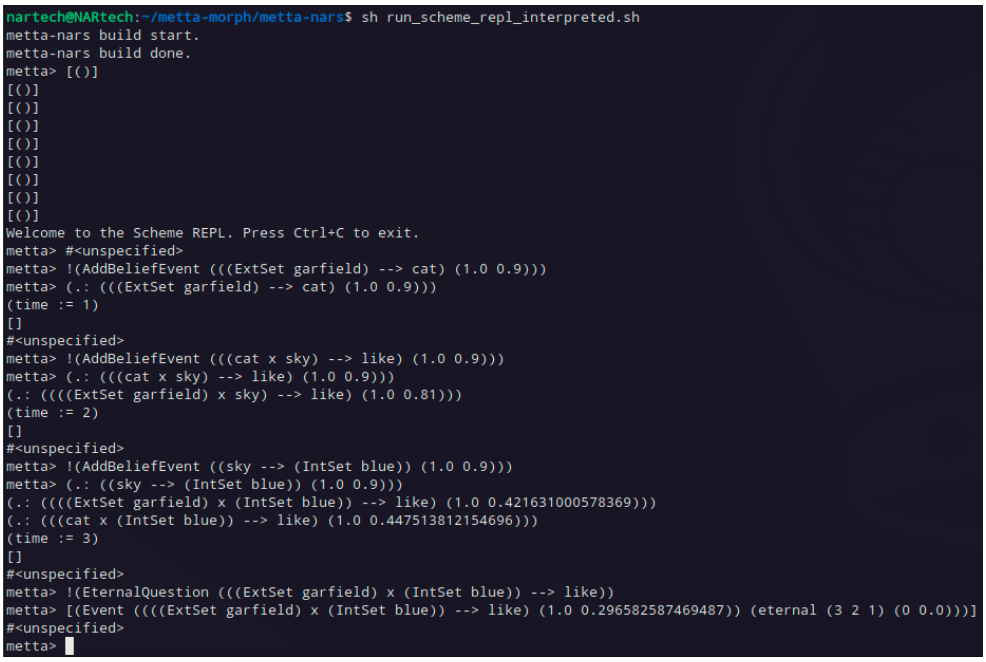


Figure 2.6: MeTTa-NARS Inputs and Derivations.

- **tests0.metta**: NAL basic inference rule application examples
- **tests1.metta**: Multi-step declarative inference example
- **tests2.metta**: Pong-like procedure learning example
- **tests3.metta**: Multi-step decision making (planning) example
- **tests4.metta**: Higher order multi-step inference
- **tests5.metta**: Higher order multi-step procedural learning example

To run any test or input using a MeTTa file, use the following command:

```
sh run_scheme.sh ./tests/tests0.metta
```

CHAPTER 3

COMPARISON WITH EXISTING NARS IMPLEMENTATIONS

This chapter outlines the key differences among the implementation designs of MeTTa-NARS, OpenNARS, and OpenNARS for Application (ONA), with a particular focus on inference control and memory aspects.

3.1 OpenNARS v3

In recent years, several stable versions of OpenNARS have been developed, notably v.3.04 *Hammer et al.* (2016) and v.3.1.2 *Wang et al.* (2020). OpenNARS v3 continues to be a solid implementation of NARS theory, encompassing all forms of reasoning, including declarative, temporal, and procedural, as well as conceptual memory representation. However, OpenNARS is more oriented toward research than practical application. Its structure is not particularly well-organized, and it incorporates several ideas in an inefficient manner, which require justification for their inclusion.

3.1.1 Data Structures

OpenNARS v3 uses *Bag* (*Wang* (2013a), *Wang* (2006)) data structure for main concept memory where the elements are sorted according to their *priority*, and the sampling operation chooses candidates with selection chance proportional to their *priority*. This makes the control strategy similar to Parallel Terraced Scan *Rehling and*

Hofstadter (1997), as it also allows to explore many possible options in parallel, with more computation devoted to options which are identified as being more promising. Notably, *Bag* is different from a priority queue, which just selects the highest priority option, in *Bag* every element even the one with the lowest *priority* has an opportunity to be selected. After selection, a candidate is reinserted into the *Bag* with a reduced *priority*. The distinction becomes more evident here: with the priority queue, even after reinsertion, a concept's priority might remain sufficiently high or even be the highest, leading to its selection in the next cycle. However, in the *Bag* structure, there remains a probability that it might not be selected, allowing a lower-priority concept to be chosen instead. As mentioned in the previous chapter, MeTTa-NARS uses an attentional focus, which is similar in nature to the priority queue, comprised of the k highest priority concepts, all of which are selected for inference during each cycle. Importantly, concepts are not removed from the attentional focus; rather, new inputs or derivations are added to the attentional focus during each cycle. Given the number of concepts exceeds the capacity of the attentional focus, concepts with the lowest priority are being moved to the long-term memory, where they may eventually be forgotten. Such approach makes the system behave in a more predictable manner, and therefore focused on a specific task, while for OpenNARS pseudo-randomness plays an important role in selection.

3.1.2 Data Items and Inference Cycle

While both MeTTa-NARS and OpenNARS utilize a concept-centric memory structure, represented as a conceptual network with prioritized nodes, they differ in the way they handle data items and process knowledge. In OpenNARS, each sentence, along with its sub-terms, forms multiple concepts. For example, a statement like $(S \rightarrow P)$ is processed into a concept for the term $(S \rightarrow P)$ itself, as well as concepts for S and P. A "concept" in this context is a data structure named by a term

that includes links to all tasks and knowledge that can be directly used in inference steps where that term is shared. In practice, a term T names a concept C_T, which contains a *Bag* of tasks and beliefs that can be directly utilized in inference with T as the shared term. Because of MeTTa-NARS focusing on temporal and procedural inference components—where premises are related on a temporal basis rather than strictly sharing terms during declarative reasoning—there is no explicit representation of task- or term-links. Instead, complete statements, including inheritance terms, similarity, negation, sequences, and implications, form concept nodes. Their sub-terms are treated as components within these concepts, and directly managed by MeTTa's pattern matcher during the inference cycle.

In OpenNARS, the use of *Bag* structures within concepts introduces additional selections for task- and term-links, featuring some probabilistic randomness. Furthermore, since all concepts compete for resources and undergo selection within the same main memory, which is also organized as a *Bag*, there's a chance that a concept representing a sub-term with only task- and term-links back to its parent task (original sentence) could be selected, leading to a wasted cycle as that term is actually not shared with other beliefs. Contrastingly, in MeTTa-NARS all the concepts within the attentional focus are deterministically selected to participate in reasoning, resulting in a robust and effective inference cycle.

The difference in data structures between MeTTa-NARS and OpenNARS becomes more pronounced when examining temporal inference. MeTTa-NARS utilizes a sophisticated yet streamlined version of the principles discussed in *Hammer and Lofthouse* (2018), employing a FIFO structure as a sliding window approach with deterministic selection of temporal compounding. This contrasts with OpenNARS v3, which uses an *event bag* with non-deterministic selection. Furthermore, in MeTTa-NARS, the decision-making rule governing whether a subgoal should be derived depends on whether the precondition is sufficiently fulfilled to achieve. In OpenNARS,

however, subgoals are derived whenever the goal is selected, leading to potential issues in coordinating behavior that may or may not be addressed by an improved version of the attentional control.

3.1.3 Attentional Control

OpenNARS uses a *Budget Value* which is a set of rational values assigned to each data item. It is used to control how much processing should be dedicated to a data item within the system. *Budget value* is a triplet (p, d, q), where p, *Priority*, measures short-term importance, d, *Durability*, a decay rate, describes how fast *Priority* of an item should decay, and q, *Quality*, indicates the long-term importance of the data item. Moreover, OpenNARS employs complex Budget Inference control which determines the budget value for every data item participating in the inference cycle. MeTTa-NARS uses a much simpler and more transparent way of measuring the importance of data items.

As previously discussed, *priority* in MeTTa-NARS is assigned to denote short-term importance and serves as a criterion for selection, while *usefulness* helps determine how long an item should stay in long-term memory based on Least Frequently Used and Least Recently Used principles. Although the Budget Inference in OpenNARS contains many solid theoretical ideas, it is often challenging to track budget value assignments in practice. Also, given that OpenNARS incorporates more types of data items such as term-links and task-links, inaccurate budget value assignments can lead to unsatisfactory derivation results. In contrast, MeTTa-NARS lacks these intricate components, resulting in a system that is easier to fine-tune and retains data items more predictably. However, this simplicity might also mean missing out on potentially relevant factors in resource allocation, although we did not observe this in any of our test cases.

3.1.4 Goal Handling and Subgoaling

In MeTTa-NARS, priority values range from 0 to 1, with the highest priority of 1 specifically allocated for events and goals as they enter the system. This means that during inference, any derived sub-goal will never have a priority higher than the original goal from which it was derived. This approach ensures that once a goal is set, the system remains focused on it and isn't distracted by its own derivations. Conversely, OpenNARS employs a different strategy, allowing the system to autonomously determine priority for goals. This can lead to sub-goals being assigned a higher priority than the initial goal, which, while theoretically feasible, often results in the system becoming unresponsive as it prioritizes its own internally generated goals over those provided by users. As the sub-goal chain extends, the priority of the original goal decreases, sometimes causing the system to completely forget the initial objective. This is akin to a person becoming distracted by circumstances and losing sight of their original intention.

3.2 OpenNARS for Applications

OpenNARS for Applications (ONA) is designed with a focus on efficient implementation and stable inference control. Its architecture supports repetitive inference across novel situations, utilizing generalized knowledge that is either provided or learned. During extended learning sessions, ONA effectively acquires knowledge from event streams while maintaining stable inference, ensuring new information enhances rather than disrupts the system's ability to apply prior knowledge to future problems. This supports a form of homeostatic inference, where learning and reasoning remain balanced over time. This stands in stark contrast to earlier NARS implementations, whose inference control mechanisms often undermined their ability to adapt to rich, prolonged streams of experience.

To reinforce inference control stability, ONA utilizes the following conditions which were derived and extended from the ALANN *Lofthouse* (2019) control model:

- Events "activate" concepts of same or matching (in case of variables) terms. However a concept's priority cannot exceeed the maximum priority of recent events that matched to it.

- The priority of a derived task is determined multiplicatively by the priority of its parent task, the priority of the associated belief concept, and its truth expectation. Consequently, a derived task's priority can never exceed that of its parent task or the belief concept from which it originates, and can only be of high priority if both parent "items" are not dormant.

3.2.1 ONA and OpenNARS

Additional design elements that differentiate ONA from OpenNARS and enhance its effectiveness include:

- Semantic linkage between concepts via its subterms is monitored through a special indexing structure. Subterms do not independently form concepts as they are not of the same nature and hence should not be judged equally in the competition.

- Concepts are forgotten purely based on use count and the time since last use—a more reliable and superior forgetting heuristic than those requiring multiple complex measures produced by Budget Inference in OpenNARS.

- Predictive knowledge is stored in fixed-capacity tables within each concept and competes solely based on the ability to correctly predict the same postcondition (the consequence of the stored temporal or procedural implications). This enables concepts to learn the conditions under which they are most likely to

encounter matching events, which makes predictive implications of same consequence compete in the same space, and allows the system to discover effectively which conditions it must realize to achieve a desired outcome.

3.2.2 ONA and MeTTa-NARS

While MeTTa-NARS incorporates numerous elements from ONA's control model, it exhibits several differences to facilitate a more streamlined and modular implementation in the MeTTa language:

- In MeTTa-NARS, predictive implications form concepts directly, rather than being stored in specialized table structures within them. This makes concepts more lightweight, easier to express in the MeTTa language, and more practical for the interpreter to handle. However, we acknowledge that this design choice reduces MeTTa-NARS's effectiveness in learning highly predictive temporal implications.

- Variable matching at the concept level for effective use of generalized procedural knowledge is currently too demanding for the MeTTa language interpreter. Therefore, events only match concepts with identical terms. As the MeTTa interpreter evolves supporting automated indexing for common queries, serving a similar role to ONA's specialized indexing structures, match statements with variables will become efficient. Enabling variable matching to effectively leverage generalized procedural knowledge is planned for future work as the interpreter evolves and will come along with variable introduction in inductive inference.

  For now, query variables are instantiated by concrete matching beliefs, allowing unification in declarative inference. This encourages generalized declarative knowledge:

```
!(AddBeliefEvent (((((ExtSet $1) x paintings) --> create) ==>
                      ((ExtSet $1) --> artist)) (1.0 0.9)))
```

while generalized procedural knowledge is not yet supported:

```
!(AddBeliefEvent ((((($1 --> (IntSet see)) &/ (^ count)) =/>
                      ($1 --> (IntSet counted))) (1.0 0.9)))
```

Further differences include special features of compound terms that will require future implementation works:

- ONA supports operator arguments rather than just atomic operations, as in MeTTa-NARS.
- ONA supports comparative relations and value terms, allowing it to reason on different values of the same property of an objects, such as size or brightness. This feature has not been implemented in either MeTTa-NARS or OpenNARS.

Due to the modularity of MeTTa-NARS, the logic is defined in MeTTa language and is strictly separate from Control and Memory. In contrast, ONA requires a special complicated handling of declarative inference within its control. Additionally, the entire system is more compact, only 20% of ONA, making it easier to support.

3.3 MeTTa-NARS Limitations

MeTTa-NARS addresses several major issues found in OpenNARS, particularly regarding its application scope and the complexity of its structure. However, MeTTa-NARS does have limitations, some of which stem from design choices aimed at ensuring predictability and reliability of the system. While the reasoning process is a bit more constrained, as previously discussed, these design choices allow for results that are more easily traceable and assessable.

The attentional span of MeTTa-NARS is more short-term, with reasoning largely concentrated on the present moments rather than on system's past or future, and induction abilities are constrained to facilitate the formation of temporal and procedural hypotheses in a robust manner. Additionally, MeTTa-NARS's reasoning process exhibits less variance, making it less sensitive to minor environmental or application variations where consistent outcomes are anticipated. On the other hand, OpenNARS functions with a more relaxed reasoning process, often oriented towards the system itself rather than remaining focused on user and application demands.

Compared to ONA, MeTTa-NARS features a significantly simpler structure, which facilitates easy comprehension. Its modular design further enhances its capacity for experimentation, allowing for quicker and easier testing of new ideas and paradigms. However, in terms of performance, ONA is considerably more efficient because it is written in the C language, and utilizes a wide range of highly efficient data structures.

We will now discuss the most important limitations of MeTTa-NARS:

- **Performance:** While the performance of MeTTa-NARS is comparable to, or even exceeds, that of OpenNARS, it remains its most significant limitation. In general, high-level interpreted functional languages tend to have slower performance compared to lower-level compiled languages. Furthermore, the novelty and immaturity of the MeTTa language mean that MeTTa-NARS is highly dependent on the version of the MeTTa interpreter being used. Although there are translators for MeTTa, such as our own Metta-Morph[1] (derived from Metamorphosis), a macro-based MeTTa to (Chicken) Scheme translator that translates MeTTa into Scheme and subsequently into C, this approach limits the full potential of MeTTa usage and still doesn't match the performance of ONA, which is fully implemented in C. It is anticipated that in the near future, the MeTTa interpreter will mature, offering more efficient and reliable interpretation and

[1]Please see the details at `https://github.com/trueagi-io/metta-morph`

execution of MeTTa code including efficient management of MeTTa's Atom spaces.

- **Temporal Compounding:** As previously mentioned, the formation of temporal sequences is exhaustively managed by a FIFO queue, which stores events in the order they occur. However, the current implementation does not allow sequences to skip events. Without skipping, the potential number of sequences ranges from polynomial $O(n^2)$ to exponential $O(2^n)$ for a particular state of the FIFO with the maximum size of n. Clearly, it is manageable only for small FIFOs. Potentially, incorporating attentional control to selectively skip events might be a solution to keep the polynomial-bounded number of sequences. Alternatively, one could limit the number of events that can be skipped in sequences. For example, given a sequence of events A, B, C, D, the system would be able to skip event B forming the following sequence: $(A, C) \Rightarrow D$. However, the impact of skipping events on the inference process remains to be tested to determine its effectiveness.

- **Variables:** At its current stage, MeTTa-NARS can utilize variables through unification, using positional roles (embedded in conjunction or implication on both or one side) to determine their logical types. However, it does not yet introduce variables of any kind during the inductive inference. Additional inference rules for variable introduction could easily be added using the same format as the existing rules implemented within logic component of MeTTa-NARS, however a complete handling would require term counting and efficient algorithmic processing of the term structure to introduce variables independently from where the terms are positioned. This would require extra work to restructure and optimize the inference control of the system which is reserved for the future work.

CHAPTER 4

NON-AXIOMATIC CAUSAL EXPLORER: SUB-SYMBOLIC GRID-WORLD AGENT

In this chapter we present a novel sub-symbolic experiential learning agent with causally-informed intrinsic reward that is capable of learning sequential and causal dependencies in a robust and data-efficient way within grid world environments. After reflecting on state-of-the-art Deep Reinforcement Learning algorithms, we provide a relevant discussion of common techniques as well as our own systematic comparison within multiple grid world environments. Additionally, we investigate the conditions and mechanisms leading to data-efficient learning and analyze relevant inductive biases that our agent utilizes to effectively learn causal knowledge and to plan for rewarding future states of greatest expected return.

4.1 Introduction

Grid world environments come in many forms and have been studied extensively in the history of Artificial Intelligence with some notable examples such as Wumpus World (*Bryce*, 2011), Minigrid (*Chevalier-Boisvert et al.*, 2024), and Tileworld (*Pollack and Ringuette*, 1990). However, the creation of intelligent grid world agents capable of learning effectively and in a data-efficient way has posed significant challenges. Current Reinforcement Learning (RL) agents struggle in some instances due to sequential dependencies, partial observability (*Wang et al.*, 2023a), continual learning (primacy bias *Kim et al.* (2024), stability-plasticity dilemma *Anand and Precup*

(2024)), relatively high-dimensional state spaces, compared to more traditional RL tasks – even when a single value per cell is provided rather than per pixel – and sometimes non-deterministic effects of actions.

Sequential dependencies usually raise the training data demand exponentially depending on the combinatorics, and, ultimately, the number of the arising options, unless the agent is capable of learning causal representations that transfer well because chaining them is favorable for reaching an intended outcome. Partial observability, on the other hand, may require the model to have access to information from prior states, which typically corresponds to the previously observed values in a grid world outside the agent's current field of view. In terms of input dimensionality, there is a trade-off between the observation window being too small for learning an effective policy and the agent's observation window being more high-dimensional, thereby demanding more training data, even though this might be the least problematic. Additionally, if the agent is able to represent values outside of its observation window, a learned policy needs to consider not only the observation window itself but also how it spatially relates to the remembered information beyond it.

With these considerations in mind, we introduce Non-Axiomatic Causal Explorer (NACE), a novel experiential learning agent, which leverages causal reasoning and intrinsic reward signals to enable more efficient learning as well as possesses learning mechanisms with the involved inductive biases. NACE is designed to induce causal rules from temporal and spatial local changes in the grid, which are often (but not always) caused by the agent. It utilizes these rules to plan for and reach future states of maximum uncertainty in order to effectively learn more (causal rules) about the environment, thereby improving predictability-based intrinsic reward formulations.

To illustrate the effectiveness of our approach, we provide a comprehensive discussion of state-of-the-art Deep Reinforcement Learning (DRL) techniques as well as our own systematic comparison within multiple grid world environments demonstrat-

ing our agent's remarkable improvement in data efficiency, achieving similar performance with about 1000 samples where DRL algorithms typically require 1 million samples, representing a 1000-fold reduction in data requirements. We thoroughly investigate the conditions and mechanisms under which learning in grid worlds can become more data-efficient and attempt to answer the question about which inductive biases can lead to close-to-optimal learning speeds in the case where the agent is not pre-equipped with particular interaction rules between grid cell types but has the inductive biases to build and use them effectively. Lastly, we analyze useful inductive biases applicable among a wide range of grid worlds and their generality to raise the data efficiency of learning in such domains.

4.2 Related Work

Current Deep Reinforcement Learning techniques, while being powerful, are often limited by their need for large quantities of data to achieve optimal performance and therefore face a significant challenge of sample efficiency, which is the ability to learn effective policies from few interactions with the environment. RL techniques such as value-based (Deep Q-Learning (*Mnih et al.*, 2013)) and policy gradient-based (Proximal Policy Optimization (*Schulman et al.*, 2017)) require millions of training iterations to succeed in grid-worlds (*Zhang et al.*, 2020b), struggling to capture causal dependencies necessary for efficient planning and transferability.

Exploration improvements, such as intrinsic rewards based on information gain (*Zhao et al.*, 2023), prediction errors (*Burda et al.*, 2018), or visitation counts (*Zheng et al.*, 2021; *Wang et al.*, 2023c), enhance sample efficiency but often lack structured reasoning for generalization. In contrast, *Tsividis et al.* (2021) propose Theory-Based RL (TBRL), exemplified by EMPA (Exploring, Modeling, and Planning Agent), which integrates Bayesian causal modeling, structured exploration, and heuristic planning to generalize efficiently across tasks with minimal training. Similarly, GALOIS

(*Cao et al.*, 2022) addresses generalization by synthesizing interpretable, hierarchical programs with strict cause-effect logic, though its reliance on predefined program sketches limits flexibility in loosely structured environments. As a model-based RL alternative, DreamerV3 (*Hafner et al.*, 2023) learns latent state dynamics and improves behavior through imagination, enabling generalization across diverse tasks with minimal domain-specific adjustments. However, its reliance on learning both latent representations and their dynamics reduces sample efficiency compared to methods that assume predefined representations.

Symbolic approaches such as STRIPS or Behavior Trees (BTs) (*Guo et al.*, 2023; *Colledanchise and Ögren*, 2018) handle human-defined causal knowledge lacking adaptive learning capabilities. POMDPs (*Spaan*, 2012) focus on probability updates rather than causal discovery, while causal networks (*Pearl*, 1995) and structure-learning methods (*Zheng et al.*, 2018) face challenges with ambiguity and scalability.

While TBRL, GALOIS, and DreamerV3 provide solutions for structured or model-based learning, their reliance on predefined logic, priors, or latent space learning introduces additional complexity. Our approach proposes lightweight "empirical" causal relations learned from recurring cause-effect patterns, supporting efficient real-time learning in grid worlds without predefined program sketches, Bayesian modeling, or latent space learning. This ensures adaptability while retaining interpretability and transferability.

4.2.1 Selected RL Techniques in Grid Worlds

DRL often struggles with sample efficiency, requiring substantial interactions with environments. This chapter examines foundational algorithms, scalable architectures, and exploration-focused methods that address these challenges.

Foundational methods include Deep Q-Network (DQN) *Mnih et al.* (2015), which combines deep neural networks with Q-learning to handle large state spaces but strug-

gles with sparse rewards; Advantage Actor-Critic (A2C) *Mnih et al.* (2016), which reduces variance in updates through synchronized parallel actors but is limited by its on-policy nature; Trust Region Policy Optimization (TRPO) *Schulman et al.* (2015), which ensures stable policy updates with trust region constraints but is computationally intensive; and Proximal Policy Optimization (PPO) *Schulman et al.* (2017), which refines TRPO with clipped objectives for improved data utilization and computational efficiency. Each of these methods has significantly contributed to the development of RL strategies.

Scalable architectures such as Importance Weighted Actor-Learner Architectures (IMPALA) *Espeholt et al.* (2018) address multi-task learning by leveraging distributed architectures with off-policy corrections, offering scalability but facing synchronization challenges, whereby exploration-focused methods aim to address sparse rewards and complex state spaces. Count-Based Exploration (COUNT) *Bellemare et al.* (2016) uses pseudo-counts for better exploration but is computationally demanding in large state spaces. Random Network Distillation (RND) *Burda et al.* (2018) stimulates novelty through prediction errors, however, it depends on high-quality state representations. Curiosity-Driven Exploration (CURIOSITY) *Pathak et al.* (2017) rewards prediction errors of action outcomes, promoting intrinsic motivation, while Rewarding Impact-Driven Exploration (RIDE) *Raileanu and Rocktäschel* (2020) focuses on impactful actions but may struggle with ambiguous state changes. Adversarially Motivated Intrinsic Goals (AMIGO) *Campero et al.* (2021) generates adversarial goals to guide exploration, requiring effective goal-generation mechanisms. These methods focus on improving exploration and sample efficiency, particularly in environments with sparse rewards or complex state spaces, such as Minigrid.

Furthermore, model-based RL techniques can learn a model of the environment and improve their behavior through imagined future scenarios *Sekar et al.* (2020), which is particularly advantageous in high-dimensional and variable state spaces.

DreamerV3 *Hafner et al.* (2023) builds on this principle by introducing a general algorithm designed to address a diverse range of tasks with minimal domain-specific adjustments. By learning latent state dynamics and planning through imagination, DreamerV3 offers broad applicability while reducing the need for extensive tuning or specialized configurations. However, the added complexity of learning latent space representations along with their dynamics leads to less sample efficiency compared to when representations are already present leaving only the dynamics to be learned.

4.3 Non-Axiomatic Causal Explorer

NACE is our proposed experiential learning technique with causality-informed intrinsic reward and strong inductive biases for grid world environments to boost sample efficiency. This section provides formal descriptions of NACE. For a comprehensive list of symbols used, refer to Appendix F.

4.3.1 States and Rule Representation

State in NACE is a tuple $s = (s_{\text{spatial}}, s_{\text{internal}})$ consisting of a two-dimensional value array $s_{\text{spatial}} \in \mathbb{N}^{m \times n}$ and a one-dimensional value array $s_{\text{internal}} \in \mathbb{N}^{k}$, as shown in Figure 4.1. The two-dimensional array reflects the spatial structure in the grid world, including remembered cells beyond the current view, while the one-dimensional array has no notion of spatial distance and is used for internal values, such as inventory items (e.g. keys to use or pick up).

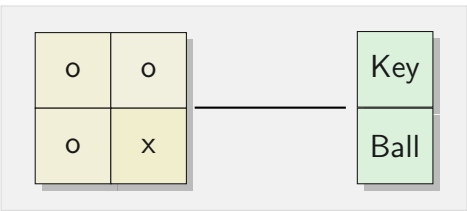


Figure 4.1: State Components.

Learned rules have the form of $(preconditions, action) \Rightarrow consequence$ where the precondition holds a conjunction of cell value constraints spatially relative to the cell value of the consequence, and the consequence predicts one particular cell's value as well as the values of the one-dimensional array at the next timestep as depicted in Figure 4.2.

Each rule tracks evidence using counters for w_+ and w_- similar to (*Wang*, 2013a), which measure the accuracy of the rule's predictions. Positive evidence (w_+) is accumulated whenever a perfectly matching rule predicts correctly, while negative evidence (w_-) increases with incorrect predictions. Tracking of evidence helps the agent refine its causal knowledge by prioritizing more reliable rules. Examples of created rules are provided in Appendix J.

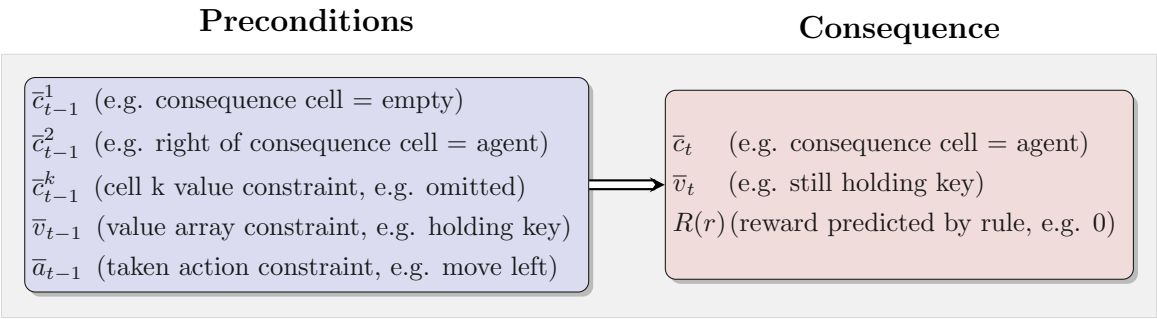


Figure 4.2: Rule schema: Preconditions with an Action Followed by Consequence.

4.3.2 Inductive Biases

It is well-known that favorable inductive biases can enhance sample efficiency. Below are inductive biases that are incorporated in NACE and relevant for grid world environments:

1. **Temporal Locality**: NACE constructs rules based solely on the current and previous state, modeling relevant dependencies locally in time. In simplest case just involving prior simulation frame, essentially Markov Assumption.

2. **Causal Representation**: NACE's knowledge representation is centered around the causal rules which can be chained and are independent of the objective. Thus, making the model able to learn knowledge about the environment that can be chained and is agnostic to its objective.

3. **Spatial Equivariance**: Ability to model causal dependencies between grid cells independently of the specific location of the cells considered in the dependency, meaning learned rules are applied at any location.

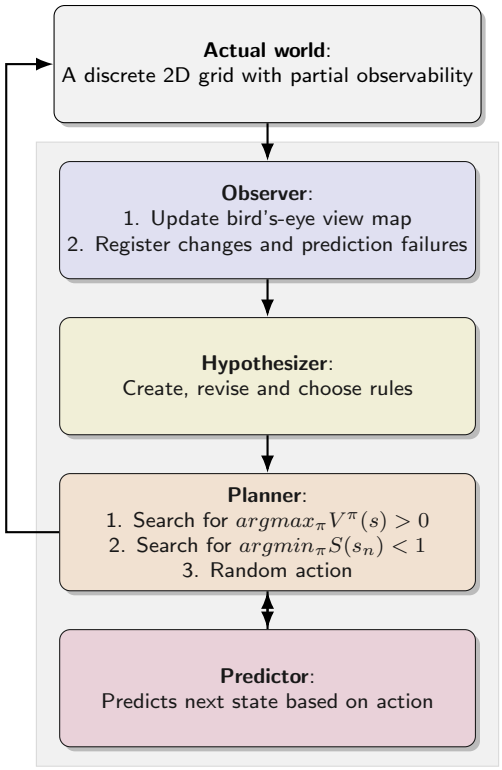
4. **State Tracking**: Ability to effectively track states outside of the field of view of the agent based on the recorded or estimated locations. NACE explicitly keeps track of a bird's-eye view map by recording observations into it, updating the values that are within its observability window.

5. **Attentional Bias**: Relevant dependencies tend to involve values that have either observably changed or a different value than predicted. Only rules that show a change from the previous to the current timestep, or differ from the predicted value, are considered for rule formation, evidence updating, and prediction.

Additional discussions on inductive biases as well as ablation studies can be found in Appendix E.

4.3.3 Curiosity Model

Curiosity model outlines the mechanism which helps NACE systematically acquire missing causal knowledge about the environment. The key principle is realized by making the agent plan to reach a state which it is most unfamiliar with. The familiarity is judged by whether existing rules match well to the situation, whereby matching is a matter of degree dependent on how many rule conditions match the cells in the known state. This motivates the following formalism:



Actual world represents the real simulated 2D grid environment (Minigrid) with a cell-granular partial observability model. In each frame, the field-of-view local to the agent is passed on to the observer.

Observer takes the field-of-view 2D array as input and detects changes in values as well as identifies prediction failures from rules that predict incorrectly.

Hypothesizer creates and updates rules based on whether their predictions align with observations, whereby only changed-cells and prediction-mismatch cells as reported by Observer are considered.

Planner searches for optimal actions that lead to greater-than-zero expected return, and if none such is found, searches for actions that lead to a state of lowest state match value greater than zero. Finally, in case such also does not exist, a random action is chosen

Predictor forecasts the next state from the current state and the taken action, utilizing individual rules to predict a state transition of the entire state, whereby for each cell its predicted value comes from the rule with the highest $M(r, c)$.

Figure 4.3: Flow Diagram of the System.

- *Match* value of a rule r is evaluated relative to consequence cell c:

$$M(r, c) = \frac{\text{Number of matched preconditions}}{\text{Total number of preconditions}}$$

- *Cell match* value of a cell c dependent on all m existing rule match values:

$$C(c) = \max(0, M(r_1, c), \ldots, M(r_m, c))$$

- *State match* value $S(s)$ of a state s is the maximum $C(c)$ value of its cells, whereby a value smaller than 1 indicates uncertainty from imperfect rule matches. This value, as we will see, is the secondary "explorative" objective in the planning process that guides the agent's decisions:

$$S(s) = max(C(c_1), \ldots, C(c_m))$$

4.3.4 NACE Architecture

Figure 4.3 illustrates the high-level architecture of NACE, which consists of several interconnected components working together to enable learning and decision-making. The related pseudocode is provided in Appendix G.

1. **Observer:** Its role is to update a bird's-eye view map via values from the partial observation 2D array, then to find changes in input, as well as prediction-observation mismatches (prediction failures). Formally this corresponds to determining the sets:

   - **Set of changes in observations**:

   $$M_t^{\text{change}} = \{c_{t,x,y}^{\text{observation}} \mid c_{t,x,y}^{\text{observation}} \neq c_{t-1,x,y}^{\text{observation}}\}$$

   This set captures all grid cells $c_{x,y}$ where the observed value has changed between timesteps $t-1$ and t, highlighting areas that have been updated or modified.

