Autonomy through real-time learning and OpenNARS for Applications

---

A Dissertation
Submitted to
the Temple University Graduate Board

---

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

---

by

Patrick Hammer
2021

Examining Committee Members:

Slobodan Vucetic, Committee Chair, Department of Computer and Information Sciences, Temple University
Pei Wang, Advisor, Department of Computer and Information Sciences, Temple University
Jamie Payton, Dept. of Computer and Information Sciences, Temple University
Claes Strannegård, Department of Computer Science and Engineering, Chalmers University of Technology

For all the people

ACKNOWLEDGEMENTS

Thanks to Professor Pei Wang for having added me to his team and guided me through the Master and PhD program at Temple University. His many decades of commitment and publications about the NARS research project, and a working implementation have convinced me that AGI is not just a dream but a research field incredible to be a part of. Also it was an honor to work with him and his team.

Special thanks to my friend Tony Lofthouse for having assisted me in creating the ONA architecture and its implementation, and who has published with me about this topic. He has dedicated many years to finding a control model which can make NARS more effective, many of his insights are inherited by ONA.

Also special thanks to Professor Slobodan Vucetic, Professor Jamie Payton, Professor Pei Wang and Professor Claes Strannegård for being part of my dissertation committee and for their valuable feedback which helped me significantly to create and summarize relevant results, also thanks to Professor Kristinn R. Thórisson and Professor Antonio Chella in this regard.

Also thanks to my friend and collaborator Francesco Lanza for having assisted me with creating Robot Operating System modules and related experiments, and to Professor Robert Johansson, Leonard M. Eberding, Christian Hahm and Robert Wuensche for having carried out experiments with my system.

Additionally I want to thank other contributors to the open source repository of the system this thesis describes: Joris Bontje, Yuhong Sun, Jarrad Hope, Paul Trojahn, and Alexander Wentz. And I want to thank my team mates Peter Isaev and Xiang Li for very valuable discussions.

And, thank you for Dr. Hugo Latapie and Dr. Ed Chow for the interesting application projects which provided both great opportunities and feedback to improve

the reasoning system.

Also, I want to additionally thank Enzo Fenoglio, Valeria Seidita for having co-authored relevant papers which results which are relevant to this thesis.

Last, thanks to my wonderful girlfriend Daria Scheer, and my cat Yobi Linski for having been so patient with me and for having supported me during the PhD program, this also applies to my whole family.

TABLE OF CONTENTS

DEDICATION ii

ACKNOWLEDGEMENTS iii

LIST OF FIGURES viii

LIST OF TABLES ix

ABSTRACT x

CHAPTER

1. Motivation 1

1.1 History and state-of-the-art of agent autonomy 1
1.2 Research question 5

2. OpenNARS for Applications 8

2.1 Theoretical Foundation 8
2.1.1 Non-Axiomatic Logic 10
2.1.2 From statements to events, sequences and temporal credit assignment 14
2.1.3 Memory and Control considerations 15
2.1.4 Representing and learning procedure knowledge . . 16
2.1.5 Goal-directed Procedure Learning: the problem . . 18
2.1.6 Goal-directed Procedure Learning: the solution . . . 20
2.2 Implementation design 22
2.2.1 Objective 22
2.2.2 Data Structures 24
2.2.3 Efficiency considerations 27
2.2.4 Architecture 28
2.2.5 Operating cycle 32
2.3 Reaching desired capabilities via a pragmatic approach 39
2.3.1 The big picture 39
2.3.2 Visual perception 40
2.3.3 Adaptation 41
2.3.4 Planning 43
2.4 Implementation usage 44

3. Comparison with prior work 48

3.1 Drescher's schemas 48
3.2 OpenNARS 52
3.2.1 Priority Queue instead of Bag 52
3.2.2 Statement concepts and implication links 53
3.2.3 Concept usefulness 54
3.2.4 Limitations in ONA 55
3.3 Adaptive Neuro-Symbolic Network Agent 57
3.3.1 Similar work and philosophical differences 58
3.3.2 Data Structures 61
3.3.3 Practical consideration: Voting Schema 64
3.3.4 ONA: why compound terms instead of SDR's 67
3.4 Belief-Desire-Intention models 68
3.5 Reinforcement Learning 69
3.5.1 Introduction 69
3.5.2 A reasoner which learns and makes decisions 71
3.5.3 Distinguishing properties of ONA and Q-Learning . 79

4. Experiments and Applications 83

4.1 ONA vs. Q-Learning 83
4.2 Tests in cognitive psychology 88
4.2.1 Acquiring object permanence 89
4.2.2 Property association 91
4.2.3 Stanford Marshmallow test 92
4.2.4 Arbitrarily Applicable Relational Responding 93
4.2.5 Active question answering 94
4.2.6 Disambiguation of nouns 96
4.2.7 Story understanding 98
4.2.8 Knowledge transfer 99
4.2.9 Other use cases 102
4.3 Robotics experiments 103
4.3.1 Practical Reasoning and Learning 103
4.3.2 ONA-ROS Interface 112
4.4 SmartCity Application 114
4.4.1 Introduction 114
4.4.2 Architecture 115
4.4.3 Results 119
4.5 TruePAL Application 122
4.5.1 Introduction 122
4.5.2 Architecture 123
4.5.3 Results 126

5. Discussion 132

5.1 Conclusion 132
5.2 Future works 134

APPENDIX A 135
A.1 ONA: Sensorimotor inference rules 136
A.2 ONA: Semantic inference rules 137

BIBLIOGRAPHY 139

LIST OF FIGURES

Figure

1.1 Specialization and Generalization 5
2.1 Data Structures utilized 24
2.2 High-level architecture showing input sequencing and cycles for sensorimotor and semantic inference 29
2.3 Shell interaction with syntax highlighting 45
3.1 Drescher schema 48
3.2 Schema chain 49
3.3 Schema extension 49
3.4 Schema extension 51
3.5 Zeroshot learning 52
3.6 SDR interpolation as illustrated in *Kanerva* (1992) 66
4.1 Space invaders, Pong and grid robot 83
4.2 Success ratio in Space invaders and Pong 84
4.3 ONA behavior graph after learning Pong 85
4.4 Success ratio in simplified Pong and Grid Robot 86
4.5 Components diagram 106
4.6 Robot in front of obstacles 107
4.7 Bottle collect mission 111
4.8 How the ROS modules are connected 113
4.9 Running modules visualized by RosGui 113
4.10 Spatial discretization by a grid 118
4.11 A pedestrian in danger due to close proximity to a quickly moving car 121
4.12 An example of Jaywalking 121
4.13 TruePAL Architecture 123
4.14 TruePAL collision detection 128
4.15 TruePAL expected first responder 129
4.16 TruePAL risky street 129
4.17 TruePAL speed warning 130
4.18 TruePAL roadside safety 131

LIST OF TABLES

Table

4.1 End performance results of all experiments. (after 10000 iterations) 88
4.2 Performance in Street Scene 122
A.1 Truth functions used for sensorimotor reasoning 136
A.2 Additional truth functions used for declarative reasoning 137

ABSTRACT

Autonomy through real-time learning and OpenNARS for Applications

by

Patrick Hammer

This work includes an attempt to enhance the autonomy of intelligent agents via real-time learning. In nature, the ability to learn at runtime gives species which can do so key advantages over others. While most AI systems do not need to have this ability but can be trained before deployment, it allows agents to adapt to changing and generally unknown circumstances. To meet this goal, in this thesis a pragmatic design (ONA) for a general-purpose reasoner incorporating Non-Axiomatic Reasoning System (NARS) theory is explored. The design and implementation is presented in detail, in addition to necessary theoretical constructs. Then, experiments related to various system capabilities are carried out and summarized, together with application projects where ONA is utilized. Also it is shown how reliable real-time learning can help to increase autonomy of intelligent agents beyond the current state-of-the-art. Here, theoretical and practical comparisons with established frameworks (such as Reinforcement Learning) and specific techniques such as Q-Learning, and Belief-Desire-Intention models are made. The capabilities are also demonstrated on real robotic hardware, the experiments there show the potential of combining learning knowledge at runtime with the utilization of only partly complete mission-related background knowledge given by the designer.

CHAPTER 1

Motivation

1.1 History and state-of-the-art of agent autonomy

As of April 28, 2021, there are no commonly available personal AI assistants which can both remember and reason about arbitrary information in different contexts. Also, despite all the progress in machine learning there are no commonly used autonomous agents which can generalise across domains without a human-in-the-loop to maintain the dataset and retrain the model between rollouts.

If the goal of AI is to create an optimised function to solve a problem in a specific domain, then we are making good progress. Optimization, in particular backpropagation with deep neural networks (DL), has proven to be better than handcrafting for many specific problems which were considered to be hard [1] [6] [11]. This turned out to be true for computer vision, language translation, speech recognition, board games, computer games, and many more. This has led to the rapid global adoption of this approach.

What all of these problems have in common is that no learning is required after model deployment. However, learning at runtime is an ability which gives intelligent animals a key survival advantage. What has made machine learning so successful is a narrower idea of learning. That is, solving an offline optimization problem which in nature is carried out by evolution and not by the intelligence of individuals. Example: find a genetic code such that the firefly will not only detect, with high accuracy, the types of prey necessary for its survival, but also catch it successfully. Here, the firefly itself does not need to learn at runtime. The same is true for autonomous cars which

are expected to be able to drive right away, with pre-wired components for navigation, localization, object detection etc. Some of these components are programmed and others have model parameters resulting from offline-optimization.

The key challenge AI has not yet convincingly completed is to move from offline optimization towards learning quickly and reliably at run time. This is not only intelligence's primary role in nature but is also the original goal of AI. AGI, like an animal in the wild, is supposed to be able to deal, at runtime, with unforeseen circumstances. An ability to adapt quickly and reliably will not only push forward the next generation of robotic explorers and personal assistants, but can also be seen as a key aspect of intelligence.

Intelligence is a term with many meanings. In volume 10 (2019) of the Journal of Artificial General Intelligence (JAGI) the article [7], "On Defining Artificial Intelligence" by Dr. Pei Wang, was published. This JAGI article is considered to be one of the strongest attempts, since the beginning of the field, to address the long-standing lack of consensus on how to define the field and topic of Artificial Intelligence (AI). The review process invited 110 leading AI experts, including researchers from DeepMind and Google Brain [8], and professors from many universities. The core of the article is centered around the author's definition of intelligence itself: "The essence of intelligence is the principle of adapting to the environment while working with insufficient knowledge and resources". While his definition has been the most agreed-upon in the related AGISI survey (with 567 participants), some responses expressed disagreements about the resource constraint and necessity to adapt in real time. Some of these disagreements stem from the difference between artificial and biological systems, the latter of which evolved to adapt under knowledge and resource insufficiency [10], while many AI systems are not equipped with an ability to adapt after deployment.

Several misconceptions about the nature of intelligence come about by leaving

learning at runtime out of the picture. For instance, genetic algorithms (GA, [5]) are sometimes sold as an alternative to reinforcement learning (RL, [12]). With the previous considerations taken into account, it is clear that GA is to RL, as evolution is to intelligence. They are fundamentally different with an overlap only occurring when learning happens in a multi-generation simulation, instantiated over an arbitrary number of generations. This instantiation property is not the case for an autonomous robot, or animal for that matter, both of which are expected to be able to adapt to unforeseen circumstances as quickly as possible and within a single lifetime. Clearly, they cannot continue learning once any fatal action is taken. These aspects are a large part of the reason why the great successes in domains with perfect simulation availability (such as [11]) have not translated into successes in real-world domains. For this, offline optimization will always be limiting compared to a system capable of adapting in real time.

From a machine learning perspective, this poses several challenges:

1. The agent needs to be able to deal with dynamic (non-stationary!) environments.

2. A proper decision theory has to be found, which for animals was achieved by evolution. This one cannot be learned during a single agent's lifetime. It needs to be innate, while exhibited behavior will be a combination of nature and nurture.

One decision theory which has had a lot of success is reinforcement learning (RL, [12]). Whilst it struggles with non-stationary environments (learning rate decay cannot be used if the agent should continue to be able to adapt!) it can at least be used for real-time learning. It has some major conceptual limitations though, let's look at the most common form of RL, Behaviorism-based RL. It is about learning a state-action response mapping (a policy) with the highest expected rewards, without

any modeling of other causal relationships encountered in the environment. Such an agent has a reward-centric world view. This means that when the utility function (which generates the rewards) changes, the agent has to relearn a proper policy. Its knowledge won't transfer to a new task, by design. Whilst a changing utility function is not an issue in computer games with a single success criteria (lap time in racing games, checkmate in chess, etc.), for biological systems, it's an everyday reality. A hungry animal behaves fundamentally differently from a thirsty one. One will search for prey or delicious leaves, and the other one for sources of water. In this case the behavior is not only dependent on external factors, but also internal needs. Causal knowledge, picked up when certain needs are pursued, automatically transfers to later when other needs will be active. This makes such an agent react instantly to changing needs, by design. The solution is the decoupling of beliefs from goals. This allows the agent to learn the consequences of its actions in different contexts (causal modeling), which is agnostic to which consequences are currently in need of achievement. This approach is pursued by many in the AGI field which pursues AI's original goal, yet is not commonly pursued in the AI subfields whose focus lies in successes in single domains.

This brings us to the next issue: measurement. Measuring is key to determine progress, but what we measure matters. If we measure agent performance for each domain separately, and allow hyperparameters to be changed between domains, we will obtain special-purpose scores for different agents. This is useful from an application perspective but will tell us nothing about the generality of a single agent. If on the other hand hyperparameter changes aren't allowed between domains, the measures will reveal scores across the domains the agent was tried on. This can be summarized into a single general-purpose score. Currently, what you'll find is that the agent with the best general-purpose score often cannot compete with special-purpose agents (the ones whose hyperparameters were allowed to be adjusted for a

single intended domain). Agents with the best special-purpose scores will not reach a high general-purpose score. Nature faced similar tradeoffs (see Fig. 1: ants have a generic body well-adapted for a large variety of tasks, not so the bug which is optimized to look like a plant, not to be seen by predators), generality makes adaptation across exceptional environmental conditions easier, though for a specific stationary environment, a special solution is often preferred.

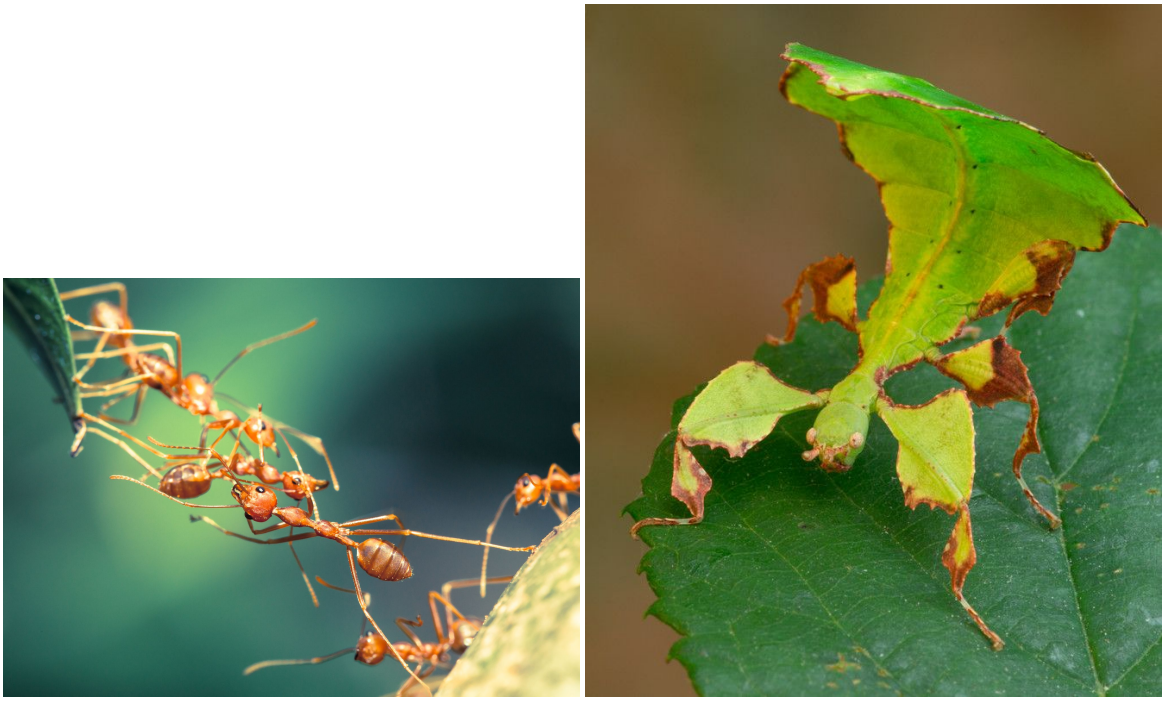


Figure 1.1: Specialization and Generalization

In this thesis we will especially address adaptation at runtime, with an attempt to measure general-purpose capability over a wide range of tasks without parameter adjustment / human in the loop.

1.2 Research question

Current intelligent systems, including autonomous agents in robotics, work well when equipped with the complete knowledge necessary to complete their mission. However, many solutions struggle to complete their mission if critical knowledge is

incomplete, or rendered invalid due to changing circumstances unforeseen by the designer. A reasoner which preserves key capabilities such as to deduce and plan ahead effectively using background knowledge, while adding learning at runtime into the picture, can account for these shortcomings and significantly extend the autonomy of intelligent agents.