   - **Set of observation mismatches**:

   $$M_{\text{mismatched},t}^{\text{observation}} = \{c_{t,x,y}^{\text{observation}} \mid c_{t,x,y}^{\text{prediction}} \neq c_{t,x,y}^{\text{observation}}\}$$

   This set includes all grid cells where the observed value differs from the predicted value at time t, indicating potential prediction failures.

   - **Set of prediction mismatches**:

   $$M_{\text{mismatched},t}^{\text{prediction}} = \{c_{t,x,y}^{\text{prediction}} \mid c_{t,x,y}^{\text{prediction}} \neq c_{t,x,y}^{\text{observation}}\}$$

   This set identifies all grid cells where the predicted value does not match the observed value at time t, from the perspective of predictions.

These sets enable the Observer to track state changes and prediction failures, ensuring an accurate understanding of the environment and supporting the system's adaptive and predictive capabilities.

2. **Hypothesizer:** Associating positive and negative evidence based on prediction success, as well as creating new rules when positive evidence is initially found.

   Formally, for each rule of the format

$$r = ((\overline{c}^1_{t-1} \wedge ... \wedge \overline{c}^k_{t-1} \wedge \overline{v}_{t-1} \wedge \overline{a}_{t-1}) \Rightarrow (\overline{c}_t \wedge \overline{v}_t \wedge R(r)))$$

   $\overline{c} := (c_r = c)$ indicates that the value in the rule precondition aligns with the actual cell value, value array in case of $\overline{v}$, and taken action in case of $\overline{a}$.

   The rule preconditions are met when all *equality constraints* $\overline{c}^1_{t-1}, ..., \overline{c}^k_{t-1}, \overline{v}_{t-1}, \overline{a}_{t-1}$ hold. Positive evidence is attributed when the equality constraints of the postcondition $\overline{c}_t, \overline{v}_t$ are met as well and the predicted reward aligns with the observed reward ($R_t = R(r)$). To increase computational efficiency only cells which changed value or have a different value than predicted are considered. Formally, this is being defined as:

$$w_+(r) = \begin{cases} w_+(r) + 1 & \text{if } \{c^1_{t-1}, ..., c^k_{t-1}, c_t\} \subseteq M \\ w_+(r) & \text{otherwise} \end{cases}$$
$$\text{where } M = (M_t^{change} \cup M^{\text{observation}}_{\text{mismatched}_t})$$

   Negative evidence is assigned when any of the postcondition equality constraints are not met:

$$w_-(r) = \begin{cases} w_-(r) + 1 & \text{if } c_t \in M^{\text{prediction}}_{\text{mismatched}_t} \\ w_-(r) & \text{otherwise} \end{cases}$$

Finally, rules r for which $w_-(r) > w_+(r)$ become inactive, and for two rules r_1, r_2 if their preconditions match (including the action) but the postconditions are different, only the rule with the higher truth expectation $T_{exp}(r)$ is selected, which is calculated according to:

$$w(r) = w_-(r) + w_+(r)$$

$$frequency(r) = \frac{w_+(r)}{w(r)}, \quad confidence(r) = \frac{w(r)}{w(r)+1}$$

$$T_{exp}(r) = (frequency(r) - \tfrac{1}{2}) * confidence(r) + \tfrac{1}{2}$$

This not only allows the system to find the relevant preconditions under which a consequence happens when the action is utilized but also gives the system tolerance to non-deterministic effects and enables accounting for uncertainty.

3. **Planner**: NACE makes use of depth- and width-bounded Breadth-First-Search algorithm with a combined search objective consisting of two components: it searches for states resulting from the different action sequences for futures that lead to the maximum expected return or, if not existing, the lowest state match value. Hence, it applies a key RL principle to maximize the expected long-term return (*Sutton et al.*, 1999), with the policy determined by the considered action sequence: $\pi(t) = a_t \quad \text{for} \quad t = 1, 2, \ldots, n$ whereby n is smaller-or-equal (dependent on where the optimum is found) to the maximum planning horizon:

$$\pi(t) = \begin{cases} \arg\max_\pi V^\pi(s_0) & \text{if } V^\pi(s_0) = R_{exp} \\ \arg\min_\pi S(s_n) < 1 & \text{otherwise} \end{cases}$$
$$\text{where } R_{exp} = \mathbb{E}\left[\sum_{t=0}^{n} \gamma^t R(s_t) \mid s_0 = s, \pi\right] > 0$$

According to this definition, if no return greater than zero can be obtained for any considered action sequence, the system instead plans for a future state of

lowest state match value, while maintaining the "uncertainty-at-the-last-step" condition:

$$(\forall t : (0 \leq t < n) \rightarrow S(s_t) = 1) \wedge S(s_n) < 1$$

meaning the action sequence is constrained to be planned in such a way that state match value is 1 except for the last action where it is minimized for the resulting state. Such constraint maximizes the agent's chance to reach the state of minimum state match while ensuring the low match value is not a consequence of predicting further from states where the knowledge was already not fully applicable.

Due to the amount of possible options, the planning algorithm dominates the asymptotics of NACE. It has the computational complexity of $O(|V| + |E|)$ where V and E are the set of nodes and edges of the search graph. Constant-bounded search depth and width can be achieved by pruning of branches by expected return and state match value, however, bounded search depth can negatively affect performance. More details are provided in the later section.

4. **Predictor**: When the planner queries for the predicted state from a given state and an action, the role of the predictor is to construct the predicted state by applying all knowledge to the given state in the following way: initializing with the cell values from the given state, where for each cell we utilize only the rule r with $M(r, c) = 1$ and maximum $T_{exp}(r)$, meaning the rule preconditions match perfectly to the given state, the action that has been considered, and r has the highest truth expectation among the rule candidates.

   In this case the postcondition cell value of the rule is applied to the corresponding cell at position (x, y) in the predicted state, while else the cell keeps the value from the previous state. Hence, for utilized rules $r^* = ((\overline{c}_t^1 \wedge ... \wedge \overline{c}_t^k \wedge \overline{v}_t \wedge \overline{a}_t) \Rightarrow$

$(\overline{c}_{t+1} \wedge \overline{v}_{t+1} \wedge R(r^*)))$, where $\overline{c}_{t+1}$ and $\overline{v}_{t+1}$ constrains the cell value and value array of the consequence:

$$c_{t+1,x,y} = \begin{cases} c_{t+1} & \text{if } r^* = \underset{r|M(r,c_{t,x,y})=1}{\operatorname{argmax}} T_{exp}(r) \\ c_{t,x,y} & \text{otherwise} \end{cases}$$

Now, while s_{t+1} is a composition of the cells at all locations at time $t+1$, the reward associated with s_{t+1} is the average reward of all N utilized rules:

$$R(s_{t+1}) = \frac{1}{N} \sum_{i=1}^{N} R(r_i^*)$$

Hereby, the average was chosen since if the reward prediction of all the used rules aligns with the observed reward, their average will also align, while the sum would overestimate the outcome.

4.4 Experiments in Minigrid

To evaluate the effectiveness of NACE compared to other DRL techniques, we conducted a series of experiments in Minigrid *Chevalier-Boisvert et al.* (2024), a 2D grid world environment featuring diverse and procedurally generated scenarios (Hardware setup and further test environments in Appendices). These scenarios demand the agent to navigate, interact with objects, and achieve specific objectives under varying degrees of observability. We focus on Minigrid levels that feature partial observability (using the default observation format, which provides values per grid cell rather than per pixel), challenging the agent to operate with limited information about its surroundings. The selected environments are categorized based on their specific characteristics:

- **Static:** fixed start & goal locations. (*MiniGrid-Empty-16x16-v0, MiniGrid-DistShift2-v0*)

- **Dynamic:** random positions of start, goal and obstacle. (*MiniGrid-LavaGapS7-v0, MiniGrid-SimpleCrossingS11N5*)
- **Dynamic with sequential dependencies:** tasks requiring specific action sequences (e.g., a door that needs a key). (*MiniGrid-Unlock-v0, MiniGrid-DoorKey-8x8-v0*)

In each environment, we recorded the average reward, episode length, and standard deviation every 100 timesteps, whereby each timestep incorporates the observed state, action taken, and obtained reward. The following sections present and discuss some representative results for each category, using the selected RL techniques mentioned in Section 4.2.1. (Configuration and hyperparameter details are in Appendix H. Additionally, Behavior Trees (BTs) and hard-coded policies are employed as performance upper bounds in non-stationary and static environments.

4.4.1 Stationary Environments

In this category, because the start and goal locations are fixed, the primary challenge for the agent is to consistently learn and optimize navigation strategies over repeated episodes.

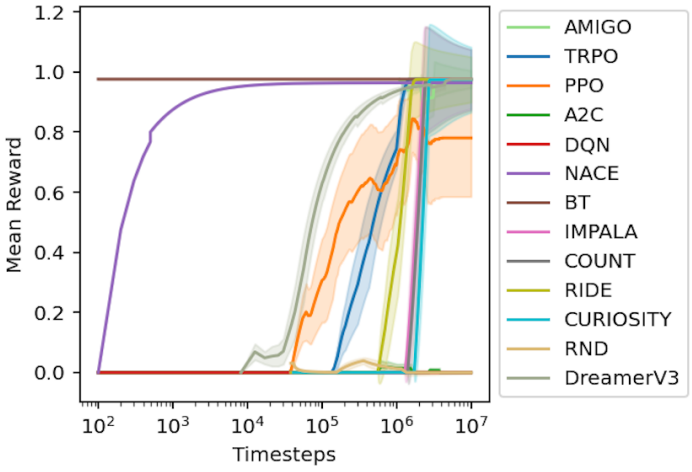


Figure 4.4: Learning Curves in *MiniGrid-Empty-16x16-v0*.

Table 4.1: Final values for *MiniGrid-Empty-16x16-v0*

Technology	$R_{avg.}$	STD
TRPO	**0.976**	**0.001**
PPO	0.781	0.390
A2C	0.000	0.000
DQN	0.000	0.000
IMPALA	**0.976**	**0.197**
COUNT	0.974	0.203
RIDE	0.975	0.151
CURIOSITY	0.974	0.217
RND	0.000	0.008
AMIGO	**0.976**	**0.059**
DreamerV3	**0.976**	**0.000**
NACE	0.964	0.002

MiniGrid-Empty-16x16-v0: This environment features a large, static grid where the agent must navigate from a fixed start location to a fixed goal. Due to the limited observation window and the sparse reward—only granted upon reaching the goal—the task can present some difficulties. In this case, NACE, following its innate strategy, first learns to move effectively within the grid by exploring its immediate surroundings. Then it systematically expands its exploration, hitting observed walls out of curiosity, and finally exploring initially invisible parts of the map until the goal object is found and moved into. Due to its intrinsic inductive bias, it explores the area systematically and associates reward with the goal location within about 10^3 timesteps. In contrast, other techniques like DreamerV3, TRPO, IMPALA, RIDE, and AMIGO, although capable of eventually learning the task, require over 10^5 timesteps to perform comparatively well (as seen in Figure 4.4, and Table 4.1).

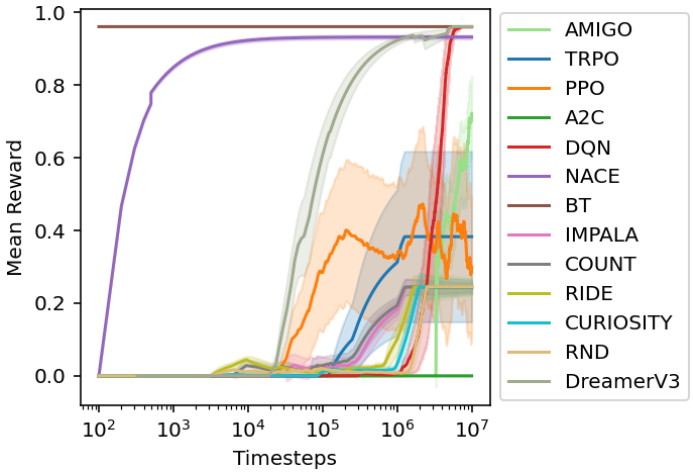


Figure 4.5: Learning Curves in *MiniGrid-DistShift2-v0.*

Table 4.2: Final values for *MiniGrid-DistShift2-v0*

Technology	$R_{avg.}$	STD
TRPO	0.383	0.469
PPO	0.763	0.381
A2C	0.000	0.000
DQN	**0.961**	**0.000**
IMPALA	0.245	0.027
COUNT	0.243	0.025
RIDE	0.245	0.036
CURIOSITY	0.245	0.041
RND	0.245	0.049
AMIGO	0.778	0.203
DreamerV3	**0.961**	**0.000**
NACE	0.932	0.011

MiniGrid-DistShift2-v0: In this case, the fixed start and goal locations are accompanied by stationary lava obstacles, which the agent must navigate around to reach the goal. DQN and DreamerV3 perform well, achieving a near-optimal policy with an average reward of 0.96, closely mirroring the performance of the BT. NACE reached a slightly smaller value of 0.93, while it was three orders of magnitudes more

sample-efficient. The next-best policies were found by AMIGO and PPO with an average reward of 0.78 and 0.76, while the other techniques were below 0.5, all of them being much less sample-efficient than NACE (as in Figure 4.5 and Table 4.2). A2C on the other hand did not converge to a policy with an average reward above 0.

4.4.2 Dynamic Environments

Given that the start and goal locations, along with obstacle positions, are randomized in each episode, these environments require the agent to continuously adapt to new and unpredictable conditions.

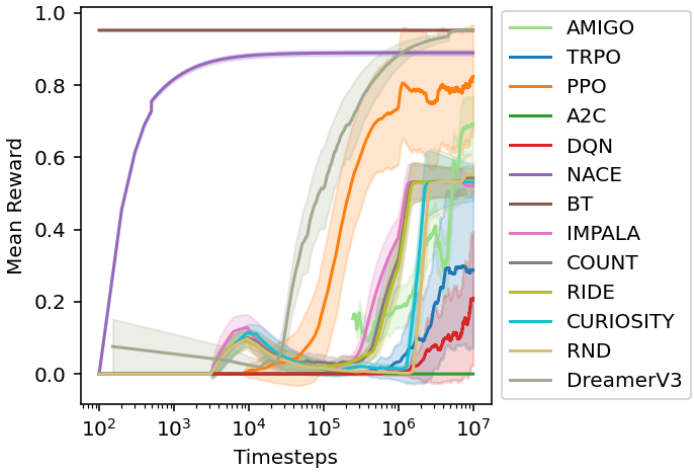


Figure 4.6: Learning Curves in *MiniGrid-LavaGapS7-v0*.

Table 4.3: Final values for *MiniGrid-LavaGapS7-v0*

Technology	$R_{avg.}$	STD
TRPO	0.187	0.375
PPO	0.838	0.309
A2C	0.000	0.000
DQN	0.114	0.309
IMPALA	0.521	0.064
COUNT	0.543	0.067
RIDE	0.535	0.070
CURIOSITY	0.531	0.103
RND	0.551	0.111
AMIGO	0.690	0.151
DreamerV3	**0.952**	**0.005**
NACE	0.922	0.013

MiniGrid-LavaGapS7-v0: In this environment, the agent must navigate around randomly placed lava obstacles to reach a fixed goal, requiring adaptability due to the varying paths between episodes. The 5x5 free space, which is mostly covered by the agent's observation window, is complicated by dynamically spawning lava, unlike the stationary obstacles in *MiniGrid-DistShift2-v0*. From the mean episode rewards (as seen in Figure 4.6, and Table 4.3), it is clear that DreamerV3 finds the most effective strategy closely followed by and NACE, whereby NACE takes about 10^3 timesteps while DreamerV3 takes 3×10^5 timesteps to reach a mean reward value of around 0.8,

while the optimal policies, as BT shows, are close to DreamerV3's. Additionally, PPO (which reached 0.838) exhibits more instability in learning and greater sensitivity to initialization, as evidenced by a higher standard deviation. Following these, AMIGO reached an average reward of only 0.69, while the remaining techniques performed poorly, despite the fact that this level is practically fully observable.

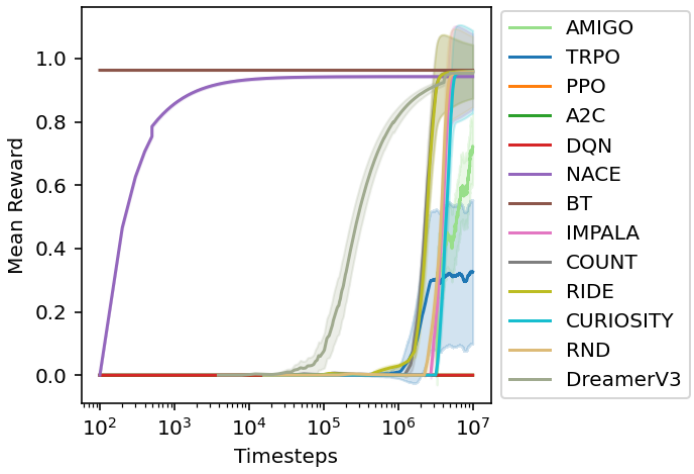


Figure 4.7: Learning Curves in *MiniGrid-SimpleCrossingS11N5-v0*.

Table 4.4: Final values for *MiniGrid-SimpleCrossingS11N5-v0*

Technology	$R_{avg.}$	STD
TRPO	0.381	0.467
PPO	0.000	0.000
A2C	0.000	0.000
DQN	0.000	0.000
IMPALA	0.958	0.238
COUNT	**0.960**	**0.168**
RIDE	0.959	0.170
CURIOSITY	0.958	0.261
RND	0.958	0.222
AMIGO	0.778	0.203
DreamerV3	0.954	0.008
NACE	0.943	0.005

MiniGrid-SimpleCrossingS11N5-v0: Here the agent faces a large grid with multiple intersections and potential dead ends. The randomized layout in each episode also forces the agent to develop a robust exploration strategy. As Figure 4.7, and Table 4.4 show, DreamerV3, IMPALA, COUNT, RIDE, CURIOSITY, RND and NACE achieved near-optimal policies since their intrinsic reward mechanisms seem to be particularly helpful in this large environment where the observable window covers only a small part. AMIGO found reasonable policies with average reward of 0.78, while the remaining techniques scored below 0.5, and some of them converged to sub-optimal policies. And NACE's shows data-efficiency due to its inductive biases, even though it does not converge to the optimal policy. TRPO was able to find a sub-optimal policy with significant less average reward, while the other techniques did not converge.

4.4.3 Dynamic Environments with Sequential Dependencies

In these environments, the need to perform actions in a specific sequence adds complexity and tests the agent's ability to plan and execute multi-step strategies.

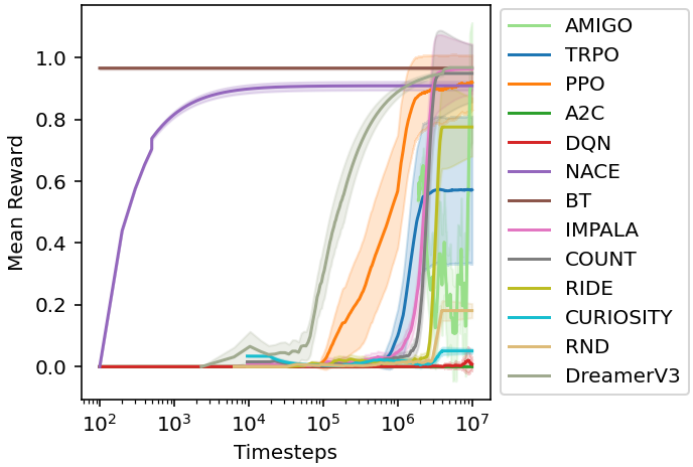


Figure 4.8: Learning Curves in *MiniGrid-Unlock-v0*.

Table 4.5: Final values for *MiniGrid-Unlock-v0*

Technology	$R_{avg.}$	STD
TRPO	0.577	0.471
PPO	0.890	0.263
A2C	0.000	0.000
DQN	0.000	0.000
IMPALA	0.964	0.162
COUNT	0.949	0.185
RIDE	0.775	0.188
CURIOSITY	0.051	0.016
RND	0.181	0.046
AMIGO	0.932	0.388
DreamerV3	**0.967**	**0.003**
NACE	0.909	0.028

MiniGrid-Unlock-v0: In this scenario, the agent must first locate and pick up a key before unlocking a door to reach the goal and obtain the reward. This sequential dependency adds a layer of complexity that challenges the agent's ability to plan ahead. Even though it is a single sequential dependency, the DRL techniques that learned the fastest initially, DreamerV3 and PPO, demands almost a million timesteps to converge to a similarly effective policy as NACE, which achieves this within just 10^3 steps again (as seen in Figure 4.8, and Table 4.5). Additionally, while PPO, which reached 0.89, shows more instability in learning, it is far less chaotic than AMIGO. IMPALA reached the optimal policy, and did so after about 2 million steps, performing similarly well as COUNT and AMIGO in the end. It is also visible, in our runs, that TRPO did not exceed a mean episode reward of 0.6, while A2C and DQN completely failed to learn any effective policy.

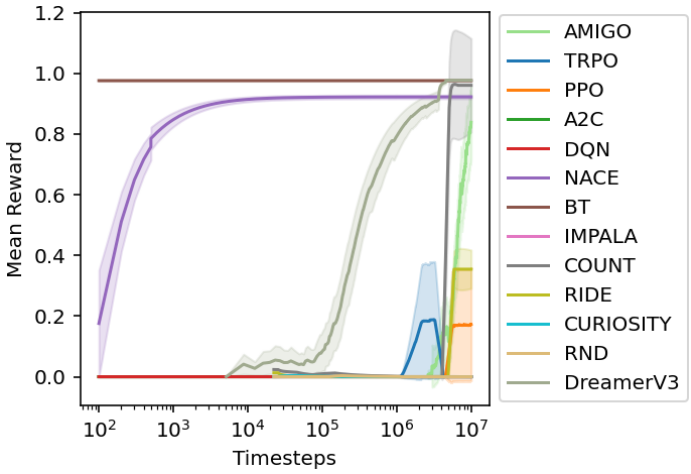


Figure 4.9: Learning Curves in *MiniGrid-DoorKey-8x8-v0*.

Table 4.6: Final values for *MiniGrid-DoorKey-8x8-v0*

Technology	$R_{avg.}$	STD
TRPO	0.000	0.000
PPO	0.156	0.357
A2C	0.000	0.000
DQN	0.000	0.000
IMPALA	0.000	0.000
COUNT	0.960	0.308
RIDE	0.354	0.126
CURIOSITY	0.000	0.000
RND	0.000	0.001
AMIGO	0.868	0.241
DreamerV3	**0.977**	**0.004**
NACE	0.922	0.012

MiniGrid-DoorKey-8x8-v0: This environment introduces an additional layer of sequential dependency by requiring the agent to navigate through an unlocked door to reach a goal in a separate room. While passing through the door adds complexity, the primary challenge lies in the sparse reward structure, as no reward is given for merely using the door, since only reaching the final goal is rewarded. As presented in Figure 4.9, Table 4.6, DreamverV3 and COUNT nearly achieved the optimal policy with an average reward of 0.975 and 0.96. AMIGO reached 0.87, but within 10^7 timesteps, which is below the average reward which NACE reached within only 10^4 steps. Overall the results suggest poor combinatorial scaling of the involved DRL techniques, while NACE, on average, required a similar amount of timesteps as for *MiniGrid-Unlock-v0* to learn an effective policy.

4.5 Discussions

NACE reaches the final average rewards after just 1000 training steps, whereby the value for the other Reinforcement Learning techniques is 0 at that time step, as they tend to require at least hundreds of thousands of training steps for effective policies

to be found. This is also consistent with the literature and more specialized methods with intrinsic rewards for faster learning (as evidenced by *Chen and Guan* (2023) and *Guo et al.* (2022)), which results we summarized in terms of average reward and training iterations in table 4.7 showing performance of NACE and different algorithms on various MiniGrid environments according to *Chen and Guan* (2023), *Guo et al.* (2022) and *Zhang et al.* (2021).

Table 4.7: Performance of NACE and different algorithms

Environment	**Reward**	**Steps**	**Algo**
MiniGrid-DoorKey-16x16-v0	0.61	3E+06	CEMP
MiniGrid-DoorKey-16x16-v0	**0.87**	1E+04	NACE
MiniGrid-DoorKey-8x8-v0	**0.98**	3E+06	CEMP
MiniGrid-DoorKey-8x8-v0	0.39	2E+05	CEMP
MiniGrid-DoorKey-8x8-v0	0.92	1E+03	NACE
MiniGrid-LavaCrossingS11N5-v0	0.81	3E+06	CEMP
MiniGrid-LavaCrossingS11N5-v0	0.01	2E+05	CEMP
MiniGrid-LavaCrossingS11N5-v0	**0.88**	1E+03	NACE
MiniGrid-UnlockPickup-v0	**0.95**	3E+06	CEMP
MiniGrid-UnlockPickup-v0	0.20	3E+06	CEMP
MiniGrid-UnlockPickup-v0	0.86	1E+03	NACE
MiniGrid-BlockedUnlockPickup-v0	0.68	1E+07	ViewX
MiniGrid-BlockedUnlockPickup-v0	**0.89**	1E+03	NACE
MiniGrid-SimpleCrossingS9N2-v0	0.21	1E+05	Graph
MiniGrid-SimpleCrossingS9N2-v0	**0.85**	1E+03	NACE

The observed sample efficiency of NACE originates from explicitly exploiting the cell-based grid world state observations for creating transition rules. This represents a strong inductive bias, which makes NACE less generic than DreamerV3. However it can nevertheless be valuable in broader applications, as we will see in the later chapter, where mapping high-dimensional input to a similar discrete world representation is feasible. This mapping, dependent on the problem domain, can be implemented with the appropriate choice of feature extraction techniques. Such approaches are commonly employed in robotics, where methods like Simultaneous Localization and Mapping and object detection models are integrated to construct semantic maps for

operating mobile robots *Zhang et al.* (2020a). However we acknowledge this demands a considerable engineering effort, while DreamerV3 can update its perceptual representations dynamically via gradient-based updates.

To analyze the factors that lead NACE to finding close-to-optimal policies, as well as factors which can in some circumstances hamper it from learning such, we present the contributing factors from a conceptual perspective, examine the impact of hyperparameter choices, and assess robustness to non-determinism arising from random action consequences.

- **Representational Limitations**: NACE's rule-based framework captures only spatially relative dependencies from one timestep to the next. It does not exploit the inherent structural statistics of environment generation, which are leveraged by various DRL techniques. While these structural dependencies are most apparent in static environments where locations remain constant, they are also present in dynamic environments. For example, the goal location consistently appears in the bottom-right corner not only in *MiniGrid-Empty* levels but also in *MiniGrid-SimpleCrossingS11N5-v0*. NACE's inability to utilize these broader environmental patterns could limit its performance compared to methods that can, even though in Minigrid this issue was less prevalent.

  Additionally, NACE's rules are tied to an action, meaning agent-external changes that are not caused by NACE need to be learned for each action separately, considerably lowering its sample efficiency by a factor of the amount of actions. The mechanism could be extended to incorporate learning of rules without an action as a precondition, leaving it to evidence collection whether the action is considered dependent on the truth expectation of the alternative rules.

- **Study of Reduced Planning Horizon**: NACE's estimation of expected returns relies heavily on its planning horizon. Short planning horizons can signifi-

cantly reduce performance, especially in tasks requiring long-term planning. To quantify this effect, we examine two cases: *MiniGrid-DoorKey-8x8-v0*, which demands longer-horizon planning, and *MiniGrid-DoorKey-6x6-v0*, which is less demanding in this regard. As shown in Table 4.8, running NACE with a planning horizon of only 8 steps in *MiniGrid-DoorKey-8x8-v0* results in convergence to an average return of 0.48, whereas extending the horizon to 100 steps improves the average reward to 0.92. In contrast, in *MiniGrid-DoorKey-6x6-v0*, NACE maintains an average reward of 0.93 regardless of the planning horizon. A similar pattern is observed in *MiniGrid-Empty-16x16-v0*, where the average reward drops from 0.91 to 0.41 when the planning horizon is reduced from 100 to 8 steps. These results highlight NACE's dependence on adequate planning horizons for effective rule chaining and the significant performance degradation that occurs when the planning horizon is too short.

Table 4.8: NACE Performance with different planning horizons

Environment	**Planning Horizon, Average Reward**
MiniGrid-DoorKey-6x6-v0	8 steps, 0.93
MiniGrid-DoorKey-6x6-v0	100 steps, 0.93
MiniGrid-DoorKey-8x8-v0	8 steps, 0.48
MiniGrid-DoorKey-8x8-v0	100 steps, 0.92
MiniGrid-Empty-16x16-v0	8 steps, 0.41
MiniGrid-Empty-16x16-v0	100 steps, 0.91

- **Robustness to Non-Determinism**: NACE's rule representation incorporates uncertainty handling through evidence counters, enabling it to cope with non-deterministic state transitions. To assess this capability, we modify the environment to invoke unintended actions with certain probabilities. In *MiniGrid-Empty-16x16-v0*, when 10% of actions result in unintended outcomes, NACE still achieves an average reward above 0.9, demonstrating basic tolerance to non-determinism. However, when the probability of unintended actions increases to 20%, NACE fails to complete the task within the maximum allowed time in all

episodes. Higher tolerance to non-determinism can be achieved by increasing the default truth expectation threshold for rule usage above the default value of 0.5. However, this adjustment reduces sample efficiency, as it requires the agent to confirm each rule multiple times before utilizing it.

4.6 Conclusion

In this chapter, we introduced NACE, an experiential learning agent designed to enhance data efficiency in grid world environments by leveraging causally-informed intrinsic rewards and strong inductive biases. NACE also have been compared with state-of-the-art DRL techniques, demonstrating that while these techniques are able to eventually achieve near-optimal policies, they often require significantly more data, especially as task complexity increases due to factors such as sequential dependencies. NACE, by contrast, extends the RL framework to empirically support causal relations, enabling effective learning and decision-making even in low data settings without relying on pre-defined causal models. Our causality-informed curiosity model, combined with the outlined inductive biases, facilitates systematic exploration and learning requiring significantly fewer timesteps. We hope that future work in the field will strike new compromises regarding the inclusion of inductive biases, leading to highly sample-efficient DRL that retains the ability to converge to optimal policies. Moving forward, we plan to generalize NACE to handle three-dimensional and continuous spaces, as well as explore neural implementations of NACE, further advancing the capabilities of learning agents in complex environments.

CHAPTER 5

TOWARD A UNIFIED FOUNDATION OF SYMBOLIC AND SUB-SYMBOLIC REASONING

5.1 Introduction

Non-Axiomatic Reasoning and Causal Explorative System (NARCES) is an integrated hybrid system featuring a tight interconnection between a semantic symbolic reasoner and a sub-symbolic learner. It represents a step toward a unified foundation of symbolic and sub-symbolic reasoning by combining the strengths of the Non-Axiomatic Reasoning System (NARS) and the Non-Axiomatic Causal Explorer (NACE). In this architecture, high-level knowledge representations and inference of MeTTa-NARS are coupled with efficient learning from experience in grid-world environments using NACE to overcome the limitations of either approach alone. The symbolic component, MeTTa-NARS, introduced in Chapter 2, operates with the formal Narsese language and Non-Axiomatic Logic (NAL) to perform reasoning under uncertainty, while the sub-symbolic component, NACE, described in Chapter 4, learns and plans in grid world environments using causal, experience-driven rules. By connecting these two components, NARCES allows the system to incorporate human-provided background knowledge and high-level semantic goals, and to realize them in real-time through data-driven learning and exploration in the environment.

The core motivation for this hybrid approach is to bring together complementary capabilities. MeTTa-NARS offers robust and expressive semantic reasoning,

encompassing deductive, inductive, abductive, temporal, and procedural inference, supported by a sophisticated semantic memory structure, as detailed in Chapter 2. It also excels in integrating and revising knowledge based on new evidence. Conversely, NACE facilitates data-efficient experiential learning directly from the grid world and effectively addresses aspects of the environment that are challenging to model symbolically, such as fine-grained spatial navigation and exploration within a grid-world context.

By bridging symbolic and sub-symbolic processing, we aim to address key challenges of Artificial Intelligence identified in Chapter 1, namely adaptation and generality in open-world, real-time scenarios. In the following sections, we provide details about the design of this hybrid system, the mechanisms by which the two reasoning paradigms interact, and how this unified approach leads to improved reasoning capabilities. We also discuss related work in neuro-symbolic domain, provide a conceptual comparison of symbolic vs. sub-symbolic reasoning, describe the architecture of the bi-directional bridge between MeTTa-NARS and NACE, and evaluate the capabilities and limitations of NARCES. Finally, we showcase the *Explorer*, a robotic agent that leverages NARCES within the ROS2 framework and a Gazebo simulation environment, as a generalized application of our hybrid approach in robotics.

5.2 Related Works

There is a long history of research on combining symbolic and sub-symbolic methods in AI. Early cognitive architectures like Soar (*Laird* (2012)) and ACT-R (*Anderson et al.* (2004)) can be considered hybrids in the sense that they incorporate both rule-based symbolic reasoning and numeric parameters for learning (e.g., activation values or reinforcement of production rules). Similarly, the LIDA architecture (*Franklin et al.* (2016)) integrates a symbolic cognitive cycle with sub-symbolic processes for attention and memory decay, inspired by cognitive science theories.