This brings us to the key research question this thesis answers: whether and how ONA (OpenNARS for Applications), due to its real-time learning ability, is able to improve the autonomy of intelligent agents beyond the current state of the art. ONA, which has been developed for this thesis and with this goal in mind, is an implementation design of NARS (Non-Axiomatic Reasoning System) which in contrast to OpenNARS *Hammer et al.* (2016) which primary focus is to try research ideas, has practical application in mind. Its strength lies in combining stable aspects of NARS theory with a focus on agency. This entails the ability to sense, reason and act, in order to achieve desired outcomes, and an ability to learn from experience underway. A corresponding multi-threaded implementation has been written in C99, which was chosen to meet high-performance goals. The wanted capabilities of the system include:

- Learn from event streams in real time, without interruption.
- Extract sensorimotor contingencies from exploration and observation.
- Use them to plan ahead to reach desired outcomes.
- Work with limited memory.

While goal-directed decision making is well-understood and has several solutions in the literature, including Practical Reasoning approaches such as BDI models *Bratman et al.* (1987) *Georgeff et al.* (1998) with planning algorithm utilization, Practical Reasoners which can learn both incrementally and reliably in real-time have not been

achieved yet. Part of the issue is that reasoners are usually only expected to carry out deductive reasoning, while Induction is usually left to a separately invoked module which can't learn incrementally (Inductive Logic Programming *Muggleton* (1991)), can learn incrementally but misses a Decision Theory (Rule Mining *Hipp et al.* (2000), *Nath* (2012), *Srinivasan et al.* (2014)) or is restricted to offline-trained perception modules (supervised ConvNet, *Bochkovskiy et al.* (2020)). What adds to the issue is that most logics do either not include Inductive Reasoning (ignoring structure learning), or the resource consumption is unmanagable to learn from incoming events in real-time *Srinivasan et al.* (2014). 'OpenNARS for Applications', developed for this thesis, is a system capable of Non-Axiomatic Reasoning *Wang* (2013a), and addresses these concerns by design.

CHAPTER 2

OpenNARS for Applications

2.1 Theoretical Foundation

Since my research is aim at a general-purpose AI system which supports high degrees of autonomy, a proper foundation had to be chosen. Among the existing theories, the one I'm mostly agreed with is the one behind the Non-Axiomatic Reasoning System (NARS) model.

NARS builds on the core belief that intelligence is the principle of adapting to the environment while working with Insufficient Knowledge and Resources (AIKR, see *Wang* (2009)). This entails three key properties:

- **Finite:** Information-processing capacity is constant during the system's lifetime, both in terms of processor speed and storage capacity.
- **Real-Time:** Learn from events as they occur at runtime. React to changing goals and circumstances timely, often by dropping tasks when they have become less important.
- **Open**: Allow to digest input data and tasks of any content, while allowing all knowledge to be challenged by new evidence.

Since NARS is realized within a Reasoning System framework, it consists of three major components, all of which have been designed to work under AIKR:

1. **Logic** A logic allows to derive new knowledge from existing information. Here, NAL (Non-Axiomatic Logic) is utilized, a term logic with evidence-based truth

value which is designed to allow for reasoning under uncertainty. In NAL, the evidence for conclusions is calculated by the available evidence of the premises. The system's beliefs are a reflection of the best evidence the system has collected so far about a certain statement given the limited resources it operates under, which can be considered a form of bounded rationality. Also, different than in Symbolic systems (such as expert systems, *Lucas and Van Der Gaag* (1991)) where symbols refer to objects and relationships in the outside world as described by some fixed mathematical model (Model-theoretic semantics), a term's meaning is determined by the experienced relationships it participates in, leading to Experience-Grounded Semantics *Wang* (2005). Ambiguities in meaning, and contradictions in truth value are both captured by Non-Axiomatic Logic, while systems based on First-Order Predicate Logic break under these conditions despite being unavoidable when learning from experience.

2. **Memory** The memory is responsible for storing the input and derived knowledge of the reasoner in a way that is efficient to work with, and in a way that AIKR is respected. NARS uses a memory model where knowledge is grouped by term and linked to each other through component terms (subterm relation). This serves the purpose that no search needs to be performed in inference, neither when evidence is summarized (finding an existing belief to update), nor when a second premise has to be selected which matches to the first to derive new knowledge from. To keep memory in bounds, "high-quality" items are kept while "less-quality" ones are forgotten. What "quality" means is implementation design dependent, in this thesis we will see a simple yet effective implementation design based on combining use count and last-used time, reflecting the idea of "useful is what is in use". This sounds like a snake biting its own tail on first look, but it's not, due to the next point.

3. **Control** The control mechanism is responsible for deciding which premises to select for inference in each reasoning step. NARS uses a control mechanism which allows for real-time reasoning under attentional control. Per time unit, a constant amount of items is processed, where items of high priority are preferred to be selected while others tend to be ignored or will even be evicted if they don't fit into the attention buffer. While this is true for all NARS despite the huge design space for AIKR-compatible memory structures, how priority is measured is again implementation design dependent. The implementation design in this thesis decides conclusion priority based on the outcome truth/desire value and priority of the premises which decays to also capture recency. Additionally it penalizes high-complexity (terms which have many subterms) conclusions.

While I mostly agree with the objective of NARS and its logic, experiments on multistep examples revealed that the memory and control mechanism of OpenNARS is not suitable for practical application, which led to the creation of ONA. The details of Memory and Control will be described in the section dedicated to the specific architecture of 'OpenNARS for Applications'. This is, because many details depend on the implementation design, while the logic does not leave much room for variations other than for decisions on which inference rules to include. Memory and Control in intelligent reasoning systems in general, and NARS in particular, can be considered ongoing research, and ONA is just one solution with a focus for being practical for application purposes.

2.1.1 Non-Axiomatic Logic

Non-Axiomatic Logic (NAL, *Wang* (2013a)) is the term logic which allows NARS to reason under uncertainty. One of its distinguishing properties is that each sentence in this logic has a non-binary truth value. It is based on positive evidence w_+ and negative evidence w_- which speaks for or against a statement / belief / hypothesis,

and the total evidence $w := w_+ + w_-$, each of which is zero or greater. Based on these evidence values, the NAL truth value is defined as the tuple (f, c) with frequency

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

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 $w = 0$ don't need to be handled as they don't contribute any evidence.

Additionally, truth expectation is defined as

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

. This measure allows to summarize the two-valued truth value into a single value 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 $\lim_{w \mapsto \infty} c = 1$.

NAL is organized into 8 layers. To make it easier for the reader to interpret the examples included in the thesis, we will use the formal language Narsese here to introduce the logical copulas. Narsese serves both for I/O and internal representation of the reasoner. The following logical copulas are introduced in each layer:

- NAL-1: About Inheritance. Inheritance $< A \to B >$ indicates A to be a special case B. For instance, that cats are animals can be encoded as an Inheritance statement $(cat \to animal)$.

- NAL-2: About Similarity, Instances and Properties. Similarity is a bi-directional

version of Inheritance, $< A \leftrightarrow B >$, additionally instance $\{instance\}$ and property terms $[property]$ are introduced.

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

- NAL-4: Relational terms. This includes products $(a * b)$ to express anonymous relationships, as well as extensional and intensional images, $(R\,/1\,x)$ and $(R\backslash 1\,x)$ respectively. The index "1" allows to distinguish "eating a pizza" $(eat\,/1\,pizza)$ from "eaten by pizza" $(eat\,/2\,pizza)$, which clearly conveys a different meaning.

- NAL-5: Truth-relationships between statements. This includes negation $(\neg\, a)$, conjunction $(a \,\&\&\, b)$, disjunction $(a \,||\, b)$, implication $< a \Rightarrow b >$ and equivalence $< a \Leftrightarrow b >$.

- NAL-6: Variables: Dependent and Independent Variables $\$name$ and $\#name$ respectively which allow to make statements more abstract by allowing unification, in inference and question answering to substitute a term as long as the structure matches.

- NAL-7: Events: this layer introduces events which only have their truth value close to their occurrence time as specified by Projection formula which weakens their confidence with increasing distance to their occcurrence time $c^* \mapsto c * \lambda^{\Delta t}$. To obtain a time-independent generalization of the event statement, eternalization $c^* \mapsto \frac{c}{c+1}$ is used, a form of induction. To be able to compose events from existing events, this layer includes temporal variants of NAL-5 copulas, in-

cluding sequential conjunction ($a \,\&/\, b$), parallel conjunction, ($a \,\&|\, b$), sequential implication ($a =/> b$) and parallel implication ($a =|> b$). Here, sequential implication is taken as a correlative rather than causal relation. As argued in *Wang and Hammer* (2015b), besides causality not being a well-defined notion, causality is not something an agent can ever know for certain. Correlation is all there is to extract from experience, though some correlation involves events the agent can invoke itself, which brings us to the next layer.

- NAL-8: Procedure: Includes operator terms $\hat{op}$, and operations as inheritance statements with an operator as predicate: $((\{SELF\} * args) \rightarrow \hat{op})$ These events the system can produce by itself to achieve what it desires. Desire is associated to goal events, where like in *Wang* (2013a) the desire value of a goal $G!$ is formalized as the truth value of $< G \Rightarrow D >$ where D stands for an implicit / virtual "desired state". This way reasoning on goals is able to utilize the belief-related inference rules without needing special treatment.

Since terms can be arbitrary nested compound terms of these types, the formal language is highly expressive. NARS implementations have a parser implementation which follows the grammar definition of the Narsese language. The formal grammar definition can be found in the Wiki of the ONA repository, though it's not as easily readable and interpretable as the previous listing which is why the listing was preferred here. Also the implementation is using an ASCII representation and not the pretty-printed Latex representation, though the visual resemblance should be sufficient to understand without ambiguities the ASCII examples part of this thesis.

We will now focus on temporal and procedural aspects, which corresponds to NAL-7 and 8. These layers are the most crucial for procedure learning from event streams.

2.1.2 From statements to events, sequences and temporal credit assignment

Since statements are just a partial description of experience the Markov property is not an acceptable design assumption. And even though statements can in principle encode complete state for some limited practical purposes, the system is designed to work also in non-Markovian environments. To allow for this, the system can temporally relate two or more events with each other by forming compound events such as sequences. Additionally, Temporal Induction, an Induction rule, allows the system to create predictive statements based on two events. The induction rule basically states

$$\{A(f_1, c_1), B(f_2, c_2)\} \vdash (A \not\Rightarrow B)(f_1, \frac{f_2 * c_1 * c_2}{f_2 * c_1 * c_2 + 1})$$

where event A happened before B. The result can then be revised on each occurrence by summing up the positive and negative evidence, allowing stronger and stronger hypotheses to form. Here, in the simplest case, A and B are events, where usually both are Inheritance statements such as $(weather \rightarrow [rainy])$ and $(street \rightarrow [wet])$ (brackets denoting properties, see *Wang* (2013a)), but they could for example also be sequences. Before the induction rule is applied, the first event is projected to the latter, this lets confidence decrease with increasing temporal distance.

These temporal aspects of the logic provide a solution to what is known as the "Temporal Credit Assignment Problem". That is, for a given outcome event, what was the contribution of the events which happened before it? Temporal projection followed by induction and revision is key here. Another key is the ability to generate negative evidence for the hypothesis whenever it fails to predict (this we call "Anticipation"), which is again followed by Revision to summarize the evidence. As projection discounts evidence of the induction result based on the temporal occurrence time

distance between the premises, it plays a similar role for temporal credit assignment (in this case the attribution of evidence) as Eligibility Trace decay in Reinforcement Learning *Sutton and Barto* (2018) Differently to Reinforcement Learning, the Temporal Credit Assignment Problem is solved for all incoming events and compositions thereof, not just reward outcomes. This allows the system to learn from observation even when reward is absent and allows it to take a past events into account when predicting others, not just current ones.

Additionally, since operations are just a special case of events, the system can form sequences such as (A, Op) and integrate them into induced hypotheses such as $(A, Op) \Rightarrow C$ where Op is an operator. This is similar to the schemas discussed before, and allows to represent the idea, that if Op is executed when A happened, C will likely follow, a key to goal-related procedure learning and procedure execution. This modular representation is also similar in nature to schemas, we will refer to them as "procedural hypotheses" and "procedural links".

2.1.3 Memory and Control considerations

While the logic is extensively explored in *Wang* (2013a), there's a central question which details have not received as much focus: given that in a real-time reasoner the time will often not allow to exhaustively process the conclusion space, how to make sure that the right inferences will be drawn at the right time. Ideally only the most contextually relevant and goal-related inferences should be made. It's the job of Attention Allocation to make this work. The architecture we will introduce exploits recency of both used premises, truth expectation and Syntactic Complexity of the derived conclusion to decide the priority of conclusions, which then again can potentially become premises for the next inference step. Additionally, we will see a memory structure which supports efficient goal-reasoning and decision making by exploiting temporal structure (learned temporal and procedural hypotheses).

Also, new information can constantly stream into the system. In this case with data structures being finite, clearly most information will ultimately need to be forgotten to not exce. 31ed a fixed maximum capacity. This raises the second question, that is, which data item should be retained, and which should be removed to make place for a new one. For the attention buffer (a form of Short-Term Memory) the system's control mechanism selects from, it's a ranking of Priority, where lowest-priority items will be removed first. For the premises to be stored in the system for longer (Long-Term Memory, which we will call "Concept memory"), it will be a usefulness metrics based on use count and last used time.

Keeping these considerations in mind helps to better understand the architecture this thesis describes, which designates separate blocks to both memory types (a form of Short Term Working Memory and Long Term Memory). But before going there, we will dive deeper into the learning of temporal and procedural links and the associated challenges.

2.1.4 Representing and learning procedure knowledge

Procedural knowledge is generally concerned with representing the preconditions and consequences of actions. In traditional planning approaches *Russell and Norvig* (2002), this knowledge is usually provided in advance, and the task is then to search for the most concise and complete plan that leads to the achievement of a certain goal, a desired end state. This approach can be modified to search for the plan that leads to the end state with the highest probability. The drawback to this approach is that the need to react to change of circumstance during the planning and execution process, as well as forming the preconditions and transition probabilities from experience, is not captured.

In Reinforcement Learning (RL) this problem is reduced to learning to act the right way in the currently observed situation, where "act the right way" is usually

taken as the selection of the action with the maximum expected utility value *Sutton and Barto* (2018). Here, no explicit plan is generated, and no subgoal derivations happen. Instead the decision-making is only considering the currently observed state, whilst assuming it is a complete description of the current situation *Sutton and Barto* (2018). While being sufficient in applications where the system's behavior serves a single purpose, this treatment becomes insufficient when novel goals spontaneously appear or existing ones disappear *Wang and Hammer* (2015a). That's clearly the case in many robotics applications, *Latombe* (2012), and also, as many argue, in the human mind *Eagleman and Sejnowski* (2000); *Pitti et al.* (2013). To improve the ability to adapt to changing circumstances, a change of goals should not require re-learning the related situation-action mappings. Instead an adaptive system should develop beliefs of the consequences of its actions in such a way, that when goals change, the actions that lead to the fulfillment of novel goals can be derived. This can be seen as an understanding of the environment in terms of how different outcomes can be achieved by the system, independently from what is currently "rewarded" or desired.

We will now combine the benefits of both traditional planning and RL, while eliminating the mentioned drawbacks of both approaches. Since learning procedural implications is about learning the preconditions and consequences of operations, the system does not need to re-learn situation-action mappings when goals change, a major advantage. With these modular representations, each piece of procedural knowledge can be independently evaluated and suggests a certain operation in a certain context to realize a certain subgoal: solving "global" problems, using "local" decisions. So, different to traditional planning approaches, no complete plan is explicitly searched for, instead the individually evaluated pieces can be learned, combined and lead to a certain behavior, influenced by current goals and circumstances. We will now see both the general problem formulation and a solution with these properties.

2.1.5 Goal-directed Procedure Learning: the problem

We regard procedure learning as the process that forms procedural hypotheses, based on temporal patterns of events and operations that appear in the system's experience. A temporal pattern can be represented as $(A_1, ..., A_n)$, which is a sequence of n consecutive events. or operations, each occurring at a certain discrete time-step. Here, events do not encode an entire state, just certain parts of it, such as temperature information coming from a sensory device, encoded by a composition of terms/IDs, called "Compound Term" (see *Wang* (2013a)). Now, as shown before, temporal patterns can become building blocks of hypotheses, which should capture useful regularities in the experience of a system. A hypothesis can be an implication $A \Rightarrow B$, where a procedural hypothesis can be defined as $(A, B) \Rightarrow C$ where A is an antecedent, B an operation and C a consequent. The antecedent can be considered as a precondition that, when followed by an operation, is believed to lead to the consequent. Additionally, with the inclusion of temporal constraints, hypotheses can be considered as predictive, such as $(A, B) \not\Rightarrow C$, whereby they imply the occurrence time of the consequent to be in the future. Here, the precondition and consequent can be arbitrarily complex pattern compositions, while the behavior is an operation which can be directly invoked. Additionally, the consequent often represents a goal or subgoal to realize. Furthermore each such hypothesis has a degree of belief corresponding to the likelihood that the prediction will be confirmed when its precondition is fulfilled, which is measured by the truth expectation of its truth value.

Given a goal $G!$, which a system wants to make as true as possible, how does it satisfy the goal? *Wang* (2013b)

There are two essential components: hypotheses formation through forward chaining on beliefs, where helpful hypotheses are formed directly from pieces of knowledge, with observed patterns as special cases, and backward chaining, where a subgoal is derived from a goal and an existing belief *Wang* (2013b). For the latter, in case where

a hypothesis for how to achieve $G!$ in the current context already exists, usually a single backward chaining step to derive the operation as a subgoal, which can be directly executed, will be sufficient. Therefore, "executing a procedural hypothesis", means to perform a single backward inference leading to an operation, that can be executed. Often however, the goal $G!$ simply cannot be reached within a single step from given circumstances, so preconditions under which $G!$ can be reached need to be derived as a subgoal first, some of these subgoals might then be reachable from current circumstances or demand to produce subgoals by themselves.

Generally, problem-solving involves an inter-play between "reason" and "execute". That is, because when the system has no appropriate procedural hypothesis, that both predicts G and also has its preconditions currently met, then usually both belief- and goal-related multi-step reasoning will be necessary, to find a solution, leading to a schema which can later be re-used *Drescher* (1989). The reasoning corresponds to creative but evidence-driven exploration of possibilities to deal with novel situations. This corresponds to re-interpretation of the situation and finding of solutions although no algorithm is known, as discussed in *Wang* (2013b). The "execute" case on the other hand relies on the memorycontrol structure's ability to remember solutions to goals in such a way that they will be easily accessible when similar goals are desired and the system finds itself in similar circumstances. This allows for faster response times to similar problems in the future and to make effective use of what has already been learned.

We believe that most of the things we do, such as driving a car, that increasingly begin to become more automatic, are due to a transition from the "reason" to the "execute" case *DeKeyser et al.* (2007). A transition from the "novel" to the "usual" is something a systems control mechanism can and should effectively support. Furthermore, a control mechanism, will need to choose between multiple hypotheses which can satisfy a goal, to a different degree, for the current context. Therefore the

formation, selection and pruning of hypotheses are crucial and each of these aspects will be described in detail below.

2.1.6 Goal-directed Procedure Learning: the solution

Proper timing estimation Additionally to the temporal reasoning capabilities introduced before, when reasoning with time the notion of 'interval' becomes important as it allows the system to estimate expected arrival of predicted or desired conclusions. Temporal implications such as $(E_1, I_t) \not\Rightarrow E_2$ include intervals which measure the time distance between E_1 and E_2. Intervals are always measured when temporal implications are induced from events, though we will omit them in the discussion whenever they don't add any value. A key challenge is how to allow hypotheses with intervals of different duration to be revised. Imagine two hypotheses $(E_1, I_{t1}) \not\Rightarrow E_2$ and $(E_1, I_{t2}) \not\Rightarrow E_2$. Both of them predict the same outcome based on the same precondition, but they expect different interval durations, I_{t1} and I_{t2}. To allow these different intervals to be revised, a confidence decay (Projection)*Hammer et al.* (2016) increasing with the time difference is applied after revision, where the projection happens towards the confidence-weighted average of both intverals. This enables the learning of the more, commonly experienced, interval durations over time.

Hypothesis Creation Hypothesis creation is the key to building relevant and useful hypotheses. A crucial insight was to separate the incoming experience stream into belief events which are operations and belief events which are not. With this separation, the task of forming meaningful preconditions becomes an easier problem to solve: operations simply become the context under which certain events cause others to occur.

Over time the total evidence of re-occurring hypotheses will increase due to the Revision rule *Wang* (2013a) being applied within the concept of the hypothesis. Now, in a proper memory structure, revised procedural hypotheses $((A, Op) \not\Rightarrow B)$ are

indexed by B, this allows term B to memorize ways of how goal events with its term can be realized, and to learn the preconditions under which itself tends to occur after.

Hypothesis Selection Assuming an incoming or derived goal $G!$, the task of hypothesis selection is to choose the highest truth-expectation and contextually applicable hypothesis (with a previously experienced event E as precondition) that can satisfy $G!$. Also assuming that such a hypothesis already exists, the Detachment rule *Wang* (2013a) can be applied twice to a matching hypothesis, which is of the form $(E, Op_i) \Rightarrow G$. The first detachment leads to $(E, Op_i)!$ and the second one to $Op_i!$. The $Op_i!$ with the highest truth expectation, or certainty, will most likely lead to the greatest satisfaction of $G!$, and therefore should be derived. $Op_i!$ can then be executed if the truth expectation if above a decision threshold *Hammer et al.* (2016). When so hypothesis with Op_i above decision threshold exists, $E!$ can be deduced as a subgoal.

Hypothesis Pruning Given the uncertain nature of input experience, it is not possible, in advance, to identify what will be relevant and useful. This, unavoidable, lack of foresight can lead to the formation of hypotheses that have little value to a system. Additionally, given the limited computational resources, only a certain number of hypotheses can be retained at any one time. This makes it even more important to keep track of the success rate of hypotheses, as to keep the most competent ones while removing the others.

The approach taken to allow for this is hypothesis pruning, to measure the success of hypotheses so that these that do not predict correctly can be removed by lowering their truth expectation. While finding positive evidence is achieved through temporal induction rules as mentioned before, finding negative evidence is the job of Anticipation: given a predictive statement of the form: *antecedent* $\not\Rightarrow$ *consequent*, we define *Anticipation* as the expectation that the antecedent will lead to the consequent being observed as predicted. With Anticipation a system is able to find negative evidence

for previously learned predictive beliefs which generate wrong predictions. *Hammer and Lofthouse* (2018); *Nivel et al.* (2013)

If the event happens, in the sense that a new input event with the same term as the anticipated event is observed, the anticipation was successful (Confirmation), in which case nothing special needs to be done, since the statement will be confirmed via the normal process of temporal induction.

If the predicted event does not happen then the system needs to recognize this. But when is the right time to do so?

The challenge is how to determine the timeout duration after which we decide the prediction failed. While in previous publications *Hammer and Lofthouse* (2018) we have attempted to estimate a deadline, the new treatment avoids a deadline altogether: whenever a hypothesis is used, a small amount of negative evidence is added to it, small enough that positive observation will overvote it. This way, the hypothesis's truth expectation will increase when the event happens as expected, while it will decrease when it won't. This has turned out to be the most effective solution attempted so far.

The recognition of the Goal-Directed Procedure Learning problem, and its solution has become a core strength of ONA, and is only partly solved by the prior works we will now discuss.

2.2 Implementation design

2.2.1 Objective

The Non-Axiomatic Reasoning System has been implemented several times (*Ivanović et al.* (2017), *Lofthouse* (2019), *Hammer et al.* (2016)). *OpenNARS* was used both as a platform for new research topics and an implementation for applications *Hammer et al.* (2020), though it was mainly intended as a research platform. Not all ideas in

OpenNARS are fully developed, and application domains require the proven aspects to work reliably. Whilst this has led to the systems capabilities being stretched to the limits it has also given us a better understanding of the current limitations. The new architecture, *OpenNARS for Applications* (ONA), has been developed to resolve OpenNARS's limitations by combining the best results from our research projects. The logic and conceptual ideas of OpenNARS *Hammer et al.* (2016), the sensorimotor capabilities of ANSNA *Hammer* (2019) and the control model from ALANN *Lofthouse* (2019) are combined in a general purpose reasoner ready to be applied.

ONA is a NARS implementation following the NARS model of intelligent reasoning *Wang* (2013a). For a system to be classified as an instance of a NARS it needs to work under the Assumption of Insufficient Knowledge and Resource (AIKR). This means the system is always open to new tasks, works under finite resource constraints, and works in real time. For the resource constraints to be respected, each inference step (cycle) must take an approximately constant time $O(1)$, and forgetting is necessary to stay within memory limits. Here, relative forgetting describes the relative ranking of items for priority based selection (a form of attention), while absolute forgetting is a form of eviction of data items, to meet space constraints. Events, beliefs and concepts compete for resource based on current importance, relevance and long term usefulness.

What all NARS implementations have in common is the use of the Non-Axiomatic Logic (NAL) *Wang* (2013a), a term logic with evidence based truth values, which allows the systems to deal with uncertainty. Due to the compositional nature of NAL, these systems usually have a concept centric memory structure, which exploits subterm relationships for control purposes. A concept centric memory structure ensures premises in inference will be semantically related. This property, together with the priority based selection, helps to avoid combinatorial explosion. An additional commonality between NARS implementations is the usage of the formal language *Narsese*,

it allows the encoding and communication of NAL sentences with the system, as well as between systems.

2.2.2 Data Structures

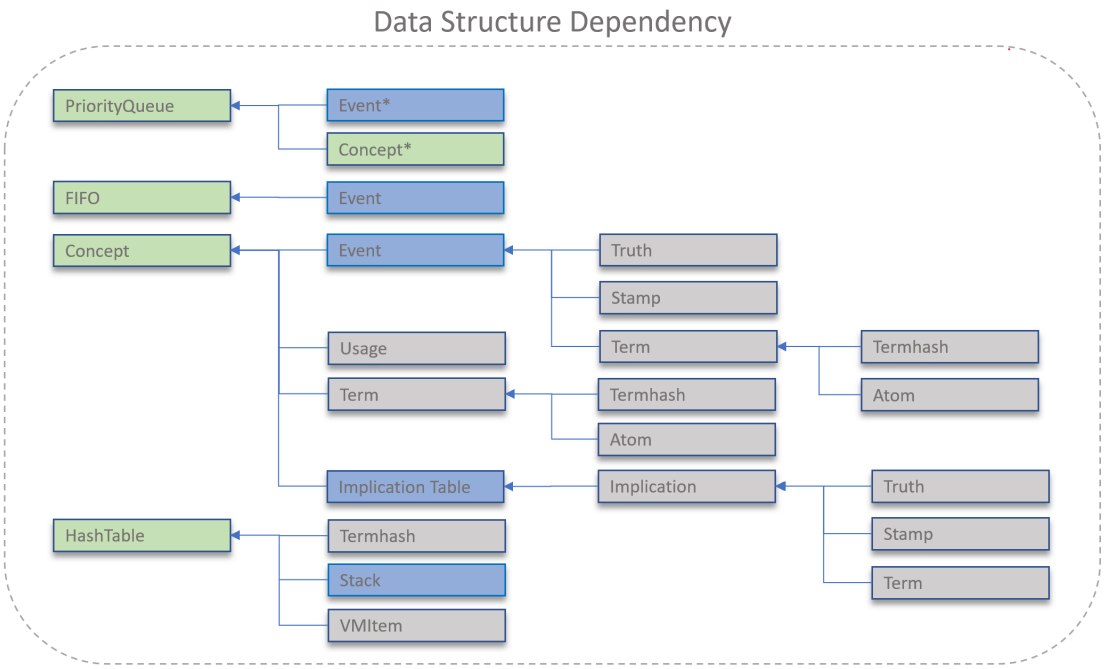


Figure 2.1: Data Structures utilized

Data structures (Fig. 2.1) can be grouped into two broad classes: Data and containers. The primary data elements are Events, Concepts, Implications and Terms; whilst the containers are FIFO, PriorityQueue, ImplicationTable and HashTable. HashTable is an optimisation and mentioned here for completeness but is not required for the functional description. It is used to efficiently retrieve a concept by its term (hash key) without searching through memory.

Term: All knowledge within the reasoner is represented as a term. Term structure is represented via a binary tree, where each node can either be a logical NAL copula or atomic. For the encoding, heap order is used, meaning the first entry is the root, and from there, a node at i's left child is at position $i * 2$ and the right child is at

position $i * 2 + 1$.

Event: Each Event consists of a term with a NAL Truth Value, a stamp (a set of IDs representing, the evidential base of any derivations or a single ID for new input), an Occurrence Time, and a priority value. The stamp is used to check for statistical independence of the premises, derivations are only allowed when there is no overlap between the stamps of the premises. Additionally, each event can either be a *judgment* (an event/piece of information) or a *goal* (an event desired to be reproduced). For input, also the *question* type is allowed, which is handled as memory lookup, simply returning the answer of highest truth expectation among the concepts matching the question.

Concept: Each concept has a term (its identifier), a priority value for attention control purposes, a *usage* value, indicating when the concept was last used and how often it was used since its creation. There is a table of pre-condition implications that act as predictive links, specifying which concepts predict which other's events. Here, each operation has its own section in the table, this way gaining more knowledge about one operators will not potentially flood out knowledge about other operators. Additionally, there is an eternal belief which summarizes all event truths which have the concept's term through revision. This eternal belief allows the system to learn, for instance, that ravens are black, from individual observations of black ravens at different points in time. Last, there is a most recent event belief reflecting the most recent event with the concept's term that happened, and a predicted event belief with the concept's term.

Implication: These are the contents of the pre-condition implication tables in the concepts. Usually its term has the form $a \Rightarrow b$ which stands for "a predicts b". Sometimes they also include an operation, such as $(a, op) \Rightarrow b$, which is the procedural form, and similar to schemas as in *Drescher* (1993), though their context is never modified. They allow the reasoner to predict outcomes (forward) and to

predict subgoals (backward). When the outcome b is predicted (with an operation execution as side effect for the procedural form), negative evidence is added to the prediction on failure, while on success positive evidence is added. The simplest way to accomplish this is to add the negative evidence right away while ensuring that the positive evidence added will outweigh the negative. In this way no anticipation deadline needs to be assumed and the truth expectation of the implication will gain truth expectation on success, and loose truth expectation on failure, anticipation realized via *Assumption of Failure*.

PriorityQueue: This is used by: Cycling Events Queue and Concepts Memory. It is a ranked, bounded priority queue which, when at capacity, removes the lowest ranked item when a new item is added. Events are ranked by priority, and concepts by usefulness, a $(lastUsed, useCount)$ which maps to raw usefulness via $usefulnessRaw = \frac{useCount}{recency+1}$, where $recency = currentTime - lastUsed$. A normalised value for usefulness is obtained with $usefulness = \frac{usefulnessRaw}{usefulnessRaw+1}$. The datastructure is realized as a max-heap, as this allows $O(1)$ access of the highest-priority item, and insertion in $O(log(n))$ time where n is the amount of items stored in the heap.

Implication Table and Revision: Implications are eternal beliefs of the form $a \Rightarrow b$, which essentially becomes a predictive link for a, which is added into an implication table (precondition implication table of b).

An implication table combines different implications, for instance $a \Rightarrow g$ and $b \Rightarrow g$ to describe the different preconditions which lead to g, stored in the implication table in concept g. Implication tables are ranked by the truth expectations of the beliefs. This makes sure that only the hypotheses which predict most successfully are kept in the table, while the memory bound is strictly respected.

2.2.3 Efficiency considerations

These include all design decisions which were made to make the system perform better, more easily adjustable, and so on.

- **Array-encoded terms** The binary tree corresponding to a Narsese sentence can be encoded in an initially empty array, by putting the root node as entry of position $i = 1$. And then, recursively, the left children is put at position $i*2$ and the right childen at position $i*2+1$. This representation, due to locality of the assigned memory region, is more efficient for CPU's than a representation which needs to follow pointers to traverse the tree. It needs more memory though, but memory nowadays is cheap.

- **Recursion-free unification** Since the terms are layed out in an array, Variable Unification can be implemented with a single traversal through the term arrays to unify. A recursive formulation of unification is more expensive as it depends on unnecessary stack manipulations for each recursive call and pointer indirections when terms are not stored in contiguous memory. While traversing, a hashmap of variable assignments is maintained, which lets the unification fail if an inconsistent assignment is made.

- **Pre-allocated memory** Since every data structure is bounded, all memory is allocated at process startup. This avoids expensive memory allocations and releases at runtime.

- **Contiguous memory allocation** Concepts are stored as contiguous memory blocks. This allows concept updates to utilize cache-locality effecively for higher performance in processing.

- **Max-heap based Priority Queue** The Priority Queue is implemented as a bounded Max-Heap structure. This datastructure allow for $O(1)$ access of the

highest-ranked element. Also insertion happens in $O(log(n))$ which allows for theoretically optimal $O(n*log(n))$ re-sort, with n being the amount of elements in the datastructure.

- **Multi-threading** Belief selection happens with a parallel OpenMP for-loop, allowing the inference to happen in parallel by multiple threads, with only memory updates being locked to store the results.

- **Inference rule code generation** Instead of interpreting inference rules at run-time or hardcoding them with code difficult to maintain, the Narsese parser is utilized to allow for the following inference rule format, established as C-macro: $(Premise1, Premise2, |-, Conclusion, TruthFunction)$. Inference rule code is automatically generated from this description, including complete unrolling of the unification. This allows modern compiler optimization to apply to make the rule matching for multiple rules more efficient.

2.2.4 Architecture

A key driver of the architectural change is the nature of how concept, task and belief are selected for inference. In OpenNARS the selection is based on a probabilistic choice from a data structure (Bag) and is concept centric *Rehling and Hofstadter* (1997). ONA takes a different approach: a task is popped from a bounded priority queue. The task determines the concept to be selected, through a one-to-one mapping between the term of the task, and concept terms. Then a subset of concepts which share a common term with the task are selected based on their priority (determined by a configuration parameter). This selection of concepts is the attentional focus as these are the concepts that will be involved in the inference cycle. Whilst the number of concepts to select is a fixed value (for a given configuration), the priority of concepts is constantly changing. A self-regulating threshold is used to maintain the priority

distribution within the necessary range to meet the selection criteria. This selection of concepts is the first stage of the inference cycle. The selected concepts are now tested for evidential overlap between the event and concept beliefs (evidence cannot be overlapping *Hammer et al.* (2016)). Finally, there is an 'inference pattern' match check, between the event and belief. If all the conditions are met the inference result is generated, and added to memory to form new concepts or to revise any pre-existing concept's belief. Then the event, or the revised one if revision occurred, is returned to the cycling events queue, with a reduced priority (if above minimum parameter thresholds).