These systems demonstrated that a degree of integration between symbolic representations (for higher-level decision making) and sub-symbolic mechanisms (for activation spreading, learning, or perception) can yield flexible intelligent behavior. However, traditional architectures often maintain a clear separation between a symbolic reasoning module and sub-symbolic modules (e.g., neural networks for perception) without a deeply unified learning process.

In recent years, the field of neuro-symbolic AI has grown substantially, aiming for tighter coupling of learning and reasoning. Researchers have recognized that purely sub-symbolic systems like deep neural networks, while powerful in pattern recognition, struggle with tasks requiring logic, abstraction, and transparency (*Ali et al.* (2023)). Conversely, purely symbolic systems lack the robustness and automated feature learning of neural networks. Modern hybrid approaches attempt to get the best of both worlds. For example, some works integrate reinforcement learning with symbolic planners or knowledge representations. *AlphaGo* famously combined deep neural networks with a symbolic tree search algorithm to master the game of Go, illustrating the power of blending learned value/policy functions with symbolic search (*Silver et al.* (2016)).

More generally, planning and reasoning algorithms can be guided by learning components, as seen in approaches that use logical rules or symbolic knowledge to guide exploration in reinforcement learning (*Yang et al.* (2018); *Cao et al.* (2022)). *Cao et al.* (2022), for instance, introduce logic synthesis into deep RL to boost generalization across tasks, and *Devidze et al.* (2022) propose an intrinsic reward mechanism based on high-level representations to improve exploration in sparse-reward environments. These examples show that integrating symbolic structures (like logic rules or relational representations) with sub-symbolic learning can yield more data-efficient and generalizable agents. Another line of work explicitly combines symbolic knowledge with neural learning in cognitive reasoning tasks. *Besold et al.* (2017) survey

neural-symbolic learning and reasoning systems, highlighting how neural networks can be trained to respect logical constraints or how symbolic reasoning can incorporate learned neural components. Approaches such as DeepProbLog (*Manhaeve et al.* (2018)) and Neural Theorem Provers (*Rocktäschel and Riedel* (2017)) embed logical inference within neural frameworks, achieving a form of differentiable reasoning. While these are more focused on classification or inference tasks (and not directly on agent acting in environments), they demonstrate the broader trend of hybridizing AI systems.

In the context of agent architectures for autonomy, perhaps the most directly relevant are systems that integrate symbolic reasoning with reinforcement learning for robotics and decision-making. For example, Sun's CLARION architecture (*Sun* (2006)) explicitly had a dual-representation with a sub-symbolic (neural network) bottom level and a symbolic rule-based top level, using the two in parallel for decision making and learning. More recently, there have been our own efforts to integrate NARS (a symbolic reasoner) with robotic control *Hammer et al.* (2022), where we demonstrated a NARS-based control system for ROS-based robots, and *Yun et al.* (2022) presented a neuro-symbolic hybrid approach to driver collision warning that combines non-axiomatic reasoning with sub-symbolic perception. These works underline a key point: a symbolic reasoner like NARS can be embedded in an agent that also leverages machine learning or other sub-symbolic techniques, yielding a system that can reason about high-level goals and knowledge while also reacting to continuous, uncertain environments.

Another noteworthy contribution is the "Unified Perception and Cognition Model" proposed by *Wang et al.* (2022), which directly integrates perception into NARS. In their framework, perception is treated as an intrinsic part of cognition rather than a separate module: what the agent perceives is determined not only by incoming sensory signals but also by the agent's existing concepts, goals, and context. Consequently,

new sensory experiences can directly inform and update the system's knowledge in a manner analogous to Piaget's processes of assimilation and accommodation *Piaget* (1954), effectively blurring the line between perceptual processing and reasoning. The work emphasizes that "perception and cognition are inextricably linked," *Wang et al.* (2022) differing primarily in their level of abstraction rather than in kind. A key difference with our approach is that Wang et al. integrate perception directly into the NARS reasoning engine customizing it to the specific task and fully relying on its inference control including resource allocation and attentional mechanisms which are not effective for processing large quantities of sensory representations, whereas we achieve unification through a hybrid modular design – coupling a stand-alone NARS-based reasoning core with a dedicated sub-symbolic exploratory learner.

Our proposed NARCES system falls squarely into this landscape of hybrid AI. It distinguishes itself by using a dedicated sub-symbolic module (NACE) specifically designed for causal exploration and experiential (trial-and-error) learning, tightly coupled with an advanced NARS implementation (MeTTa-NARS). In contrast to earlier hybrid cognitive architectures, NARCES features a bidirectional exchange of information between the symbolic and sub-symbolic components at runtime, rather than treating one as a fixed pre- or post-processor for the other. This design is influenced by dual-process theories of cognition (*Kahneman* (2011); *Gronchi and Perini* (2024)), where a fast subconscious system (analogous to NACE's reactive learning) works in tandem with a slower deliberative system (analogous to NARS's reasoning). By situating our work in the context of these related efforts, we underscore that NARCES is part of a broader attempt toward hybrid intelligent systems that leverage both logical reasoning and sub-symbolic learning.

5.3 Interlude: Symbolic vs Sub-symbolic

Before detailing the hybrid approach, it is important to clarify how we view the roles of symbolic and sub-symbolic reasoning within a single system. In NARCES, the two types of reasoning coexist within one architecture, with a carefully designed interface between them. Symbolic reasoning, as embodied by MeTTa-NARS, uses knowledge explicitly represented as statements (beliefs, goals, etc.) expressed in Narsese formal language with an experience-grounded semantics along with formal inference rules. A statement can carry a truth value indicating the system's confidence, and NARS continuously revises its beliefs and priorities based on incoming evidence. MeTTa-NARS operates using terms that have semantic meaning (e.g., grounded in concepts such as "object", "goal", or specific entities) and produces derivations utilizing NAL logic which are transparent, traceable and explainable. This allows for logical reasoning that is flexible, enabling the system to handle uncertainty, learn new knowledge incrementally, and to incorporate human-provided knowledge. It also allows the system to derive and interpret high-level goals and to perform multiple types of inference (as discussed in Chapter 2). However, MeTTa-NARS alone cannot learn directly from raw sensory data—it relies on its inputs already being relatively abstract (e.g. event predicates in Narsese) and does not necessarily capture the causal dependencies of grid-world environments.

By contrast, sub-symbolic reasoning (the role of NACE) deals with raw or low-level inputs without requiring predefined symbolic labels. It learns implicit knowledge in the form of patterns, associations, and causal representations. In our case, NACE efficiently learns from observations obtained directly from the environment represented as a 2D grid and induces low-level causal rules from temporal and spatial local changes within the grid. It utilizes these rules to plan for and reach future states of maximum uncertainty in order to effectively learn more (causal rules) about the environment, thereby improving predictability-based intrinsic reward formulations. However, rules

are not represented as logical propositions and nowhere during the rule learning or formation process 'symbols' do occur neither any semantic description emerges nor it is being provided. Sub-symbolic learning is excellent for capturing causal dependencies directly from raw representation of the environment, and discovering patterns that are not known in advance. It is analogous to a human's intuitive, experiential learning process – recognizing a scenario or performing a skill without necessarily verbalizing a rule for it.

The strength of a hybrid system lies in the synergy between these two forms of reasoning. In NARCES, we enable a seamless flow of information across the symbolic/sub-symbolic boundary via a bi-directional bridge. This bridge performs a mapping between semantic content and sub-symbolic experience: it translates high-level symbolic knowledge or objectives into a form that the sub-symbolic component can utilize, as well as it translates important experiential outcomes from the sub-symbolic side back into symbolic representations. In doing so, we ensure that important observations within the grid-world of the NACE agent are grounded in the MeTTa-NARS within its semantic memory, and conversely that the NACE agent's low-level rule learning is guided by meaningful semantic concepts of NARS. Figure 5.1 (conceptually) illustrates this dual-processing architecture.

The symbolic side maintains a conceptual model of the environment along with the background knowledge provided as prior knowledge or input by a user at runtime. It can reason upon possible actions and produce derivations. For example, if MeTTa-NARS knows what it means to prepare a table, and the goal "to prepare the table" is provided, the system deduces the prerequisites necessary to prepare the table and translates them as low-level objectives to NACE for further realization. The sub-symbolic side simultaneously interacts with the environment, gathering observations (e.g., "the cell at coordinate (x, y) is empty") and learning from trial and error (e.g., discovering which actions lead to goal achievement). The bridge ensures that

key observations (like detecting an object or a condition being met) are sent to the symbolic memory of NARS as new beliefs and that the goals formed by the symbolic system are dispatched to the sub-symbolic learner as objectives to pursue.

It's worth noting that maintaining both reasoning modes requires careful design so that they assist rather than interfere with each other. NARCES features a hybrid memory: the NARS side uses a network of semantic concepts and their relations, which can be enriched by knowledge originated from NACE's observations of the environment, and the NACE side retains a knowledge base of learned state-action rules along with the latent representation of a grid-world as a 2D grid with values. MeTTa-NARS and NACE operate on different representations, yet through the bridge they refer to the same reality. In essence, the bridge provides a shared *representation medium* where symbolic and sub-symbolic information can be aligned. For example, a spatial relation like "Object A is *left-of* Object B" can be represented symbolically as a Narsese statement and at the same time corresponds to a certain configuration in the grid (sub-symbolic state) that NACE can recognize or achieve. By mapping this relation into both realms, the system ensures consistency between what it *believes* (symbolically) and what it *experiences* (sub-symbolically). In summary, symbolic and sub-symbolic reasoning in NARCES are not competing approaches, but complementary ones. Symbolic reasoning provides structure, interpretability, and the ability to incorporate prior knowledge and perform high-level reasoning. Sub-symbolic reasoning provides the ability to efficiently learn from raw experience and execute action directly in the environment. The next section shows how a hybrid approach to reasoning leverages this complementarity: rather than solving a problem purely with one mode or the other, NARCES interlaces symbolic inference with sub-symbolic learning throughout the reasoning process.

5.4 A Hybrid Approach to Reasoning

In a hybrid reasoning approach, an intelligent agent can dynamically invoke either symbolic or sub-symbolic processing (or both) depending on the context, and more importantly, allow each mode to inform the other. We now describe how NARCES employs such an approach to solve tasks that neither MeTTa-NARS nor NACE could easily solve alone. The key idea is that semantic knowledge in MeTTa-NARS is interlaced with NACE's observations and learned experiences from the environment. This manifests in several ways:

- **High-level goals and low-level planning:** The agent's high-level goals are formulated in MeTTa-NARS or provided by the user. For instance, consider a simple example, where a user asks NARCES to "store a container" as:

  ```
  !(AddGoalEvent ((container --> (IntSet stored)) (1.0 0.90)))
  ```

  Such goal initially requires some reasoning that generates derivations or subgoals which are subsequently translated to the low-level objective for NACE such as:

  ```
  world[{y+1}][{x}] == '{cup}' and world[y][x] == '{bin}'
  ```

  meaning "the cup needs to be under the bin" where the cup and the bin must satisfy particular spatial relation in the grid-world environment (in this case the cup must be one cell below the bin). Given that MeTTa-NARS is provided with the background property groundings and categorical knowledge such as:

  ```
  !(AddBeliefEternal ((((bin x cup) --> up) ==>
                      (container --> (IntSet stored))) (1.0 0.9)))
  !(AddBeliefEternal (((container --> (IntSet stored)) ==>
                      ((bin x cup) --> up)) (1.0 0.9)))
  !(AddBeliefEternal ((cup --> container) (1.0 0.9)))
  ```

which tells "what it means to store cup" and "cups are containers", using inductive and deductive inference it can derive `((bin x cup) --> up)` indicating that the bin has to be above the cup. This derivation will be sent to the bridge, where it is translated to the planning objective for NACE, which will proceed with the planning and execution utilizing its existing causal rules or by learning new ones as it explores the environment as described in Chapter 4. This illustrates how MeTTa-NARS, using its symbolic reasoning capabilities, can decompose high-level objectives into sub-goals or derive required preconditions. However, to actually execute those goals in an environment, the agent relies on NACE. The hybrid approach means that once NARS derives required pieces of knowledge based on system's experience, it delegates them to NACE via the bridge. NACE then treats this as an objective condition in the grid-world and uses its learning and planning mechanisms to find a sequence of actions to fulfill it. In this way, symbolic reasoning is complimented by the sub-symbolic planning and execution.

- **Informed exploration and learning:** Conversely, the sub-symbolic side enhances symbolic reasoning by providing data-driven insights. NACE continuously explores the environment, often driven by curiosity and causality-informed intrinsic reward (see Chapter 4). Through this exploration, NACE discovers new details about the world, for example, "there is an obstacle left of the agent" or "possession of a key to open a door", which stay implicit in NACE in the form of learned causal rules or latent representation of a grid-world. However, in NARCES, these discoveries are translated into symbolic form and grounded in MeTTa-NARS's conceptual memory. This could be done proactively, whereby the bridge sends a new observation as Narsese statements to MeTTa-NARS, or reactively when the observations are queried, such as "Is there a cup on the table?". The effect is that NARS's experience is enriched with new beliefs ac-

quired from the environment, which it can then utilize in subsequent reasoning. For instance, if NACE learns a cause-effect rule in the grid-world, NARS might not incorporate the exact rule (as we will discuss later in this section), but it could receive specific observed instances that become grounded in its memory such as "cup is on the table". Symbolically, this new evidence can form a new belief as well as support or contradict existing knowledge triggering NARS's revision mechanism.

- **Dual control mechanisms:** Because the two components run in parallel, NARCES effectively employs two distinct control mechanisms. One mechanism explicitly invokes high-level inference rules and performs various types of inference (as seen in MeTTa-NARS), while the other efficiently constructs knowledge about grid-world environments in a sub-symbolic manner, operating without any high-level knowledge (as facilitated by NACE). In a well-functioning hybrid system, these controls need to stay coordinated. In NARCES, both components operate independently, where NARS loops through its inference cycles by processing inputs and providing derivations, while NACE continuously explores the environment. Coordination, however, is achieved through the bi-directional bridge. When a high-level goal is provided by the user, NARS produces derivations and sub-goals which are registered in NACE's Planner as a new planning objective, and effectively directs NACE agent toward the specified objective, overriding the exploration if the objective is reachable. Once NACE realizes the objective, a confirmation event is generated by the bridge and fed back into MeTTa-NARS, which may shift its attentional focus. For example, a realization of NACE's objective can be crucial for NARS to chain multiple sub-goals. Hence, in the context of our example with the table being prepared, given the high-level goal "to prepare the table", NARS first derives a planning objective for NACE to "make the coffee", and only after the corresponding confirmation

event is received, the next derived objective will be "to prepare the table". The major advantage of such combined control is flexibility: NARS can deliberately switch strategies or try alternative reasoning approaches influenced by inputs from the NACE agent, while NACE ensures the agent always continues exploration in case symbolic guidance is not yet provided or unavailable.

- **Unified knowledge representation:** The major advantage of NARCES is that the system uses a combined form of data representation, which includes a semantic graph of prioritized concepts along with a large collection of sub-symbolic causal rules. Each representation has its own strengths. The concept graph can represent abstract knowledge about the environment including relations between entities and their properties, while the sub-symbolic causal rules capture state transitions in the agent's egocentric view (e.g., a rule that if the agent is at position (x, y) facing north and the cell ahead is empty, moving up leads to the agent's position changing accordingly). NARS explicitly distinguishes different types of inference (declarative, temporal, procedural) and applies them to its symbolic beliefs. NACE, on the other hand, utilizing its curiosity model to plan for and reach future states of maximum uncertainty, effectively learns and updates its knowledge about the environment. By running both, NARCES is capable of reasoning in ways a pure sub-symbolic agent cannot (for instance, using background knowledge provided by a human), as well as learning in a data-efficient way directly from observations of the environment in ways a pure symbolic system cannot. This unified approach proved particularly useful in scenarios where some background prior knowledge is known and other aspects must be learned on-line. For example, we might pre-load MeTTa-NARS with background knowledge about object categories and their general properties, while NACE learns the layout of a specific new environment. NARCES can then, in real-time, apply the general knowledge to the specific situation.

5.5 Architecture and Design Principles

This section outlines the technical architecture of NARCES and clarifies the design principles that guide the integration of MeTTa-NARS with NACE. Figure 5.1 provides a visual representation of the NARCES architecture, highlighting the primary components and their interactions[1]. At a high level, the system is composed of two main processing modules: the symbolic reasoner and the sub-symbolic learner, connected by a bridging interface that facilitates communication between them. We have previously examined these components in Chapters 2 and 4; therefore, this section will focus on the Bi-Directional Bridge and elaborate on the design principles that ensure the coherent operation of the hybrid system.

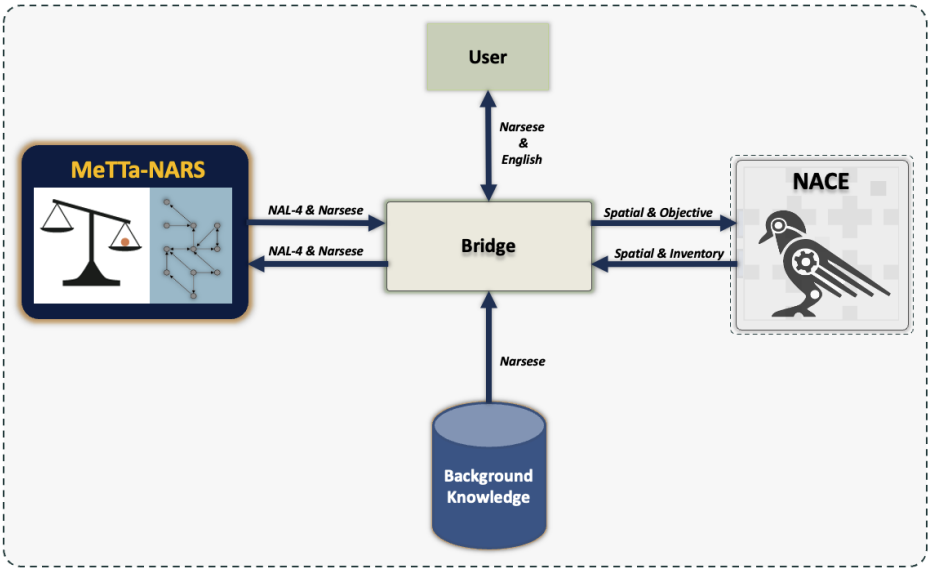


Figure 5.1: NARCES Architecture.

5.5.1 Design Principles

Below are the important design principles governing NARCES's architecture:

[1]In Figure 5.1, the NARS-to-Bridge interface distinguishes NAL-4 spatial relations from general Narsese for illustrative purposes only, because the former is exclusively required for NACE, while the latter handle general I/O of MeTTa-NARS.

- **Modularity and Encapsulation:** Each component, MeTTa-NARS and NACE, operates as a standalone module equipped with a well-defined API. MeTTa-NARS is not required to be aware of NACE's operation and learning process, just as NACE remains uninformed about NARS's inference mechanisms. Their interaction is exclusively facilitated through the bridge's API. This modularity enhances ease of maintenance and presents the potential for upgrading one component without necessitating changes to the other. For instance, one could replace NACE with a different sub-symbolic learner by re-implementing the bridge's translation layer, all without making changes to MeTTa-NARS.

- **Shared Semantic Layer:** Even though internal workings differ, there is a shared semantic understanding achieved via the bridge. The bridge essentially implements a schema for translation, where certain key relations and types of events are predefined to ensure consistent interpretation by both components. We have established a set of spatial relations (`up`, `down`, `left`, `right`, `near`, `far`) along with NAL-4 declarative statements that NARS can reference, all of which possess clear meanings within the grid-world context and functions as a common language. The design principle here resembles the creation of a symbol grounding layer: NAL-4 statements utilized by NARS correspond directly to tangible elements within NACE's world model, while sub-symbolic observations of NACE are simultaneously grounded in NARS's memory, aligning with the system's experience.

- **Asynchronous Yet Coordinated Execution:** MeTTa-NARS and NACE operate on differing timescales. NARS's inference cycle can potentially execute several steps during the time it takes the agent to perform a single action, or vice versa. The architecture does not enforce synchronized operation; rather, the bridge facilitates data synchronization as it becomes available. In practice,

our implementation allows NACE (the environment loop) to invoke the bridge at each time step to check for new objectives, beliefs, and environmental observations. This asynchrony enables NARS to continue its internal reasoning processes even while the NACE agent is engaged in executing a planning objective. Similarly, NACE can persist in its exploration if NARS is momentarily occupied with another problem or if there are no pertinent goals or derivations available.

- **Consistency and Feedback:** Our architecture incorporates feedback mechanisms to ensure consistency between components. A prime example of this is the handling of objective completion. When NACE successfully achieves a planning objective set by NARS, the bridge not only informs NARS of this achievement but also creates a corresponding belief that is fed back into NARS. This process effectively confirms that the state desired by NARS has been realized, updating NARS's experience to reflect the new state of the world. Conversely, if NACE fails or the objective becomes unreachable, the absence of a confirmation event prompts an adjustment in NARS's reasoning. This feedback principle ensures that both the symbolic and sub-symbolic components remain aligned in their understanding of the agent's current actions and what has happened.

5.5.2 Planning Objective for NACE

To enhance the functionality of NARCES, modifications were made to the Planner module of NACE, allowing spatial relations to be set as objectives by MeTTa-NARS or a user. This is achieved by introducing a predicate within the Planner that serves as a planning objective. The revised Planner prioritizes actions that fulfill the specified planning objective and, if it proves unreachable, follows its original strategy. Specifically, it searches for optimal actions that lead to greater-than-zero expected return, and if none such is found, searches for actions that lead to a state of low-

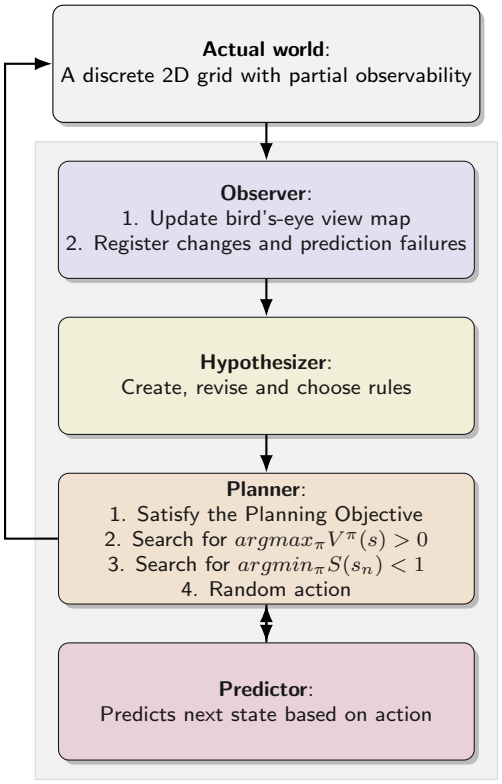


Figure 5.2: NACE with Modified Planner.

est state match value greater than zero. Finally, in case such also does not exist, a random action is chosen. Figure 5.2 illustrates the NACE diagram with modified Planner module.

To ensure clarity in the subsequent sections, it is important to distinguish goals in NARS and objectives in NACE. In the context of NARS, a goal is defined as an event with a desire value, which is realized by executing an operation that is a part of Procedural Inference (we covered it extensively in Section 2.3.9). A planning objective in NACE refers to a certain state of the environment the NACE agent needs to achieve by searching for and executing appropriate actions.

In our implementation, this is realized by constructing a lambda function that evaluates to true if and only if the spatial relation holds between two entities, `S` and `P`, within NACE's representation of the environment. For example, if the objective is to move "a cup on top of the table," represented as `(cup x table) --> up`, the

bridge constructs a lambda function that examines each cell in the environment to locate `table`, and, if found at (x, y), it verifies whether `cup` occupies the top neighbor position $(x, y + 1)$. Similarly, planning objectives incorporating spatial relations of `down`, `left`, `right`, `near`, and `far` are represented through lambda functions, which essentially formalize the objective in precise terms according to NACE's state representation.

5.5.3 Bi-Directional Bridge

The Bi-Directional Bridge serves as the essential conduit linking MeTTa-NARS and NACE. It comprises a set of functions and routines that transform data and messages between different representations. On one side, the bridge listens for outputs from MeTTa-NARS, such as high-level goals or semantic derivations, which need to be translated to NACE. On the other, it monitors NACE's state and achievement of objectives to send meaningful events back to NARS.

Central to the bridge's functionality is its mapping between symbolic representations and their corresponding descriptions within NACE. For instance, if NARS sets a high-level goal as "put the cup on top of the table", the bridge translates it into a spatial relation first and consequently sets up a planning objective for NACE's planner. In general, the mappings can be straightforward, such as directly associating objects with coordinates in the grid to align with given beliefs—for instance, if a user or NARS asserts that "the cup is on the table." Alternatively, the process can be more abstract, involving the creation of a spatial relation and a planning objective represented by a specific grid cell configuration.

The bridge facilitates two primary operations: first, the translation from symbolic to sub-symbolic form, where semantic goals and beliefs from MeTTa-NARS are converted into objectives for NACE, directly influencing NACE's internal representation of the environment. Secondly, it enables the translation from sub-symbolic to sym-

bolic form, where environmental observations or confirmations of objective realization from NACE are transformed back into events for NARS. Operating continuously in sync with the cycles of each module, the bridge employs a loop that processes new outputs from NARS and updates from NACE at each time step. This seamless integration ensures that the system can function coherently, leveraging the strengths of both symbolic and sub-symbolic reasoning.

To make this discussion more concrete, we now break down the bi-directional interactions into two directions: from MeTTa-NARS to NACE (symbolic goals and beliefs influencing sub-symbolic behavior), and from NACE to MeTTa-NARS (sub-symbolic findings influencing symbolic knowledge).

5.5.4 MeTTa-NARS to NACE

Here we cover how information flows from the symbolic side of NARCES to the sub-symbolic one. The primary form of information in this direction are the MeTTa-NARS beliefs represented as ordinary relations encoding spatial information (NAL-4 Narsese statements *Wang* (2013a)). These beliefs are being translated into a spatial relation either representing a specific state of the environment or a planning objective for NACE. The process begins with MeTTa-NARS reasoning upon background knowledge as well as the information about objects within the environment and producing derivations including sub-goals. Whenever a derivation encoding ordinary relation is formulated that requires environment interaction, it is being captured by the bridge and translated into NACE sub-symbolic domain by either grounding it as a planning objective or a belief about agent's observed world.

In particular, we can distinguish the following types of MeTTa-NARS to NACE interconnection:

1. **NACE Planning Objectives:** These consist of NARS derivations and user-provided low-level environmental inputs, both becoming planning objectives

for NACE's Planner. NARCES is always open to new inputs, allowing users to specify low-level objectives that direct the NACE agent. Technically, these derivations and user inputs are encoded as NAL-4 statements that define the spatial relation for NACE to achieve, typically involving a relation between two entities, `S` and `P`. For example, NAL-4 relation `(S x P) --> up` signifies that `S` must be above of `P`, realizing which requires NACE agent to move `S` one cell above `P`. The entities `S` and `P` could represent object labels present within the environment, such as a cup or a table.

2. **NARS Beliefs:** NARS can form specific beliefs about the environment, which may arise either from the provided background knowledge or through derivations made during the inference process. For instance, NARS might hold the belief that "a cup is on top of the table", which corresponds to the following NAL-4 relation: `(cup x table) --> up`. This belief is grounded in the agent's observed world, with the bridge actively modifying the agent's latent representation of the grid-world to reflect this, thereby causing the NACE agent to "perceive" the cup as being on the table. During exploration process, the NACE agent may subsequently confirm or contradict this belief based on its observations of the environment.

3. **NARS Subgoals:** In the context of NARS, realizing a goal involves executing an operation that leads to the Desired State of the system, provided that the relevant procedural hypotheses of the form $(A, Op) \Rightarrow G$ already exist within the system's experience. However, if the precondition event A has not recently happened it is being derived as a subgoal as discussed in Chapter 2. In certain situations, the precondition can be expressed as a NAL-4 spatial relation that the NACE agent must achieve. That is, a newly derived NARS' subgoal initially needs to be realized by the NACE agent as part of fulfilling its planning

objective. Only after receiving a confirmation event, NARS can proceed with executing its designated operation. For instance, consider a robotics example where NARS has a defined `^pick` operation to pick an object from the table. To carry out this operation, the robot (agent) must either be positioned next to the table or move to that location. The requirement of being next to the table can be encoded as a NAL-4 spatial relation, `(agent x table) --> near`, which NARS derives as a subgoal. This subgoal is then translated into a planning objective for the NACE agent directing it to move next to the table. Once the NACE agent positions itself next to the table, a confirmation event is sent back to NARS, enabling it to execute the `^pick` operation.

In summary, the flow from MeTTa-NARS to NACE enables the symbolic reasoner to guide the sub-symbolic agent. By translating semantic beliefs into concrete objectives within the environment, it adds semantic meaning to NACE's actions. The advantage is that NACE is no longer blindly exploring, instead it has a purposeful direction guided by NARS's reasoning about what actions should be undertaken.

5.5.5 NACE to MeTTa-NARS

The information flow from the sub-symbolic learner (NACE) to the symbolic reasoner (MeTTa-NARS) primarily involves observations and learned facts within grid-world environments that can be integrated into NARS's semantic memory. As NACE explores the environment, its spatial observations are transmitted to NARS, allowing for reasoning and updates on the agent's spatial relations with other objects or specific environmental conditions. Furthermore, information about the agent's current inventory is communicated to NARS, enabling it to incorporate this knowledge into its inference processes. NARS's effective mechanism for revising existing knowledge ensures that information initially provided by NARS (as a belief about the environment), which becomes inaccurate after NACE explores the relevant grid-world area or

an inventory item, which is dropped after being picked, is reported back as negative evidence, prompting a revision. All spatial relations observed by NACE or specific environmental conditions, such as confirmation of planning objective achievement or inventory possession, are encoded using NAL-4 ordinary relation statements, as previously discussed.

In general NACE to MeTTa-NARS interconnection allows for the following knowledge sharing:

1. **Agent's Surround View:** As the NACE agent explores the environment, its spatial observations are transmitted to MeTTa-NARS, facilitating reasoning and updates regarding the agent's spatial relations with other objects in proximity. This can be particularly valuable if the NACE agent rediscovers information about the environment which is no longer accurate (discussed below) or just interacts with other objects thereby grounding these interactions and object's properties within NARS's semantic memory. In practice, this is realized on a cyclic basis where at each timestep, four spatial relations (`up`, `down`, `left`, `right`) are generated to create the surround view of the agent. These relations are being translated to NAL-4 statements and fed back to MeTTa-NARS.

2. **Objective Confirmations:** It is crucial for NARS to determine whether the NACE agent's planning objectives are achieved, as this can be particularly valuable in scenarios requiring procedural inference. As previously explained, when the NACE agent successfully fulfills a planning objective set by NARS, the bridge communicates this back to MeTTa-NARS by generating a confirmation event, encapsulated as a NAL-4 ordinary relational statement that encodes spatial relation about this specific environmental condition.

3. **Inventory Items:** The NACE agent continuously provides updates about its current inventory items, which the bridge translates into NAL-4 relational

statements. This translation enables NARS to reason about the items the agent currently holds, such as a key needed to open a door, which can be vital for certain procedural inference scenarios. Whenever the agent picks an item—for example, a key—the bridge generates a NAL-4 statement like: `(((agent x key) --> has) (1.0 0.9))`, which is then communicated back to NARS. Here, the numerical values represent the truth value, as discussed in Chapter 2. When the agent drops the item, the bridge generates a similar statement with a low frequency, such as `(((agent x key) --> has) (0.0 0.9))`, thereby introducing negative evidence and prompting the revision.

4. **Belief Contradictions:** This occurs when the observed spatial relation contradicts NARS's existing knowledge. As previously discussed, NARS can hold beliefs about the environment that the bridge uses to directly alter the agent's latent environmental representation. If, during exploration, these beliefs become inaccurate, the bridge generates a contradiction event and communicates it back to NARS for knowledge revision. This is done similarly to the process for Inventory Items. For example, if the table is outside the agent's observation view, and NARS holds the belief `(((cup x table) --> up) (1.0 0.9))`, indicating "the cup is on the table," the bridge adjusts the agent's representation accordingly. However, if the agent later observes that "nothing is on the table," the bridge creates a contradicting event `(((cup x table) --> up) (0.0 0.9))` and relays it to NARS for correction.

In summary, the NACE-to-NARS communication ensures that the symbolic reasoning is continually informed by the ground truth within the environment. This process completes the feedback loop on objectives by confirming both successes and failures, introduces new perceptual knowledge into the symbolic domain, and facilitates the integration of low-level learned knowledge into higher-level abstract reasoning.

5.5.6 NACE Causal Rules and NARS Procedural Implications

One crucial design decision in our hybrid approach is the scope of information that is exchanged between MeTTa-NARS and NACE. Not every piece of sub-symbolic knowledge is transferred to the symbolic side, and vice versa. We deliberately restrict the interchange to avoid overloading either component with information it cannot effectively use. For instance, NACE learns a large number of causal rules such as:

```
<(v=[1], c[0, 0]=' ', c[0, 1]='x', ^up) =/> (v=[1], c[0, 0]='x', R(r)=0)>
```

which encodes agent's interaction with empty spaces and signifies that if the empty space (`' '`) is one cell above the agent (`x`), and the action is to move (`'up'`), then the agent will move to the corresponding cell, while there is no reward awarded for that (`R(r)=0`), as well as no change in agent's current inventory.