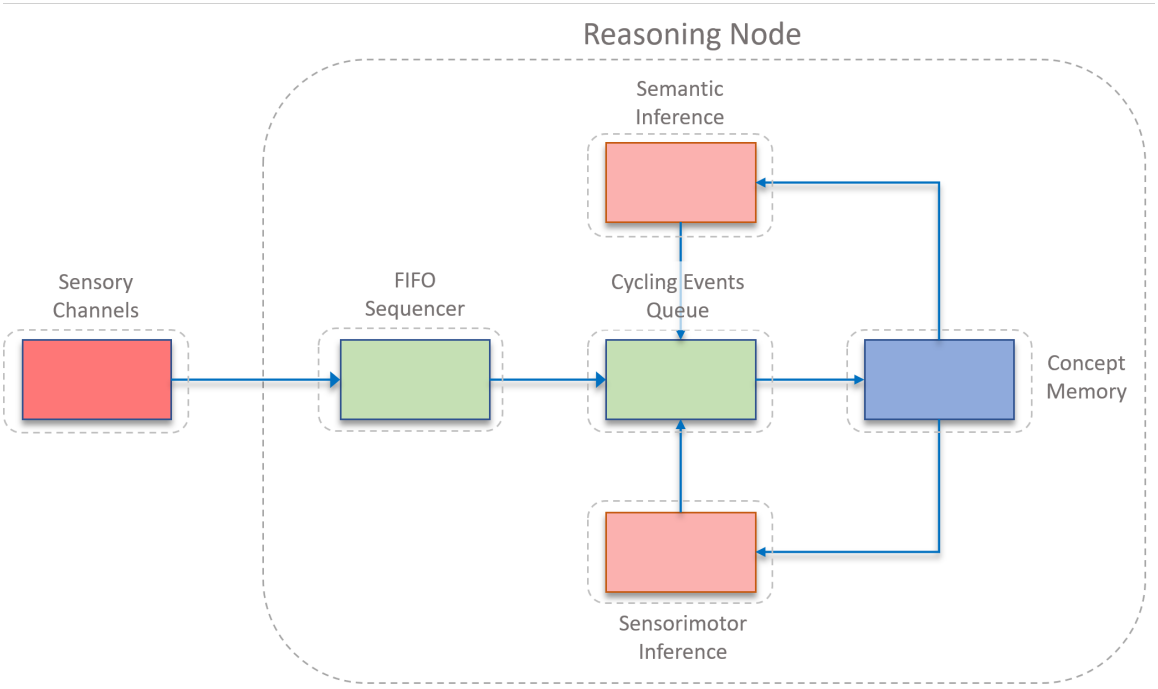


Figure 2.2: High-level architecture showing input sequencing and cycles for sensorimotor and semantic inference

Sensory Channels: The reasoner allows for sensory input from multiple modalities. Each sensory channel essentially converts sensory signals to Narsese. Dependent on the nature of the modality, its internals may vary. As an example for application purposes, a Vision Channel could consist of a Multi-Class Multi-Object Tracker for

the detection and tracking of instances and their type, and an encoder which converts the output into: the instances which were detected in the current moment, their type, visual properties, and spatial relationships among the instances *Hammer et al.* (2020). But also a simpler treatment is often sufficient, where only a detector is used and the detected class and location input into NARS. In this case, no explit instance-based distinction is made.

FIFO Sequencer: The Sequencer is responsible for multi-modal integration. It creates spatio-temporal patterns (compound events) from the events generated by the sensory channels. It achieves this by building both sequences and parallel conjunctions, dependent on their temporal order and distance. These compositions will then be usable by sensorimotor inference (after concepts for the sequence have been added to concept memory and the compound event added as belief event within the concept). As shown in figure 2.2, these compound events go through cycling events first, ideally to compete for attention with derived events to be added to memory. The resource allocation between input and derivations is a difficult balance, for now, we let input events and the compound events (from FIFO sequencer) be passed to memory before derivations. We acknowledge that this simple solution might not be the final story.

Cycling Events queue: This is the global attention buffer of the reasoner. It maintains a fixed capacity: items are ranked according to priority, and when a new item enters, the lowest priority item is evicted. For selection, the highest-priority items are retrieved, both for semantic and sensorimotor inference, the retrieved items and the inference results then go back into the cycling events queue after the corresponding inference block. The item's priority decays on usage, but also decays in the queue, both decay rates are global parameters.

Sensorimotor Inference: This is where temporal and procedural reasoning occurs, using NAL layers 6-8. The responsibilities here include: Formation and strength-

ening of implication links between concepts, driven both by input sequences and derived events. Prediction of new events based on input and derived events, via implication links. Efficient subgoaling via implication links and decision execution when an operation subgoal exceeds decision threshold *Hammer and Lofthouse* (2018).

The truth functions, some of which are also used by semantic inference as summarized in Table A.1 in the appendix, and so are the inference rules.

Semantic Inference: All declarative reasoning using NAL layers 1-6 occurs here as described in *Wang* (2013a), meaning no temporal and procedural aspects are processed here. As inheritance can be seen as a way to describe objects in a universe of discourse *Wang* (2006), the related inference helps the reasoner to categorize events, and to refine these categorizations with further experience. Ultimately this allows the reasoner to learn and use arbitrary relations, to interpret situations in richer ways and find crucial commonalities and differences between various knowledge. Also, due to the descriptive power of NAL and its experience-grounded semantics, semi-natural communication with the reasoner becomes possible, and high-level knowledge can be directly communicated. This also works when the meaning of some terms is not yet clear and needs to be enriched to become useful.

The truth functions which are only used by semantic inference rules are summarized in Table A.2 in the appendix, and also the inference rules.

Concept Memory: The concept store of the reasoner. Similar to the cycling events queue, it maintains a fixed capacity: but instead of being ranked by priority, items are ranked according to usefulness, and when a new item enters, the lowest useful item is evicted. Usefulness takes both the usage count and last usage time into account, to both, capture the long term quality of the item, and to give new items a chance. All events from the cycling events queue, both input and derived, that weren't evicted from the queue, arrive here. A concept node is created for each event's term, or activates it with the event priority if it already exists. Now revision

of knowledge, of the contained beliefs, takes place. It also holds the implications which were formed by the sensorimotor component, which manifest as implication links between concepts. The activation of concepts allows the reasoner's inference to be contextual: only beliefs of the highest priority concepts, which share a common term with the event selected from the Cycling Events queue (for Semantic Inference), or are temporally related (through an implication link or in temporal proximity, for Sensorimotor Inference), will be retrieved for inference.

2.2.5 Operating cycle

The operating cycle of the reasoner makes use of the following attentional control functions for resource management, these are crucial to make sure the reasoner works on contextually relevant information.

- *Forget event:* Forget an event using monotonic decay. This happens in the cycling events queue, where the decay after selection can differ from the decay applied over time, dependent on the corresponding event durability system parameters. (multiplied with the priority to obtain the new one)

- *Forget concept:* Decay the priority of a concept monotonically over time, by multiplying with a global concept durability parameter.

- *Activate concept:* Activate a concept when an event is matched to it in Concept Memory, proportional to the priority of the event (currently simply setting concept priority to the matched event's when its priority exceeds the concept's). The idea here is that events can activate concepts while the concept's priority leaks over time, so that active concepts tend to be currently contextually relevant ones (temporally and semantically). Additionally, the usage counter of the concept gets increased, and the last used parameter set to the current time, which increases the usefulness of the concept.

- *Derive event:* The inference results produced (either in Semantic Inference or Sensorimotor Inference), will be assigned a priority, the product of: belief concept priority or truth expectation in case of an implication link (context), Truth expectation of the conclusion (summarized evidence), Priority of the event which triggered the inference, and $\frac{1}{log_2(1+c)}$ where c is the syntactic Complexity of the result. (the amount of nodes of the binary tree which represents the conclusion term)

  The multiplication with the parent event priority causes the child event to have a lower priority than its parent. Now from the the fact that event durability is smaller than 1, it follows that the cycling events queue elements will converge to 0 in priority over time when no new input is given. This, together with the same kind of decay for concept priority, guarantees that the system will always recover from its attentional states and be ready to work on new input effectively after busy times.

- *Input event:* The priority of input events is simply set to 1, it will decay via relative forgetting as described.

The following overview describes each component of the main operating cycle, in which the attentional control functions are utilized:

1. Retrieve EVENT_SELECTIONS events from cycling events priority queue (which includes both input and derivations)

2. Process incoming belief events from FIFO, building implications utilizing input sequences and selected events (from step 1)

3. Process incoming goal events from FIFO, propagating subgoals according to implications, triggering decisions when above decision threshold

4. Perform inference between selected events and semantically/temporally related, high-priority concepts to derive and process new events

5. Apply relative forgetting for concepts according to CONCEPT_DURABILITY and events according to EVENT_DURABILITY

6. Push selected events (from step 1) back to the queue as well, applying relative forgetting based on EVENT_DURABILITY_ON_USAGE

Semantic Inference: After an event has been taken out of cycling events queue, high-priority concepts which either share a common subterm or hold a temporal link from the selected event's concept to itself will be chosen for inference. This is controlled by adapting a dynamic threshold which tries to keep the amount of selected belief concepts as close as possible to a system parameter. The selected event will then be taken as the first premise, and the concept's belief as the second premise. Here the concept's predicted or event belief is used when it's within a specified temporal window relative to the selected event, otherwise its eternal belief. The NAL inference rules then derive new events to be added to cycling events queue, which will then be passed on to concept memory to form new concepts and beliefs within concepts of same term.

Implication Link formation (Sensorimotor inference): Sequences suggested by the FIFO form concepts and implications. For instance event a followed by event b, will create a sequence (a, b), but the sensorimotor inference block will also make sure that an implication like $a \Rightarrow b$ will be created which will go into memory to form a link between the corresponding concepts. In addition, a second link $a \Rightarrow b$ is derived, but with all subterms which appear extensionally more than once (extensionally meaning being the subject of an inheritance subterm) being replaced with a variable. And a third candidate where the same happens intensionally (meaning being the predicate of an inheritance subterm). This variable introduction strategy is quite general but also

has its limits, as some implications cannot be induced this way. However, generating all possible candidates is not a good solution either, controlling induction remains a challenge.

In all cases, a itself can be a sequence coming from the FIFO sequencer, or a derived event from the cycling events queue which can help to predict b in the future. Also if $a \Rightarrow b$ exists as link and a was observed, assumption of failure will be applied to the link for implicit anticipation: if the anticipation fails, the truth expectation of the link will be reduced by the addition of negative evidence (via an implicit negative b event), while the truth expectation will increase due to the positive evidence in case of success. To solve the Temporal Credit Assignment problem such that delayed rewards can be dealt with, Eligibility Traces have been introduced in Reinforcement Learning (see *Sutton* (1988) and *Sutton and Barto* (2018)). The idea is to mark the parameters associated with the event and action which was taken as eligible for being changed, where the eligibility can accumulate and the eligibility decays over time. Only eligable state-action pairs will undergo high changes in utility dependent on the received reward. NARS realizes the same idea via projection and revision: when a conclusion is derived from two events, the first event will be penalized in truth value dependent on the temporal distance to the second event, via Projection. If both events have the same term, they will revise with each other forming a stronger event of same content, capturing the accumulation aspect of the eligibility trace. If they are different, the implication $a \Rightarrow b$ can be derived as mentioned before, and if this implication already exists, it will now revise with the old one, adding the new evidence to the existing evidence to form a conclusion of higher confidence. If b is a negative event, the truth expectation will decrease (higher confidence but less frequency), while a positive observation b will increase it. This is similar to the utility update in RL, except with one major difference: the learning rate is not given by the designer, but determined by the amount of evidence captured so far. In RL implementations this

deficit is compensated by decreasing the learning rate over time with the right speed (by trial and error carried out by the designer). However given amount of additional time is not a guarantee that more evidence will be collected for a specific state-action entry, its state might simply not have re-appeared within the time window, yet the next time it's encountered the learning rate for its adjustment will be lower, leading to inexact credit assignment.

Subgoaling and Decision (Sensorimotor inference): When a goal event enters memory, it triggers a form of sensorimotor inference: subgoaling and decision. The method to decide between these two is: the event concept precondition implication links are checked. If the link is strong enough, and there is a recent event in the precondition concept (Event a of its concept when $(a, op) \Rightarrow g$ is the implication), it will generate a high desire value for the reasoner to execute op. The truth expectations of the incoming link desire values are compared, and the operation from the link with the highest truth expectation will be executed if over a decision threshold. If not, all the preconditions (such as a) of the incoming links will be derived as subgoals, competing for attention and processing in the cycling events queue. This can be summarized as follows:

```
Input: Goal g Result: Execution of Op with feedback, or subgoal derivation
i=j=0
 Subgoals = {}
 bestDesire = 0.0
 bestOp = None
 implications = concept(g).preconditionImplications
 while i<implications.size do
    (subject, copula, predicate) = implications_i
     seqOpGoal = Deduction(implication_i, g)
     seqGoal = StructuralDeduction_Arg1Detachment(seqOpGoal)
     precondition = projected(concept(seqGoal).belief_event, currentTime)
     if precondition ≠ NULL then
        Subgoals = Subgoals ∪{seqGoal}
         Op = Deduction_Arg2Detachment(seqOpGoal, precondition)
         if Op.truth.expectation > bestDesire then
            bestDesire = Op.truth.expectation
             bestOp = Op
        end
    end
    i=i+1
end
(babbled, babbleOp) = Motorbabbling(implications)
 if babbled then
    Execute(babbleOp)
     inputAsNewBeliefEvent(babbleOp)
     return
end
if bestDesire > DECISION_THRESHOLD then
    Execute(bestOp)
     inputAsNewBeliefEvent(bestOp)
else
    while j < n_Subgoals do
        derive(Subgoals_j)
         j+=1
    end
end
```

Algorithm 1: Decision and subgoaling

Prediction Also, event a leads to the prediction of b via Deduction, assuming $a \Rightarrow b$ exists as implication in concept b. Here, it suffices for the event to be able to unify with the precondition of the implication, it doesn't have to be the same term.

Motor Babbling and Curiosity: The system is not only motivated by user

goals and goals related to bodily/device function. During its operation, it is expected to be able to explore its environment in such a way, that potentially useful knowledge will be found. This way, whenever the knowledge the system possesses, is, or has become, insufficient to complete some of its tasks, the system can still succeed. Clearly in the beginnings this is necessary as nothing about the environment might yet be known. But also later the search for better solutions demands the agent to learn more about the environment. In the beginnings, to trigger executions when no procedure knowledge yet exists, the reasoner periodically invokes random motor operations, a process called Motor Babbling. Without these initial operations, the reasoner would be unable to form correlations between action and consequence, effectively making procedure learning from experience impossible *Nivel et al.* (2013), *Hammer* (2019) and *Hammer et al.* (2016). Once a certain level of capability has been reached (sufficient confidence of a procedural implication $(a, op) \Rightarrow g$), the motor babbling is disabled for op in context a to allow for more competent behavior. Additionally, rather than just sampling uniformly among the operations, a sampling which prefers operations which consequences in current context have low confidence encourages the agent to explore more systematically what it does not yet know. This can be seen as a form of artificial curiosity which improves the exploration behavior of the system. They way this is realized is to make the function $Motorbabbling$ return a random operation, with a higher chance to select a candidate attached to an implication which has high precondition truth expectation (it contextually applies) and low confidence (it is uncertain).

2.3 Reaching desired capabilities via a pragmatic approach

2.3.1 The big picture

Here, we will discuss considerations about cognitive functioning of this system which ultimately give raise to the ability to act autonomously and in real-time, for agents and robots in particular. Usage in robots is especially a goal for this system, as it's obviously the only way for an AI-driven machine to directly operate in the real-world. This is aligned with the author's goal to show that NARS can be well-applied in environments where practically unlimited task-related training data and/or a perfect simulation isn't available, such as in many real-world scenarios. As mentioned, such circumstances are a large part of the reason why great recent AI successes in domains with perfect simulation availability (such as *Schrittwieser et al.* (2020), *Mnih et al.* (2013)) have not translated into successes in real-world domains where unexpected situations occur more regularly than tolerable. But most importantly, environments do change, and often do change fundamentally. In this case the assumption that the system receives new data from the same data-generating process is not even a reasonable assumption to make. Regular concept drift eliminates techniques which cannot deal with non-stationary environments (these which can only adapt to a single, static, probability distribution). Also it eliminates systems which need to update a single model describing all relevant aspects of the environment, as they cannot properly capture when only specific aspects of the environment change, while other aspects might remain the same. To deal with this, compositional representations are crucial, such as schemas which will be introduced in this chapter. Also crucial however is the ability to deal with the flood of high-dimensional input rich visual, auditory, ranging etc. sensors provide. In this thesis a practical approach is taken, where objects are first detected using state-of-the-art Deep Learning models, and then related spatially and temporally by the reasoning machinery we will introduce.

2.3.2 Visual perception

While just a special case of many possible modalities, visual perception is often crucial for certain tasks to be performed in real-world environments, especially for robotic explorers, rescue robots, robotic housemaids etc. For biological systems this is equally crucial, clearly most birds and mammals utilize sight to get around in their environment, to hunt for prey, or find other sources of food or water, etc. This fact is part of what drove research in artificial vision systems initially, and led to a task that was not nearly as easy to solve as Marvin Minsky wanted it to be. It seems fair to claim that it took many decades of progress in Machine Learning and hardware development till Deep Convolutional Neural Networks were found and practical variants became widespread in use.

Hence the approach taken here is combining vision, ranging, and tracking approaches, and the overall solution can be seen as a form of 3D perception. Such a 3D visual perception system needs to be able to identify instances of specific object classes in the world, relevant visual properties, and how these objects are spatially related and move relative to each other in 3-dimensional space, and relative to the agent. Additionally, the time required to identify objects in the scene needs to be within a small constant time, to not get in the way of real-time operating ability of the overall system. Here it's essentially crucial to make sure it's easy to utilize state-of-the-art object detectors which are fast in execution time (such as *Bochkovskiy et al.* (2020)), and real-time trackers which build on the detectors (such as *Bewley et al.* (2016) and newer). For object detection, a Convolutional Neural Network (YOLOv4), together with a ranging sensor to give distance information, was utilized for all applications which demand 3D perception, and the related examples we will see in this thesis. The YOLOv4 model represents the state-of-the-art of object detection as of April 2021, and is widely used in industry.

Additionally, this pragmatic approach is still compatible with a form of crude but

active vision: embedded in a robotic device, the system still decides what to see based on motor output, can use attention control to focus on specific objects, and can use visual cues to navigate and to gain information which is yet unknown. This allows the agent to seek for new information which will potentially help it to fulfill a mission goal, or to answer questions about objects which are not currently in sight, both will be demonstrated.

However, this pragmatic approach to perception, to utilize the best available Deep Learning models also comes with restrictions: it inherits the key limitation that it cannot learn to detect new types of objects at runtime. However, despite Unsupervised Learning and more recently Self-Supervised Learning having become popular recently, to my knowledge there are no robust solutions which allow to showcase this capability in real-world scenarios, so this property remains a wish also this thesis has no solution for. Desired properties such an adaptive, active visual perception system would ideally have have been summarized in *Wang and Hammer* (2018), but how exactly to realize such a system in a practical form has yet to be seen. Until then, relying on state-of-the-art Deep Learning solutions is arguably the most practical decision one can make, and is, as we will see, totally compatible with having an adaptive general-purpose reasoner on top rather than just merely application-specific hardcoding like seen in current non-adaptive autonomous driving solutions.

2.3.3 Adaptation

As just mentioned, ideally the system could learn to recognize new objects and their properties at operating time (we will generally refer to "adaptation" as learning at operating time in this thesis), but for now we have to take into account that this is outside of what current technology can realistically deliver. But while the perception system is not adaptive and the detected object types and properties are assumed to be fixed for the purpose of this thesis, ONA, as we will be shown, can build cate-

gories on top by combining and relating detected object properties with each other. This form of learning operates on top of the detection of the individual objects which are identified. Additionally, associative learning can happen by associating detected objects and properties to other perceived events which are not necessarily events originating from the visual perception system, then the association becomes multi-modal. Multi-modal patterns are not only crucial for Natural Language Understanding to allow agents to relate words and phrases to ongoing events (such as seeing a person approaching the fridge after hearing the word "hungry"), they are even more crucial for autonomous robots with different sensor types attached to it: for instance a rescue robot can adjust its search direction when hearing a person scream from a certain side, where the connection between scream and person either happened from past experience, or in this case most likely through background knowledge the rescue robot was provided with, before it started its mission.

A different form of Adaptation falls under Operant Conditioning, *Staddon and Cerutti* (2003) which is about the learning of behaviors which ideally lead the system to get what it currently desires, an essential part of the ONA architecture is concerned about solving this problem in particular. Interestingly, solutions which address what Operant Conditioning describes are relatively rare, and so far mostly unsuccessful: they are either proposed and not implemented, or implemented and not effective. Methods in Reinforcement Learning *Sutton and Barto* (2018), the arguably most overlapping paradigm in AI provides no difference here: most of it can't deal with multiple or contradicting objectives, or even handle the simple requirement of changing objectives. The latter isn't an issue in board games but an every-day reality for biological systems which behavior depends heavily on current demands. Most importantly, the most fascinating property of Operant Conditioning as observed in various birds and mammals is data efficiency. Often a single key observation for an animal is enough to learn the crucial knowledge which drive radical changes in the be-

haviors they exhibit. It's not unreasonable to suspect that some form of uncertainty logic is what allows for this kind of capability, leading to logics such as Probabilistic Logic Networks *Goertzel et al.* (2008) or Non-Axiomatic Logic (NAL, *Wang* (2013a)) which are more data efficient than approaches which don't track the sample space size and instead assume a sufficiently large sample space to work.

In ONA, all forms of adaptation are carried out via reasoning with NAL *Wang* (2013a) introduced in the NARS chapter. This is different than in Inductive Logic Programming, where facts are given prior to training, and induction is performed on the basis of facts via various rule extraction techniques. (*Muggleton* (1991) and *Muggleton and De Raedt* (1994)) Once a rule set is extracted, the output rules lack the statistical information / the evidence which was inherent to build them, hence if the fact basis changes, the entire algorithm has to run again.

Highly related are association rule miners which work on a data stream in real-time, especially the ones which work with a sliding windows approach. While these solutions usually don't introduce variables (as the events usually don't have internal structure like being predicates with arguments) and lack inference facilities necessary for prediction and decision making, they face a similar knowledge extraction problem from input events as ONA addresses also with a sliding window approach, a FIFO structure.

2.3.4 Planning

A large part of intelligent behavior depends on the ability to plan ahead to reach desired outcomes, outcomes which sometimes might not have been experienced before and might not be approached ever again exactly in the same manner. Planning relies on the feasibility of finding a plan based on the chaining of believed-to-be causal relationships. The outcome, the plan, is a series of steps which an agent needs to perform in succession to get from the current situation to the desired one. Since time

passes by during plan execution, and things might go wrong on the way, especially agents operating in the real world (robots) usually depend on periodic re-planning to allow the agent to take into account changes which are not under its control. If such a system should allow for real-time response additionally, then the planning time spent usually needs to be within a certain deadline. However, when the planning space is too large, exhaustive processing of alternatives is no longer an option. In this case certain paths need to have priority over others, leading to planning techniques such as Monte Carlo Tree Search *Chaslot et al.* (2008) which allow to explore more promising paths with higher probability, which can also be seen as a form of Attention Allocation as it redirects the computational resources the system has available.

A related problem is that the results of planning processes would ideally be memorized, so that next time a similar planning problem is encountered, large parts of the solution will already be known, reducing planning time significantly. In cognitive architectures such as Soar this is usually referred to as a form of "Chunking" (*Laird* (2019) and *Laird et al.* (1986)). To support this, a plan cannot just be judged according to its overall success, but individual pieces the plan is composed of need to be judged by whether they were able to produce associated intermediate outcomes, allowing for a form of temporal credit assignment. Additionally to the procedural implications we have seen, we will now get to a similar kind of modular representations which allow for such chunking to happen, Drescher's Schemas.

2.4 Implementation usage

To download the software from *Hammer* (2020a), all that is required is Git, and Clang or GCC. Any POSIX-compatible OS suffices.

To get started, download:

```
git clone https://github.com/opennars/OpenNARS-for-Applications
```

```
cd OpenNARS-for-Applications
sh build.sh
```

Then to run the tests which confirm the system to work, execute:

```
./NAR (C Tests only)
python3 evaluation.py (all tests)
```

Finally, the Shell can be executed to interact with the system in an interactive manner:

```
./NAR shell
```

The shell waits for input, which can either be a Narsese events such as

```
<world --> [round]>. :|:
<world --> [?what]>? :|:
```

or just inference steps, indicated by a number and enter, or only enter. The shell also supports directly streaming example files in, and piping it to syntax highlighting for prettier output is supported as well, as shown by Fig. 2.3. Hereby, green lines show input, yellow are derivations (omitted for length reasons), red ones are answers and operator executions, and pink ones are expected outputs for testing purposes which have been calculated by hand.

Figure 2.3: Shell interaction with syntax highlighting

Also several program examples are included:

1. Two versions of Pong: Reinforcement Learning-style behavior learning
2. Cartpole: Reinforcement Learning-style behavior learning
3. Space Invaders: Reinforcement Learning-style behavior learning
4. Test Chamber: demonstrates abilities to learn goal-dependent behaviors and to utilize them for planning purposes
5. Robot: combines all the abilities, includes real-time multi-step decision making
6. Various Narsese example files which demonstrate different reasoning capabilities
7. Various English files for the included English-to-Narsese channel

and scripts for conveniences such as english_to_narsese.py which translate English input to Narsese. This script is based on a pipeline of tokenization, Part-of-Speech-tagging, Wordnet-based lemmatization, and relationship extraction with a preliminary implication-based grammar learning attempt. Then, there is vision_to_narsese.py which uses YOLOv4 to stream in vision input (class ID and discretized X and Y location) into ONA. Both are preliminary sensor channels implementations which make it easier to utilize the system when vision and natural language input is required. These can also be chained, and attached to the system via simple pipe in the shell (including syntax highlighting output via colorize.py):

```
python3 english_to_narsese.py | python3 vision_to_narsese.py |
./NAR shell | python3 colorize.py
```

Last, there is a Python script available which makes it easy to use the system in a Python application. It can be imported with

```
import NAR
```

and input events can be fed like this:

```
NAR.AddInput("""<(<{switch1} --> [off]> &/
                  <({SELF} * {switch1}) --> ^activate>)
                         =/> <{switch1} --> [on]>>.""")
NAR.AddInput("<{switch0} --> [on]>. :|:")
NAR.AddInput("<{switch1} --> [off]>. :|:")
```

Questions are returned with the input question:

```
print(NAR.AddInput("<{switch0} --> [?1]>? :|:")["answers"])
```

The by ONA returned output is represented as a Python dictionary by the wrapper, which makes further processing for various application purposes easier:

```
[{'occurrenceTime': '2', 'punctuation': '.',
'term': '<{switch0} --> [on]>',
'truth': {'frequency': '1.000000,', 'confidence': '0.900000'}}]
```

Similarly, operation executions are returned with the input goal (sometimes a few calls later if the decision takes multiple inference steps):

```
print(NAR.AddInput("<{switch1} --> [on]>! :|:")["executions"])
```

which prints:

```
[{'operator': '^activate', 'arguments': '{switch1}'}]
```

In addition to the Shell and Python API, there is also a way to invoke ONA directly in a C/C++ project. This is outside of the scope of this chapter which is only meant to give a brief introduction to the ONA software. More information can be found in the project Wiki *Hammer* (2020a).

CHAPTER 3

Comparison with prior work

3.1 Drescher's schemas

One way to achieve this modularity are Jean Piaget's schemas, which he described as "a cohesive, repeatable action sequence possessing component actions that are tightly interconnected and governed by a core meaning". Schemas, according to Piaget, are the basic building block of intelligent behavior. They can be seen as a way to organize procedure knowledge. Drescher, in his publication *Drescher* (1986) and *Drescher* (1993), formalized schemas and visualized them as shown in Fig. 3.1:

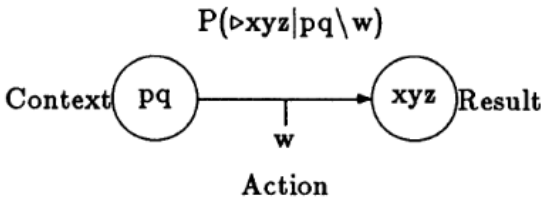


Figure 3.1: Drescher schema

His formalization describes schemas as a transitition from context to result via an action. Hereby, the transition probability is a tuple with the probability of the result happening with the action being taken, and probability of the result happening with the action not being taken. This allows to evaluate, how much the taking of the action actually contributes to making the outcome happen. To illustrate, let's consider both probabilities to be the same. This means the outcome will happen in the circumstance no matter if the action is taken or not, in which there is no merit in taking it. If however, the chance to achieve the desired outcome is higher when the

action is invoked, it should be taken. Among the competing schemas, the schemas which most likely will lead to the desired outcome are chosen and their associated actions executed, which also demands the schema's context to be fulfilled.

However, to achieve an outcome, in most cases multiple steps are necessary to get there. To address this, schemas can be chained, where chaining happens from a currently satisfied context to a goal, which corresponds to a form of planning outcome, again as visualized in *Drescher* (1986) by Fig. 3.2:

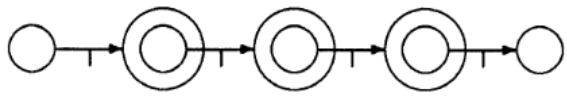


Figure 3.2: Schema chain

Differently than what we will see in NARS, schemas are not immutable and bounded in size, but can be extended indefinitely. This can raise computational demands, breaking real-time operating constraints if allowed to happen in an unbounded way. Fig. 3.3 illustrates the schema expansion process *Drescher* (1986):

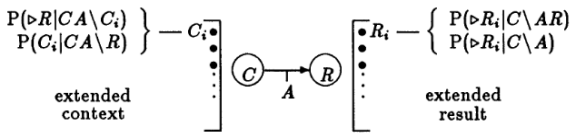


Figure 3.3: Schema extension

The key criteria for extending the schema is to increase the "predictive strength" of the schema when the additional contextual cue is added. This usually happens when the previous schema context was not specific enough to allow for similarly reliable prediction. To measure this reliability, Drescher explicitly introduced "reliance" to replace probability, as he noticed that the probabilistic approach does not scale well computationally, as whenever the probability of a schema is adjusted, all other schemas need their probabilities to be adjusted as well to not violate probability space

axioms (the sum of probabilities equals 1 requirement). Hence he defined *reliance* as $\frac{successes}{sucesses+failures}$, a measure which he proposed to be used instead of probability.

More importantly though, Drescher noticed a key issue with making the decisions dependent on *reliance* or probability alone. If a schema wasn't tested often enough yet, usually meaning successes and failures are both still small, the probability estimate will not be proper and consequently decisions will not lead to the desired outcomes. To fix this, he proposed $successes + failures$ to be large enough for a schema to be considered in decision making. However, whatever the threshold might be, there will be cases where the choice will turn out to be suboptimal. From a Machine Learning perspective it's safe to make the choice larger if this becomes an issue, as data efficiency / learning speed can often be ignored there. This is especially true in large datasets and in simulations where arbitrary amounts of data can be generated. From an autonomous system operating in the real world perspective however this is unacceptable and will hamper the solution to adapt timely to changing circumstances. Drescher didn't go deep enough to resolve this fundamental issue, and we saw how NARS addresses this via a 2-valued truth value.

Their basic idea is however valid, and to demonstrate their feasibility, we will look at the success stories of schemas as proposed so far, including data-hungry variants with 1-valued reliability measures. In the ICML 2017 paper, *Kansky et al.* (2017), schema networks were proposed, which essentially are entity-related schemas which can span multiple frames (see 3.4) learned through a modern optimization technique, Integer Linear Programming (see *Schrijver* (1998)).

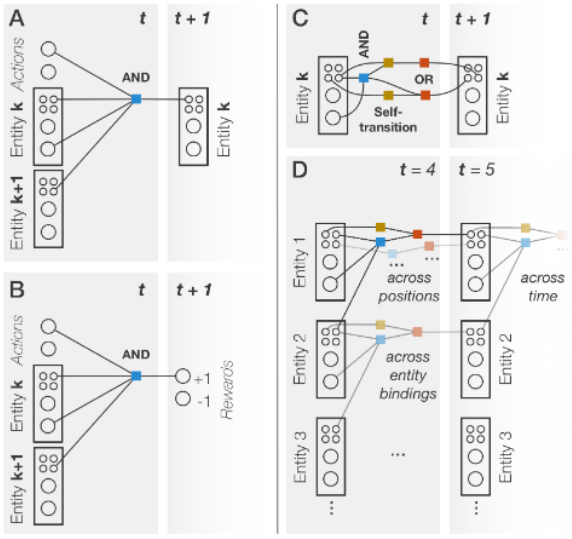


Figure 3.4: Schema extension

While their approach supports no stochasticity (their schemas need to be consistent with the entire observation history of the agent!) and cannot learn in real time as the resource effort raises with every observed frame (it's an ever-growing Integer Linear Programming problem to solve), their approach nevertheless shows a core strength of the schema representation: it has a capability to deal with previously unseen scene variations in a zero-shot way, due to the utilization of causal object-related representations. For instance, as they show in their paper (as visualized by Fig. 3.5), their agent is able to deal with a novel rock arrangement in the Breakout game immediately, while the Asynchronous Actor-Critic Agents demands to be trained for this specific spatial rock arrangement extensively and fails to generalize:

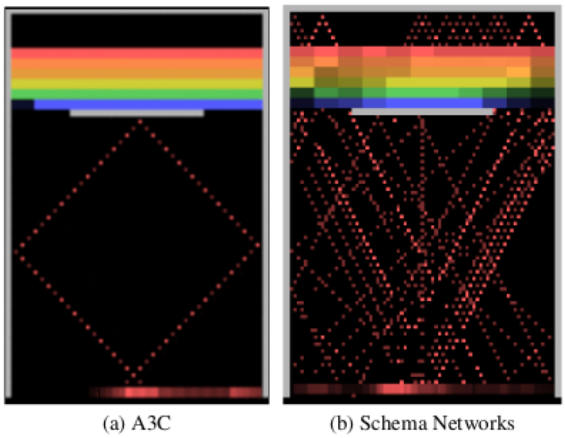


Figure 3.5: Zeroshot learning

This shows that schema representation can be a key component to enhance intelligent decision making, one we re-encountered in NARS (procedural implications), while overcoming the discussed limitations.

3.2 OpenNARS

ONA was built as a more practical system compared to OpenNARS as published in *Hammer et al.* (2016). This chapter summarizes the most notable differences between both implementation designs, which includes especially memory and control considerations.

3.2.1 Priority Queue instead of Bag

Differently than Bag in (*Wang* (2013a), *Wang* (2006) and *Hammer et al.* (2016)) which selects items with a selection chance proportional to the priority of the item (as in Parallel terraced scan, *Rehling and Hofstadter* (1997)), a priority queue selects always the item of highest priority. This means, that in Bag the lowest-priority item would have a chance to be selected too, while a priority queue would never choose them (unless it's the only item, obviously). Especially when items are re-inserted and hence can get re-selected, the difference can be significant. In ONA however there

is no concept selection step and events are not re-inserted into the attention buffer. After selection from Cycling events queue, the selected event can produce "offspring" via reasoning, but is then removed permanently from this data structure, at least until a event with same term is either input or derived again. This design decision also makes the system behave in a more predictable manner, while for OpenNARS pseudo-randomness plays an important role in selection. Also it makes ONA more reactive than OpenNARS, in the sense that its reasoning is mostly (though not completely) driven by input rather than internal activity.