Given MeTTa-NARS ability to perform Procedural Inference as described in Chapter 2, it is easy to find structural similarities between NAL-8 procedural implications of NARS and causal rules of NACE which will allow for simple and efficient translation in both directions. However, we found little use for that, moreover, directly translating all such rules to NARS would be counterproductive because of the following:

- **The truth-value semantics mismatch:** NACE's rules come with a certain truth value and truth expectation which has little to no value within the learned experience of NARS because the semantics of experience in NARS and NACE differ. As a result, the nuances of the truth expectation do not translate cleanly, and therefore, while converting sub-symbolic rules into NAL-8 statements is feasible, we found little practical benefit in doing so.

- **The context focus disparity:** NACE's learned rules are mostly focused on the agent's immediate context by design as they describe spatial relations centered

on the agent (e.g., what is in front, left, right relative to the agent). Incorporating all these rules systematically into MeTTa-NARS could skew its attentional focus towards the agent's egocentric perspective at the expense of other considerations. Overloading attentional control with numerous agent-centric rules would distract MeTTa-NARS from reasoning upon other important knowledge such as user-provided input or its own derivations.

Therefore the sub-symbolic rules remain in the NACE component, where they are highly effective for rapid local planning and learning, and we do not attempt to systematically import them into the NARS memory. Instead, the bridge selectively transfers information that has semantic significance or task-level relevance.

5.6 Capabilities and Limitations

Having outlined the design and functionality of NARCES, we proceed to evaluate the capabilities this hybrid system offers, while also addressing its existing limitations. By integrating MeTTa-NARS and NACE, NARCES not only inherits numerous strengths from each subsystem but also acquires new abilities from their synergy. These capabilities include, but are not limited to, the following:

- **Goal-Driven Autonomy and Real-Time Reasoning:** NARCES can handle open-ended goals in a dynamic environment in real-time. The symbolic part allows a user to specify goals in a high-level descriptive manner, and the system will autonomously attempt to realize them. Unlike a classical planner, NARCES doesn't require a perfect world model to start, and unlike an RL agent, it doesn't require a massive trial-and-error learning process to learn the right policy. It blends different kinds of reasoning, planning, and execution seamlessly, what makes NARCES highly adaptive to evolving scenarios.

- **Data-Efficient Learning:** One of the motivations for integrating NARS with NACE was to leverage NARS's knowledge to enable efficient low-level planning and execution. NACE by itself was designed for causal data-efficient learning in grid-world environments, and therefore, presented itself as an ideal candidate for inclusion within NARCES. The addition of symbolic guidance allows for high-level descriptions to enhance the agent's exploration efficiency and directs the exploration in a semantically informed manner.

- **Knowledge Transfer:** Due to its symbolic component, NARCES can effortlessly transfer knowledge across different grid-world environments. For instance, if NARCES learns in one environment that "a key opens a door," this information is stored within the semantic memory of NARS. When the agent is relocated to a new environment with a different layout but still containing keys and doors, NARS already holds the pertinent beliefs in its memory and can apply them immediately. Additionally, due to the explicit nature of its rules and the inductive biases it employs, NACE does not need to relearn these concepts from the ground up.

- **Explainability and Transparency:** NARCES can explain its reasoning to large extent. Due to high-level semantic learning of MeTTa-NARS, its derivation and decision-making processes are fully traceable and explainable. NARS can also answer questions about what it knows in symbolic terms at runtime. If the agent does something unexpected, one can inspect NARS's beliefs to find the causes of such unexpected outcome. Unlike most sub-symbolic and neural systems, the rules and state representations in NACE are explicitly defined and consistently logged, making it possible to trace and inspect the learning process.

- **Handling of Uncertainty and Insufficient Knowledge:** NARCES retains NARS's ability to operate under insufficient knowledge and adapt as new infor-

mation arrives. Initially, NARS lacks much knowledge about the environment, or some of its knowledge may be misleading. However, as NACE explores the environment, information is systematically communicated to NARS via the bridge. NARS naturally incorporates the new evidence, updates its knowledge as needed and corrects any inaccuracies. The system doesn't require a complete model beforehand, unlike classical planners; it also doesn't require extensive training, unlike RL agents, what makes NARCES well-suited for open-world environments.

While these capabilities are promising, NARCES also has its constraints, which are inevitably inherited from its two main components. Specifically, there are two main limitations: the performance of MeTTa-NARS, and the ability of NACE to generalize effectively to domains beyond grid environments.

- **Performance and Scalability:** While the performance of MeTTa-NARS is comparable to, or even exceeds, that of OpenNARS, or other reasoning systems, it remains its most significant limitation. As previously discussed, high-level interpreted functional languages tend to have slower performance compared to lower-level compiled languages. Furthermore, the novelty and immaturity of the MeTTa language mean that MeTTa-NARS is highly dependent on the version of the MeTTa interpreter being used. It is expected that, in the near future, advancements in the MeTTa interpreter will lead to more efficient and reliable execution of MeTTa code, including the effective management of MeTTa's Atom spaces. This will potentially allow for the expansion of MeTTa-NARS's data structures and memory capacity, thereby enhancing its performance and scalability.

- **Generalization:** The observed sample efficiency of NACE originates from explicitly exploiting the cell-based grid world state observations for creating tran-

sition rules. This represents a strong inductive bias, which makes NACE less generalizable to non-grid environments as some DRL techniques such as DreamerV3 (*Hafner et al.* (2023)). However, it can nevertheless be valuable in broader applications, as we will see in the next section, where mapping high-dimensional input to a similar discrete world representation is feasible. This mapping, dependent on the problem domain, can be implemented with the appropriate choice of feature extraction techniques. Such approaches are commonly employed in robotics, however we acknowledge this demands a considerable engineering effort.

5.7 Explorer: Generalized Robotics Application

We demonstrate the Explorer, an autonomous robotic agent that leverages the NARCES framework in a realistic setting. Explorer is implemented within the ROS2 (Robot Operating System) ecosystem and operates in a Gazebo simulated environment, using standard robotics tools for sensing and navigation. By employing industry-standard simulation and middleware, the Explorer is positioned for eventual deployment on physical robots in real-world settings. This section presents the framework of Explorer, as well as illustrates how the robot builds a semantic understanding of its environment and uses hybrid reasoning to achieve complex goals in an open-world scenario. Additionally, we discuss the implications for future research.

5.7.1 Explorer Framework and Structure

The Explorer's architecture is built using standard ROS2 nodes that integrate the NARCES hybrid reasoning system with multiple modules for perception, mapping, localization and navigation. The nodes and modules within, interface directly with the Gazebo environment via ROS2 standard communication mechanisms, such as topics, actions and services, for sensor readings and motor commands. The overall framework

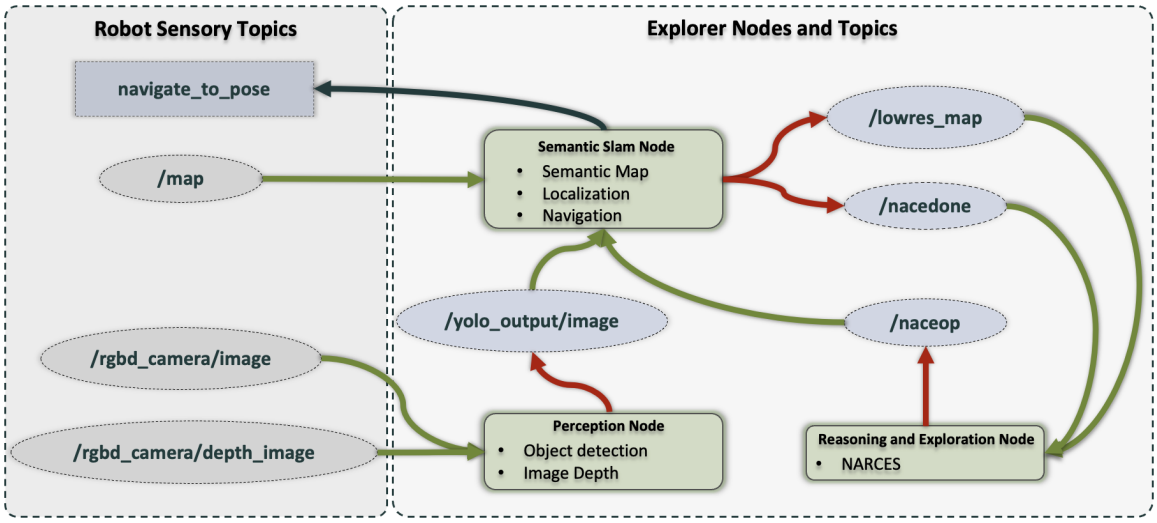


Figure 5.3: Explorer ROS2 Diagram.

consists of the three nodes incorporating multiple modules working in unison which are connected through various topics. Figure 5.3 provides a visual illustration[2] of Explorer integration within ROS2.

Reasoning and Exploration node is at the core of Explorer, it incorporates full NARCES (Symbolic & Sub-Symbolic Reasoner): the integration of the MeTTa-NARS symbolic reasoner with the NACE learning agent. The symbolic component (NARS) handles high-level reasoning, goal decomposition, and knowledge management, while the sub-symbolic component (NACE) handles low-level exploration, learning, and execution of actions. These two are connected via a custom bridge interface within the same ROS node, enabling bi-directional communication. This design allows the robot to reason about its situation and the environment leveraging the capacity of NARS's inference and uncertainty handling, as well as simultaneously learn and explore using NACE's causal rule learning principles.

Perception node handles vision, object detection, and image depth. Explorer makes use of the robot's on-board RGB camera sensor that provides color images

[2]The diagram uses the following visual elements: green squares for Nodes, light blue and gray ovals for Topics, light blue squares for Actions, green arrows for subscription, red arrows for publishing and dark blue arrow for actions.

and depth camera that provides depth information of the image. A vision module processes these inputs to detect and identify objects of interest in the robot's field of view. We implement a real-time object detector using the YOLOv4 deep learning model, which is lightweight enough for on-board operation. Each incoming camera frame is analyzed by YOLO model to produce bounding boxes and class labels for recognized objects (for example: person, chair, table, fridge, etc.). The depth image is used in conjunction with the RGB camera to estimate the distance to each detected object. For every detection above a confidence threshold (we selected 0.5), the system computes the object's 3D coordinates relative to the camera. This is achieved by projecting the pixel coordinates of the detection into the camera's coordinate frame using the depth value and known focal length parameters. The ROS *tf* transform library then converts these coordinates into the global map frame, yielding a position in the world for the object. Thus, the perception subsystem provides a steady stream of labeled observations, which form the basis for semantic mapping and also serve as grounding information for the symbolic reasoner.

Semantic SLAM and Map node integrates several modules that collectively support the creation of a low-resolution map, encompassing functionalities for localization and navigation. This node includes the following components:

- **Semantic Map:** To navigate and reason about space, Explorer builds a semantic map of the environment. The foundation of this map is a standard occupancy grid obtained via SLAM (Simultaneous Localization and Mapping) algorithms in ROS2. This occupancy grid is continuously updated as the robot explores, providing a metric representation of free space versus obstacles. On top of this, the Semantic SLAM module overlays semantic information from the object detector. To integrate with NACE's discrete state representation, the occupancy grid is downsampled to a coarser grid. Each cell in this low-resolution grid carries an integer value representing its semantic state: open

space, unknown, or containing a specific object or structure. Thus, the robot's continuous world is effectively abstracted into a grid-based world model, similar to the grid world environments where NACE was originally developed, thereby bridging the gap between simulation grid worlds and real environments. We will discuss semantic Slam in more detail in the next section.

- **Localization:** The Explorer uses ROS2's transformation and localization utilities to always know the robot's position and orientation on the map. The Localization module queries the robot's pose (e.g. using *tf* transform library to obtain the 'base_link' relative to the 'map' frame) at each update cycle. This provides the robot's coordinates, which are then converted into the low-resolution grid index (by applying the same scaling/downsampling factor). Knowing the robot's grid position is essential for both updating the map (marking where the robot is) and for reasoning (e.g., establishing spatial relations). Additionally, the orientation of the robot is also tracked, which allows the system to determine which direction (north, south, east, west in grid terms) the robot is facing or needs to face when executing a discrete move. In our implementation, the orientation is simplified into four cardinal directions for integration with NACE's action space.

- **Navigation:** To move the robot within the environment according to the subsymbolic agent's decisions, we leverage the ROS2 Navigation stack (Nav2). The Navigation module acts as the executor of action commands, bridging NACE's discrete actions with continuous motion. For example, when NACE creates a plan, which is a sequence of moves (e.g., a plan like "up, up, right, left" meaning the agent should go north twice then right, then left on the grid), the Navigation module translates this into actual target coordinates on the original map. Each grid action of NACE corresponds to a displacement by one cell's

width in the low-resolution semantic map. A destination cell's coordinates are first translated back to the original map and then sent to the Nav2 action server along with the orientation aligned to the direction of travel. The robot, using Nav2's planner and controller, attempts to move to that waypoint while avoiding obstacles. The Navigation module includes safety checks against the semantic map before dispatching a move: for example, if the target cell is marked as occupied, the module can preemptively flag a collision. In such cases, it may modify or truncate the commanded sequence or, if no alternative, report failure of the navigation attempt. This prevents the agent from getting stuck by repeatedly attempting known impossible moves. As each navigation goal is executed, the Navigation module monitors the result via ROS action callbacks. If the objective succeeds (the robot reached the desired location), it notifies MeTTa-NARS through the bridge that the action is complete, thus informing of the execution status. In essence, the Navigation component enables NACE's learned sequence of actions to be realized on a real robot: a discrete "step forward" becomes a smooth drive to the next physical location, and the agent can rely on the lower-level navigation system to handle path planning and obstacle avoidance intricacies in a continuous space.

All these nodes and their modules operate simultaneously and share information through ROS topics, actions and services. For instance, the object detector continuously feeds the semantic map, which in turn is used by the navigation and reasoning modules to decide where to go next. The symbolic reasoner receives established spatial relations from the low-resolution map (through the bridge interface), updates its beliefs about the environment, and derives an objective for NACE to realize on the same low-resolution map. The design of Explorer's framework emphasizes modularity: each capability (vision, mapping, navigation, reasoning) can be improved or replaced independently, and the ROS2-based integration ensures that replacing the

simulated sensors or actuators with real hardware is straightforward. The next subsections delve into key aspects of this framework: the creation and use of the semantic map, and the hybrid reasoning process in an open-world context with a step-by-step example of the robot performing a task.

5.7.2 Semantic SLAM

Semantic SLAM is the crucial module in the Explorer which refers to the process of building a low-resolution map that is both spatial (like a traditional occupancy grid) and semantic (augmented with object labels). This map serves as the common ground where raw sensory data meets symbolic representation. The output of the Semantic SLAM module is a dynamic semantic grid that the robot updates as it explores, and creates spatial relations allowing the agent to understand "what is where" in its environment in a discrete form.

The mapping process begins with standard SLAM: as the robot moves, it uses its sensors (e.g., LiDAR or depth cameras) to estimate the occupancy of the surrounding area, producing a 2D occupancy grid. Each cell in this grid holds a value indicating whether that area is free (navigable), occupied by an obstacle, or unknown. This occupancy grid is fine-grained (on the order of centimeters per cell) to accurately capture walls and small obstacles. However, directly using such a fine grid for high-level reasoning or causal rule learning would be impractical due to the state space size. Therefore, the Semantic SLAM module performs a *spatial abstraction*: it partitions the occupancy map into larger blocks and generates a coarser grid that represents the environment at a higher level. For example, with a downsampling factor of 28, an area of 28×28 cells in the original map corresponds to a single cell in the low-resolution semantic grid. A coarse cell is considered occupied if any obstacle exists within its area, or free if it is completely free space. This conversion dramatically reduces the state dimensionality, yielding a grid that is similar to those on which

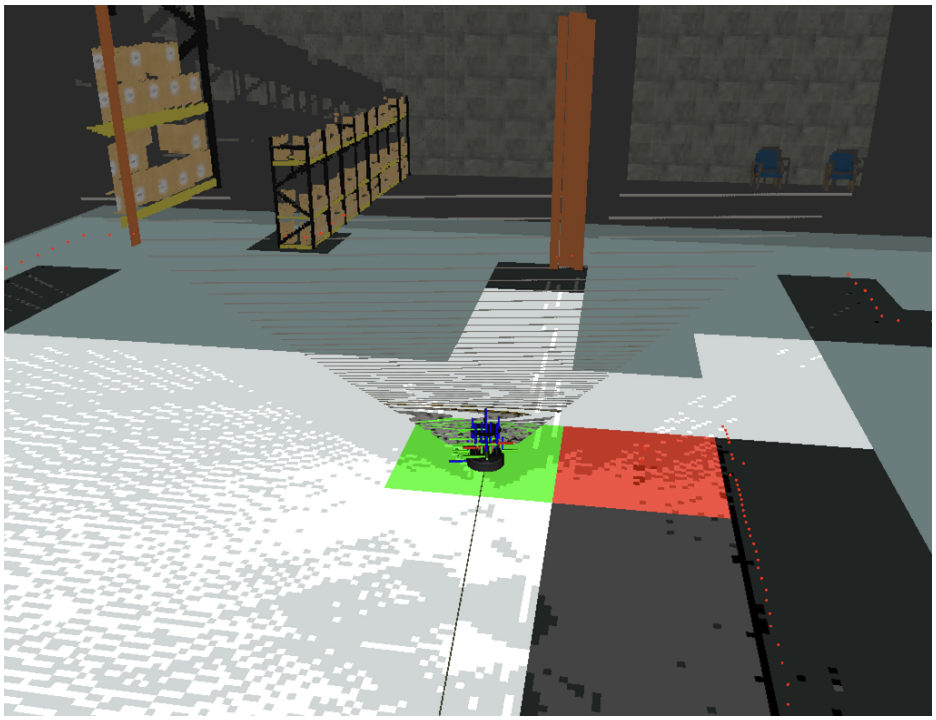


Figure 5.4: Semantic Grid Shown Within RViz Suite.

NARCES has been tested. The downsampling factor is chosen deliberatively such that one cell in the low-resolution grid covers the full area occupied by the base of the robot facilitating reliable and efficient navigation.

Once this discretized spatial layout is obtained, the semantic layer is added. As the object detection system identifies objects, these are integrated into the grid. The integration involves two steps: determining which grid cell corresponds to the object's location and labeling that cell with the object's category. The transformation from camera detection to grid cell is handled by using the robot's pose and camera geometry – effectively projecting the detected object into the global map frame and then finding the nearest grid cell coordinates in the low-resolution grid. That cell's value is then set to a code representing a certain object. We maintain a dictionary of semantic codes that maps labels to cell values, these codes are specifically chosen to be distinguishable as they represent a certain color within the RViz standard visualization suite such as "green" is reserved for the robot while "red" is for a human as seen in Figure 5.4.

By writing these codes into the grid, the occupancy map becomes a semantic map: an array where each index indicates not just "occupied" but "occupied by a certain object".

The semantic grid is dynamically updated at a fixed interval. Every few seconds, the module refreshes the grid based on the latest occupancy data and the latest detections. Previously detected objects remain marked on the map for a certain persistence duration or until they are observed to be absent. This prevents, for example, a detected chair from "disappearing" from the agent's memory just because the robot looked away. However, the system will remove or update an object's entry if new sensor information contradicts the old. This fusion of memory and perception ensures the robot gradually builds a more and more complete semantic picture of the environment as it explores.

An important consequence of semantic SLAM is that it enables symbol grounding for the reasoning system. The symbolic reasoner (MeTTa-NARS) does not directly operate on pixel intensities or raw point clouds, instead, it receives grounded facts derived from this semantic map. For example, as the robot moves, four spatial relations can be formulated at each step: what is "north" of the robot, "south", "east", and "west" in the grid. Because the map encodes certain objects in those directions, the facts input to NARS become meaningful assertions about the environment. The semantic map thus plays a critical interoperability role: it speaks the language of the low-level sensors, NACE sub-symbolic relations and the high-level NARS symbolic AI. It translates continuous sensory input into a symbolic-friendly discrete format, allowing NACE to function in this real-world scenario much as it did in idealized grid worlds.

In summary, semantic SLAM provides the Explorer with a continuously evolving model of its environment that is both spatially and semantically rich. This model is the substrate on which hybrid reasoning is performed, as we discuss next.

5.7.3 Hybrid Reasoning in Open World

Equipped with the semantic mapping capability, the Explorer operates as a hybrid reasoning system in an open-world environment. In this context, "open-world" means the robot cannot assume a fully known environment or a fixed set of facts; it must be prepared to discover new information and adapt continuously. The hybrid approach of NARCES – combining symbolic and sub-symbolic reasoning – is specifically advantageous here, as it marries the adaptability of learning with the interpretability and guidance of symbolic logic.

High-level Guidance of Learning and Acting: The MeTTa-NARS component in Explorer acts as a high-level deliberative layer. It maintains and reasons upon background knowledge and its own derivations about the environment such as objects, location, object properties, and objectives for the robot to realize. For example, NARS may have a belief such as "the kitchen is a room that might contain a person" or derive a goal such as "Find a person". However, NARS alone does not have direct access to the continuous world—this is where it relies on NACE and the connected modules. For instance, if the goal is to find a person, NARS might infer sub-goals such as "navigate to the kitchen area" (if it believes a person is likely there). It then issues objectives to the sub-symbolic layer accordingly. The NACE agent, which interfaces with the Navigation and Mapping modules, takes these objectives and executes them in the environment.

Sub-Symbolic Execution and Learning: On the other side, the NACE component (augmented by the robot's perception and navigation subsystems) handles the uncertainty and complexity of real-world execution. When NARS requests the robot to perform a task, the exact method to achieve is not fully specified—this is where NACE comes into play. NACE interprets the current state from the semantic map (e.g., the layout of obstacles around the robot, or the presence/absence of certain objects) and selects primitive actions that it has learned are effective. In the grid-world

experiments of earlier chapters, NACE learned causal rules through intrinsic reward; similarly, in the open-world setting, NACE continues to learn from trial and error. For example, if a certain corridor consistently leads to dead-ends, NACE's experience (captured as learned rules) will bias against spending too much time in that corridor in the future. The sub-symbolic layer thus provides robustness: it copes with sensor noise, unpredictable outcomes, and environment dynamics by continuously adapting.

Bi-directional Knowledge Exchange: A key aspect of hybrid reasoning in Explorer is the tight feedback loop between the symbolic and sub-symbolic parts. As the robot explores, it encounters new facts: perhaps it discovers a previously unknown room, or it finds that an assumed passage is actually blocked. These observations are encoded in the semantic map and sent back to the symbolic reasoner. The bridge translates observations into Narsese statements that update NARS's semantic memory. For example, when the robot observes a fridge or a stove in the area, the system informs NARS of the objects' presence allowing NARS to reason and deduce that "this area is a Kitchen" (if such a relational concept is given as background knowledge). NARS can then derive new knowledge, or hypothesize that "if a kitchen is found, a person might be nearby" if our goal is "to find a person". Similarly, if NARS initially believed a certain location was clear but the robot's SLAM reports an object, a negative evidence event is generated, causing NARS to adjust its beliefs. This way, the symbolic layer is never static; it is constantly being refined by what the sub-symbolic layer experiences.

Adaptation to the Unknown: The hybrid system demonstrates considerable capability when addressing novel situations. For instance, the robot encounters an unfamiliar object type, one not previously recognized by either NARS or NACE or even maybe labeled as "unknown" in the semantic map because of being absent from the semantic codes of the object detector. In navigating these situations, NACE treats the object as an obstacle initially and then given its experiential learning nature tries

moving it around. If the object moves, it generates the causal rules accordingly and provides corresponding observations to NARS. NARS then creates a new concept for it within its semantic memory and gathers evidence further associating it with this concept, thus enriching knowledge about this particular object and its properties over time. The ability to introduce new symbolic concepts on the fly from sub-symbolic observations is a step toward solving the symbol grounding problem because the system is not limited to a fixed set of pre-defined symbols; it can form new ones based on what it encounters. While our current Explorer prototype doesn't fully implement concept formation for arbitrary unknowns, it lays the groundwork by feeding all perceptual events into NARS. Given NARS's mechanisms for generalization and continuous learning from a stream of events, the robot is well-equipped to build a richer knowledge base than it initially possessed over prolonged deployments.

Operating in an environment inspired by real-world conditions compels the hybrid system to manage continuous time and asynchronous events. The robot may need to perform reasoning while in motion or re-evaluate its plans if an unexpected event occurs, such as the emergence of a dynamic obstacle. Our framework supports the interleaving of reasoning and action: NARS operates within its own control thread, continuously evaluating the inputs and generating outputs at a frequency limited by its inference cycle. Concurrently, NACE and the ROS navigation stack manage the physical execution loop. If an urgent event arises—such as an impending collision with an obstacle—lower-level reflexes, such as those within Nav2's local planner or a potential emergency stop protocol, address the issue immediately. Subsequently, NACE/NARS can integrate the event into their processes, treating it as negative evidence or as a novel condition to be avoided in the future. This approach ensures safety and responsiveness without requiring NARS to manage decisions at the millisecond level. Instead, NARS remains focused on higher-level reasoning, a role suited to its relatively slower, more deliberative process.

In conclusion, hybrid reasoning in the Explorer demonstrates that a robot can simultaneously reason and learn in an open-ended environment. The symbolic reasoner provides a form of common-sense understanding and goal-directed planning, while the sub-symbolic learner provides adaptability and sensorimotor skills. Together, they address each other's weaknesses: NARS gives NACE direction beyond exploration, and NACE gives NARS real-world grounding and the ability to carry out complex tasks that would be infeasible to script purely symbolically.

5.7.4 Future Works

The development of the Explorer agent provides a proof-of-concept for integrating MeTTa-NARS's high-level reasoning with NACE's experiential learning in robotics. However, there are several avenues to expand and enhance this framework:

- **Manipulator Arm Integration:** In the current simulation, the Explorer system operates using the TurtleBot3 platform — a small, affordable, and customizable ROS-based mobile robot that does not include additional actuators such as a manipulator arm or gripper. An important extension to the Explorer is the integration of such a manipulator, which would enable the robot not only to perceive and navigate through its environment but also to physically interact with it. This enhancement would allow the robot to perform manipulation tasks such as picking up, moving, or placing objects, which are essential capabilities for applications in service robotics, warehouse logistics, and domestic assistance. Given that NARS already possess an extensive and robust procedural inference control, the inclusion of a manipulator would fit naturally within the hybrid reasoning paradigm of NARCES as it will utilize NARS high-level planning and decision making while relying on the sub-symbolic experiential learning.

  Consider a toy example: the robot has detected a cup on the table, and NARS receives a high-level goal from the user to "pick up the cup and bring it to the

human". In order to be able to execute "pick" operation, NARS has to receive a recent event such that "robot is near the table". Given that the robot is in the different region of the map, NARS derives a sub-goal and publish a planning objective for NACE to "move next to the table". Once the robot reaches the table, it provides feedback back to NARS and NARS starts executing "pick" operation to grab a cup. After a successful grasp, NARS will similarly derive and communicate the goal of delivering the cup to the human, issuing additional navigation and manipulation objectives. This example highlights the interleaving of high-level sub-goal derivation and low-level execution, demonstrating the strength of hybrid reasoning in practical, embodied tasks.

However, integrating a manipulator also introduces new technical challenges. On the perception side, the system must reliably estimate the 3D pose of objects, assess graspability, and potentially identify interaction affordances from visual or depth data. On the planning side, arm motion planning must be coordinated with mobile base positioning, particularly in cluttered or constrained environments. Incorporating manipulation into the Explorer agent significantly increases its embodied intelligence, moving from passive observation and movement toward active physical interaction with its surroundings.

- **Real-World Deployment:** One important next step is to validate the Explorer system on actual robotic hardware outside of the simulation. While ROS2 and Gazebo allowed us to simulate sensors and environments with reasonable fidelity, real environments introduce additional challenges such as sensor noise, actuator uncertainty, and unforeseen physical interactions. Deploying Explorer on a physical mobile robot will test the robustness of the system's modules. It may require integrating more advanced localization (e.g., using wheel odometry and visual SLAM together) and robustifying the object detection (handling varying lighting and image noise). The expectation is that the hybrid reasoning

approach will continue to offer advantages in the real world, especially its ability to adapt at runtime; but performance may need tuning. For instance, real-world testing could lead to improvements in how the semantic map handles false positive detections or how the navigation module deals with dynamic obstacles (like people moving around). Overcoming these challenges will move NARCES closer to a deployable autonomous agent for tasks like domestic service robots or search-and-rescue missions.

In summary, the Explorer and the underlying NARCES framework open up a rich landscape of future research. The immediate next steps involve scaling up to more complex environments and tasks, and moving from simulation to the real world. On the algorithmic side, enhancing the integration between symbolic and sub-symbolic components will push the boundaries of what autonomous agents can achieve. Ultimately, this line of work aims to contribute to a "unified foundation for reasoning and learning" in embodied agents – one that combines the adaptability of learning systems with the clarity and generality of symbolic reasoning, enabling machines to handle the complexity and uncertainty of the real world in a robust and intelligent manner.

CHAPTER 6

SELECT APPLICATIONS AND EXPERIMENTS

6.1 Introduction

In this chapter, we present two integrated neural-symbolic frameworks, each leveraging the Non-Axiomatic Reasoning System (NARS) as a core component for reasoning under uncertainty. These frameworks exemplify early attempts to hybrid systems, wherein sub-symbolic neural architectures are combined with the symbolic reasoning capabilities of NARS to harness the strengths of both paradigms for effective performance.

The first framework involves the integration of NARS with mobile robot to improve planning and decision-making processes. NARS enables robots to effectively handle uncertainty in real-time with complete sensor and actuator integration, including a neural object detection model, thereby ensuring adaptability to evolving scenarios. This integration is part of its deployment as a ROS node. A case study is provided demonstrating the system's proficiency to carry out a garbage collection task in an open-air environment by operating a mobile robot with manipulator arm, and we demonstrate its ability to learn about the place-dependent accumulation of garbage items. Case study also reveals that our approach performs more effectively on the overall task than the Belief-Desire-Intention model we compared with.

We then present NARS-GPT, an integrated multi-component system, which combines the power of the Generative Pre-Trained Transformer (GPT) with the reasoning capabilities of NARS. Such combination enables the system to effectively respond to questions posed in natural language while retaining the capacity to store inferred

important information for future use. The GPT element readily converts natural language into formal representations enabling seamless user interaction, while NARS performs real-time reasoning on these representations and grounds them automatically by relating them to observed events. This represents a novel solution to the symbol grounding problem which does not depend on the designer to link a selected set of pre-defined symbols to the perception model, hence allowing for autonomous acquisition of grounded concepts from natural language input at runtime. NARS-GPT is capable of long-term learning through interactive Q&A sessions with users and continuously enhances the system's knowledge base, thereby ensuring adaptability to evolving scenarios.

6.2 Non-Axiomatic Reasoning for an Autonomous Mobile Robot

In the rapidly evolving field of robotics, the development of autonomous systems has become a focal point of research to allow for the automation of crucial tasks within society that are currently performed manually. This includes not only inspection tasks but also tasks which demand some form of manipulator such as garbage collection, which is labor-intensive and critical for maintaining cleanliness in urban environments. To address such challenges, we introduce a novel approach by employing a Non-Axiomatic Reasoning System (NARS) for planning and decision making of a mobile robot with manipulator arm. (illustrated in Fig. 6.1).

NARS is an advanced reasoning system that handles uncertainty and learns cause-effect relations from observation in real-time, rather than solely using a predefined set of rules, making it a valuable asset for applications and domains requiring adaptability and flexibility. NARS comprises a logic and control mechanism with associated memory structure. For this application, NARS's Temporal and Procedural infer-

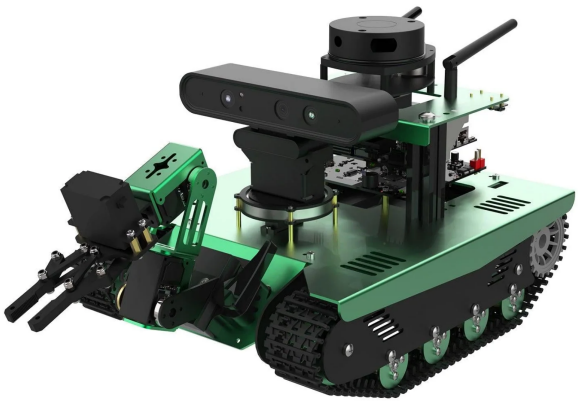


Figure 6.1: Yahboom Transbot.

ence mechanisms are crucial for permitting NARS-based robots to perform their task efficiently and robustly. The essential components of the logic as well as relevant definitions, concepts, and the key operational principles of the Temporal and Procedural mechanisms, have been thoroughly covered in Chapter 2, specifically in sections 2.3.8 and 2.3.9. Thus, a detailed discussion about NARS will be omitted in this section.