3.2.2 Statement concepts and implication links

In OpenNARS each derived sentence, together with all its subterms forms a concept node. This means the term of statement $(A \rightarrow B)$ itself names a concept, and so does A and B. In ONA, only statements (including inheritance terms, similarity, negation and sequences) form concept nodes, and their subterms are handled implicitly via keys in a indexing structure (these are closer to NARS concepts than the statement nodes which are called concepts in ONA). This was also necessary because in ONA memory is pre-allocated, and assigning a belief table etc. to A and B would make the system way less space efficient. Another reason is that implications $(A \implies B)$ do not form concepts in ONA, instead this implication resides in the implication table of concept B, which also only makes sense when the term of B is a statement which can be the term of an event. The implication table allows to compare candidates effectively, as to decide how to best achieve B. This, as we have seen is crucial for subgoal derivation and decision making in ONA. This structure has also turned out to be necessary in OpenNARS as our research has revealed, the solutions in ONA are essentially a more sophisticated yet simpler form of the principles in *Hammer and Lofthouse* (2018), using a FIFO structure with deterministic selection for temporal compounding instead of an event bag with non-deterministic

selection. Also, while the decision making rule decides whether a subgoal should be derived or not dependent on whether the precondition is already fulfilled enough to reach B, in OpenNARS the derivation of subgoals happens in any case when the goal is selected, causing issues in coordination of behavior which an improved attention control mechanism might or might not resolve. ONA's memory structure is especially designed for the learning and utilization of multi-step behaviors.

3.2.3 Concept usefulness

While OpenNARS has a 3-valued Budget Value (p, d, q) of which the third value, Quality, represents long-term value of the concept to the system, ONA utilizes a usefulness value calculated by (lastUsed, useCount). This is similar to a combination of Least Frequently Used and Least recently used.

Quality on the other hand is a combination of:

- Average task priority within the concept
- Balance between intension and extension of the concept
- Effectiveness in goal achievement
- Rescale factor r where the priority of tasks does not fall below $q * r$, indirectly where q is the quality of the task. (indirectly via task priority)
- How truth maps into quality of tasks. (indirectly via task priority)

and many more. These are all relatively hard to tune and weight (which one should have how much contribution to quality), and that task priority is produced by a multitude of factors coming from complex link structures (see *Hammer et al.* (2016)) doesn't help either.

Usefulness on the other hand doesn't have these moving parts and leads to a system which keeps items in a more predictable way, but it might miss potentially relevant factors.

3.2.4 Limitations in ONA

While ONA addresses the biggest issues of OpenNARS for application purposes, some limitations do exist: compared to OpenNARS *Hammer et al.* (2016) the attention span is more short-term (reasoning concentrated mainly at the current moment rather than the past or future) and induction abilities (which hypotheses can be formed). Some of them result from design decisions which were made to make the system perform in a more predictable and reliable manner, so that its abilities can be judged more easily and experiment outcomes are resistant to example variations in cases where the outcome isn't expected to be affected by slight changes in input representation or order, though some timing dependence is inevitable.

In this chapter we will go through a list of current limitations, discussing how they can be overcome:

Temporal induction As seen earlier, the formation of sequences is controlled by the FIFO structure. Events in this datastructure are stored in the order they appear in. The sequences currently however cannot skip an event. The amount of sequences one could generate would go from polynomial $O(n^2)$ to 2^n for a given FIFO state with a FIFO of maximum size n. Clearly this is only managable for small FIFO's. For larger FIFO's, ideally attentional control would be necessary to make skip events an option (rather than just filtering generated candidates in the Cycling Events Queue). Or one can restrict the amount of events the FIFO can skip in sequences, this way the amount of sequences would stay polynomial-bounded. Since ONA, for simplicity, treats the Cycling Events Queue as the only attentional control point in the system, only the latter has been tried so far: an ONA branch SkipEventsFIFO with this

capability is available. For instance, from a given event sequence a, b, c, d, this branch can generate

```
<(a &/ c) =/> d>.
```

where b has been "skipped" in the sequence.

Variable introduction Often there are many ways variables could be introduced. ONA uses the following strategy to introduce variables:

1. Only introduce variables for atomic terms which appear at least twice.
2. Only count on left/right side of inheritances, never mix them.
3. Always introduce all variables which can be introduced.

To exemplify this strategy, an example:

```
<({a} * {b}) --> equal>. :|:
<({b} * {a}) --> equal>. :|:
```

will lead to:

```
<<({a} * {b}) --> equal> =/> <({b} * {a}) --> equal>>.
<<({$1} * {$2}) --> equal> =/> <({$2} * {$1}) --> equal>>.
<<({a} * {b}) --> $1> =/> <({b} * {a}) --> $1>>.
```

The first candidate is the version without any variables introduced, and the second one the one which introduces variables only on the left side of inheritances (extensional side), and the third one only on the right side (intensional side).

The options missed are:

```
<<({$1} * {b}) --> equal> =/> <({b} * {$1}) --> equal>>.
<<({a} * {$1}) --> equal> =/> <({$1} * {a}) --> equal>>.
```

```
<<({a} * $1) --> equal> =/> <($1 * {a}) --> equal>>.
<<($1 * {b}) --> equal> =/> <({b} * $1) --> equal>>.
<<($1 * {$2}) --> equal> =/> <({$2} * $1) --> equal>>.
<<({$1} * $2) --> equal> =/> <($2 * {$1}) --> equal>>.
<<($1 * $2) --> equal> =/> <($2 * $1) --> equal>>.
<<({$1} * {$2}) --> $3> =/> <({$2} * {$1}) --> $3>>.
<<($1 * {$2}) --> $3> =/> <({$2} * $1) --> $3>>.
<<({$1} * $2) --> $3> =/> <($2 * {$1}) --> $3>>.
<<($1 * $2) --> $3> =/> <($2 * $1) --> $3>>.
```

of which at least the first seven could be useful. The others don't include any atomic term, hence the hypothesis is too abstract to be meaningful as they are not contextually constrained in any way. While it is easy to filter out the latter, generating all possible options is not necessary a good idea, especially as the resource effort will heavily depend on the premises, jeopardizing real-time response ability. Ideally the introduction of candidates with intermediate abstraction level should be guided by hypotheses which have already been used extensively. This would demand usefulness values to be also attached to implications though, a complicating I wanted to avoid but points to an unexplored future direction.

3.3 Adaptive Neuro-Symbolic Network Agent

An alternative knowledge representation is based on Sparse Distributed Representations (SDR) *Ahmad and Hawkins* (2017), rather than based on Compound Terms that are typical for NARS. Using SDR's a system can build event sequences by taking into account the compositionality of bit vectors as proposed by *Kanerva* (2009). Operations on SDR's captures union and difference operations between bit vectors, and ways to encode hierarchical structure within them.

Making use of Non-Axiomatic Reasoning System theory, my earlier work Adaptive Neuro-Symbolic Network Agent (ANSNA) is able to learn directional correlative links between concept activations that were caused by the appearing of observed and derived event sequences. Like in ONA, which emerged from this research, these directed correlations are encoded as predictive links between concepts, and the system uses them for directed concept-driven activation spreading, prediction, anticipatory control and decision-making.

3.3.1 Similar work and philosophical differences

Like ONA, ANSNA borrows most of its theory from the Non-Axiomatic Reasoning System proposed by Pei Wang (see *Wang* (2013a)), while using the inference control theory of ALANN *Lofthouse* (2019). What makes ANSNA really different from a usual NARS is however the complete absence of *Terms* and explicit *Inheritance* relationships, coming from a philosophically very different path: while NARS tries to model a general-purpose thinking process with highly flexible ways to compare, transform, and generally deal with any kind of information that can somehow be expressed in *Narsese* (NARS's formal internal and I/O language), ANSNA concentrates completely on sensorimotor.

For NARS, sensorimotor capability, which consists mainly of procedural and temporal inference on sensor & motor events, is just a special case of the rich reasoning abilities Non-Axiomatic Logic (NAL) supports. NAL also includes declarative reasoning abilities about sets, arbitrary relations, and inheritance-relationships that are all there to support dealing with conceptual knowledge that doesn't necessarily have any grounding in actual sensorimotor experience. ANSNA takes the position where knowledge that has no possible grounding in the system's sensorimotor experience is not necessarily meaningless (as it can clearly relate to other knowledge), but surely was so far useless to a goal-driven decision-maker, as it would indicate that

the meaning of that knowledge is completely orthogonal to everything ANSNA has ever experienced through its sensors so far, both external and internal. In NARS this situation is by far not unusual, a user entering a new Inheritance relationship (term123 $\rightarrow$ term242) consistent only of new terms, term123 and term242, leaves the system's memory with a floating pair of concepts that have so far no relation to any other concepts whatsoever, meaning also no relation to sensorimotor concepts, and how such a relation should be established through correlations is a difficult problem. Such a problem does not exist in ANSNA, as it is assumed that all information is consumed through external (vision, touch, sound, temperature, other modalities...) and internal sensors (battery level, structural integrity, etc.).

According to ANSNA philosophy, relating new user-given abstract terms to sensorimotor experience is not necessary, only building compositions of sensorimotor patterns. In ANSNA every composition is automatically "grounded", since every information, without exception, ultimately is forced to enter ANSNA through the system's sensors. Also in a NARS operating in a robot without Narsese-communication channel, it is usually not happening, and not at all necessary, that new atomic terms will be created, in such a case the set of atomic terms are pre-defined by the designer, consisting of pre-defined sensor encodings and probably revisable background knowledge that was loaded on the robot beforehand. In that sense, a semantic code is inevitable, meaning the universe of mental discourse will be spanned by possible compositions of events following pre-defined encodings of sensory data (plus combinations with background knowledge, in NARS). Even though NARS itself does not assume a fixed semantic code, in that case it is undeniably present. This is however no contradiction with that such a system can acquire the meaning of observed events, where the meaning of an event has both structural and empirical aspects.

Structural meaning is determined by the composition following the semantic code, which encodes how the pattern is observed/composed from sensorimotor experience.

For instance there is no way for the system to see the observation of a red ball as structurally identical to an observed blue ball. However, it needs to be possible for the system to learn that a blue ball carries overlapping meaning, not only by being a similar structural composition / semantic code word, but also that nudging a blue ball in similar circumstances, will have similar consequences like nudging a red ball in similar contexts. And that can be done without having the user entering an explicit Inheritance relationship into the system, and without an explicit Inheritance altogether, as whether experienced event a is a special case of another event b can implicitly be represented by sensorimotor relations, that is, if a leads to the consequences we expect from b, it is naturally a special case of the former even though it may structurally differ.

Of course, the semantic code needs to be rich, not in quantity, but in quality. Same as a set of lego technic pieces needs to be rich in variety and fit together nicely to support the construction of a large variety of machines, the semantic code needs to be rich in variety and fit together in such a way, that the agent is able to conceptualize experienced aspects of its environment in an effective way. This can happen through a large variety of perceptual attributes, such as, for example, Color, PositionX, PositionY, Pitch, Frequency, Temperature, Pressure and Battery Level. Color, PositionX and PositionY can encode information from a visual field, for instance. Once a basic semantic code is in place, the encoders are present, everything the system experiences will be seen in terms of the attributes these encoders present, by ANSNA. The more comprehensive, the richer the context will be, and the better will ANSNA be able to make sense of its environment through compositions of sensorimotor events. This leads to the last key difference to OpenNARS and ANSNA, the usage of Sparse Distributed Representations (long, sparse bit vectors, SDR's), and usage of Pentti Kanerva's *Kanerva* (2009) insights about how hierarchical structure can be encoded in them. Clearly, differently than Sparse Distributed Memory (SDM)

Kanerva (1992), ANSNA is not just a model of memory, and thus, as we will see, its event-based design requirements make its memory architecture different than SDM, while preserving some of SDM's key properties. For instance, mapping events with similar SDR's to similar concepts, supporting content-addressable memory.

3.3.2 Data Structures

ANSNA's memory consists of two priority queues, one contains concepts and the other current events (Events Buffer).

Event: Each Event consists of a SDR with a NAL Truth Value, an Occurrence Time, and a Attention Value that consists of the priority of the event and a durability value that indicates the decay rate of the priority over time.

A SDR is a large bit-vector with most bits being zero, in ANSNA all SDR's are of equal length n.

SDR structure: With a,b being SDR's we can now define the following functions calculating a new SDR based on a existing one, using theory borrowed from P. Kanerva *Kanerva* (2009): $\text{SDRSet}(a, b) := a|b$ where $|$ is the bitwise or operation. $\text{SDRTuple}(a, b) := \Pi_S(a) \oplus \Pi_P(b)$ where Π_S and Π_P are two random permutations selected when ANSNA starts up, they remain the same after that.

Additionally encoding functions E as proposed in *Purdy* (2016) are used to encode similar numbers to similar SDR's, and terms are encoded into random SDR's deterministically. This way, arbitrary hierarchical compositions can be encoded into ANSNA, and as we will see later, effectively compared with each other based on a per-bit basis. For now it is sufficient to see that two input encodings, such as $SDRTuple(E(brightness), E(3.23))$ and $SDRTuple(E(brightness), E(3.5))$ will lead to similar SDR's, meaning most 1-bits will overlap. We will omit E from now on, and see that $SDRSet(green, light)$ will have more 1-bits in common with *light* than *sound*. Of course $SDRTuple$ and $SDRSet$ can be arbitrary nested with each other,

essentially forming a tree which leafs are for instance SDR-encoded terms or numbers, and structurally similar trees will lead to similar SDR's.

Concept: Concepts in ANSNA are summarized sensorimotor experience, they are the components of ANSNA's content-addressable memory system and are named by interpolations of the events SDR's that matched to it (described in more detail in the next section). Processed events can match to different concepts with various degree, but in a basic implementation a winner-takes-all approach can be taken, matching the event only to the most specific matching case that was kept in memory, and processing it as such.

Each concept has a SDR (its identifier), and Attention value consisting of a priority and a durability value, a Usage value, indicating when the concept was last used (meaning it won the match competition for an event, as we will see later) and how often it was used since its existence. Also it has a table of pre- and post-condition implications that are essentially predictive links, specifying which concepts activate which others, and a FIFO for belief and goal events, and has multiple responsibilities:

To categorize incoming events by matching them to its SDR: to become good representatives, concepts have to encode useful and stable aspects of a situation, conceptual interpolation, explained in the next section, helps here; To support revision, prediction and explanation for native events, events for which this concept wins the matching competition; To maintain how relevant the concept is currently and how useful it was in total so far; Learning and revising preconditons and consequences by interacting with an for temporal inference incoming event.

Matching events to concepts: An event can match to multiple concepts with a truth value "penalty" according to the match. Let S and P be a SDR. We want that S can be said to be a special case of P, or can stand for P, denoted by $S \rightarrow P$, if most of the bits in P also occur in S, but not necessarily vice versa. So S =SDRSet(red,ball) should be a special case of P =SDRSet(ball). It has most the features of ball, but

also has the redness feature, meaning a red ball can effectively stand for, or be treated as a ball too.

We will now formalize this idea using a NAL truth value, which is a frequency-confidence tuple $(f, c) = (\frac{w_+}{w_+ + w_-}, \frac{w_+ + w_-}{w_+ + w_- + 1})$ where w_+ is positive evidence and w_- negative evidence. The truth value of $S \rightarrow P$ can be established as follows: Let's define each 1-bit in the SDR to be a NAL sentence (see *Wang* (2013a)), where each of these 1-bits, at position i, in S, encode $bit_i = 1$.

One case of positive evidence for $S \rightarrow P$, is a common property S and P both share. Such as the fact that bit_5 is a 1-bit. On the other hand, a case of negative evidence would be a property possessed by P that S does not possess. Given that, we can define the positive evidence as: $w_+ := |\{i \in \{1, ..., n\} | S_i = P_i = 1\}|$ and the negative evidence as $w_- := |\{i \in \{1, ..., n\} | S_i = 1 \wedge P_i = 0\}|$.

If the event E has truth value T_E, to apply the penalty of "treating it as concept C", the truth value becomes Truth_Deduction(T_{match}, T_E), which will then be used in the inference rule within the concept for deriving further events.

That is motived by that if event E is a special case of the pattern it is encoded by, $SDRE$, and $SDRE$ is a special case of $SDRC$, as the match determined, then we have $E \rightarrow SDRE$ with truth value T_E and $SDRE \rightarrow SDRC$ with truth value $T_{match} := SDR_Inheritance(S, P)$. Using the deduction rule as specified in *Wang* (2013a), we end up with $E \rightarrow SDRC$, allowing to treat the event as if it would have the SDR SDRC.

Please note there is also a symmetric match defined by Truth_Intersection(SDR_Inheritance(a,b), SDR_Inheritance(b,a)) as we will need later. For a tuple of truth values $((f, c), (f_2, c_2))$ Truth_Intersection leads to $(f * f_2, c * c_2)$ and Truth_Deduction to $(f * f_2, f * f_2 * c * c_2)$, for the other truth functions we will use, please see *Wang* (2013a), they have all been described by Pei Wang in detail.

3.3.3 Practical consideration: Voting Schema

Matching new SDR-events to concepts is more difficult than in a Compound Term approach like OpenNARS where the concept can be uniquely determined by an event's term. It would be expensive to evaluate the SDRInheritance for every concept in the system's memory. So somehow we want to try only these that are guaranteed to be a good match. Events can match to multiple concepts in ANSNA. The voting schema allows for this, comparing only these SDR's that have overlapping 1-bits, and keeping track of the best result: Given concepts with the corresponding SDR $a = 00010$ concept $b = 01010$ and concept $c = 00101$, from the creation of these concepts in memory, the voting table got updated to: $bit_1 = \{\}$ $bit_2 = \{b\}$ $bit_3 = \{c\}$ $bit_4 = \{a, b\}$ $bit_5 = \{c\}$ based on the procedure, that for every i, every concept with a SDR with the $i-th$ bit being 1 will be added into the set bit_i. Now when a new event, say with SDR 01110 comes into the system (derived or input) we iterate through all the bits, and keep track of the counts of the concepts that have a 1-bit at the same position, in this case giving the following result: $\{(b, 2), (c, 1), (a, 1)\}$ where b is the best match: its 1-bits were matched by 2 of the 1-bits of the task, bit_2, bit_4. A detailed runtime analysis is outside of the scope of this paper, but this method exploits the sparsity of long bit vectors, and will be faster than matching an event to all concepts in a brute-force way. when the vectors are long enough, and sparse enough.

Of course, the revision (which sums up the positive, and negative evidence of both premises) can also happen in the belief event FIFO, this make sure that two conflicting sensory signals that happen concurrently, will be merged, allowing to better deal with contradicting sensory information. [1]

Implication Table and Revision: In NARS terms, Implications in ANSNA are eternal beliefs of the form $a \Rightarrow b$, which essentially becomes a predictive link for a

[1] A detail: As in *Hammer et al.* (2016), only revise if the evidental base does not overlap, and only if the revised element when projected to the occurrence-time middle between both elements is higher than the premises's.

and a retrospective link in b, each going to a separate implication table (preconditions and postconditions).

An implication table combines different implications, for instance $a \Rightarrow b$ and $a \Rightarrow c$ to describe the different consequences of a in the postcondition table of concept a. Implication tables are ranked by the truth expectations of the beliefs, which for a given truth value (f, c) is defined as $(c * (f - \frac{1}{2}) + \frac{1}{2})$.

Different than in OpenNARS, where it is clear whether revision can happen dependent on whether the terms are equal, two items in ANSNA can have different degree of SDR overlap. To deal with this, both revision premises are penalized with symmetric SDR match SDR_Similarity, leading to $Truth1$ and $Truth2$ using Truth_Intersection, and revision will only occur if revision(Truth1,Truth2) has a higher confidence than both Truth1 and Truth2. When a new item enters the table, it is both revised with the closest SDR candidate (the revised result will be added to the table, if it was a proper result), and also the original Implication will be added to the table.

Conceptual Interpolation: Conceptual interpolation, inspired by *Kanerva* (1992), is the process by which concept's SDR adapts to the SDR's of the matched events, in such a way that the SDR of the concept becomes the average case among the matched event SDR's. This allows the concepts to become useful "prototypes" under the presence of noise, useful in the sense that a newly seen noisy pattern can be reconstructed.

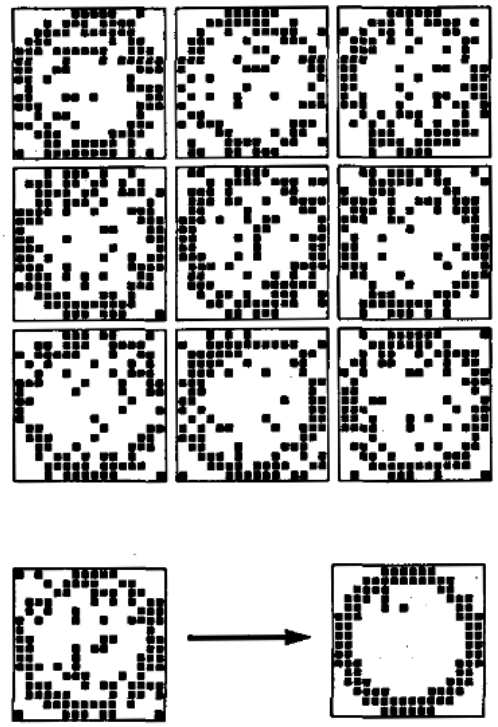


Figure 3.6: SDR interpolation as illustrated in *Kanerva* (1992)

A way to implement this is idea is to add a counter for each bit in the SDR. Each 1-bit of the matched event increases the corresponding counter by 1*u, and each 0-bit decreases it by 1*u, where u =Truth_Expectation(SDR_Inheritance(e,c)), meaning an event that better matches to the concept will have a stronger influence on it. If the counter is 0 or smaller, the corresponding concept SDR's bit will be 0, else 1. This effectively means that iff there is more positive evidence for the bit in the matched event SDR's to be 1 than 0, it will be 1 in the concept SDR they were matched to too.

Note: conceptual interpolation is highly related to, and a special case of centroid-based clustering.

3.3.4 ONA: why compound terms instead of SDR's

SDR's did not make it into ONA, however a prototype system was published *Hammer* (2019) and developed which included improved procedure learning ideas which were first described in *Hammer and Lofthouse* (2018), and later developed into ONA. Let's shed light on their pro and cons over a compound term representation:

Pros:

- Noise resistance by structural match, while compound terms demand similarity reasoning to achieve the same.
- Inheritance and Similarity-related reasoning in particular demands many inference steps and revision steps to achieve the same as what 1 SDR match can do efficiently.

Cons:

- Nesting structure encoded in SDR's is not straightforward to work with, though is crucial for higher-level cognitive functioning beyond association and recall / recognition.
- As an important special case of the previous, compositionality of representations is hard to capture with SDR's. (such as to encode a face by describing its parts)
- Memory structure hard to make efficient despite the described voting structure, better suited for hardware such as GPU's where many SDR's can be matched in parallel.
- Similarity is mostly structural rather than being evidence-based, though the centroids concept containers are associated with can be moved by using conceptual interpolation of the concept SDR which is best matched by a processed event's SDR.

- SDR's need to be very long to profit from their properties and to make sure there won't be too many overlaps for union operation, like easily 8K bits and more, while quite complex compound terms can fit into 1K bits even as it's the default in the ONA implementation.

Most of the benefits of SDR's are in efficiency for low-level pattern matching, which is why in ONA it was dropped. This is also because ONA comes with sensory channels which includes the ability to connect state-of-the-art object detectors etc., hence the benefits of SDR's.

3.4 Belief-Desire-Intention models

Belief-Desire-Intention models *Bratman et al.* (1987) *Georgeff et al.* (1998) are a very widespread paradigm for autonomous systems, it consists of the following notions:

1. Beliefs: encode current state of affairs as First-Order-Predicate-Logic (FOPL) sentences, which includes both sensor inputs and mental states.

2. Desires: State of affairs the agent wants to realize. When selected to pursue, they become goals.

3. Intentions: From commitment to a goal, linked to a plan which is meant to realize it.

A key of BDI model's success is its ability to apply goal-oriented (FOPL) reasoning / planning on task-related background knowledge (beliefs and plans). This also includes the ability to plan ahead over many decision steps, and to reach various goals in different contexts, for which learning state-action mappings / policies in Reinforcement Learning fashion isn't sufficient. These aspects are shared by NARS, however in NARS, differently to BDI models, plans and intentions are both treated as beliefs and

are not just either true or false: procedure knowledge is learnable and revisable by NARS, and is of various certainty, instead of being provided as fact by the user. Just selecting a plan according to desires / goals to become an intention, based on current circumstances (beliefs), is a much simpler problem to solve, as it ignores the learning aspect of behaviors and consideration of uncertainties which are both so critical to let agents operate in largely unknown and/or changing environments. BDI model-based agents are restricted in behavior to utilize what has been specified by the designer, though the knowledge can be combined in flexible ways by the planning process.

As we will now see also for Reinforcement Learning (RL), NARS combines and extends the key aspects of both BDI models and RL without inheriting some of their biggest limitations. We will see experiments which map to both paradigms, also including an experiment with robots for which BDI models are typically the preferred choice.

3.5 Reinforcement Learning

3.5.1 Introduction

In this chapter we will see NARS being compared with a common Reinforcement Learning technique (Q-Learning *Watkins* (1989), *Sutton and Barto* (2018)) in a set of Reinforcement Learning problems. Reinforcement Learning is usually seen as orthogonal to Practical Reasoning (*Wooldridge* (1996), *Shams et al.* (2017)), as two sub-areas of AI which serve different purposes. The former is expected to learn behaviors which maximize expected future reward, while the latter is expected to find ways to reach goals based on already available knowledge which can be used for planning purposes. The outcome of the planning process is a sequence of steps which is expected to lead to the desired goal state when starting from current circumstances. In Reinforcement Learning usually the goal is fixed and represented as a utility func-

tion which provides the agent with different reward values in different circumstances. Also, the planning is implicit: when a certain action is taken it's chosen exactly because it is expected to lead to the highest future reward, but this can still mean that additional steps are required to get the reward, which demands dealing with Temporal Credit Assignment, that is, to evaluate to what degree the taken actions were responsible for the reward outcome *Sutton* (1988) . When this situation is common (that only specific states provide reward feedback) we usually refer to this as rewards being "sparse" *Riedmiller et al.* (2018). In Practical Reasoning terms this is usually the default, as it just means that the only outcome of interest is the achievement of the goal state, intermediate outcomes don't provide good or bad value by themselves other than making achieving the goal state easier or more difficult.

To allow for meaningful comparison, in this paper we will see 'OpenNARS for Applications' (ONA, see *Hammer and Lofthouse* (2020)) which can reason under uncertainty, competing with a table-based Q-Learner (see *Watkins* (1989), *Sutton and Barto* (2018)) in domains with sparse rewards. This will show the key point this paper is about to convey: that uncertainty reasoning can be used to learn behaviors typically learned by Reinforcement Learners, and reach comparable results in certain domains model-free Reinforcement Learning techniques are typically applied in. Additionally, we will see how the reasoning approach performs better when the Markov property of next state and reward only being dependent on previous state and previous action is violated, and without relying on state merging heuristics to make the Markov property for reward hold again such as in *Gaon and Brafman* (2020). Most importantly, this work establishes uncertainty reasoning (based on Non-Axiomatic Logic in particular) as an additional Machine Learning technique to deal with Reinforcement Learning problems. In addition, a real-world robotics experiment will be presented, where YOLOv4 *Bochkovskiy et al.* (2020), a Convolutional Neural Network *Khan et al.* (2020) which represents the state-of-the-art at object detection (success or *Redmon*

et al. (2016)), will be utilized. In this experiment we will see that the reasoner inherits some of the strengths means-end reasoning solutions are known for, while being able to deal with knowledge insufficiencies at runtime. Knowledge insufficiencies *Wang* (2009) include incomplete knowledge, outdated knowledge, and various forms of concept drift, all of which is addressed by supporting robust learning at runtime.

3.5.2 A reasoner which learns and makes decisions

Whilst Practical Reasoning Systems have multiple existing instantiations (such as *Ferrein et al.* (2012), *Purang et al.* (1999), *Wooldridge* (1996), *Shams et al.* (2017), Belief-Desire-Intention models *Bratman et al.* (1987) *Georgeff et al.* (1998) like *Bordini and Hübner* (2005), etc.), most are not designed to allow knowledge to be uncertain but rely on it to be sufficient for the task at hand. Multiple logics have been proposed to support reasoning under uncertainty, such as: Markov Logic Networks **?**, ProbLog *De Raedt et al.* (2007), Fuzzy Logic: *Zadeh* (1988), Probabilistic Logic Networks: *Goertzel et al.* (2008), Non-Axiomatic Logic: *Wang* (2013a). What all these have in common is extending truth value of prepositions from boolean to a degree of belief. This allows them to capture knowledge which is not either true or false, but somewhere in-between. Of these logics, *De Raedt et al.* (2007), **?** and *Goertzel et al.* (2008) operate with probability values associated to the prepositions. To take into account the size of the sample spaces, *Goertzel et al.* (2008) and *Wang* (2013a) use a second value which intuitively speaking corresponds to the stability of the probability in light of new evidence. This allows them to allocate a higher certainty to say a 50/50 over a 5/5 coin flip scenario, while still converging to the same truth value in the limit of infinite samples. This makes these two logics 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 and the amount of samples needs to be considered in addition when comparing competing hypotheses.

For this purpose, NAL *Wang* (2013a) was chosen over PLN *Goertzel et al.* (2008) since it incorporates goal reasoning and decision making, hence can be considered a Practical Reasoner able to learn from experience. For this chapter, especially to make it more self-contained, we will present again, but in a narrower style, the NAL definitions necessary to replicate the experiments.

Truth Value Truth Value in NAL is based on positive evidence w_+ and negative evidence w_- which speaks for or against a statement / belief / hypothesis, and the total evidence $w := w_+ + w_-$, each of which is zero or greater. Based on these evidence values, the NAL truth value is defined as the tuple (f, c) with frequency

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

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 $w = 0$ don't need to be handled as they don't contribute any evidence.

Additionally, 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 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 $\lim_{w \mapsto \infty} c = 1$.

Implications For the sake of this paper we will restrict ourselves to 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 happened. To calculate the truth values of these correlative implications *Wang and Hammer* (2015b), the evidences of w_+ and w_- are needed. If events would have binary truth, $(A \Rightarrow B)$, w_+ would 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, for $((A, op) \Rightarrow B)$, w_+ would be the amount of cases in which A happened, *op* was executed and B happened after it. And 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, to make the temporal reasoning more robust in regards to timing variations, the following treatment is used instead:

Event uncertainty Events are not "true" at only a specific moment in time (with some unique identifier attached to them, which can be an integer, string, or as we will see later, logical statements with internal structure), instead they have an occurrence time and truth value attached to them. Hereby, the confidence decreases with increasing time distance to the second premise (also called Projection in *Wang* (2013a)). The way this is realized is that when two premises are used in inference, the confidence of the second premise is discounted by the factor $\beta^{|\Delta t|}$ with $\Delta t = time(B) - time(A)$, where β is the truth projection decay, a hyperparameter.

Now, the way implications are formed is via the Induction rule

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

with Δt stored as metadata and the truth of the conclusion being (as described in more detail in *Wang* (2013a)):

$$truth((A \Rightarrow B)) = f_{ind}((f_1, c_1), (f_2, c_2)) = (f_1, \frac{f_2 * c_2 * c_1}{f_2 * c_2 * c_1 + 1})$$

Now, when the same implication is derived multiple times, their truth values are revised, by simply adding up the evidences 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 events which support it (the antecedent and consequent) do occur, exactly as we intended. But with the addition that evidence is discounted based on temporal distance, which is what makes the temporal credit assignment succeed. On this matter, Projection plays the same role as Eligibility Traces do for Reinforcement Learners *Sutton* (1988).

As 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. We will need this soon to decide the occurrence time of derived events.

Learning To form the temporal and procedural implications from input events (to calculate their evidence), a sliding-window approach is taken, where the sliding window (a first-in-first-out buffer) only holds the latest k events. This way evidence for implication $(A \Rightarrow B)$ is only attributed (based on the Induction rule we just described) 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, like (X, Op), encoding that X happened and then operation event Op happened. In principle sequences don't need to contain operation events and can contain more than just two elements, this allows ONA to learn temporal patterns which span a larger

time distance (up to the sliding window size). This helps especially in environments where the Markov property does not hold, but since we compare with Q-Learning which assumes the Markov property to hold (next state and reward only being dependent on current state plus current action), we will leave this out for now to make the comparison fair.

Collecting negative evidence for an implication is slightly more tricky (Anticipation in NAL, see *Wang* (2013a)), as it is supposed to be added when the consequent won't happen, but how long to wait for the consequent? Ideally this wouldn't depend on the buffer size, and would be dependent on the averages of the experienced timings and related variances. However timing estimations can go wrong if certain distributional assumptions aren't met, which is why we went for a simpler solution for now which is at least not dependent on the size of the sliding window: to add a small amount of negative evidence immediately when the antecedent arrives, small enough that should be consequent arrive as predicted by the implication, the truth expectation of the implication will still increase (the positive evidence over-votes the negative), while else it would decrease due to the negative evidence which was added. Overall, the accumulation of positive and negative evidence leads to frequency values which encode the hypotheses (the implications) proficiency to predict successfully, whereby truth expectation can be seen as the expected frequency, which as we will now see is used in decision making (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).

Decision Making: Goal events $G!$ are represented as temporal implication $(G \Rightarrow D)$ where D is implicitly present and stands for "desired state", and their desire value is the truth value of this implication. When processed, goals either trigger decisions or lead to the derivations of subgoals. For this purpose, the existing procedural implications are checked. If the implication $((A, op) \Rightarrow B)$ has a suf-

ficiently high truth value, and event A recently happened, it will generate a high desire value for the reasoner to execute *op*. The truth expectations of the implications with G as consequent are compared, and the operation from the candidate with the highest expectation desire value will be executed if above decision threshold (a hyperparameter). If not, all the preconditions (such as A) of the implications with G as consequent will be derived as subgoals, competing for attention and processing in a bounded priority queue ranked by truth expectation (this way only the most desired goals are pursued). Hereby, the desire value of the subgoal is evaluated using deduction between the implication and the goal *Wang* (2013a). And to determine the operation's desire value one additional deduction step to take the precondition truth value into account is necessary. This corresponds to the inference rule

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

where the conclusion goal's occurrence time (the time at which X would have to have occurred if G had to happen right now) is G's occurrence time minus the Δt stored as metadata of the implication. And the following inference rule in case X is of the form (Y, op):

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

which encodes that *op* is wanted to be executed if *op* is wanted to be executed after Y happened and Y happened.

The conclusion goal desire values are:

$$desire(X) = f_{ded}(desire(G), truth(((X, op) \Rightarrow G))$$

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

$$desire(op) = f_{ded}(desire((X, op)), truth(X))$$

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

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

Using this model, decision making is concerned with realizing a goal by executing an operation which most likely, and sufficiently likely leads to its fulfillment. And when no such candidate exists to get this done in a single step, subgoals are derived from which a candidate will fulfill this requirement or again lead to further subgoaling. This is like backward planning from a goal to current circumstances, but preferring to process more attainable goals by taking uncertainties (of events and implications) into account. Differently to *Chaslot et al.* (2008) *Browne et al.* (2012) *Wang and Sebag* (2012), this process goals backwards and random rollouts (random action action till the game finishes) are not assumed to be to be possible, which also allows for usage in open-ended environments. This process can be summarized as follows:

Input: Goal G **Result:** Execution of Op, or subgoaling
subgoals = {}, bestDesire = 0.0
forall $((X, Op) \Rightarrow G), (X \Rightarrow G) \in memory$ **do**
  Subgoals = Subgoals ∪{X}
  if $desire(Op)$ *projected to current time* > *bestDesire* **then**
    bestDesire = $desire(Op)$, bestOp = Op
  end
end
if *bestDesire* > *DECISION_THRESHOLD* **then**
  execute(bestOp)
else
  forall $s \in subgoals$ **do**
    derive s projected to current time
  end
end

Algorithm 2: Decision and subgoaling

Also to make usage of implications effective in implementations, the procedural implications should be indexed by their consequent, where only a constant amount of implications is allowed for each consequent. This can be achieved by ranking them according to their truth expectation, so that too weak and wrongly predicting implications are removed while these which predict successfully are kept (similarly as in *Hammer and Lofthouse* (2018)), which keeps the resource demands bounded *Wang* (2009). Also, through the indexing, the competing hypothesis to lead to the goal don't need to be searched for, they only need to be iterated and compared in the way Alg. 2 describes.