Our contribution is an integration of NARS into Robot Operating System (ROS) for mobile robots which we provide as a portable open-source package with a comprehensive set of sensory information encoders and an interface for the reasoning system to instruct motor actions. Hereby we will answer the question whether NARS is a suitable reasoning technique for autonomous ROS-based mobile robots and if its distinguished features such as real-time learning and uncertainty handling can be beneficial for such applications.

To illustrate the effectiveness of our solution, we present a case study conducted in the city of Aarhus, Denmark, using Yahboom Transbot (Fig. 6.1). We experimentally demonstrate the system's ability to successfully carry out a garbage collection task in an open environment, and more reliably than a Belief-Desire-Intention model. Furthermore we show the systems's capacity to learn about the expected accumulation of trash objects at different places. This adaptability is a cornerstone of our system,

as it enables the robot to respond to changes in the environment and to deal with unexpected disturbances which can appear at runtime.

6.2.1 Related Works

The strength of practical means-end reasoning lies in the ability to effectively utilize high-level knowledge for planning purposes, which is easily expressible and communicable by humans. However, available Practical Reasoning solutions with logic derived from First Order Predicate Logic, such as *Bordini and Hübner* (2005); *Ferrein et al.* (2012); *Andres et al.* (2015); *Rovida et al.* (2017), lack structure learning and handling of uncertainty which could enhance the adaptability of task planners *Li and Ding* (2023). This also includes non-monotonic reasoning systems *Vilchis-Medina et al.* (2021) which, albeit may retract existing hypotheses when facing contradicting information, are not robust in settings with inherent uncertainties where even the best hypotheses sometimes are contradicted by new data. There also has been a work regarding structure learning from data *Diehl et al.* (2021), such as the learning of new cause-effect relations using Schema Network *Kansky et al.* (2017) and Inductive Logic Programming (ILP) *Karapinar and Sariel* (2015). Nevertheless, both schemas and induced FOPL predicates demand to be consistent with all previously observed events making them inapplicable in non-deterministic and complex environments with inherent uncertainties, an issue also faced with description logics *Tenorth and Beetz* (2009). While probabilistic extensions of ILP have been created, solutions such as *Riguzzi et al.* (2014) are not suitable for real-time application on mobile robots.

Other common systems operating with fuzzy logic *Qiu et al.* (2012) or known probability values, such as Probabilistic STRIPS extensions *Teichteil-Königsbuch et al.* (2011) that utilize Probabilistic Planning Domain Definition Language (PPDDL *Younes and Littman* (2004)) take uncertainties into consideration, but they depend on both pre-defined causal structure and value assignments *Hellström* (2021). There

also has been works *Zhang et al.* (2012) to utilize *Lauri and Ritala* (2016) or update probability values in Partially Observable Markov Decision Processes *Spaan* (2012), but these lack structure learning, i.e. learning of new causal relationships. Following these approaches, the sample space sizes on which the probability evaluations are based are not explicitly tracked, which, besides having to adhere to the axioms of probability spaces, makes incremental updates difficult. In contrast, NARS provides a way to efficiently support structure learning with local memory updates, and perform incremental evidence updates of existing hypotheses, whereby memory capacity constraints are respected *Wang* (2009a).

Our continued work *Hammer et al.* (2022) is largely motivated by the fact that this type of reasoning system can enhance a robot's ability to handle uncertainty, including "unknown unknowns".

6.2.2 ROS Integration

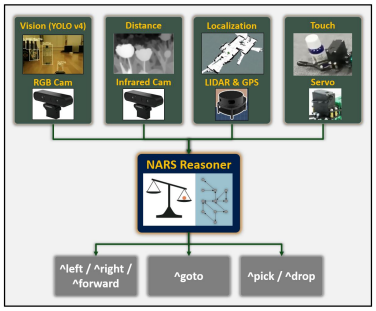


Figure 6.2: Multi-modal Integration Illustration.

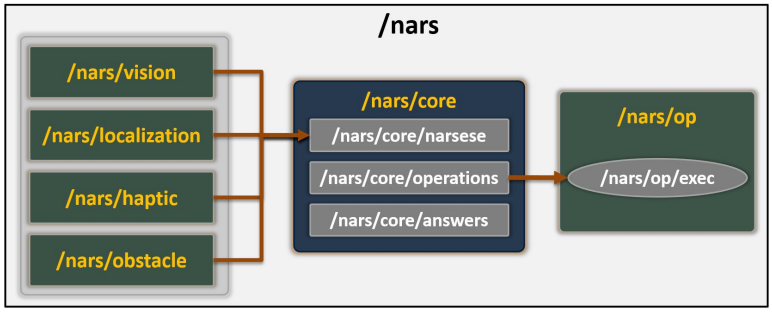


Figure 6.3: Corresponding ROS Network for NARS.

To make NARS easily portable on mobile robots, we provide an open-source software package which embeds the reasoner into ROS Melodic, where more widely useful nodes for mobile robots have been integrated and connected to the reasoning node. This includes standard ROS nodes for Simultaneous Localization and Mapping (SLAM), the navigation stack, nodes for visual object detection, and related encoders

to map the pre-processed information into a format the reasoner can work with. In Fig. 6.3 we see the channel nodes providing perceptual input to the NARS reasoner from the various sensors illustrated in Fig. 6.2.

- **Vision:** A broadly applicable vision channel that can interface convolutional neural network models for object detection such as provided by darknet_ros with the reasoning system. The vision channel uses discretized relative to the center location information along with the output label to form statements of the form
  $< objectLabel \rightarrow [direction] >$ similar as in *Hammer et al.* (2021) but with the relative location encoding where *position* for the X-axis is either $left$, *center* or $right$. This form of encoding makes NARS aware of the detected object types and their position in the camera's field of view.

- **Obstacles:** An event coding providing nearest-obstacle awareness based on laser scan input. Hereby distance and direction to the nearest object within 2 meters of range are determined. This is utilized to make the reasoning system aware of objects to be avoided in close proximity. The event encodings in this case are $< obstacle \rightarrow [direction] >, < obstacle \rightarrow [free] >$ with $direction \in \{left, front, right\}$.

- **Localization:** Encoders from the output of localization estimations such as based on SLAM, as mobile robots often utilize. The estimated location is periodically (every second) sent to the reasoner as a relational statement $< (SELF * location) \rightarrow [at] >$ whereby *location* encodes the robot's pose (location and rotation).

- **Haptic** is responsible for manipulator feedback of a gripper. It indicates, with a Boolean whether the robot has managed to get a hold on an object via an

applicable motor procedure. The feedback is encoded as an event $< gripper \rightarrow [holding] >$ or $< gripper \rightarrow [free] >$.

- **Navigation:** NARS is able to send navigation goals to the ROS navigation stack. Thus, the reasoning system can invoke local base movements for moving *^forward*, *^left*, *^right*, and may send goals to reach distant locations on the map obtained from SLAM (via *^goto*).

- **Manipulation:** The system can invoke motor procedures for object manipulation (such as *^pick* and *^drop* which can be manually coded or defined with MoveIt).

The reasoner core node (as in Fig. 6.2), encapsulating NARS, accepts new inputs as strings formatted in the formal language the reasoner works with, by subscribing to the Narsese topic of the ROS node. To let NARS control the robot, operation executions are redirected to the *Exec* node for further dispatch. Hereby the ROS navigation stack is utilized for robot base movements, by publishing *move_base/goal* desired target location goals (both local-relative, and map-absolute) and canceling the previous ones when new operations are invoked by NARS (via *move_base/cancel*). Also, the manipulator arm of the robot is controlled by adjusting the angles of the servo motors based on the location of the detected object to pick and its corresponding depth estimate.

6.2.3 Case Study: Garbage Collection

The case study presents the testing and optimization of an autonomous crawler robot (Fig. 6.4) controlled by our NARS ROS package in a real-world environment for the purpose of garbage collection. The chosen test site is located in Aarhus, Denmark and is recognized as one of the largest of its kind for autonomous robots in Europe, offering a diverse range of environments for comprehensive testing. The

testbed encompasses a bustling recreational area, a concrete football field, and a park. These diverse settings allowed us to simulate various real-world scenarios that the robot might encounter during its operation. The testbed's design and public utilization closely mimics the typical trash accumulation patterns observed in urban areas, thereby providing a realistic environment for testing the robot's performance.

6.2.4 Integration on Transbot

We demonstrate that NARS ROS package is capable of:

- operating the robot with sub-second perception-reason-action cycle featuring full sensor & actuator integration
- learning and updating hypothesis evidence to deal with failure and learning about the garbage items distribution within the environment
- performing the overall task more effectively than the Belief-Desire-Intention model we compared with
- utilizing high-level knowledge and operator input regarding checkpoints during the reasoning process.

We deliberately utilized Yahboom Transbot (Fig. 6.1) to integrate and demonstrate the capabilities of our ROS package on low-cost hardware. Transbot is a mobile robot with tracks and a manipulator arm with 3 degrees of freedom. It incorporates the following sensors: an RGB and depth camera (combined, via Astra Pro), a 2D

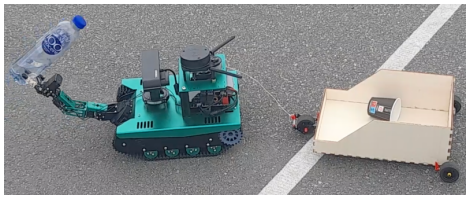


Figure 6.4: Garbage Collection in Aarhus Robotics Testbed.

Lidar sensor (RPLidar), and an angle reading sensor for each servo motor of its manipulator. In terms of compute power, Transbot uses a Jetson Nano with an ARM CPU, low-powered NVIDIA Tegra GPU and 4 GBs of RAM.

We chose the following software configuration:

- **Reasoner:** OpeNARS for Applications (ONA) v0.9.2[1] with default parameter configuration.
- **Vision:** YOLOv4 from tensorrt_demos[2] was utilized as object detection model, as it is small enough to be accelerated using the on-board GPU of Jetson Nano, providing low but task-sufficient 4 frames per second.
- **Haptic:** Since the servo motors of the manipulator arm do not provide a sensor measuring the current, we instead check whether the rate, with which angle of the gripper changes during the relevant part of the pick operation, is below a threshold. This is being used to generate the previously described Boolean flag indicating whether the gripper is holding an object.
- **Manipulation:** For manipulation we created a handcrafted motor procedure based on proportional control, using the detected bounding box location as control input, as well as the feedback event generated from the haptics handling. While this is clearly not state-of-the-art, it is sufficient for this part of the task.

6.2.5 Testing Conditions and Observations

The robot was tested to collect different types of trash objects such as bottles and cups of different materials including plastic, paper and glass. The robot's performance was evaluated under various weather and lighting conditions, including sunny, cloudy, and low lighting conditions. Because the solution consists of multiple components

[1]Repository: https://github.com/opennars/OpenNARS-for-Applications
[2]Repository: https://github.com/jkjung-avt/tensorrt_demos

Table 6.1: Object Detection Performance

Condition	Detection Rate	Time
Evening Sun	61%	4 PM - 6 PM
Cloudy	70%	11 AM - 1 PM
Sunny	83%	11 AM - 1 PM

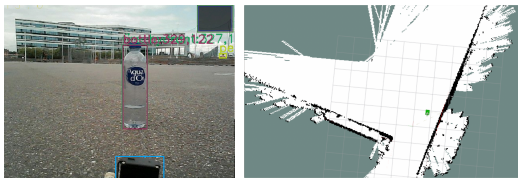


Figure 6.5: Visual Object Detection and Corner Mapping.

and failure is often caused by one or more of them, we report the success rate of the different components under the different conditions. For reproducibility purposes the final quantitative measures were taken in a 22x22 meters square area within the football field of the testbed. Also the conditions present during testing and their respective times were recorded.

Object detection model: We found the YOLOv4 object detection model of the robot which was pre-trained on ImageNet to be sensitive to the lighting conditions during the day, with the detection performance for bottles and cups decreasing from approximately 83% to 61% in the evening sun. The detection rate was around 70% in cloudy conditions, and optimal lighting conditions were observed between 11 AM and 1 PM under clear skies, as illustrated in Table 6.1. These numbers are reported with a YOLOv4's minimum confidence value of 0.3. Further reduction in the confidence level led to a strong increase of false positive detections, and sometimes to duplicate bounding boxes, while an increase resulted in missing relevant detections more frequently.

SLAM: Localization beyond the Lidar range of 10m led the robot to fully depend on its odometry and IMU. This proved to be sufficient when a corner or significant markings were mapped beforehand and when the Lidar-blind area was only traversed

by the robot without any checkpoints in that area which would cause intermediate rotations of the base.

However, extensive navigation with waypoints within such blind spots of the map beyond Lidar range to obstacles resulted in failures of SLAM building a consistent map and in successive localization errors. Therefore, in our setup, we ensured the corners were mapped out before operation and checkpoints were laid out at the boundaries of the testbed so that the robot could traverse the middle of the testbed in a relatively straight line without causing localization issues.

Manipulation: The proportional manipulator control approach exhibited issues on-site that were not observed in the lab in a way that friction on a concrete surface, due to gravel and other factors, behaved more unpredictably. This initially caused bottles to lean over in most cases before the gripper was able to secure them, as the controller approached the target object too aggressively. The issue was resolved by setting smaller constants in our proportional actuator controller, which avoids overshooting the target object when aiming at and grasping. Additionally, outdoor imagery reduced manipulation performance in conditions with strong shadows (evening sun) due to less accurate bounding boxes.

6.2.6 Final Test Setup and Results

The final test setup involved the robot collecting trash items in a designated testbed within the football field of the area in sunny conditions. Hereby we followed a repeated procedure that ensures reproducibility between the test runs and software adjustments, as well as replication of the same setup with other techniques for comparison purposes. While the test runs were performed within the public area where quantitative measures were taken, the area was fenced off to avoid any external interference because of safety regulations.

The **Test Procedure** we employed is as follows:

Table 6.2: Object Collection Performance

Method	Success Rate	Runs
NARS	68%	30
BDI	56%	30

1. The reasoner is given the background knowledge:

```
// Go to next checkpoint to arrive there:
<(<gripper --> [open]> &/ <({SELF} * (next * $P)) --> ^goto>) =/>
 <({SELF} * {$P}) --> at>>.
// If gripper is open and trash is seen, pick it up to hold it
<((<trash --> [see]> &/ ^pick) =/>  <gripper --> [hold]>>.
// If the gripper is holding an object, drop it to the trailer
<(<gripper --> [hold]> &/ ^drop) =/> G>.
// bottles and cups are both trash
<bottle --> trash>.
<cup --> trash>.
```

2. The robot was positioned at a starting point (left-upper corner in Fig. 6.6)[3] within the testbed and sent on an initial control route to generate the initial map which has the corners of the testbed included.

3. Bottles and cups of different sizes (10 objects, within the gripper width limits) and materials (plastic, paper and glass) were placed at the checkpoints in upright position, and the success rate in object detection and manipulator control was recorded during the mission.

4. Checkpoints were established in a circular route through the testbed (as illustrated in Fig. 6.6), which the robot used as named guidance points to navigate.

[3]Checkpoint placement (orange) and example trajectory (blue), collected garbage items (green), missed garbage items (red)



Figure 6.6: Checkpoint Placement, Trajectory and Garbage Items.

While for grasping well-detected objects we observed a success rate of 85%, the overall success rate of finding and collecting the existing trash objects peaked at 68%, resulting in slightly more than two-thirds of the placed objects being collected. This corresponds roughly to the 83% detection rate multiplied by the 85% success rate of the pick operation, minus 2% error introduced by rare occasions of improper decision making by the reasoning system (such as moving on to another checkpoint even though trash was detected).

6.2.7 Comparison with BDI Model

We compared our solution with a Belief-Desire-Intention (BDI) model as a baseline (via Jason v3.2.0 *Bordini and Hübner* (2005)), for which above background knowledge was directly translated. With the BDI model we only reached 56% success rate on the task, which can largely be explained by a lack of evidence updating for the success of picking up individual objects. In our runs the BDI model running with same knowledge tried to repeatedly re-pick objects it could not pick (such as objects that had been leaned over by the robot, as it still matched the plan precondition) until it did not see an object and moved on to the next checkpoint, while NARS, due to its evidence updating mechanisms, moved on to the next observed object or another checkpoint after repeated failure. In NARS, consistent with the working principles introduced before, this is due to updating the hypothesis for picking up a specific object instance, e.g.

```
<(<{instanceX} --> [see]> &/ ^pick) =/> <gripper --> [hold]>>.
```

receives negative evidence when the object instance is seen however the confirmative event indicating the gripper is holding an object is not provided after executing the pick operation. Now, while we are aware that a similar principle of applying success rate counting could have been programmed into a BDI system by extending the knowledge base accordingly, it is important to note that this kind of evidence updating is central to NARS and is useful not just to learn which objects can or cannot be picked up, but also to learn other potentially useful knowledge about the environment and to deal with different kinds of disturbances in that way. In this particular use case, NARS also learned at which checkpoints trash is seen more frequently, which can be used to make the robot first explore areas where trash accumulates more often. In our setting for a particular checkpoint P_i, NARS forms individual hypotheses as:

```
<<({SELF} * {Pi}) --> at> =/> <trash --> [see]>>.
```

with confidence value representing how often the robot has visited checkpoint P_i, and its frequency at which fraction thereof trash has been observed there. While truth values are automatically utilized in decision making using the mechanisms described in earlier sections, it can also be explicitly queried which makes the robot's behavior transparent. For example, the reasoner may be asked whether and where it has seen trash with the question query as follows:

```
<<({SELF} * {?1}) --> at> =/> <trash --> [see]>>?
```

It is important to note, that the above question is not required for the hypothesis formation or its evidence update, but serves only for the user to check the evidence of the statement.

6.3 NARS-GPT: An integrated system for natural language interactions

The capacity to understand, reason and communicate in natural language is a cornerstone of human intelligence. However, the creation of machines capable of performing these tasks has posed a significant challenge. A recent breakthrough in natural language processing is the Generative Pre-Trained Transformer (GPT, *Radford et al.* (2018)), which has resulted in a variety of tools capable of handling natural language inputs *Min et al.* (2021) and been prompted to perform a broad range of tasks *Bubeck et al.* (2023). Nevertheless, GPT models face limitations, in particular, they are constrained by a context size limit of a few thousand tokens and require augmentation with long-term memory to facilitate the retention of user input over extended periods. When the information is retained for longer, the need for updating the system's knowledge arises, as the newly encountered knowledge can either support or contradict previously acquired information. In such scenarios, it is beneficial for the system to be able to formulate and store its conclusions for long-term use and to reason in a dependable manner. Additionally, it is necessary to relate natural language terms to sensorimotor representations in order to become useful in the real-world scenario, unless there is a human interpreter in the loop solving this problem for the system (e.g. a human asking ChatGPT for advice and then using the knowledge in the real world). This is known as Symbol Grounding problem discussed in *Harnad* (1990).

To address these challenges, we introduce NARS-GPT, a multi-component integrated system that combines the strengths of the Non-Axiomatic Reasoning System (NARS) with the Generative Pre-trained Transformer (GPT) language model. This integration enables the system to reason efficiently and reliably, to ground language terms by relating them to sensorimotor representations, and effectively respond to

questions directly from natural language inputs, making it a valuable asset for various applications and domains. NARS-GPT also features a distinguished ability of learning over prolonged periods through interactive Q&A sessions. The ongoing learning process is accomplished using the system's advanced attention control and memory management mechanisms which allows for adaptability to evolving circumstances. Furthermore, NARS-GPT adds the novelty to interact with NARS via natural language. This is particularly valuable because NARS has been successfully applied in domains including Traffic Surveillance *Hammer et al.* (2021), Mobile Robotics *Hammer et al.* (2024) and Driving Assistance *Yun et al.* (2022), where natural language-based user interaction can add additional merit

6.3.1 Related Works

Concerning Long-Term Memory, Ret-LLM *Modarressi et al.* (2023) includes a read-write memory for LLMs which extracts memory triplets from user-provided knowledge. These triplets are then stored and queried from a tabular database using vector similarity search. While Ret-LLM demonstrates promising recall capabilities, the system cannot derive and explicitly store inferred information for later use. Additionally, quantitative results measuring benefits over existing techniques are lacking. Memorizing Transformers *Wu et al.* (2022) store key-value pairs in long-term memory, which are retrieved via k-Nearest-Neighbours (kNN) search and then included in the final transformer layers of a LLM model. Their result requires fine-tuning for memory updates. LongMem *Wang et al.* (2023d), which outperforms Memorizing Transformers, includes a non-differentiable memory bank and uses sparse attention to avoid quadratic cost of self-attention. Finally, Memory Bank *Zhong et al.* (2023) was created to support sustained long-term interaction with chatbots. It utilizes a large array structure with timestamps for the memory items and memory updating mechanism, and while this supports memory updates without involving Backpropa-

gation, it does (as *Kynoch and Latapie* (2023)) not incorporate a reasoning system. Approaches combining LLMs with reasoning systems however do exist. In LOGIC-LM *Pan et al.* (2023) and LARK *Choudhary and Reddy* (2023) the integration is restricted to deductive reasoning and not combined with a Long-Term Memory.

Furthermore, previous solutions exhibit limitations to Symbol Grounding, such as designers specify an interface of pre-defined symbols being deliberately grounded into perceptual representations *Taddeo and Floridi* (2005), or left to representation learning *Guo et al.* (2019), whereby human interpretability is lacking. A proper solution to the symbol grounding problem would allow new terms to be introduced and grounded at the runtime in a data-efficient way.

6.3.2 NARS-GPT Architecture

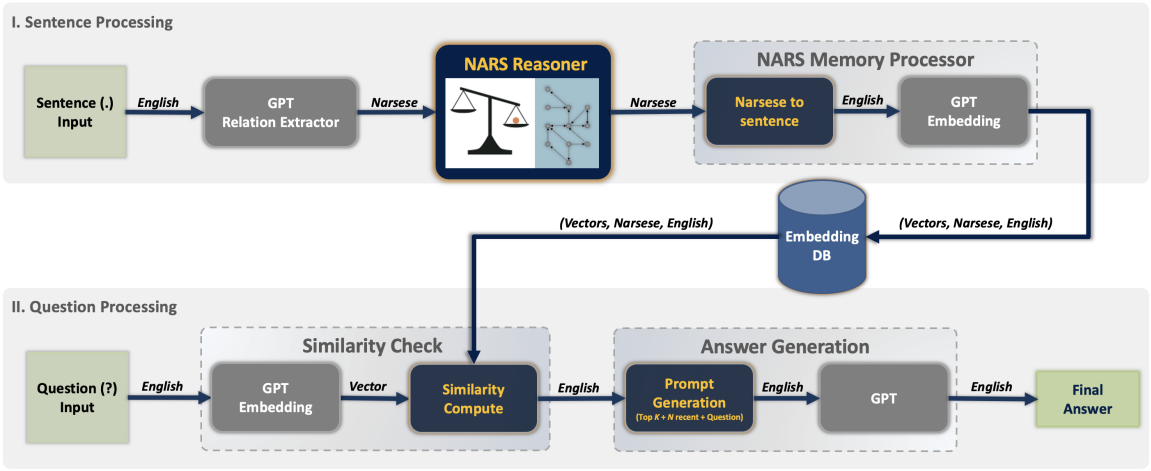


Figure 6.7: NARS-GPT Overview.

The NARS-GPT architecture is composed of two main components: GPT and NARS. While GPT is tasked with transforming natural language inputs into formal representations as well as calculating sentence embeddings, NARS is responsible for evidence gathering, belief revision, uncertainty estimation, and reliable multi-step reasoning using these formal representations.

The GPT component is a LLM which was trained on a vast text data corpus by OpenAI. Its ability to generate coherent and grammatically accurate text makes it an excellent tool for processing natural language, but more importantly GPT can be prompted to transform the natural language inputs into formal representations that NARS can process.

The NARS component is a stand-alone reasoning system capable of executing various forms of reasoning, including declarative, most importantly deductive, inductive, and abductive, as well as temporal and procedural reasoning. It can also estimate uncertainty in the real-time and revise its beliefs based on available new evidence. NARS consumes and processes the formal representations produced by GPT and carries out the necessary reasoning steps in a reliable, data-efficient and human-interpretable way.

As shown in Figure 6.7, the overall information flow occurs in two main phases, Sentence Processing and Question Processing, depending on the punctuation of the input.

6.3.3 Sentence Processing

1. Sentences are input into the system in natural language, and relations are being extracted from the sentences as *Subject-Verb-Object* triplets by "GPT Relation Extractor" (the gray box in Figure 6.7).

2. The triplets are being transformed into *Narsese* statements (formal language), where for each term the embeddings are calculated to use existing terms if the embedding is similar enough, and fed into the reasoner to obtain derivations.

3. Inputs and derivations in Narsese are translated back into English by "NARS Memory Processor", and a sentence embedding is being computed and associated with each item.

4. NARS belief (*Narsese* statement), its corresponding sentence in natural language, and its sentence embedding are stored in the vector database for future use during the Question Processing phase.

This can be summarized in the following algorithm:

Algorithm 2: Belief Processing

procedure BELIEFPROCESSING(sentence)
    $triplets \leftarrow$ extractTriplets($sentence$)
    $narsese \leftarrow$ transformToNarsese($triplets$)
    $inputs, derived \leftarrow$ inputAndReasonOn($narsese$)
    $beliefs \leftarrow$ toEnglish($inputs, derivations$)
    $embed \leftarrow$ computeEmbedding($beliefs$)
    storeToDB($(inputs, derived), beliefs, embed$)

6.3.4 Question Processing

1. Question is input into the system in the form of natural language, and a sentence embedding vector is being computed for it using "GPT Embedding" (the gray box within "Similarity Check" of question processing in Figure 6.7).

2. NARS beliefs are being retrieved from the vector database according to their similarity[4] to the question embedding vector.

3. Prompt generation is performed using k-"similar" NARS beliefs from the previous step and additional n-recent items retrieved from the vector database along with the question; all in the form of natural language sentences.

4. GPT is invoked to answer using the generated prompt and to reference the correct index of the included NARS belief and its truth value.

Pseudo-code summarizes the Question Processing below:

[4]Similarity check is performed using OpenAI API

Algorithm 3: Question Processing

```
function QuestionProcessing(question)
    embedding ← computeEmbedding(question)
    relevant ← relevantFromDB(embedding)
    recent ← recentFromDB(embedding)
    prompt ← PromptEncode(relevant, recent)
    answer ← promptGPT(prompt)
return answer
```

6.3.5 Translation Process

Translations between English and Narsese formal language are essential for enabling NARS-GPT functionality. In order to translate English to Narsese during the Sentence Processing phase, the GPT Relation Extractor utilizes a prompt with a description of the *Subject-Verb-Object* relations and negated counterparts:

```
RelationClaim(noun,verb,noun)
... this relation is claimed true in the sentence
RelationClaim(noun,"IsA",noun)
... this relation is claimed true in the sentence
PropertyClaim(noun,"HasProperty", adjective)
... this relation is claimed true in the sentence
NegatedRelationClaim(noun,verb,noun)
... this relation is claimed false
    in the sentence with an explicit 'not' word
NegatedRelationClaim(noun,"IsA",noun)
... this relation is claimed false
    in the sentence with an explicit 'not' word
NegPropertyClaim(noun,"HasProperty",adjective)
... this relation is claimed false
    in the sentence with an explicit 'not' word
```

Table 6.3: Translation from the various triplets to Narsese

Triplet	**Narsese**
RelationClaim(noun,verb,noun)	$< (noun * noun) \longrightarrow verb >$
NegatedRelationClaim(noun,verb,noun)	$!(< (noun * noun) \longrightarrow verb >)$
RelationClaim(noun,IsA,noun)	$< noun \longrightarrow noun >$
NegatedRelationClaim(noun,IsA,noun)	$!(< noun \longrightarrow noun >)$
PropertyClaim(noun,"HasProperty", adjective)	$< noun \longrightarrow [adjective] >$
NegPropertyClaim(noun,"HasProperty", adjective)	$!(< noun \longrightarrow [adjective] >)$

Please note, the descriptions include "IsA" and "HasPropertyOf" as a special case, since they directly map to logical relations, as we will see later. In addition, the following exact instructions are provided within the prompt, verbally describing how the relations are to be extracted from the input sentence:

```
Capture the complete sentence meaning with
code that calls the four functions. Please
make sure that the word "not" is not included
in your call, just use the functions and
Negated functions instead. And use verbs for
comparative relations! Encode all relations in
the sentence, and the sentence has to be believed!
```

Above prompt has been tuned particularly for GPT-4 to make it utilize the description for the translation to triplets. Once the triplets are generated, translation between Narsese and English is carried out according to the cases captured in Tables 6.3 & 6.4. Please note that the triplets are initially obtained from the input sentence and then are translated into Narsese, while NARS derivations already come in Narsese form and are being translated back to English.

Below we can observe the example of the translation process:

Table 6.4: Translation from Narsese to English

Narsese	**English**
$< (noun * noun) \longrightarrow verb >$	noun verb noun
$!(< (noun * noun) \longrightarrow verb >)$	noun not verb noun
$< noun \longrightarrow noun >$	noun is a noun
$!(< noun \longrightarrow noun >)$	noun is not a noun
$< noun \longrightarrow [adjective] >$	noun has property adjective
$!(< noun \longrightarrow [adjective] >)$	noun does not have property adjective

Input: Sam is a cat
Input: Sam is eating a pizza

NARS Memory:
('<sam ⟶ cat>', 1)(1, 1, (1.0, 0.9), [1])
('<(sam * pizza) ⟶ iseating>', 2)(2, 1, (1.0, 0.9), [2])

Prompt view of memory:
i=0: sam isa cat. time=1 conf=0.576
i=1: sam iseating pizza. time=2 conf=0.720

The structure of each memory item consists of two tuples: a key-tuple and a value-tuple. The key-tuple is of the form $(term, occurrenceTime)$ because the same statement may appear at multiple times with different truth value, while the value-tuple has the form of $(lastUsed, useCount, (f, c), evidentialBases)$ holding the metadata. Here, $lastUsed$ describes when the knowledge item has been used last in inference, while $useCount$ tells how often it has been utilized. These two values determine the usefulness of statements. The pair (f, c) is truth value for uncertainty estimation (introduced later), and finally, $evidentialBases$ shows the input IDs (premises) upon which the knowledge is derived by NARS which is used within the reasoner to avoid double-counting evidence.

6.3.6 Answer Generation

Answering the questions employs the following prompt template specifying the desired output which should both be brief and mention the involved certainty values and memory references:

```
Mention concrete memory contents with certainty values.
Use the minimum involved certainty value.
Memory: *Prompt view of memory*
The question: *User question*
according to Memory and which memory item i?
Answer in a probabilistic sense and within
15 words based on memory content only.
```

whereby the missing parts are the actual question and the memory items which are relevant and recent to the particular question. Importantly, the embedding vectors within the vector database, which are necessary to compute the similarity to the user question, are calculated from the English form of each Narsese statement using the OpenAI's GPT-based embedding model, and the recent memory items are included based on their occurrence time. In the particular example, using the entire prompt view along with the user question and relevant memory items we get the following Q&A result:

Who is eating a pizza?

Memory item i=1 states with 72% certainty that Sam is eating pizza.

Please note that the system references the correct memory index and includes the linearly time-distance-discounted confidence value as shown in the prompt view of the memory. After a certain time distance to the current moment is reached, the 'eternalized' version of the belief (which drops the specific occurrence time value) is incorporated in the prompt with revised truth value reflecting the confidence-weighted average frequency value of all the specific events of the same term content. This way the system can induce and use common beliefs (e.g. chairs are typically found next to tables, and ravens are typically black, even though specific exceptions can be observed) when it doesn't have specific recent evidence about a current case of relevance available. This crucial temporal mechanism is responsible for a fluid transition from

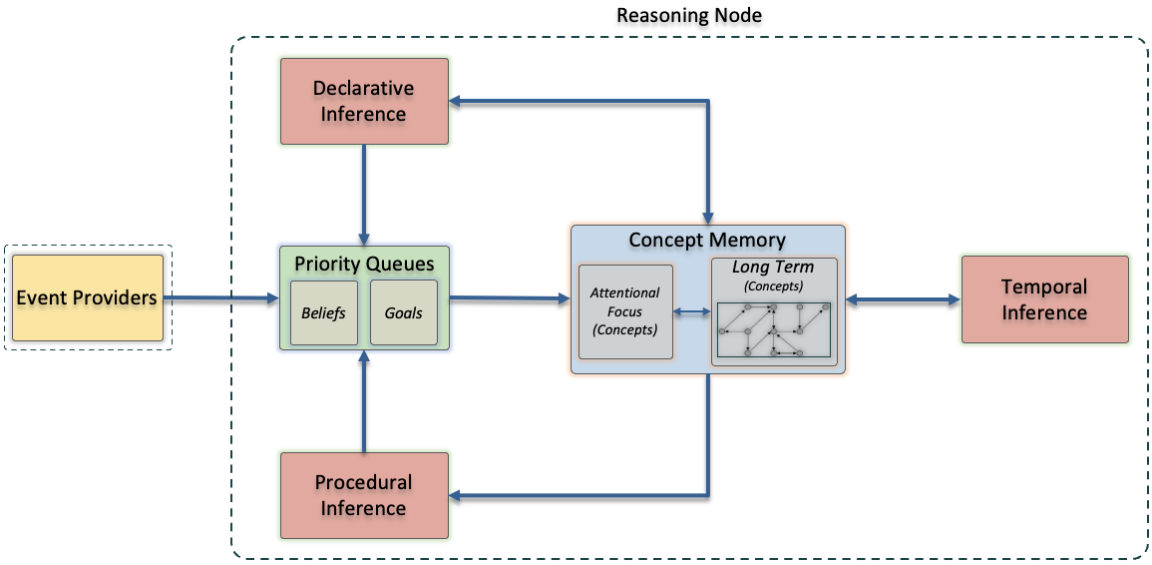


Figure 6.8: Overview of MeTTa-NARS, the NARS Implementation.

episodic events to long-term memory and the learning of time-independent declarative knowledge from specific observation while being able to handle exceptions which can temporally occur within a time window. Implementation-relevant details of this mechanism are described in related works regarding NARS, including *Wang* (2013a) and in *Hammer* (2021).

6.3.7 Real-time Reasoning Capabilities

The reasoning capabilities of NARS-GPT are realized through the use of MeTTa-NARS as its symbolic reasoning component, which integrates Inference Control, Memory, and Logic. Since Chapter 2 provides a comprehensive overview of MeTTa-NARS, below we reiterate certain aspects that are particularly relevant to the NARS-GPT system.

Logic specifies which information can be derived from combinations of at most two premises. In case with NARS, it utilizes NAL, a term logic that supports uncertainty estimation. To establish the semantics of the logical statements and associated notion

of uncertainty, we first focus on

$$SUBJECT \longrightarrow PREDICATE\ (f, c)$$

relations, which encode an inheritance relationship between the terms $SUBJECT$ and $PREDICATE$ with truth value of (f, c). The extension and intension of a term is defined as:

$$T^E = \{x | x \longrightarrow T\},\ T^I = \{x | T \longrightarrow x\}$$

Based on these notions we introduce Positive Evidence and Negative Evidence as:

$$w_+ = |(S^E \cap P^E)| + |(P^I \cap S^I)|$$

$$w_- = |(S^E \setminus P^E)| + |(P^I \setminus S^I)|$$

Now, with Total Evidence $w = w_+ + w_-$, Truth Value (f, c) is being defined as:

$$f = \frac{w_+}{w}, c = \frac{w}{w+1}$$

With the definition of truth value, inference rules are established as in *Wang* (2013a):

Deduction: $\{A \longrightarrow B, B \longrightarrow C\} \vdash A \longrightarrow C, f_{ded}$

Abduction: $\{A \longrightarrow C, B \longrightarrow C\} \vdash A \longrightarrow B f_{abd}$

Induction: $\{C \longrightarrow A, C \longrightarrow B\} \vdash A \longrightarrow B f_{ind}$

with the corresponding truth value functions as:

$$f_{ded}((f_1, c_1), (f_2, c_2)) = (f_1 * f_2, f_1 * f_2 * c_1 * c_2)$$

$$f_{abd}((f_1, c_1), (f_2, c_2)) = (f_2, \frac{f_1 * c_1 * c_2}{f_1 * c_1 * c_2 + 1})$$

$$f_{ind}(T_1, T_2) = f_{abd}(T_2, T_1)$$

Additionally, for A having a property B can be expressed as $A \longrightarrow [B]$. Also, relational statements of the form $(A * B) \longrightarrow R$ can be transformed to logically equivalent statements where A and B become the subject of an inheritance statement $A \longrightarrow (R/_1 B)$ and $B \longrightarrow (R/_2 A)$, which allows all the previous inferences to apply on relational statements for them to be generalized or specialized. This selected collection of NAL inference rules captures the most relevant inferences required for NARS-GPT.

Memory in NARS stores the information for long-term use and contains usefulness evaluations (based on use count and last usage time) as well as a related usefulness-based eviction criteria to keep memory consumption within the specific bound. Implementation-wise, it is exploited that the aforementioned inference rules always share a common term between the premises. NARS's memory is a collection of concepts representing a conceptual network with prioritized nodes and links, whereby priority specifies the amount of system resources allocated to a specific statement.

Inference Control: Being a general-purpose reasoner, MeTTa-NARS provides rich Declarative, Temporal and Procedural Inferences in real-time with reasoning steps taking only up to a maximum sub-second constant amount of time (less than 10 ms in the MeTTa-NARS shown in Figure 6.8). However, for the purpose of NARS-GPT, we focus solely on the inference loop that utilizes Declarative Inference. The declarative capabilities and declarative cycle of MeTTa-NARS were covered extensively in the previous chapter (2.3.7). In this setup, the natural language input sentence is being translated into Narsese statement which enters the reasoner and is being added to the priority queue with a priority value of 1. Hence, GPT Relation Extractor of NARS-GPT acts as an Event Provider in Figure 6.8. Then the inference cycle is initiated and proceeds as described in section 2.3.7.

6.3.8 Belief Revision Mechanism

A key advantage of NARS-GPT is the ability to execute belief revision through the accumulation of evidence, a considerably more efficient method than the modification of Transformer weights through use of Backpropagation (as in *Gupta et al.* (2023)). ML models, such as the Transformer-based GPT, have fixed weights that are challenging to adjust once the model is trained, even when weights are located *Meng et al.* (2022a). A common approach, when the model generates incorrect predictions, is to adjust the model with new data using Backpropagation in a computationally expensive fine-tuning process *Radford et al.* (2018).

Contrastingly, the belief revision mechanism of NARS-GPT enables the system to update its beliefs in a computationally efficient way (constant if the input mentions the same relation) based on new evidence presented in natural language and eliminates the need for fine-tuning the system. This is facilitated by the NARS reasoner, which gathers evidence and updates the system's beliefs in real-time. Therefore, if the system generates an incorrect prediction, it can revise its beliefs based on new evidence, resulting in a corrected answer output when the question is posed again. This is especially useful when the language is grounded (related) to perceptual terms, as we will see, the relations which are claimed to hold in the language can be checked directly by the system without any additional human input or reward 'wire' from human feedback (as in *Ouyang et al.* (2022)).

Moreover, the belief revision mechanism of NARS-GPT equips it to manage uncertainty and ambiguity more effectively. The system can assign degrees of certainty to its beliefs, based on the available evidence which can accumulate over a very long time-period, far beyond the contextual window size limitation of GPT-4. This also implies that if the system is uncertain about a specific prediction, it can communicate this uncertainty to the user (as we have seen in Section 6.3.6), instead of simply stating a potentially incorrect answer as true.

In summary, the belief revision mechanism of NARS-GPT through evidence accumulation is a computationally less expensive and hence sometimes more practical approach than the modification of Transformer weights. It enables the system to update its beliefs in real-time and to manage uncertainty appropriately, as illustrated in the following example:

What is kaeferbohnensalat?

No memory item. I believe kaeferbohnensalat is a German salad made with green beans and potatoes. Certainty: 80%.

Input: Kaeferbohnensalat is an Austrian salad made with runner beans, onion, and pumpkin seed oil.

What is kaeferbohnensalat?

Kaeferbohnensalat is a salad made with runner beans, according to memory item i=1 with a certainty of 0.69

In which country is kaeferbohnensalad made?

Memory item i=1 suggests that kaeferbohnensalad is made in Austria with a certainty of 0.69

What are all the ingredients of kaeferbohnensalat, list them all?

According to memory, kaeferbohnensalat contains runner bean and onion (i=1,3, 4,5,6). Pumpkin seed oil may also be an ingredient (i=11) with a certainty of 0.788.

As one can observe, the outputs are being modified based on the natural language input from the user. Below are corresponding constructed belief statement representations in NARS:

```
<kaeferbohnensalat --> austrian_salad>

<(kaeferbohnensalat * runner_bean) --> made_with>

<(austrian_salad * onion) --> made_with>

<(austrian_salad * pumpkin_seed_oil) --> made_with>
```

6.3.9 Experiments and Evaluation

The efficacy of NARS-GPT was assessed through a series of experiments conducted on two benchmark datasets: the Facebook AI research bAbI dataset *Dodge et al.* (2016) and the Aigo.ai INT_inf benchmark. The bAbI dataset comprises 20 reasoning tasks, each containing 2000 questions that span across deductive and non-deductive reasoning, spatial, temporal, and declarative inference, as well as short-term Q&A. Conversely, the Aigo.ai INT_inf benchmark is composed of over 1000 lines of personal assistant input, encompassing spatial, temporal, declarative reasoning, and long-term Q&A.

NARS-GPT demonstrated remarkable inductive reasoning performance on the bAbI dataset, achieving a 73% accuracy rate on the QA16 task (Table 6.5). This particular task measures inductive reasoning ability such as when knowing an instance of a class having a certain property, concluding that another instance of the same class (in the absence of additional information) might also have the same property. This is in stark contrast to GPT-4's 46% accuracy. These results underscore the superior performance of NARS-GPT in inductive reasoning compared to using GPT-4 without a reasoning system.

However, in the other BabI subtasks which only require deductive reasoning we found the performance numbers are comparable as illustrated for task 15 in Table[5] 6.5. The reason we identified is that babI tasks only contain a small number of input sentences after which questions are raised. For a pair of sentence such as "X is a C_1" and "C_1, C_2 are in relation R" the deductive reasoning of NARS essentially leads to a prompt window entry "X, C_2 are in relation R". While the latter provides a direct answering to the question what category X is in relation R with, GPT-4 can answer it from the few original sentences reliably, making the information essentially redundant,

[5]Performance comparison of NARS-GPT and GPT-4 on benchmark datasets with reported standard deviations across 10 runs each

similarly as with other patterns of deductive reasoning. This is in contrast to inductive reasoning which is inherently speculative and hence not the desired case LLM's are typically trained for. There the desired output usually consists of conclusions of high reliability, whereby questions to the user are generated in cases where clarity is lacking in the input.

In prolonged Q&A sessions NARS-GPT nevertheless has a benefit for deduction, as intermediate inference results can be independently remembered and ranked for inclusion in the prompt window, whereby the multiple statements these results are derived from might not be included. This allows NARS-GPT to be applicable in evolving circumstances, such as handling interactive long-term Q&A sessions with users, where information given by the users is long-term remembered and reasoned upon. The extendable INT_inf benchmark was created by Aigo.ai to measure question-answering performance of AI systems with Long-Term Memory in such situations, demanding the ability to operate beyond the contextual window size of GPT models. In INT_inf, we report a 52% accuracy rate of NARS-GPT, compared to the 38% of GPT-4.

Table 6.5: Performance comparison of NARS-GPT and GPT-4

Task	**Type**	**NARS-GPT**	**GPT-4**
babI-15	Deduction	$98 \pm 2\%$	$98 \pm 2\%$
babI-16	**Induction**	**73±3%**	**46±4%**
INT_inf	Long-term Q&A	$52 \pm 3\%$	$38 \pm 3\%$

6.3.10 Grounding Language in Event Streams

Prior to this work, the symbol grounding problem *Harnad* (1990) has been addressed in a relatively limiting way within reasoning systems. Usually a designer needed to take care of symbols' vocabulary which are perceptually grounded as in *Coradeschi et al.* (2013) and *Taddeo and Floridi* (2005), thus restricting the system to the pre-defined groundings. This is however incompatible with the idea of learning new notions from language interaction. The involved Model-Theoretic Semantics,

which state that symbols' meaning is either defined by a pre-existing reference to objects or pre-defined by static domain knowledge, are also incompatible with learning new notions. This is in contrast with the Experience-Grounded Semantics of NARS, where meaning is subjective to the system and dynamically altered with additional experience of relations a term participates in.

The utilization of NARS along with computed sentence embeddings enables to reason on input information and to establish logical relations. As a result, it allows the system to respond to queries about its perceptual events and to inductively relate representations from verbal input to sensorimotor events. Along with the formation of hypotheses through inductive inference and revision which NARS features *Hammer* (2022), it assists with contextualizing the information, recalling the context in which it was mentioned, and identifying what they refer to, thus, allowing for real-time question-answering from event streams. Below is an example of such capability (":|:" marks events):

Input: <chair ⟶ [left]>. :|:
Input: <table ⟶ [right]>. :|:

Is there a chair left to the table?

Memory item i=0 states that there is a chair left to the table with a 57.6% certainty.

Here it is also noteworthy that events in the Narsese formal language are efficient to process as they do not depend on a Large Language Model to be recognized and processed by the reasoning system. This allows a high information throughput from sensory encodings, while the information can be introspected with queries in natural language whenever desired. However, the terms in the sensory encodings must utilize valid English words, as the embeddings for the terms will else not suffice to retrieve related events for inclusion into the prompt.

6.3.11 Future Work

Future work will include comparisons on further reasoning datasets as well as extending perceptual grounding with multi-modal embeddings as in the Ethosight project *Latapie et al.* (2023) of Cisco DeepVision. Consistent with the presented results, we especially expect to demonstrate further benefits in inductive, abductive and temporal inference, as well as benefits (also for deductive inference) in domains where faulty or even non-transparent reasoning cannot be tolerated. Additionally, dynamic belief updating from new evidence will require new datasets and benchmarks to be evaluated on, hereby NARS-GPT is expected to excel particularly at data-efficiency. In this context both comparisons with finetuning approaches and other form of weight isolation and modification (as in *Meng et al.* (2022b)) will be carried out, where we will stress on the benefits for this process to happen autonomously and without supervision. And last, we will investigate further how to overcome current limitations. This includes addressing information loss when complex sentences are translated into symbolic statements, and the use of more advanced embedding models to reduces errors when mapping to previously utilized terms.

6.4 Conclusion

In this chapter we showed a notable progress in integrating autonomous mobile robots with reasoning systems by incorporating the principles of NARS into ROS. This integration demonstrates its effectiveness through a garbage collection task, where our approach surpassed traditional BDI models. This highlights NARS's ability to effectively utilize human-interpretable knowledge and adapt through inductive reasoning and revision mechanism, driven by new evidence gained from input events. Our open-source software is positioned to benefit the robotics community with its compatibility with standard ROS components and sophisticated object detection capabilities.

We also presented, the NARS-GPT system, which signifies a noteworthy progression in the integration of large language models with reasoning systems such as NARS. It provides a seamless integration allowing to benefit from the strengths of both components as well as enabling natural language representations to get automatically related to sensorimotor events and results of reasoning. This is achieved via proper use of hybrid encodings associated to the beliefs of the reasoning system. These contain both the structural knowledge and a corresponding vector embedding, ensuring compatibility with the reasoning mechanisms and the querying from GPT-4.

The combination of GPT and NARS represents a novel form of neural-symbolic integration and empowers the system to handle tasks which require reliable forms of reasoning (including inductive and abductive), and to acquire and ground (for long-term use) new concepts from natural user interactions, while also providing transparent and explainable outcomes. Furthermore, the system's ability to learn over prolonged periods from interactive Q&A sessions with users augments its knowledge base and performance, ensuring its adaptability to users and evolving situations.

In essence, NARS-GPT can be a valuable asset in various applications where these capabilities are crucial, since it is widely applicable and adoptable to a wide range of domains, while supporting the reliable reasoning and natural user interaction with the system to extend its knowledge at runtime as needed to create personal assistants which are better adapted to their users.

CHAPTER 7

CONCLUSION AND SUMMARY

7.1 Revisiting the Research Problem

This thesis addresses a central challenge in artificial intelligence and autonomous robotics: endowing an intelligent agent with the capacity for abstract reasoning under uncertainty while simultaneously enabling it to learn from raw experiential data at runtime. As articulated in Chapter 1, the core research problem lies in bridging the gap between symbolic reasoning systems, which can perform logical inference under uncertainty and having incomplete knowledge, and sub-symbolic learning systems, which excel at trial-and-error learning from the environment as well as dealing with high-dimensional state spaces. Traditional AI approaches often focus on one paradigm in isolation, leading to agents that either cannot adapt to new knowledge or cannot readily perform higher-level reasoning or knowledge updating at runtime. The problem motivating this work was how to integrate these two paradigms into a single architecture for an autonomous agent, thereby combining the strengths of both. In particular, Non-Axiomatic Reasoning System (NARS) was identified as a powerful framework for reasoning under the Assumption of Insufficient Knowledge and Resources, and a complementary exploratory learning mechanism was needed to provide the agent with efficient exploration of the environment supported by causal dependency extraction from sub-symbolic observations. The overarching question was: Can a hybrid agent be designed that leverages non-axiomatic reasoning alongside sub-symbolic experiential learning to achieve more robust autonomy in complex, uncertain environments? By clearly formulating this problem, the thesis lays the

groundwork for developing a novel system that bridges the gap between high-level reasoning and data-driven learning in autonomous intelligent agents.

7.2 A Hybrid Solution: The NARCES Framework

In response to the above challenge, this research introduces Non-Axiomatic Reasoning and Causal Explorative System (NARCES), a hybrid architecture that uniquely integrates MeTTa-NARS' high-level reasoning with the low-level experiential learning. NARCES combines two previously disparate components into a unified framework: a MeTTa-NARS high-level reasoner and the Non-Axiomatic Causal Explorer (NACE) experiential learning module. The MeTTa-NARS component is an implementation of the NARS cognitive reasoning system within the MeTTa functional logic programming language (part of the OpenCog Hyperon project), enabling the agent to perform non-axiomatic logical inference, manage uncertain beliefs, and make decisions through reasoning. The NACE component is a complementary learning system that allows the agent to explore its environment, discover causal relationships, and adapt through reinforcement signals. By integrating these two, NARCES allows an agent to think and learn concurrently: the high-level reasoning side provides understanding, generalization, and goal-driven guidance, while the experiential learning aspect provides trial-and-error experience, pattern discovery, and adaptability. This hybrid integration is the unique contribution of NARCES. Unlike purely symbolic systems, NARCES can improve its knowledge through self-guided exploration; unlike purely sub-symbolic (e.g. deep learning or conventional reinforcement learning) agents, NARCES can leverage an existing knowledge base and logical inference to direct its learning more efficiently at runtime.

The MeTTa-NARS reasoner continuously derives new conclusions and forms plans using Non-Axiomatic Logic (NAL) rules applied to the agent's experiences. Simultaneously, NACE generates intrinsic motivations for the agent to explore aspects of the

environment that are most uncertain or informative. Crucially, the two components share information: the hypotheses and causal rules induced by NACE's experiential learning are fed into the symbolic reasoner as new beliefs, and conversely, the reasoner's inferences can identify goals or focus areas for NACE to explore. This tight coupling means that the agent can use reasoning to guide exploration, leading to more directed and data-efficient learning, and use exploration to enrich reasoning, ensuring the knowledge base is continually updated with relevant experiences. The resulting NARCES architecture is therefore an instance of a sub-symbolic and symbolic hybrid system that operates in an online, anytime fashion. It leverages NARS's ability to handle uncertainty and contradiction in a principled way, alongside NACE's ability to improve through interaction with the environment. For example, in the context of a grid-world environment (as discussed in Chapter 5), NARCES was able to induce causal rules from observed state transitions (e.g. discovering which actions lead to which outcomes) and use those rules in the reasoning process to plan exploratory actions that maximize information gain. This led to significantly improved data efficiency in learning: the agent could achieve competent performance with orders of magnitude fewer training samples than a baseline deep reinforcement learning agent, thanks to its knowledge-guided exploration. In summary, the NARCES framework demonstrates that combining symbolic and sub-symbolic AI principles is not only feasible but highly effective. It provides a proof-of-concept that an autonomous agent can be designed to learn new knowledge while reasoning with it in real time, addressing the research problem posed at the outset.

7.3 Contributions of this Research

This dissertation yields several important contributions in the realms of cognitive architectures, reasoning systems, and robotics. The main contributions of the work are as follows:

- **Implementation of NARS in the MeTTa Language:** The research successfully implemented the full Non-Axiomatic Reasoning System within the MeTTa functional logic programming language. This involved translating the theoretical inference rules and memory structures of NARS into MeTTa, resulting in the MeTTa-NARS engine. This contribution is significant, as it provides a modular and portable implementation of NARS (following the NARS model of intelligence) that can be easily integrated with other components. The MeTTa-NARS system replicates the key capacities of NARS: including its declarative reasoning (deriving beliefs from events), temporal inference (handling sequences and causal implications), and procedural control (goal-driven decision making through sub-goaling), all within a modern runtime. By realizing NARS in MeTTa, the work lays a foundation for embedding advanced reasoning in a variety of platforms and for extending NARS with new features. (The source code for this contribution is provided in the following repository `https://github.com/pisaev1/metta-nars`)

- **NACE formalization and development:** Key research and development contributions regarding formalization and implementation of Non-Axiomatic Causal Explorer, to work on the foundations for sample-efficient curiosity-driven exploration in grid worlds and the formal analysis which is part of this dissertation. The NACE component is a key part of NARCES, covering necessary aspects of lower-level sub-symbolic learning. The source code for this component is available here: `https://github.com/patham9/NACE`

- **Integration of Symbolic and Sub-symbolic AI aspects into a Hybrid Framework:** Building on the above, the dissertation designed and developed the NARCES hybrid architecture that tightly couples the MeTTa-NARS reasoner with the NACE learning module. This integration required novel mech-

anisms for communication between the logical inference system and the exploratory learning system. A unified knowledge representation was defined so that NACE's learned causal relations could be translated into NARS-compatible statements, and conversely so that NARS's chosen goals could trigger NACE's exploration routines. The resulting NARCES framework is capable of dynamic knowledge exchange: the causal learning module (NACE) supplies the agent with causal hypotheses learned from interactions, and intrinsic reward signals for efficient exploration, while the higher-level reasoning module (NARS) supplies inferences, predictions, and decision-making logic. The thesis demonstrates how this two-way integration leads to emergent capabilities that neither component could achieve alone: notably, intrinsically motivated reasoning (where the logical system allows the agent to pursue high-level goals) and explainable exploration (where the agent can learn explicit causal rules). This contribution is providing a concrete example of merging a reasoning engine with a learning algorithm within a single agent. (Bi-directional communication bridge, which is the key integration component, is available in the following repository: `https://github.com/pisaev1/NARCES`)

Additionally, **NARS-GPT** was described, which provides the ability to communicate with MeTTa-NARS using natural language. This further exemplifies the merging of a sub-symbolic technique (Large Language Model) with high-level logical reasoning, with the ability to perform reliable and transparent reasoning on the information provided by the user. Hereby, key contributions were focused on the interaction interfaces between both systems as well as extending the memory with embedding vectors to effectively use term encodings, as well as extending the evaluation on the Facebook AI Research BaBi datasets.

- **ROS Implementation and Robotic Applications:** A crucial part of this research was the implementation of the Explorer agent, featuring NARCES

system within the Robot Operating System 2 (ROS2) framework, enabling the application of the hybrid architecture to real robotic scenarios. The thesis describes how the NARCES agent was embodied in a ROS2-based robot: the MeTTa-NARS reasoning core and the NACE learning module were encapsulated in ROS2 nodes that could process sensor input and command actuators in real time. The author integrated the AI architecture with standard robotic components (e.g. perception, navigation, and motor control nodes) to create an autonomous robot capable of reasoning about its environment and learning from it on the fly. This involved developing interfaces for translating raw sensor data into the symbolic representations that NARS can reason about, as well as converting the agent's chosen high-level goals into low-level grounded objectives. Through this implementation, NARCES was tested in robotic exploration tasks: for instance, having a mobile robot explore an unknown environment while avoiding obstacles and gathering knowledge about object locations. The ROS2 integration validates that the hybrid system can operate under real-world conditions (with asynchronous sensor data, real-time constraints, and physical dynamics), and not just in abstract simulations. It also provides a practical framework for other researchers to deploy NARCES on different robotic platforms. (The integration code and ROS2 launch configurations have been released in the explorer repository: `https://github.com/pisaev1/explorer`)

This was a natural extension of author's prior work of using NARS on a Autonomous Mobile robot (Transbot) which was presented at ICRA 2024 *Hammer et al.* (2024), where key contributions involved the integration of NARS with ROS1 and sensory data (object detection and distance estimation) and fluid actuator control involving semantic information (target object to pick) beyond the simplistic prior attempts where depth information was ignored and actuator control still unreliable.

In addition to the above contributions, all developed software components have been made available as open-source repositories to encourage reuse and extension. In summary, the author's work provided key contributions (both conceptually and in implementation) to the software artifacts: MeTTa-NARS, the implementation of the non-axiomatic reasoning engine in MeTTa; NACE, the non-axiomatic causal exploration module; NARCES, a hybrid system which combines high-level reasoning and experiential learning; and Explorer agent, the integration of these components into a robotic middleware. These resources not only demonstrate the contributions but also serve as a foundation for future research in the community.

7.4 Research Outcomes and Future Prospects

By combining the contributions above, this thesis accomplishes its goal of a functioning hybrid intelligent agent and provides empirical evidence of the approach's benefits. The research outcomes indicate that NARCES can significantly outperform comparable single-paradigm systems in learning efficiency and adaptability. The agent was shown to quickly learn accurate causal models of its environment and utilize them for decision-making. For example, in grid-world experiments, NARCES achieved similar or better task performance with dramatically fewer interactions compared to deep reinforcement learning baselines, thanks to its ability to focus exploration on informative states. In robotic trials, the NARCES-driven Explorer agent demonstrated robust behavior in the face of uncertainty: it could reason about unforeseen changes (using NARS) and adapt its strategy by learning (using NACE) without needing exhaustive retraining. These results validate the central hypothesis that integrating reasoning with learning yields a more powerful and data-efficient form of autonomy. Furthermore, the agent's decisions remained interpretable at a high level (through the logical propositions in NARS), addressing the explainability issues often found in purely sub-symbolic approaches. While the current work has

answered the primary research questions, it also opens up several exciting avenues for future research. One direction is to expand the scope of NARCES to more complex domains and perceptual inputs. In this thesis, the focus was on relatively structured environments (grid worlds, basic mobile robot sensors); an important next step is to integrate richer sensory data (such as vision or auditory inputs beyond the SLAM and object detection provided) by coupling the NARCES framework with further deep neural networks for perception. This would further test the hybrid architecture's scalability and its ability to handle high-dimensional, continuous observations. Another avenue is to further refine the interaction between the high-level reasoning and low-level learning components. Currently, NARCES uses a predefined interface for exchanging knowledge and goals between NARS and NACE, in future works the interface could also involve learning methods. Additionally, the theoretical underpinnings of NARCES warrant further investigation. Formal analysis of how and when the hybrid system yields efficiency gains (e.g. in terms of convergence properties or bounds on sample complexity improvement) would deepen our understanding and could guide the design of next-generation neuro-symbolic algorithms.

From an engineering perspective, future developments might involve using NARCES for real-world problems requiring long-term autonomy and open-ended learning, such as lifelong learning robots in homes or factories, to observe how the system manages continual learning and reasoning over extended periods. In this context, since MeTTa-NARS is implemented in a modular way, it could be connected with larger frameworks (such as the OpenCog Hyperon platform or other AGI architectures) to provide a reasoning module within a broader cognitive system. Exploring these integrations could lead to an even more general autonomous cognitive agent, inching closer to the vision of AGI that can learn and reason in the wild.

In conclusion, the work presented in this thesis demonstrates a successful synthesis of symbolic and sub-symbolic AI in the form of the NARCES hybrid architecture.

The research problem posed at the beginning as a unifying logical reasoning with experiential learning for an autonomous agent, has been addressed by a novel solution that was implemented, tested, and shown to be effective. The author's personal contributions in building this system have not only advanced the state of the art in reasoning systems and robot intelligence, but also provided practical tools (code and frameworks) for the AI community. NARCES stands as a proof-of-concept that an agent can concurrently learn new knowledge and reason with it, offering a pathway to more intelligent and adaptive machines. Looking forward, this work lays a rich foundation for future explorations in neuro-symbolic AI. By building on the ideas and systems developed here, researchers can continue to bridge the gap between high-level cognition and low-level learning, ultimately bringing us closer to autonomous agents with the versatility and understanding necessary to operate in the complexity of the real world.

BIBLIOGRAPHY

Ali, S., et al. (2023), Explainable artificial intelligence (xai): What we know and what is left to attain trustworthy artificial intelligence, *Information Fusion*, *99*, 101,805, doi:https://doi.org/10.1016/j.inffus.2023.101805.

Anand, N., and D. Precup (2024), Prediction and control in continual reinforcement learning, *Advances in Neural Information Processing Systems*, *36*.

Anderson, J., D. Bothell, M. Byrne, S. Douglass, C. Lebiere, and Y. Qin (2004), An integrated theory of the mind., *Psychological review*, *111*, 1036–60, doi:10.1037/0033-295X.111.4.1036.

Andres, B., D. Rajaratnam, O. Sabuncu, and T. Schaub (2015), Integrating asp into ros for reasoning in robots, in *Logic Programming and Nonmonotonic Reasoning: 13th International Conference, LPNMR 2015, Lexington, KY, USA, September 27-30, 2015. Proceedings 13*, pp. 69–82, Springer.

Baars, B., S. Franklin, and U. Ramamurthy (2007), How deliberate, spontaneous and unwanted memories emerge in a computational model of consciousness.

Bellemare, M., S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos (2016), Unifying count-based exploration and intrinsic motivation, *Advances in neural information processing systems*, *29*.

Besold, T. R., et al. (2017), Neural-symbolic learning and reasoning: A survey and interpretation, *ArXiv*, *abs/1711.03902*.

Bordini, R. H., and J. F. Hübner (2005), BDI agent programming in AgentSpeak using Jason, in *International workshop on computational logic in multi-agent systems*, pp. 143–164, Springer.

Browne, C. B., et al. (2012), A survey of Monte Carlo Tree Search methods, *IEEE Transactions on Computational Intelligence and AI in games*, *4*(1), 1–43.

Bryce, D. (2011), Wumpus world in introductory artificial intelligence, *Journal of Computing Sciences in Colleges*, *27*(2), 58–65.

Bubeck, S., et al. (2023), Sparks of artificial general intelligence: Early experiments with gpt-4, *arXiv preprint arXiv:2303.12712*.

Burda, Y., H. Edwards, A. Storkey, and O. Klimov (2018), Exploration by random network distillation, *arXiv preprint arXiv:1810.12894*.

Campero, A., R. Raileanu, H. Küttler, J. Tenenbaum, T. Rocktäschel, and E. Grefenstette (2021), Learning with amigo: Adversarially motivated intrinsic goals, in *ICLR 2021-9th International Conference on Learning Representations*, International Conference on Learning Representations.

Cao, Y., Z. Li, T. Yang, H. Zhang, Y. ZHENG, Y. Li, J. Hao, and Y. Liu (2022), Galois: Boosting deep reinforcement learning via generalizable logic synthesis, in *Advances in Neural Information Processing Systems*, vol. 35, edited by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, pp. 19,930–19,943, Curran Associates, Inc.

Chaslot, G., S. Bakkes, I. Szita, and P. Spronck (2008), Monte-Carlo Tree Search: A new framework for game AI., in *AIIDE*.

Chen, Z., and Q. Guan (2023), Continuous exploration via multiple perspectives in sparse reward environment, in *Chinese Conference on Pattern Recognition and Computer Vision (PRCV)*, pp. 57–68, Springer.

Chevalier-Boisvert, M., B. Dai, M. Towers, R. Perez-Vicente, L. Willems, S. Lahlou, S. Pal, P. S. Castro, and J. Terry (2024), Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks, *Advances in Neural Information Processing Systems*, *36*.

Chong, H.-Q., A.-H. Tan, and G.-W. Ng (2007), Integrated cognitive architectures: A survey, *Artificial Intelligence Review*, *28*, 103–130, doi:10.1007/s10462-009-9094-9.

Choudhary, N., and C. K. Reddy (2023), Complex logical reasoning over knowledge graphs using large language models, *arXiv preprint arXiv:2305.01157*.

Colledanchise, M., and P. Ögren (2018), *Behavior trees in robotics and AI: An introduction*, CRC Press.

Coradeschi, S., A. Loutfi, and B. Wrede (2013), A short review of symbol grounding in robotic and intelligent systems, *KI-Künstliche Intelligenz*, *27*, 129–136.

De Raedt, L., A. Kimmig, and H. Toivonen (2007), ProbLog: A probabilistic Prolog and its application in link discovery., in *IJCAI*, vol. 7, pp. 2462–2467, Hyderabad.

Devidze, R., P. Kamalaruban, and A. Singla (2022), Exploration-guided reward shaping for reinforcement learning under sparse rewards, in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Curran Associates Inc., Red Hook, NY, USA.

Diehl, M., C. Paxton, and K. Ramirez-Amaro (2021), Automated generation of robotic planning domains from observations, in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6732–6738, IEEE.

Dodge, J., A. Gane, X. Zhang, A. Bordes, S. Chopra, A. Miller, A. Szlam, and J. Weston (2016), Evaluating prerequisite qualities for learning end-to-end dialog systems.

Espeholt, L., et al. (2018), Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures, in *International conference on machine learning*, pp. 1407–1416, PMLR.

Ferrein, A., G. Steinbauer, and S. Vassos (2012), Action-based imperative programming with YAGI., in *CogRob@ AAAI*.

Franklin, S., T. Madl, S. D'Mello, and J. Snaider (2013), Lida: A systems-level architecture for cognition, emotion, and learning, *IEEE Transactions on Autonomous Mental Development*, *6*, doi:10.1109/TAMD.2013.2277589.

Franklin, S., T. Madl, S. Strain, U. Faghihi, D. Dong, S. Kugele, J. Snaider, P. Agrawal, and S. Chen (2016), A lida cognitive model tutorial, *Biologically Inspired Cognitive Architectures*, *16*, doi:10.1016/j.bica.2016.04.003.

Goertzel, B., M. Iklé, I. F. Goertzel, and A. Heljakka (2008), *Probabilistic Logic Networks: A comprehensive framework for uncertain inference*, Springer Science & Business Media.

Gronchi, G., and A. Perini (2024), Dual-process theories of thought as potential architectures for developing neuro-symbolic ai models, *Frontiers in Cognition*, doi: 10.3389/fcogn.2024.1356941.

Guo, H., F. Wu, Y. Qin, R. Li, K. Li, and K. Li (2023), Recent trends in task and motion planning for robotics: A survey, *ACM Computing Surveys*, *55*(13s), 1–36.

Guo, W., J. Wang, and S. Wang (2019), Deep multimodal representation learning: A survey, *Ieee Access*, *7*, 63,373–63,394.

Guo, Y., Y. Fu, R. Peng, and H. Lee (2022), Learning exploration policies with view-based intrinsic rewards, in *Deep Reinforcement Learning Workshop NeurIPS 2022*.

Gupta, A., D. Mondal, A. K. Sheshadri, W. Zhao, X. L. Li, S. Wiegreffe, and N. Tandon (2023), Editing commonsense knowledge in gpt, *arXiv preprint arXiv:2305.14956*.

Hafner, D., J. Pasukonis, J. Ba, and T. Lillicrap (2023), Mastering diverse domains through world models, *arXiv preprint arXiv:2301.04104*.

Hammer, P. (2021), *Autonomy through real-time learning and OpenNARS for Applications*, Temple University.

Hammer, P. (2022), Reasoning-learning systems based on non-axiomatic reasoning system theory, in *International Workshop on Self-Supervised Learning*, pp. 89–107, PMLR.

Hammer, P., and T. Lofthouse (2018), Goal-directed procedure learning, in *International Conference on Artificial General Intelligence*, pp. 77–86, Springer.

Hammer, P., and T. Lofthouse (2020), 'OpenNARS for Applications': Architecture and control, in *International Conference on Artificial General Intelligence*, pp. 193–204, Springer.

Hammer, P., T. Lofthouse, and P. Wang (2016), The OpenNARS implementation of the Non-Axiomatic Reasoning System, in *International conference on artificial general intelligence*, pp. 160–170, Springer.

Hammer, P., T. Lofthouse, E. Fenoglio, H. Latapie, and P. Wang (2021), A reasoning based model for anomaly detection in the smart city domain, in *Intelligent Systems and Applications: Proceedings of the 2020 Intelligent Systems Conference (IntelliSys) Volume 2*, pp. 144–159, Springer.

Hammer, P., P. Isaev, T. Lofthouse, and R. Johansson (2022), Ona for autonomous ros-based robots, in *International Conference on Artificial General Intelligence*, pp. 231–242, Springer.

Hammer, P., P. Isaev, L. Feng, R. Johansson, and J. Tumova (2024), Non-axiomatic reasoning for an autonomous mobile robot, in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 17,079–17,085, IEEE.

Harnad, S. (1990), The symbol grounding problem, *Physica D: Nonlinear Phenomena*, *42*(1-3), 335–346.

Helgason, H. (2013), General attention mechanism for artificial intelligence systems, Ph.D. thesis.

Hellström, T. (2021), The relevance of causation in robotics: A review, categorization, and analysis, *Paladyn, Journal of Behavioral Robotics*, *12*(1), 238–255.

Hofstadter, D., and M. Mitchell (1995), The copycat project: A model of mental fluidity and analogy-making.

Ikle, M., J. Pitt, B. Goertzel, and G. Sellman (2009), Economic attention networks: Associative memory and resource allocation for general intelligence, doi:10.2991/agi.2009.19.

Isaev, P., and P. Hammer (2020), An attentional control mechanism for reasoning and learning, in *Proceedings of the Thirteenth Conference on Artificial General Intelligence*, to appear.

Isaev, P., and P. Hammer (2023), Memory system and memory types for real-time reasoning systems", in *Artificial General Intelligence*, edited by P. Hammer, M. Alirezaie, and C. Strannegård, pp. 147–157, Springer Nature Switzerland, Cham.

Ivanović, M., J. Ivković, and C. Bădică (2017), Role of Non-Axiomatic Logic in a distributed reasoning environment, in *International Conference on Computational Collective Intelligence*, pp. 381–388, Springer.

Kahneman, D. (2011), *Thinking, Fast and Slow*, Farrar, Straus and Giroux.

Kansky, K., et al. (2017), Schema networks: Zero-shot transfer with a generative causal model of intuitive physics, in *International Conference on Machine Learning*, pp. 1809–1818, PMLR.

Karapinar, S., and S. Sariel (2015), Cognitive robots learning failure contexts through real-world experimentation, *Autonomous Robots*, *39*, 469–485.

Kim, W., Y. Shin, J. Park, and Y. Sung (2024), Sample-efficient and safe deep reinforcement learning via reset deep ensemble agents, *Advances in Neural Information Processing Systems*, *36*.

Kotseruba, I., and J. K. Tsotsos (2020), 40 years of cognitive architectures: core cognitive abilities and practical applications, *Artificial Intelligence Review*, *53*, 17–94, doi:10.1007/s10462-018-9646-y.

Kynoch, B., and H. Latapie (2023), Recallm: An architecture for temporal context understanding and question answering, *arXiv preprint arXiv:2307.02738*.

Laird, J. (2012), *The Soar Cognitive Architecture*, doi:10.7551/mitpress/7688.001.0001.

Latapie, H., K. R. Thorisson, S. Yu, V. Petrosyan, P. Hammer, P. Wang, B. Kynoch, H. Chen, and T. Li (2023), Ethosight: A joint-embedding based system for nuanced perception using contextual label affinity metric and reasoning based iterative learning, *arXiv preprint arXiv:2307.10577*.

Lauri, M., and R. Ritala (2016), Planning for robotic exploration based on forward simulation, *Robotics and Autonomous Systems*, *83*, 15–31.

Li, H., and X. Ding (2023), Adaptive and intelligent robot task planning for home service: A review, *Engineering Applications of Artificial Intelligence*, *117*, 105,618.

Lofthouse, T. (2019), ALANN: An event driven control mechanism for a Non-Axiomatic Reasoning System (NARS), NARS Tutorial and Workshop at AGI 2019.

Lucas, P., and L. Van Der Gaag (1991), Principles of Expert Systems, Addison-Wesley Wokingham.

Manhaeve, R., S. Dumancic, A. Kimmig, T. Demeester, and L. De Raedt (2018), Deepproblog: Neural probabilistic logic programming, in *NeurIPS*.

Meng, K., D. Bau, A. Andonian, and Y. Belinkov (2022a), Locating and editing factual associations in gpt, *Advances in Neural Information Processing Systems*, *35*, 17,359–17,372.

Meng, K., A. S. Sharma, A. Andonian, Y. Belinkov, and D. Bau (2022b), Mass-editing memory in a transformer, *arXiv preprint arXiv:2210.07229*.

Min, B., H. Ross, E. Sulem, A. P. B. Veyseh, T. H. Nguyen, O. Sainz, E. Agirre, I. Heintz, and D. Roth (2021), Recent advances in natural language processing via large pre-trained language models: A survey, *ACM Computing Surveys*.

Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller (2013), Playing atari with deep reinforcement learning, *arXiv preprint arXiv:1312.5602*.

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*, pp. 1928–1937, PMLR.

Mnih, V., et al. (2015), Human-level control through deep reinforcement learning, *nature*, *518*(7540), 529–533.

Modarressi, A., A. Imani, M. Fayyaz, and H. Schütze (2023), Ret-llm: Towards a general read-write memory for large language models, *arXiv preprint arXiv:2305.14322*.

Nivel, E., K. R. Thórisson, H. Dindo, G. Pezzulo, M. Rodriguez, et al. (2013), Autocatalytic Endogenous Reflective Architecture, Citeseer.

Ouyang, L., et al. (2022), Training language models to follow instructions with human feedback, *Advances in Neural Information Processing Systems*, *35*, 27,730–27,744.

Pan, L., A. Albalak, X. Wang, and W. Y. Wang (2023), Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning, *arXiv preprint arXiv:2305.12295*.

Pathak, D., P. Agrawal, A. A. Efros, and T. Darrell (2017), Curiosity-driven exploration by self-supervised prediction, in *International conference on machine learning*, pp. 2778–2787, PMLR.

Pearl, J. (1995), From bayesian networks to causal networks, in *Mathematical models for handling partial knowledge in artificial intelligence*, pp. 157–182, Springer.

Piaget, J. (1954), *The Construction of Reality in the Child*, Basic Books, New York.

Pollack, M. E., and M. Ringuette (1990), Introducing the tileworld: Experimentally evaluating agent architectures, in *AAAI*, vol. 90, pp. 183–189.

Pramod, S., and O. Vyas (2012), Data stream mining: A review on windowing approach, *Global Journal of Computer Science and Technology Software & Data Engineering*, *12*(11), 26–30.

Qiu, R., et al. (2012), Towards robust personal assistant robots: Experience gained in the srs project, in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1651–1657, IEEE.

Radford, A., K. Narasimhan, T. Salimans, I. Sutskever, et al. (2018), Improving language understanding by generative pre-training.

Raffin, A., A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann (2021), Stable-baselines3: Reliable reinforcement learning implementations, *Journal of Machine Learning Research*, *22*(268), 1–8.

Raileanu, R., and T. Rocktäschel (2020), Ride: Rewarding impact-driven exploration for procedurally-generated environments, in *International Conference on Learning Representations*.

Rehling, J., and D. Hofstadter (1997), The parallel terraced scan: an optimization for an agent-oriented architecture, in *1997 IEEE International Conference on Intelligent Processing Systems (Cat. No.97TH8335)*, vol. 1, pp. 900–904 vol.1, doi: 10.1109/ICIPS.1997.672938.

Richardson, M., and P. Domingos (2006), Markov Logic Networks, *Machine Learning*, *62*(1-2), 107–136.

Riguzzi, F., E. Bellodi, and R. Zese (2014), A history of probabilistic inductive logic programming, *Frontiers in Robotics and AI*, *1*, 6.

Rocktäschel, T., and S. Riedel (2017), End-to-end differentiable proving, in *NeurIPS*.

Rovida, F., M. Crosby, D. Holz, A. S. Polydoros, B. Großmann, R. P. Petrick, and V. Krüger (2017), Skiros—a skill-based robot control platform on top of ros, *Robot Operating System (ROS) The Complete Reference (Volume 2)*, pp. 121–160.

Russell, S., and P. Norvig (2002), Artificial intelligence: a modern approach.

Schulman, J., S. Levine, P. Abbeel, M. Jordan, and P. Moritz (2015), Trust region policy optimization, in *International conference on machine learning*, pp. 1889–1897, PMLR.

Schulman, J., F. Wolski, P. Dhariwal, A. Radford, and O. Klimov (2017), Proximal policy optimization algorithms, *arXiv preprint arXiv:1707.06347*.

Sekar, R., O. Rybkin, K. Daniilidis, P. Abbeel, D. Hafner, and D. Pathak (2020), Planning to explore via self-supervised world models, in *International conference on machine learning*, pp. 8583–8592, PMLR.

Silver, D., A. Huang, C. J. Maddison, et al. (2016), Mastering the game of go with deep neural networks and tree search, *Nature*, *529*(7587), 484–489.

Spaan, M. T. (2012), Partially Observable Markov Decision Processes, in *Reinforcement Learning*, pp. 387–414, Springer.

Sun, R. (2006), The clarion cognitive architecture: Extending cognitive modeling to social simulation, *Cognition and multi-agent interaction*, pp. 79–99.

Sutton, R. S. (1988), Learning to predict by the methods of Temporal Differences, *Machine learning*, *3*(1), 9–44.

Sutton, R. S., and A. G. Barto (2018), *Reinforcement Learning: An introduction*, MIT press.

Sutton, R. S., A. G. Barto, et al. (1999), Reinforcement learning, *Journal of Cognitive Neuroscience*, *11*(1), 126–134.

Taddeo, M., and L. Floridi (2005), Solving the symbol grounding problem: a critical review of fifteen years of research, *Journal of Experimental & Theoretical Artificial Intelligence*, *17*(4), 419–445.

Teichteil-Königsbuch, F., V. Vidal, and G. Infantes (2011), Extending classical planning heuristics to probabilistic planning with dead-ends, in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 25, pp. 1017–1022.

Tenorth, M., and M. Beetz (2009), Knowrob—knowledge processing for autonomous personal robots, in *2009 IEEE/RSJ international conference on intelligent robots and systems*, pp. 4261–4266, IEEE.

Tsividis, P. A., J. Loula, J. Burga, N. Foss, A. Campero, T. Pouncy, S. J. Gershman, and J. B. Tenenbaum (2021), Human-level reinforcement learning through theory-based modeling, exploration, and planning, *arXiv preprint arXiv:2107.12544*.

Vilchis-Medina, J.-L., K. Godary-Déjean, and C. Lesire (2021), Autonomous decision-making with incomplete information and safety rules based on non-monotonic reasoning, *IEEE Robotics and Automation Letters*, *6*(4), 8357–8362.

Wang, A., A. C. Li, T. Q. Klassen, R. T. Icarte, and S. A. McIlraith (2023a), Learning belief representations for partially observable deep rl, in *International Conference on Machine Learning*, pp. 35,970–35,988, PMLR.

Wang, C., S. X. Liu, and A. H. Awadallah (2023b), Cost-effective hyperparameter optimization for large language model generation inference.

Wang, K., K. Zhou, B. Kang, J. Feng, and Y. Shuicheng (2023c), Revisiting intrinsic reward for exploration in procedurally generated environments, in *The Eleventh International Conference on Learning Representations*.

Wang, P. (1995), Non-axiomatic reasoning system: Exploring the essence of intelligence, Ph.D. thesis, Indiana University.

Wang, P. (2004), Problem solving with insufficient resources, *International Journal of Uncertainty, Fuzziness and Knowledge-based Systems*, *12*(5), 673–700.

Wang, P. (2005), Experience-Grounded Semantics: a theory for intelligent systems, *Cognitive Systems Research*, *6*(4), 282–302.

Wang, P. (2006), *Rigid Flexibility*, Springer.

Wang, P. (2009a), Insufficient Knowledge and Resources - a biological constraint and its functional implications., in *AAAI Fall Symposium: Biologically Inspired Cognitive Architectures*, Citeseer.

Wang, P. (2009b), Case-by-case problem solving, in *Proceedings of the Second Conference on Artificial General Intelligence*, edited by B. Goertzel, P. Hitzler, and M. Hutter, pp. 180–185.

Wang, P. (2013a), *Non-Axiomatic Logic: A model of intelligent reasoning*, World Scientific.

Wang, P. (2013b), Solving a problem with or without a program, *Journal of Artificial General Intelligence*, *3*(3), 43–73.

Wang, P. (2019), On defining artificial intelligence, *Journal of Artificial General Intelligence*, *10*, 1 – 37.

Wang, P., and P. Hammer (2015), Issues in temporal and causal inference, in *International Conference on Artificial General Intelligence*, pp. 208–217, Springer.

Wang, P., P. Isaev, P. Hammer, and X. Li (2020), The conceptual design of opennars 3.1.0, *Tech. Rep. 11*, AGI Team, Temple University, Philadelphia, USA.

Wang, P., C. Hahm, and P. Hammer (2022), A model of unified perception and cognition, *Frontiers in Artificial Intelligence*, *Volume 5 - 2022*, doi:10.3389/frai.2022.806403.

Wang, W., and M. Sebag (2012), Multi-objective Monte-Carlo Tree Search, in *Asian conference on machine learning*, pp. 507–522, PMLR.

Wang, W., L. Dong, H. Cheng, X. Liu, X. Yan, J. Gao, and F. Wei (2023d), Augmenting language models with long-term memory.

Wu, Y., M. N. Rabe, D. Hutchins, and C. Szegedy (2022), Memorizing transformers, *arXiv preprint arXiv:2203.08913*.

Yang, F., D. Lyu, B. Liu, and S. Gustafson (2018), Peorl: Integrating symbolic planning and hierarchical reinforcement learning for robust decision-making, in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pp. 4860–4866, International Joint Conferences on Artificial Intelligence Organization, doi:10.24963/ijcai.2018/675.

Younes, H. L., and M. L. Littman (2004), Ppddl1. 0: An extension to pddl for expressing planning domains with probabilistic effects, *Techn. Rep. CMU-CS-04-162*, *2*, 99.

Yun, K., T. Lu, A. Huyen, P. Hammer, and P. Wang (2022), Neurosymbolic hybrid approach to driver collision warning, in *Pattern Recognition and Tracking XXXIII*, vol. 12101, pp. 134–141, SPIE.

Zadeh, L. A. (1988), Fuzzy Logic, *Computer*, *21*(4), 83–93.

Zhang, J., W. Wang, X. Qi, and Z. Liao (2020a), Social and robust navigation for indoor robots based on object semantic grid and topological map, *Applied Sciences*, *10*(24), 8991.

Zhang, S., F. S. Bao, and M. Sridharan (2012), Asp-pomdp: Integrating non-monotonic logical reasoning and probabilistic planning on mobile robots, in *Proceedings of the AAMAS*, pp. 185–201, Citeseer.

Zhang, T., H. Xu, X. Wang, Y. Wu, K. Keutzer, J. E. Gonzalez, and Y. Tian (2020b), Bebold: Exploration beyond the boundary of explored regions, *arXiv preprint arXiv:2012.08621*.

Zhang, T., R. Kirk, T. Rocktäschel, E. Grefenstette, et al. (2021), Graph backup: Data efficient backup exploiting markovian data, in *Deep RL Workshop NeurIPS 2021*.

Zhao, Q., J. Han, and M. Xu (2023), Boosting policy learning in reinforcement learning via adaptive intrinsic reward regulation, *IEEE Access*.

Zheng, L., J. Chen, J. Wang, J. He, Y. Hu, Y. Chen, C. Fan, Y. Gao, and C. Zhang (2021), Episodic multi-agent reinforcement learning with curiosity-driven exploration, *Advances in Neural Information Processing Systems*, *34*, 3757–3769.

Zheng, X., B. Aragam, P. K. Ravikumar, and E. P. Xing (2018), Dags with no tears: Continuous optimization for structure learning, *Advances in neural information processing systems*, *31*.

Zhong, W., L. Guo, Q. Gao, H. Ye, and Y. Wang (2023), Memorybank: Enhancing large language models with long-term memory.

APPENDIX

A MeTTa-NARS INFERENCE RULES

Below is the list of inference rules present within Logic module of MeTTa-NARS. Only those which are structurally matched by MeTTa inbuilt pattern matcher, create derivation results during the Inference Cycle. Each rule adheres to the following format:

- Up to two premises each incorporating a copula and a truth value such as: `(($a --> $b) $T1) (($b --> $c) $T2))`, where " -->" stands for Inheritance copula and `$T1` and `$T2` are the truth values of the premises
- The derivation result with calculated truth value by a specific truth function such as: `(($a --> $c) (Truth_Deduction $T1 $T2))`

```
;;NAL-1
;;!Syllogistic rules for Inheritance:
(= (|- (($a --> $b) $T1) (($b --> $c) $T2)) (($a --> $c) (Truth_Deduction $T1 $T2)))
(= (|- (($a --> $b) $T1) (($a --> $c) $T2)) (($c --> $b) (Truth_Induction $T1 $T2)))
(= (|- (($a --> $c) $T1) (($b --> $c) $T2)) (($b --> $a) (Truth_Abduction $T1 $T2)))
(= (|- (($a --> $b) $T1) (($b --> $c) $T2)) (($c --> $a) (Truth_Exemplification $T1 $T2)))

;;NAL-2
;;!Rules for Similarity:
(= (|- (($S <-> $P) $T)) (($P <-> $S) (Truth_StructuralIntersection $T)))
(= (|- (($M <-> $P) $T1) (($S <-> $M) $T2)) (($S <-> $P) (Truth_Resemblance $T1 $T2)))
```

```
(= (|- (($P --> $M) $T1) (($S --> $M) $T2)) (($S <-> $P) (Truth_Comparison $T1 $T2)))
(= (|- (($M --> $P) $T1) (($M --> $S) $T2)) (($S <-> $P) (Truth_Comparison $T1 $T2)))
(= (|- (($M --> $P) $T1) (($S <-> $M) $T2)) (($S --> $P) (Truth_Analogy $T1 $T2)))
(= (|- (($P --> $M) $T1) (($S <-> $M) $T2)) (($P --> $S) (Truth_Analogy $T1 $T2)))
;;!Dealing with properties and instances:
(= (|- (($S --> (ExtSet $P)) $T)) (($S <-> (ExtSet $P)) (Truth_StructuralIntersection $T)))
(= (|- (((IntSet $S) --> $P) $T)) (((IntSet $S) <-> $P) (Truth_StructuralIntersection $T)))
(= (|- (((ExtSet $M) --> $P) $T1) (($S <-> $M) $T2)) (((ExtSet $S) --> $P) (Truth_Analogy $T1 $T2)))
(= (|- (($P --> (IntSet $M)) $T1) (($S <-> $M) $T2)) (($P --> (IntSet $S)) (Truth_Analogy $T1 $T2)))
(= (|- (((ExtSet $A) <-> (ExtSet $B)))) (($A <-> $B) (Truth_StructuralIntersection $T)))
(= (|- (((IntSet $A) <-> (IntSet $B)))) (($A <-> $B) (Truth_StructuralIntersection $T)))


;;NAL-3
;;!Set decomposition:
(= (|- (((ExtSet $A $B) --> $M) $T)) (((ExtSet $A) --> $M) (Truth_StructuralDeduction $T)))
(= (|- (((ExtSet $A $B) --> $M) $T)) (((ExtSet $B) --> $M) (Truth_StructuralDeduction $T)))
(= (|- ((M --> (IntSet $A $B)) $T)) ((M --> (IntSet $A)) (Truth_StructuralDeduction $T)))
(= (|- ((M --> (IntSet $A $B)) $T)) ((M --> (IntSet $B)) (Truth_StructuralDeduction $T)))
;;!Extensional and intensional intersection decomposition:
(= (|- ((($S | $P) --> $M) $T)) (($S --> $M) (Truth_StructuralDeduction $T)))
(= (|- (($M --> ($S & $P)) $T)) (($M --> $S) (Truth_StructuralDeduction $T)))
(= (|- ((($S | $P) --> $M) $T)) (($P --> $M) (Truth_StructuralDeduction $T)))
(= (|- (($M --> ($S & $P)) $T)) (($M --> $P) (Truth_StructuralDeduction $T)))
(= (|- ((($A ~ $S) --> $M) $T)) (($A --> $M) (Truth_StructuralDeduction $T)))
(= (|- (($M --> ($B - $S)) $T)) (($M --> $B) (Truth_StructuralDeduction $T)))
(= (|- ((($A ~ $S) --> $M) $T)) (($S --> $M) (Truth_StructuralDeductionNegated $T)))
(= (|- (($M --> ($B - $S)) $T)) (($M --> $S) (Truth_StructuralDeductionNegated $T)))
;;!Extensional and intensional intersection composition: (sets via reductions)
(= (|- (($P --> $M) $T1) (($S --> $M) $T2)) ((($P | $S) --> $M) (Truth_Intersection $T1 $T2)))
(= (|- (($P --> $M) $T1) (($S --> $M) $T2)) ((($P & $S) --> $M) (Truth_Union $T1 $T2)))
(= (|- (($P --> $M) $T1) (($S --> $M) $T2)) ((($P ~ $S) --> $M) (Truth_Difference $T1 $T2)))
(= (|- (($M --> $P) $T1) (($M --> $S) $T2)) (($M --> ($P & $S)) (Truth_Intersection $T1 $T2)))
(= (|- (($M --> $P) $T1) (($M --> $S) $T2)) (($M --> ($P | $S)) (Truth_Union $T1 $T2)))
(= (|- (($M --> $P) $T1) (($M --> $S) $T2)) (($M --> ($P - $S)) (Truth_Difference $T1 $T2)))
;;!Extensional and intensional intersection decomposition:
(= (|- (($S --> $M) $T1) ((($S | $P) --> $M) $T2)) (($P --> $M) (Truth_DecomposePNN $T1 $T2)))
(= (|- (($P --> $M) $T1) ((($S | $P) --> $M) $T2)) (($S --> $M) (Truth_DecomposePNN $T1 $T2)))
(= (|- (($S --> $M) $T1) ((($S & $P) --> $M) $T2)) (($P --> $M) (Truth_DecomposeNPP $T1 $T2)))
(= (|- (($P --> $M) $T1) ((($S & $P) --> $M) $T2)) (($S --> $M) (Truth_DecomposeNPP $T1 $T2)))
(= (|- (($S --> $M) $T1) ((($S ~ $P) --> $M) $T2)) (($P --> $M) (Truth_DecomposePNP $T1 $T2)))
(= (|- (($S --> $M) $T1) ((($P ~ $S) --> $M) $T2)) (($P --> $M) (Truth_DecomposeNNN $T1 $T2)))
```

```
(= (|- (($M --> $S) $T1) (($M --> ($S & $P)) $T2)) (($M --> $P) (Truth_DecomposePNN $T1 $T2)))
(= (|- (($M --> $P) $T1) (($M --> ($S & $P)) $T2)) (($M --> $S) (Truth_DecomposePNN $T1 $T2)))
(= (|- (($M --> $S) $T1) (($M --> ($S | $P)) $T2)) (($M --> $P) (Truth_DecomposeNPP $T1 $T2)))
(= (|- (($M --> $P) $T1) (($M --> ($S | $P)) $T2)) (($M --> $S) (Truth_DecomposeNPP $T1 $T2)))
(= (|- (($M --> $S) $T1) (($M --> ($S - $P)) $T2)) (($M --> $P) (Truth_DecomposePNP $T1 $T2)))
(= (|- (($M --> $S) $T1) (($M --> ($P - $S)) $T2)) (($M --> $P) (Truth_DecomposeNNN $T1 $T2)))

;; NAL-4
;;!Transformation rules between product and image:
(= (|- ((($A x $B) --> $R) $T)) (($A --> ($R /1 $B)) (Truth_StructuralIntersection $T)))
(= (|- ((($A x $B) --> $R) $T)) (($B --> ($R /2 $A)) (Truth_StructuralIntersection $T)))
(= (|- (($R --> ($A x $B)) $T)) ((($R \1 $B) --> $A) (Truth_StructuralIntersection $T)))
(= (|- (($R --> ($A x $B)) $T)) ((($R \2 $A) --> $B) (Truth_StructuralIntersection $T)))
;;other direction of same rules (as these are bi-directional)
(= (|- (($A --> ($R /1 $B)) $T)) ((($A x $B) --> $R) (Truth_StructuralIntersection $T)))
(= (|- (($B --> ($R /2 $A)) $T)) ((($A x $B) --> $R) (Truth_StructuralIntersection $T)))
(= (|- ((($R \1 $B) --> $A) $T)) (($R --> ($A x $B)) (Truth_StructuralIntersection $T)))
(= (|- ((($R \2 $A) --> $B) $T)) (($R --> ($A x $B)) (Truth_StructuralIntersection $T)))
;;!Optional rules for more efficient reasoning about relation components:
(= (|- ((($A x $B) --> $R) $T1) ((($C x $B) --> $R) $T2)) (($C --> $A) (Truth_Abduction $T1 $T2)))
(= (|- ((($A x $B) --> $R) $T1) ((($A x $C) --> $R) $T2)) (($C --> $B) (Truth_Abduction $T1 $T2)))
(= (|- (($R --> ($A x $B)) $T1) (($R --> ($C x $B)) $T2)) (($C --> $A) (Truth_Induction $T1 $T2)))
(= (|- (($R --> ($A x $B)) $T1) (($R --> ($A x $C)) $T2)) (($C --> $B) (Truth_Induction $T1 $T2)))
(= (|- ((($A x $B) --> $R) $T1) (($C --> $A) $T2)) ((($C x $B) --> $R) (Truth_Deduction $T1 $T2)))
(= (|- ((($A x $B) --> $R) $T1) (($A --> $C) $T2)) ((($C x $B) --> $R) (Truth_Induction $T1 $T2)))
(= (|- ((($A x $B) --> $R) $T1) (($C <-> $A) $T2)) ((($C x $B) --> $R) (Truth_Analogy $T1 $T2)))
(= (|- ((($A x $B) --> $R) $T1) (($C --> $B) $T2)) ((($A x $C) --> $R) (Truth_Deduction $T1 $T2)))
(= (|- ((($A x $B) --> $R) $T1) (($B --> $C) $T2)) ((($A x $C) --> $R) (Truth_Induction $T1 $T2)))
(= (|- ((($A x $B) --> $R) $T1) (($C <-> $B) $T2)) ((($A x $C) --> $R) (Truth_Analogy $T1 $T2)))
(= (|- (($R --> ($A x $B)) $T1) (($A --> $C) $T2)) (($R --> ($C x $B)) (Truth_Deduction $T1 $T2)))
(= (|- (($R --> ($A x $B)) $T1) (($C --> $A) $T2)) (($R --> ($C x $B)) (Truth_Abduction $T1 $T2)))
(= (|- (($R --> ($A x $B)) $T1) (($C <-> $A) $T2)) (($R --> ($C x $B)) (Truth_Analogy $T1 $T2)))
(= (|- (($R --> ($A x $B)) $T1) (($B --> $C) $T2)) (($R --> ($A x $C)) (Truth_Deduction $T1 $T2)))
(= (|- (($R --> ($A x $B)) $T1) (($C --> $B) $T2)) (($R --> ($A x $C)) (Truth_Abduction $T1 $T2)))
(= (|- (($R --> ($A x $B)) $T1) (($C <-> $B) $T2)) (($R --> ($A x $C)) (Truth_Analogy $T1 $T2)))
(= (|- ((($A x $B) --> $R) $T1) ((($C x $B) --> $R) $T2)) (($A <-> $C) (Truth_Comparison $T1 $T2)))
(= (|- ((($A x $B) --> $R) $T1) ((($A x $C) --> $R) $T2)) (($B <-> $C) (Truth_Comparison $T1 $T2)))
(= (|- (($R --> ($A x $B)) $T1) (($R --> ($C x $B)) $T2)) (($A <-> $C) (Truth_Comparison $T1 $T2)))
(= (|- (($R --> ($A x $B)) $T1) (($R --> ($A x $C)) $T2)) (($B <-> $C) (Truth_Comparison $T1 $T2)))
```

```
;;NAL-5
;;!Negation conjunction and disjunction decomposition:
(= (|- ((neg $A) $T)) ($A (Truth_Negation $T)))
(= (|- (($A && $B) $T)) ($A (Truth_StructuralDeduction $T)))
(= (|- (($A && $B) $T)) ($B (Truth_StructuralDeduction $T)))
(= (|- (($A && $B) $T)) (($B && $A) (Truth_StructuralIntersection $T)))
(= (|- ($S $T1) (($S && $A) $T2)) ($A (Truth_DecomposePNN $T1 $T2)))
(= (|- ($S $T1) (($S || $A) $T2)) ($A (Truth_DecomposeNPP $T1 $T2)))
(= (|- ($S $T1) (((neg $S) && $A) $T2)) ($A (Truth_DecomposeNNN $T1 $T2)))
(= (|- ($S $T1) (((neg $S) || $A) $T2)) ($A (Truth_DecomposePPP $T1 $T2)))
;;!Syllogistic rules for Implication:
(= (|- (($A ==> $B) $T1) (($B ==> $C) $T2)) (($A ==> $C) (Truth_Deduction $T1 $T2)))
(= (|- (($A ==> $B) $T1) (($A ==> $C) $T2)) (($C ==> $B) (Truth_Induction $T1 $T2)))
(= (|- (($A ==> $C) $T1) (($B ==> $C) $T2)) (($B ==> $A) (Truth_Abduction $T1 $T2)))
(= (|- (($A ==> $B) $T1) (($B ==> $C) $T2)) (($C ==> $A) (Truth_Exemplification $T1 $T2)))
;;!Conditional composition for conjunction and disjunction:
(= (|- (($A ==> $C) $T1) (($B ==> $C) $T2)) ((($A && $B) ==> $C) (Truth_Union $T1 $T2)))
(= (|- (($A ==> $C) $T1) (($B ==> $C) $T2)) ((($A || $B) ==> $C) (Truth_Intersection $T1 $T2)))
(= (|- (($C ==> $A) $T1) (($C ==> $B) $T2)) (($C ==> ($A && $B)) (Truth_Intersection $T1 $T2)))
(= (|- (($C ==> $A) $T1) (($C ==> $B) $T2)) (($C ==> ($A || $B)) (Truth_Union $T1 $T2)))
;;!Multi-conditional inference:
(= (|- ((($S && $P) ==> $M) $T1) (($S ==> $M) $T2)) ($P (Truth_Abduction $T1 $T2)))
(= (|- ((($C && $M) ==> $P) $T1) (($S ==> $M) $T2)) ((($C && $S) ==> $P) (Truth_Deduction $T1 $T2)))
(= (|- ((($C && $P) ==> $M) $T1) ((($C && $S) ==> $M) $T2)) (($S ==> $P) (Truth_Abduction $T1 $T2)))
(= (|- ((($C && $M) ==> $P) $T1) (($M ==> $S) $T2)) ((($C && $S) ==> $P) (Truth_Induction $T1 $T2)))
;;!Rules for equivalence:
(= (|- (($S <=> $P) $T)) (($P <=> $S) (Truth_StructuralIntersection $T)))
(= (|- (($S ==> $P) $T1) (($P ==> $S) $T2)) (($S <=> $P) (Truth_Intersection $T1 $T2)))
(= (|- (($P ==> $M) $T1) (($S ==> $M) $T2)) (($S <=> $P) (Truth_Comparison $T1 $T2)))
(= (|- (($M ==> $P) $T1) (($M ==> $S) $T2)) (($S <=> $P) (Truth_Comparison $T1 $T2)))
(= (|- (($M ==> $P) $T1) (($S <=> $M) $T2)) (($S ==> $P) (Truth_Analogy $T1 $T2)))
(= (|- (($P ==> $M) $T1) (($S <=> $M) $T2)) (($P ==> $S) (Truth_Analogy $T1 $T2)))
(= (|- (($M <=> $P) $T1) (($S <=> $M) $T2)) (($S <=> $P) (Truth_Resemblance $T1 $T2)))
;;!Higher-order decomposition
(= (|- ($A $T1) (($A ==> $B) $T2)) ($B (Truth_Deduction $T1 $T2)))
(= (|- ($A $T1) ((($A && $B) ==> $C) $T2)) (($B ==> $C) (Truth_Deduction $T1 $T2)))
(= (|- ($B $T1) (($A ==> $B) $T2)) ($A (Truth_Abduction $T1 $T2)))
(= (|- ($A $T1) (($A <=> $B) $T2)) ($B (Truth_Analogy $T1 $T2)))
```

B MeTTa-NARS TRUTH FUNCTIONS

Below is the full set of Truth Functions for corresponding inference rules. Truth Functions calculate the truth value for corresponding inference rules during the inference cycle.

```
(= (Truth_c2w $c) (/ $c (- 1 $c)))
(= (Truth_w2c $w) (/ $w (+ $w 1)))
(= (Truth_Deduction ($f1 $c1) ($f2 $c2)) ((* $f1 $f2) (* (* $f1 $f2) (* $c1 $c2))))
(= (Truth_Abduction ($f1 $c1) ($f2 $c2)) ($f2 (Truth_w2c (* (* $f1 $c1) $c2))))
(= (Truth_Induction $T1 $T2) (Truth_Abduction $T2 $T1))
(= (Truth_Exemplification ($f1 $c1) ($f2 $c2)) (1.0 (Truth_w2c (* (* $f1 $f2) (* $c1 $c2)))))
(= (Truth_StructuralDeduction $T) (Truth_Deduction $T (1.0 0.9)))
(= (Truth_Negation ($f $c)) ((- 1 $f) $c))
(= (Truth_StructuralDeductionNegated $T) (Truth_Negation (Truth_StructuralDeduction $T)))
(= (Truth_Intersection ($f1 $c1) ($f2 $c2)) ((* $f1 $f2) (* $c1 $c2)))
(= (Truth_StructuralIntersection $T) (Truth_Intersection $T (1.0 0.9)))
(= (Truth_or $a $b) (- 1 (* (- 1 $a) (- 1 $b))))
(= (Truth_Comparison ($f1 $c1) ($f2 $c2))
   (let $f0 (Truth_or $f1 $f2)
   ((If (== $f0 0.0) 0.0 (/ (* $f1 $f2) $f0)) (Truth_w2c (* $f0 (* $c1 $c2))))))
(= (Truth_Analogy ($f1 $c1) ($f2 $c2)) ((* $f1 $f2) (* (* $c1 $c2) $f2)))
(= (Truth_Resemblance ($f1 $c1) ($f2 $c2)) ((* $f1 $f2) (* (* $c1 $c2) (Truth_or $f1 $f2))))
(= (Truth_Union ($f1 $c1) ($f2 $c2)) ((Truth_or $f1 $f2) (* $c1 $c2)))
(= (Truth_Difference ($f1 $c1) ($f2 $c2)) ((* $f1 (- 1 $f2)) (* $c1 $c2)))
(= (Truth_DecomposePNN ($f1 $c1) ($f2 $c2))
   (let $fn (* $f1 (- 1 $f2)) ((- 1 $fn) (* $fn (* $c1 $c2)))))
(= (Truth_DecomposeNPP ($f1 $c1) ($f2 $c2)) (let $f (* (- 1 $f1) $f2) ($f (* $f (* $c1 $c2)))))
(= (Truth_DecomposePNP ($f1 $c1) ($f2 $c2)) (let $f (* $f1 (- 1 $f2)) ($f (* $f (* $c1 $c2)))))
(= (Truth_DecomposePPP $v1 $v2) (Truth_DecomposeNPP (Truth_Negation $v1) $v2))
(= (Truth_DecomposeNNN ($f1 $c1) ($f2 $c2))
   (let $fn (* (- 1 $f1) (- 1 $f2)) ((- 1 $fn) (* $fn (* $c1 $c2)))))
(= (Truth_Eternalize ($f $c)) ($f (Truth_w2c $c)))
(= (Truth_Revision ($f1 $c1) ($f2 $c2))
   (let* (($w1 (Truth_c2w $c1)) ($w2 (Truth_c2w $c2)) ($w  (+ $w1 $w2))
          ($f (/ (+ (* $w1 $f1) (* $w2 $f2)) $w)) ($c (Truth_w2c $w)))
          ((min 1.00 $f) (min 0.99 (max (max $c $c1) $c2)))))
(= (Truth_Expectation ($f $c)) (+ (* $c (- $f 0.5)) 0.5))
```

C MeTTa-NARS TERM-REDUCTION RULES

Additional Term Reduction rules:

```
;;!Extensional intersection, union, conjunction reductions:
(= (reduced ($A & $A)) $A)
(= (reduced ($A | $A)) $A)
(= (reduced ($A && $A)) $A)
(= (reduced ($A || $A)) $A)
;;!Extensional set reductions:
(= (reduced ((ExtSet $A) | (ExtSet $B))) (ExtSet $A $B))
(= (reduced ((ExtSet $A $B) | (ExtSet $C))) (ExtSet ($A SetElement $B) $C))
(= (reduced ((ExtSet $C) | (ExtSet $A $B) )) (ExtSet $C ($A SetElement $B)))
;;!Intensional set reductions:
(= (reduced ((IntSet $A) & (IntSet $B))) (IntSet $A $B) )
(= (reduced ((IntSet $A $B) & (IntSet $C))) (IntSet ($A SetElement $B) $C))
(= (reduced ((IntSet $A) & (IntSet $B $C))) (IntSet $A ($B SetElement $C)))
;;!Reduction for set element copula:
(= (reduced (ExtSet ( $A SetElement $B ))) (ExtSet $A $B))
(= (reduced (IntSet ( $A SetElement $B ))) (IntSet $A $B))
```

D MeTTa-NARS REVISION AND CHOICE

Algorithm 4 specifies the logic for Revision and Choice. *Truth_Revision* is defined in Appendix B

Algorithm 4: Revision and Choice

Input: Event $E1$, Evidential Base $b1$, $(f1, c1)$, Event $E2$, Evidential Base $b2$, $(f2, c2)$
Result: Revised truth value, or a chosen event with the highest confidence
if *Overlap(*$b1$*,* $b2$*)* **then**
  if $c1 > c2$ **then**
    return $Ev1$
  else
    return $Ev2$
  end
else
  return Truth_Revision(($f1$ $c1$), ($f2$ $c2$))
end

E ABLATION STUDY

Causal Rule Representation

The causal rule representation is foundational to NACE's operation and cannot be omitted. However, we have previously analyzed the effects of reducing the planning horizon, which limits the depth of chaining.

Temporal Locality and Attentional Bias

Omitting these biases with larger environment sizes is infeasible due to the combinatorial explosion of potential rules as we will now analyze.

- For an environment of size $w \times h$, the number of possible rule preconditions for a single timestep is $2^{w \cdot h}$, as each particular cell can either be considered or not be considered in the precondition of a rule.
- For a time window of duration d, this expands to $2^{w \cdot h \cdot d}$, leading to $1.8447 * 10^{19}$ possible rule preconditions for an 8×8 grid within a single timestep.
- NACE is tied to Markov Assumption particularly within the observational window, as all rule construction and updating considers only the previous and current state. However, its bird view map representation also contains values from observations of previous time steps which are currently out of view of the agent, which brings us to the next point, state tracking.

State Tracking

Without state tracking, NACE lacks memory of prior observations and memory of observations which are outside of its field-of-view. This results in oscillatory behavior caused by the exploration strategy of the agent, as it can only utilize information the visible information.

- In our experiments across 10 runs in MiniGrid-Empty-8x8-v0, this led the agent to turn indefinitely due to the curiosity model assigning low match values to previously visited areas (due to the lack of state tracking they are always considered to be of unknown value) which are now outside of the field-of-view of the agent. The closest such cell is immediately behind the agent with the default partial observation model in Minigrid, which explains the behavior.

- State Tracking plays a critical role in ensuring purposeful exploration and decision-making, for the agent to know which places have been visited and what it has been observing at the particular locations, as well as which locations have yet to be observed.

- Sequential dependencies often depend on state tracking. An example of this is when a door has to be opened with a key, where the key and the door is too far apart to be observed concurrently, demanding some form of spatial memory. Another form of state tracking lies in the observable inventory array, which when absent would need the modeling of long-range temporal dependencies (e.g. did the agent already pick up the key?) which would demand a suitable model structure to be learnable by the agent.

Spatial Equivariance

The absence of spatial equivariance significantly impacts sample efficiency.

- Each rule must be learned independently for every location, meaning in an 8x8 grid, the agent has to learn 64 times the same set of rules. However, since particular arrangements of cell values will not re-appear through the environment generation, it can take significantly longer to learn the relevant knowledge without this bias.

- Hence for the general case with an environment of size $w \times h$, this increases the

required sample count at least by a factor of $w \cdot h$, harming significantly the sample efficiency of the technique.

- Conceptually, we also would like to point out that the rule learning mechanisms do not allow to learn spatial equivariance retrospectively either, while some DRL techniques, dependent on the model structure, could potentially acquire it.

These results highlight the necessity of each inductive bias in ensuring the scalability, efficiency, and functionality of NACE.

Inductive Biases Present in the DRL Techniques

In the Chapter 4 we outlined the inductive biases of NACE. However we would like to point out that some of them are also inherent in the DRL techniques, complementing our discussion on inductive biases in DRL and NACE:

- **Temporal Locality**: The DRL methods perform best when the Markov Assumption is met, despite LSTM allowing to cope with partial observability, the need to capture long-range temporal dependencies makes sample efficient learning more difficult.

- **Causal Representation**: While not explicitly stated as a set of cause-effect relations, DreamerV3's learned dynamics model can predict the consequence states of actions, which is not the case for the model-free DRL methods. Such modeling is to some extent independent from the objective (what is rewarding), and allows an agent to train itself from simulated experience by predicting novel states, and to reach novel goals.

- **Spatial Equivariance**: Clearly the DRL techniques do not have an explicit rule representation, however the Convolution layers in the DRL policies allow for learned features to be identified at different locations, improving generalization.

- **State Tracking**: Is not explicitly handled by the DRL techniques as a separate point, instead it is handled in the same way as non-local temporal dependencies in the LSTM-including policies, while NACE builds a bird view map explicitly, which can be considered to be a form of spatial memory.

- **Attentional Bias**: While NACE has a strong prior to which cells to consider based on observably changed values and prediction mismatches, the DRL policies with Convolution layers are more flexible and allow an agent to learn which values are relevant in relation to each other.

F NOTATION AND SYMBOLS IN CHAPTER 4

Symbol	Description
s	State, represented as a combination of a 2D grid (s_{grid}) and a 1D array (s_{array})
a	Action taken by the agent
r	Causal rule in the form (preconditions, action) $\Rightarrow$ consequence
$c_{t,x,y}$	Cell value at position (x, y) in the 2D grid at time t
$\overline{c}$	Equality constraint on a cell value (e.g., $c_r = c$)
v_t	Value array at time t
$\overline{v}$	Equality constraint on value array (e.g., $v_r = v$)
$M(r, c)$	Match value of a rule r for cell c, based on the fraction of preconditions satisfied
$C(c)$	Cell match value for cell c, derived from the maximum match value across all rules
$S(s)$	State match value for state s, calculated as the average $C(c)$ for cells with $C(c) > 0$
$w_+(r)$	Positive evidence counter for rule r, incremented when predictions align with observations
$w_-(r)$	Negative evidence counter for rule r, incremented when predictions differ from observations
$w(r)$	Total evidence count for rule r, defined as $w(r) = w_+(r) + w_-(r)$
$frequency(r)$	Fraction of positive evidence for rule r, defined as $f(r) = \frac{w_+(r)}{w(r)}$
$confidence(r)$	Confidence factor for rule r, defined as $c(r) = \frac{w(r)}{w(r)+1}$
$f_{\text{exp}}(r)$	Expected truth value for r, calculated as $f_{\text{exp}}(r) = (f(r) - \frac{1}{2}) \cdot c(r) + \frac{1}{2}$
M_t^{change}	Set of cells with changes in observed values between timesteps $t-1$ and t
$M_{\text{mismatched},t}^{\text{observation}}$	Set of cells where observed values differ from predicted values at timestep t
$M_{\text{mismatched},t}^{\text{prediction}}$	Set of cells where predictions differ from observations at timestep t
$R(r)$	Reward associated with rule r
$R(s)$	Reward associated with a state s, defined as the average reward of rules applied to generate s
$V(s)$	Value of state s, used in planning for maximizing long-term returns
$\pi(t)$	Planned action sequence or policy at timestep t
γ	Discount factor for future rewards

G NACE PSEUDOCODE

The system can be described by the following pseudocode:

Algorithm 5: Pseudocode of NACE

- **Actual World:** perceived_array = perceive_partial(world)
- **Observer:**
  s_t = update_bird_view(s_{t-1}, perceived_array)
  $calculate(M_{change}, M^{observation}_{mismatched}, M^{prediction}_{mismatched})$
- **Hypothesizer:**
  - Create new rules for which $w_+(r) = 1$.
  - Update rule evidences according to $w_+(r)$ and $w_-(r)$.
  - Choose rules r_1 with $w_+(r_1) > w_-(r_1)$ for which there does not exist a rule r_2 with same precondition and action, but different postcondition with $T_{exp}(r_2) > T_{exp}(r_1)$.
- **Planner utilizing Predictor:**
  $a_1, ..., a_n = BFS_with_Predictor(V(s) > 0)$
  $a^*_1, ..., a^*_n = BFS_with_Predictor(min(S(s)) < 1)$
  //whereby BFS_with_Predictor is bounded breadth first search with Predictor as state transition function
  If found$(a_1, ..., a_n)$:, return $a_1, ..., a_n$
  If found$(a^*_1, ..., a^*_n)$:, return $a^*_1, ..., a^*_n$
  Else Perform a random action

H HYPERPARAMETER DETAILS FOR DRL MODELS

Foundational Algorithms

Core Models and Their Mechanisms

- **Deep Q-Network (DQN):** DQN integrates deep neural networks with classical Q-learning, making it effective for handling large state spaces. To stabilize training, DQN uses experience replay and a separate target network. The Q-value update in DQN follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a)\big)$$

  where:

  - s, a: current state and action,
  - s', a': next state and action,
  - $R(s)$: reward received,
  - γ: discount factor for future rewards,
  - α: learning rate.

- **Advantage Actor-Critic (A2C):** A2C builds on the actor-critic framework, synchronizing multiple parallel learners to reduce variance in policy updates. It calculates an **advantage function** to evaluate actions relative to the current policy's value estimate, stabilizing training but requiring frequent environmental interactions due to its on-policy nature.

  Advantage Function:

$$A(s,a) = Q(s,a) - V(s)$$

Policy Update: The policy is updated using the gradient:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a|s) A(s,a)$$

where:

- $Q(s,a)$: action-value function,
- $V(s)$: state-value function,
- $\pi_\theta(a|s)$: policy parameterized by θ,
- α: learning rate.

- **Trust Region Policy Optimization (TRPO):** TRPO addresses stability in policy updates by enforcing a trust region constraint, ensuring small policy changes during optimization. This constraint is implemented via a KL-divergence bound, preventing drastic shifts in behavior but requiring computationally expensive second-order optimization.

  Objective Function:

$$\max_\theta \mathbb{E}_{s\sim\pi_{\theta_{\text{old}}}} \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} A(s,a) \right]$$

  Constraint:

$$\mathbb{E}_{s\sim\pi_{\theta_{\text{old}}}} \left[D_{\text{KL}}(\pi_{\theta_{\text{old}}} || \pi_\theta) \right] \leq \delta$$

  where:

  - $\pi_\theta(a|s)$: new policy,
  - $\pi_{\theta_{\text{old}}}(a|s)$: previous policy,
  - $A(s,a)$: advantage function,
  - D_{KL}: KL-divergence,

- δ: trust region size.

- **Proximal Policy Optimization (PPO):** PPO refines TRPO by introducing a clipped surrogate objective, which simplifies computation and allows for multiple updates per batch. This approach improves data utilization while maintaining policy stability.

  Clipped Surrogate Objective:

$$\max_{\theta} \mathbb{E}_{s,a}\left[\min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}A(s,a), \text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}, 1-\epsilon, 1+\epsilon\right)A(s,a)\right)\right]$$

  where:

  - $\pi_\theta(a|s)$: new policy,
  - $\pi_{\theta_{\text{old}}}(a|s)$: old policy,
  - $A(s,a)$: advantage function,
  - ϵ: clipping threshold.

Hyperparameter Configuration for Foundational Algorithms

We utilize the Stable Baselines3 framework (*Raffin et al.*, 2021) to train and evaluate foundational algorithms, leveraging its pre-implemented models and customizable configurations. All algorithms use the same convolutional neural network architecture to process observations, ensuring consistency across experiments. The hyperparameters for each algorithm were selected based on achieving the best average performance across all tasks, rather than optimizing for a single task, to ensure generalizability. The details of the network architecture and training setup for each algorithm are outlined below.

Network Architecture: Observations ($7 \times 7 \times 3$) from the Minigrid environment are processed through four convolutional layers. Each layer is configured as follows:

- Kernel size: 2×2
- Activation: ReLU
- Increasing number of filters: 16, 32, 64, and 128

The output of the final convolutional layer is flattened and passed to a fully connected layer with:

- Output dimension: 128
- Activation: ReLU

Training Configurations:

Table H.1: DQN and A2C Training Parameters

Parameter	**DQN**
Learning rate	0.0001
Buffer size	1,000,000
Learning starts	100
Batch size	32
Soft update coefficient	1
Discount factor	0.99
Train frequency	4
Gradient steps	1
Target update interval	10,000
Exploration fraction	0.1
Initial exploration epsilon	1.0
Final exploration epsilon	0.05
Max gradient norm	10.0

Parameter	**A2C**
Learning rate	0.0007
Number of steps	5
Discount factor	0.99
Entropy coefficient	0.0
Value function coefficient	0.5
Max gradient norm	0.5

Table H.2: TRPO and PPO Training Parameters

Parameter	TRPO
Learning rate	0.001
Number of steps	2048
Batch size	128
Discount factor	0.99
Conjugate gradient max steps	15
Conjugate gradient damping	0.1
Line search shrinking factor	0.8
Line search max iterations	10
Number of critic updates	10
Target KL divergence	0.01

Parameter	PPO
Learning rate	0.0003
Number of steps	2048
Batch size	64
Number of epochs	10
Discount factor	0.99
Clip range	0.2
Entropy coefficient	0.0
Value function coefficient	0.5
Max gradient norm	0.5

Model-Based Algorithm: DreamerV3

DreamerV3 is a model-based RL algorithm designed to enhance sample efficiency by learning a latent world model of the environment. It optimizes both the world model and the policy within the latent space, reducing the computational demands of interacting with the environment.

World Model: The latent dynamics model predicts future latent states z based on prior latent state z_{t-1}, action a_{t-1}, and reward R_{t-1}. This model facilitates long-term planning without requiring explicit rollouts in the actual environment.

Policy Optimization: The policy maximizes expected rewards in the learned latent space by leveraging the dynamics model to simulate trajectories. Policy updates use gradient-based methods informed by imagined rollouts.

Loss Function:

$$\mathscr{L}_{\text{DreamerV3}} = \mathscr{L}_{\text{Reconstruction}} + \mathscr{L}_{\text{Dynamics}} + \mathscr{L}_{\text{Policy}}$$

where:

- $\mathscr{L}_{\text{Reconstruction}}$: Measures the accuracy of reconstructing environment observations,

- $\mathscr{L}_{\text{Dynamics}}$: Captures consistency in latent state transitions,
- $\mathscr{L}_{\text{Policy}}$: Maximizes imagined rewards.

Hyperparameter Configuration for DreamerV3: The hyperparameter configuration has been chosen to match the settings provided in `https://github.com/qxcv/dreamerv3`. To avoid redundancy and maintain brevity, we do not include the full configuration here due to its extensive nature.

Model-Free Exploration and Scalability Extensions

All experiments for the other model-free methods are based on the Torchbeast implementation of IMPALA (*Espeholt et al.*, 2018), which has been modified to support intrinsic reward algorithms as described in *Raileanu and Rocktäschel* (2020) and *Campero et al.* (2021). The hyperparameters were selected following the configurations used in these references. For clarity, we first list the values shared across all algorithms, followed by the specific details unique to each one.

Shared Hyperparameters

- **Network Architecture:** Observations ($7 \times 7 \times 3$ for Minigrid) are processed through three convolutional layers:
  - Number of filters: 32 per layer
  - Kernel size: 3×3
  - Stride: 2
  - Padding: 1
  - Activation: Exponential Linear Unit (ELU)

  The output of the convolutional layers is passed to:

  - An LSTM layer to address partial observability by maintaining temporal dependencies and encoding sequences of observations.
  - A fully connected layer for computing:
    * Policy logits: Unnormalized scores for each action, converted to probabilities using a softmax function.
    * Value estimates: Predictions of expected future returns, used in actor-critic methods.

- **Training Setup:**
  - Number of actors: 40
  - Number of buffers: 80
  - Unroll length: 100
  - Number of learner threads: 4
  - Batch size: 32
  - Discount factor: 0.99
  - Learning rate: 0.0001
  - Policy entropy loss: 0.0005
  - Gradient clipping: Norm of 40
  - Save interval: Every 20 minutes

- **Special Parameters (Only When Applicable):**
  - Count reset probability: 0.001 (COUNT, RIDE)
  - Hash bits: 128 (COUNT)

Intrinsic Rewards and Coefficients

Intrinsic rewards address sparse rewards and inefficient exploration. Each algorithm applies scaling coefficients to normalize its intrinsic rewards. Additionally, all techniques incorporate policies enhanced with LSTMs to address partial observability by maintaining memory of past observations and actions.

- **IMPALA:** No intrinsic reward ($r_i = 0.0$).
- **COUNT:** $r_i = 0.005$.
- **RIDE:** $r_i = 0.1$.
- **CURIOSITY:** $r_i = 0.1$.
- **RND:** $r_i = 0.1$.
- **AMIGO:** $r_i = 0.1$ (applies to the teacher's intrinsic rewards).

The formal definitions of the intrinsic rewards are:

COUNT: The intrinsic reward is based on state visitation counts, encouraging exploration of less-visited states:

$$r_i = \frac{1}{N(s_0)},$$

where $N(s_0)$ is the (pseudo)count of visits to state s_0. Counts are never reset during training.

RIDE (Rewarding Impact-Driven Exploration): The intrinsic reward combines state novelty and state-change impact:

$$r_i = \|\phi(s) - \phi(s_0)\|_2 \cdot \frac{1}{N(s_0)},$$

where ϕ is trained to minimize both forward and inverse dynamics prediction errors. Counts $N(s_0)$ are reset at the beginning of each episode.

CURIOSITY: The intrinsic reward comes from the prediction error of a forward dynamics model f, which predicts the next state embedding $\phi(s_0)$ from the current embedding $\phi(s)$ and action a:

$$r_i = \|f(\phi(s), a) - \phi(s_0)\|_2.$$

RND (Random Network Distillation): The intrinsic reward is computed as the prediction error of a trainable network ϕ attempting to match the output of a fixed random network $\hat{\phi}$:

$$r_i = \|\phi(s_0) - \hat{\phi}(s_0)\|_2.$$

AMIGO: The teacher policy generates goals g for the agent, with rewards given as:

$$r_i = v(s_t, g) = \begin{cases} +1 & \text{if } s_t \text{ satisfies } g, \\ 0 & \text{otherwise.} \end{cases}$$

The total reward is a weighted sum of intrinsic and extrinsic rewards:

$$r_t = \beta r_i + \alpha r_e, \quad \text{with } \beta = 0.3, \alpha = 0.7.$$

Algorithm-Specific Hyperparameters and Architectures

Algorithm	**Details**
IMPALA (Baseline)	Intrinsic reward: None. Loss: Policy gradient, baseline, entropy.
COUNT	Intrinsic reward: State visitation counts. Count reset probability: $p = 0.001$.
CURIOSITY	Intrinsic reward: Forward prediction error. State embedding: 256-dimensional. Forward model predicts next state. Inverse model predicts actions. Loss weights: Forward 10.0, Inverse 0.1.
RIDE	Intrinsic reward: Counts $\times$ norm of state change. Modules: Same as CURIOSITY.
RND	Intrinsic reward: Prediction error. Target net: Fixed embeddings. Predictor: Trained for target match. Loss weight: 0.1.
AMIGO	Intrinsic reward: Teacher-generated. Reward coefficients: Intrinsic $\beta = 0.3$, Extrinsic $\alpha = 0.7$. Batch size: 150. Entropy cost: 0.05. Threshold: -0.5.

I HARDWARE AND RUNTIME

In this section, we describe the hardware setup used to run the techniques and provide runtime characteristics, including the duration of 5000 representative timesteps for each technique. While we report this information for reproducibility, we emphasize that the focus of our analysis is not on computational cost, but rather on sample efficiency.

- CPU: Intel Core i7-9750H with 32GB RAM
- GPU: Geforce GTX-1660 Ti with 6GB RAM

Table I.1: Runtime (in seconds) across *MiniGrid-v0* environments.

Algorithm	Empty-16x16	DistShift2	LavaGapS7	SimpleCrossingS11N5	Unlock	DoorKey-8x8	Average
TRPO	751.46	84.89	84.05	374.76	241.70	489.37	337.04
PPO	745.70	149.06	128.53	384.06	240.66	493.73	357.29
A2C	625.22	181.67	157.37	323.45	207.82	413.42	318.83
DQN	593.23	130.77	122.55	284.60	156.81	321.42	268.90
IMPALA	268.98	282.99	274.11	262.70	267.95	249.48	267.70
COUNT	357.52	403.85	281.94	332.05	257.58	273.81	317.13
RIDE	361.53	354.95	288.58	377.65	338.89	265.41	331.17
CURIOSITY	392.20	299.62	329.57	358.67	323.51	296.38	333.66
RND	357.41	298.23	308.14	378.99	317.51	373.73	339.67
AMIGo	49.32	77.84	68.38	47.68	59.88	44.30	57.90
DreamerV3	102.38	78.67	84.39	76.01	73.38	78.62	82.91
NACE	314.12	301.93	302.06	301.58	302.24	360.11	313.67

J EXAMPLE WITH LEARNED RULES

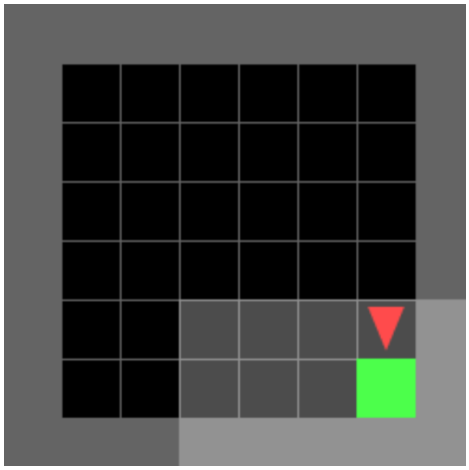


Figure J.1: Illustration of Minigrid-Empty-8x8.

The following are all the rules NACE learns in the Minigrid-Empty-8x8 environment as illustrated in Figure J.1:

```
Agent interacting with goal location:
<(v=[1], c[ 0, 0 ]='x', c[ 0, 1 ]='H', ^down ) =/> (v=[0], c[ 0, 0 ]='.', R(r)=1)>.
<(v=[1], c[-1, 0 ]='H', c[ 0, 0 ]='x', ^left ) =/> (v=[0], c[ 0, 0 ]='.', R(r)=1)>.
<(v=[1], c[ 0,-1 ]='H', c[ 0, 0 ]='x', ^up   ) =/> (v=[0], c[ 0, 0 ]='.', R(r)=1)>.
<(v=[1], c[ 0, 0 ]='x', c[ 1, 0 ]='H', ^right) =/> (v=[0], c[ 0, 0 ]='.', R(r)=1)>.
Goal location interacting with agent:
<(v=[1], c[ 0,-1 ]='x', c[ 0, 0 ]='H', ^down ) =/> (v=[0], c[ 0, 0 ]='.', R(r)=1)>.
<(v=[1], c[ 0, 0 ]='H', c[ 1, 0 ]='x', ^left ) =/> (v=[0], c[ 0, 0 ]='.', R(r)=1)>.
<(v=[1], c[ 0, 0 ]='H', c[ 0, 1 ]='x', ^up   ) =/> (v=[0], c[ 0, 0 ]='.', R(r)=1)>.
<(v=[1], c[-1, 0 ]='x', c[ 0, 0 ]='H', ^right) =/> (v=[0], c[ 0, 0 ]='.', R(r)=1)>.
Agent interacting with empty space:
<(v=[1], c[ 0, 0 ]=' ', c[ 0, 1 ]='x', ^up   ) =/> (v=[1], c[ 0, 0 ]='x', R(r)=0)>.
<(v=[1], c[-1, 0 ]='x', c[ 0, 0 ]=' ', ^right) =/> (v=[1], c[ 0, 0 ]='x', R(r)=0)>.
<(v=[1], c[ 0,-1 ]='x', c[ 0, 0 ]=' ', ^down ) =/> (v=[1], c[ 0, 0 ]='x', R(r)=0)>.
<(v=[1], c[ 0, 0 ]=' ', c[ 1, 0 ]='x', ^left ) =/> (v=[1], c[ 0, 0 ]='x', R(r)=0)>.
Empty space interacting with agent:
<(v=[1], c[ 0,-1 ]=' ', c[ 0, 0 ]='x', ^up   ) =/> (v=[1], c[ 0, 0 ]=' ', R(r)=0)>.
<(v=[1], c[ 0, 0 ]='x', c[ 1, 0 ]=' ', ^right) =/> (v=[1], c[ 0, 0 ]=' ', R(r)=0)>.
<(v=[1], c[ 0, 0 ]='x', c[ 0, 1 ]=' ', ^down ) =/> (v=[1], c[ 0, 0 ]=' ', R(r)=0)>.
<(v=[1], c[-1, 0 ]=' ', c[ 0, 0 ]='x', ^left ) =/> (v=[1], c[ 0, 0 ]=' ', R(r)=0)>.
Agent interacting with wall:
```

```
<(v=[1], c[ 0,-1 ]='o', c[ 0, 0 ]='x', ^up   ) =/> (v=[1], c[ 0, 0 ]='x', R(r)=0)>.
<(v=[1], c[ 0, 0 ]='x', c[ 1, 0 ]='o', ^right) =/> (v=[1], c[ 0, 0 ]='x', R(r)=0)>.
<(v=[1], c[ 0, 0 ]='x', c[ 0, 1 ]='o', ^down ) =/> (v=[1], c[ 0, 0 ]='x', R(r)=0)>.
<(v=[1], c[-1, 0 ]='o', c[ 0, 0 ]='x', ^left ) =/> (v=[1], c[ 0, 0 ]='x', R(r)=0)>.
Wall interacting with agent:
<(v=[1], c[ 0, 0 ]='o', c[ 1, 0 ]='x', ^left ) =/> (v=[1], c[ 0, 0 ]='o', R(r)=0)>.
<(v=[1], c[ 0, 0 ]='o', c[ 0, 1 ]='x', ^up   ) =/> (v=[1], c[ 0, 0 ]='o', R(r)=0)>.
<(v=[1], c[-1, 0 ]='x', c[ 0, 0 ]='o', ^right) =/> (v=[1], c[ 0, 0 ]='o', R(r)=0)>.
<(v=[1], c[ 0,-1 ]='x', c[ 0, 0 ]='o', ^down ) =/> (v=[1], c[ 0, 0 ]='o', R(r)=0)>.
```

The amount of learned rules required to deal with the Minigrid environments required typically vary between 16 (minimum with walls and free space) and usually less than 100 dependent on the amount of cell types, whereby for two cell types to interact with m actions, at least $2 * m$ additional rules are learned.