Exploration Additionally, 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. This is also common for Reinforcement Learners, and for the reasoner is necessary especially in the beginnings where no procedural implication does exist thus far, hence no decision can be derived to lead to the desired outcome. Yet sometimes an action should be tried so that the first implications will form and "informed decision" can increasingly replace random trial (exploitation taking over

exploration).

3.5.3 Distinguishing properties of ONA and Q-Learning

Reinforcement Learning such as in *Sutton and Barto* (2018) is the most common approach used for behavior learning. While it works well for environments where infinite data can be generated (simulations primarily, *Schrittwieser et al.* (2020), *Mnih et al.* (2013)), it does not feature compositional representations that would allow for data-efficient learning. There are additional distinctions which this chapter summarizes, in addition we will later see experimental comparisons.

Differences between the decision-making models Before we move on to comparison on concrete experiments, there are some relevant differences in both decision making models which we will need to address to allow for a fair comparison. Since ONA is a NARS implementation design, many properties of the ones described in *Wang and Hammer* (2015a) are inherited by it. Compared to Reinforcement Learning formalizations some of the most significant differences are:

- **Statements instead of states**: ONA, as a NARS, does not assume states which fully describe the current situation, instead events are usually partial descriptions of the current situation as perceived by the agent, consistently with this idea also the Markov property is not assumed to hold. To make the practical comparison possible, the events however will hold the same information as the corresponding states the Q-Learner will receive in the simulated experiments. However, in the last example, a robotics use case, we will see events from different sources, coming from different modalities without blowing up the state space as would be the case when simply combining their values into a single state vector.

- **Unobservable information** Related to the previous point, there is a major difference between unobservable states which aren't known (not just their values

being unknown), and known unobservable states which values can be estimated from observable state due to known observation probabilities as commonly handled by a POMDP *Spaan* (2012) in a model-based RL setting. While ONA inherits some limited capability to address both via its uncertainty reasoning machinery, partial observability is outside of the scope of this manuscript.

- **Hierarchical abstraction** Complex environments often demand a higher level of abstraction of behaviors to allow for data-efficient learning of policies or hypotheses in general. This is by far not fully solved yet by any AI model, though attempts like *Nachum et al.* (2018) and *Zhou et al.* (2019) which arrange Reinforcement Learners in a hierarchical way do exist. However also abstraction of state matters, here, Deep Learning, especially Convolutional Neural Networks *Khan et al.* (2020) allow to deal with high-dimensional image input, and models which work in real-time *Shanahan and Dai* (2019) have become an essential technology in robotics and many real-world applications, including self-driving cars *Do et al.* (2018) *Nugraha et al.* (2017) *Farag* (2018) *Zhang et al.* (2019). In the last use case we will see YOLOv4 *Bochkovskiy et al.* (2020) being utilized for object detection, additionally we will see further abstraction of behavior (abstracting away from particular object types) happening via inductive reasoning.

- **One action in each step** ONA does not assume that in every step an action has to be chosen. To make the techniques comparable, we will hence add an additional $\hat{n}othing$ action for the Q-Learner in each example.

- **Multiple objectives** ONA can work on multiple goals simultaneously. A common approach to deal with multi objectives in Reinforcement Learning is to combine together the individual objectives into a single reward function *Sutton and Barto* (2018) *Yang et al.* (2019). This is most commonly done by formulating a scalarisation function *Van Moffaert et al.* (2013), measuring the utility of a

linear combination of expected return values of the objectives. In *Zintgraf et al.* (2015) however it is argued that the approach does not work if the parameters of the scalarisation function are not known in advance, and that in such cases a model that expresses the multiple objectives explicitly is required.

- **Changing objectives** Most Reinforcement Learning solutions are not designed to allow dealing with changing objectives / changing utility function. For game-playing this is fine, RL had a lot of success as the objective of a game usually does not change while playing it *Schrittwieser et al.* (2020). However, in robotics scenarios the situation is different, behavior of robots (such as household robots and robotic explorers) is usually expected to satisfy dynamic user goals. In this case planning methods (route planning, motion planning, etc.) remain to be crucial in autonomous robotics and self-driving cars *Karpas and Magazzeni* (2020) *Badue et al.* (2020) *Aradi* (2020), but can for example also be combined with RL-based path tracking approaches *You et al.* (2019).

Reasoner and background The particular implementation we will use for comparison purposes and follows these principles is 'OpenNARS for Applications' (ONA, see *Hammer and Lofthouse* (2020)), an implementation of a Non-Axiomatic Reasoning System **?** the author proposed previously. While ONA has been compared with Actor Critic (AC) and Double-Deep Q (DDQ) on Cartpole-variants (by our collaborators, Professor Kris Thórisson's team, especially Leonard Eberding, *Eberding et al.* (2020)), the input representations were not the same between the compared methods (mostly because ONA does not accept numeric inputs without preprocessing), lowering the strength of the results. Additionally, more tasks are required to make the case to establish ONA as an additional technique to address Reinforcement Learning problems stronger. In this section we will compare ONA with a standard table-based Q-Learner *Watkins* (1989) implementation with Eligibility Traces *Sutton* (1988), while ensuring that both the Q-Learner and ONA receive the exact same

input. For completeness, and to allow to relate the hyperparameter choices with the Q-Learning model used for the experiments:

$$
\begin{aligned}
Q(s_{t+1}, a_{t+1}) &= max_a Q(s_{t+1}, a) \\
\delta Q_t &= r + \gamma * Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \\
e_t(s_t, a_t) &\leftarrow e_t(s_t, a_t) + 1 \\
\forall s, a : [Q(s, a) &\leftarrow Q(s, a) + \alpha * \delta Q_t * e_t(s, a) \\
e_t(s, a) &\leftarrow \gamma \lambda e_t(s, a)]
\end{aligned}
$$

where α is the learning rate, γ controls how much to favour future rewards over short-term reward, and λ controls the decay speed of egilibility traces. Additionally ϵ which is not mentioned here, is the exploration rate. It encodes the chance to select a random action as a_{t+1} instead of the one with the highest expected reward.

Goal achievement as reward While ONA can deal with the goals in the described way, the Q-Learner needs a reward signal. Hence if both should receive the exact same input to make comparison more meaningful, there needs to be a mapping from goal achievement to reward (or from reward to goal achievement). The way this is achieved is the following way: when an event X is input it is interpreted by ONA as event, and by the Q-Learner simply as current state. If X however corresponds to the outcome to achieve, and the reward for the Q-Learner will be 1 (while ONA receives event $goodNar$), and else 0. This of course assumes that the goal doesn't change, as else the Q-table entries would have to be re-learned, meaning the learned behavior would often not apply anymore. For the purposes of this paper, and for a fair comparison with Reinforcement Learning, the examples include a fixed objective.

CHAPTER 4

Experiments and Applications

4.1 ONA vs. Q-Learning

The experiments chosen for comparison are two typical Reinforcement Learning examples, Space invaders and Pong, plus a grid world experiment where the agent has to find food while maneuvering around obstacles underway. (Fig. 4.1) Both techniques are run multiple times in each experiment, and the example-specific success measure is kept track of for each time point across 10000 iterations, together with the average summarized over all runs of the particular technique.

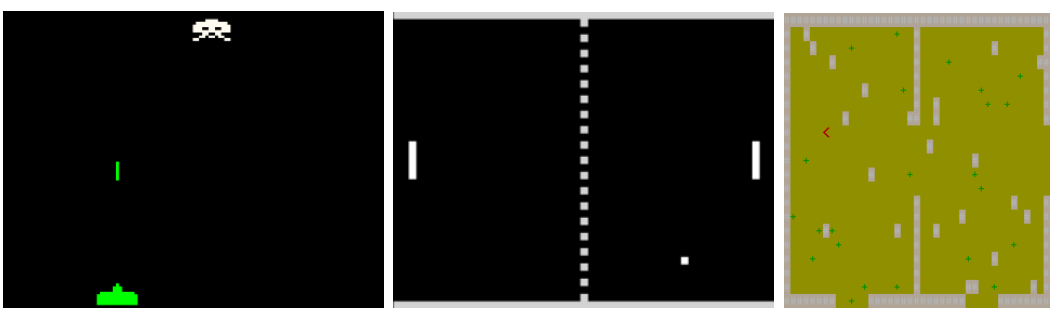


Figure 4.1: Space invaders, Pong and grid robot

Space invaders In this game (Fig. 4.1) the player controls a spaceship and is supposed to shoot down aliens, whereby the success ratio we will use is the amount of hits over the total shots. Both models receive the enemy location as either $enemy_left$, $enemy_right$, or $enemy_aligned$ when aligned with the player. Additionally the actions the agent can take are $\hat{l}eft$, $\hat{r}ight$ which move the agent to the left/right side by some step size (set to be 5 percent of the game screen width, so 20 actions to get from one side to the other), and *shoot*. Additionally the $\hat{n}othing$ action exists for

fair comparison between both techniques, as the reasoner can decide to do nothing according to NAL decision theory *Wang* (2013a). To hit the enemy, the *shoot* action needs to be taken when aligned with the enemy. The hyperparameters for the Q-Learner are $\alpha = 0.1, \gamma = 0.1, \lambda = 0.8, \epsilon = 0.3$. The ONA parameters are the default config in ONA v0.8.7 *Hammer and Lofthouse* (2020), and use the same epsilon as motor babbling rate. Each technique is run 10 times with different random seeds each, meaning also variations in starting positions in addition to outcomes of when to choose explorative actions.

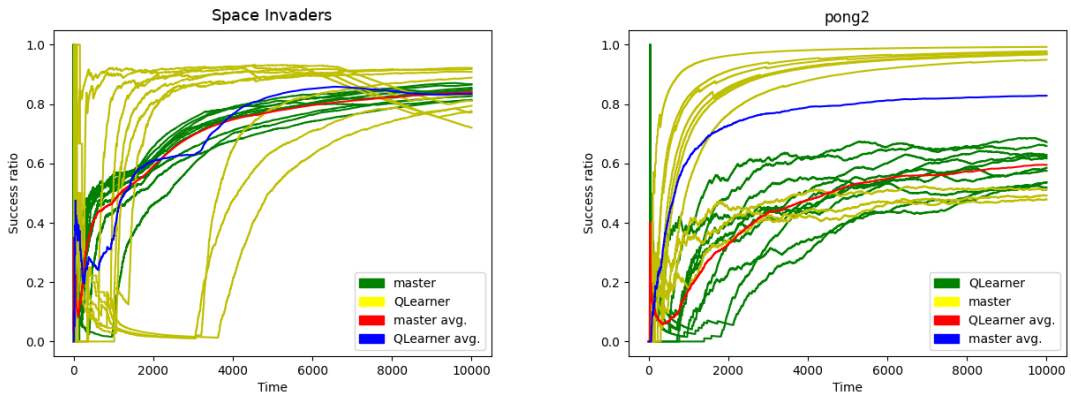


Figure 4.2: Success ratio in Space invaders and Pong

The results as seen in Fig. 4.2 indicate that both techniques (Q-Learning and ONA) converge to the same capability on average, and on average with approximately the same learning speed. However, the learning behavior of ONA is more consistent, while Q-Learning revealed both cases where it learns significantly quicker and slower than all ONA runs.

Pong In Pong (Fig. 4.1), the location of the ball, which is able to reflect at the walls, is encoded as relative $ballLeft$, $ballRight$, $ballEqual$ in the same fashion as in Space invaders. This time $\hat{l}eft$ and $\hat{r}ight$ initiate left/right movement of the bat, and an additional operator $\hat{s}top$ stops the movement. Like before $\hat{n}othing$ exists as alternative action to make the comparison fair. The goal is for the ball to hit the bat

repeatedly, and the success rate represents the hits over the hits plus misses. This time the hyperparameters of the Q-Learner are $\alpha = 0.1, \gamma = 0.1, \lambda = 0.8, \epsilon = 0.2$, meaning a slightly lower epsilon than before.

Surprisingly, while the Q-Learner learned the use of $\hat{l}eft$ and $\hat{r}ight$, it consistently failed to learn to use the stop operator and hence reached lower success ratios than ONA (4.2) which only failed to do so in 3 runs.

When visualizing the reasoner's knowledge in a behavior graph as in Fig. 4.3, we can see why this can be tricky. In this graph, nodes are the event, and links encode transitions via some operation. The operations are drawn at the outgoing side of the edge. Additionally, the edge colors are of interest here, where red encodes positive evidence and blue encodes negative evidence, hence violet is a mixture thereof:

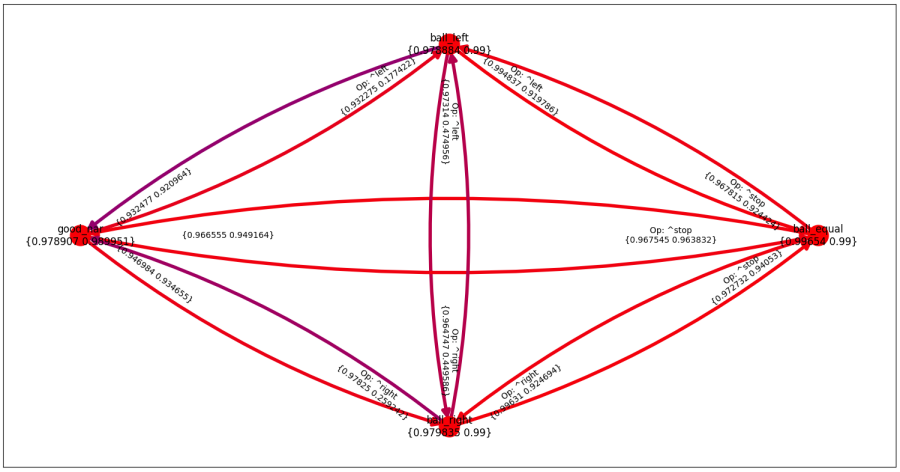


Figure 4.3: ONA behavior graph after learning Pong

There is a violet rather than red connection from $ballLeft$ to $goodNar$ via $\hat{l}eft$, which indicates that this link has received an significant amount of negative evidence. This makes sense, as moving left when the ball is left is often not by alone sufficient, also the stop operator has to be invoked when reaching the ball location (same for the

other direction). The edge from $ballLeft$ to $ballEqual$ via $\hat{l}eft$ on the other hand is very successful (indicated by red color), and so is the edge from $ballEqual$ to $goodNar$ via $\hat{s}top$. This makes the issue at heart clear: an operator that initiates movement instead of performing a step of a certain size once breaks the Markov property of the reward, that is, that the last state and action determines fully what state the agent will observe next and the reward it will obtain. With action $\hat{l}eft$ being used lastly, the bat continues to move left when $\hat{n}othing$ is invoked, hence the state transition and also reward in this case can depend on a state-action combination which happened prior to the last one. In this case also state merging heuristics like in *Gaon and Brafman* (2020) cannot resolve the issue, as the side effect is caused by the action and not part of an observable state.

To confirm that this is indeed the explanation for the worse performance, the same experiment was tried without the $\hat{s}top$ action, with everything else being the same. In this case the Q-Learner turned out to indeed perform comparably well like ONA on average (Fig. 4.4):

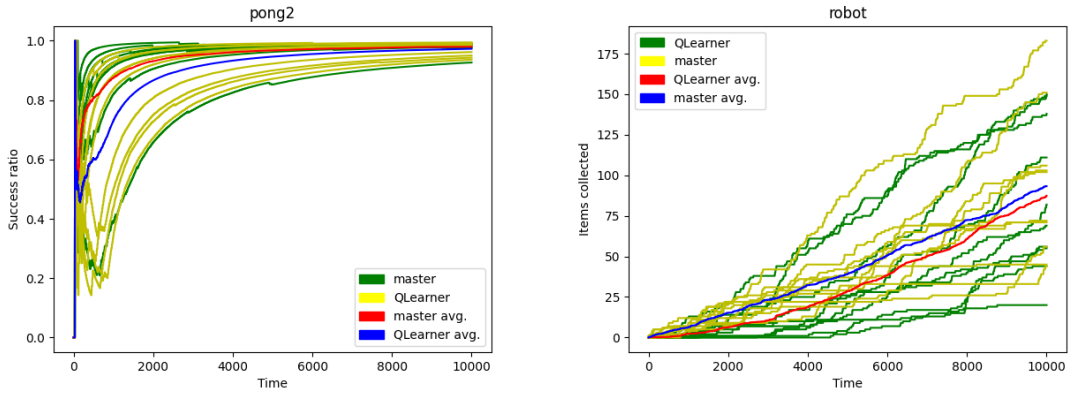


Figure 4.4: Success ratio in simplified Pong and Grid Robot

though in this case the problem regresses to learning to invoke only the actions $\hat{l}eft$ and $\hat{l}eft$, which is clearly a simpler problem to solve. Nevertheless, what this suggests is that ONA can deliver the same learning performance as Q-Learning, while

additionally delivering better results when the Markov property does not hold. The sliding window approach to mine for patterns in input event stream, although local in time, is less restricted than only considering the current and last state in Q-table updates.

Robot This experiment (Fig. 4.1) features a robot in a grid with walls and food objects to collect, whereby the amount of food objects collected we will directly use as success measure. Perceived events are from the perspective of the robot, which can turn around by using $\hat{r}otateLeft$ and $\hat{r}otateRight$. Additionally it can move forward by one grid cell by invoking $\hat{m}oveForward$. The robot can see what is in front, left and right to it within a 10 cells distance. It can either be, in this preference order, $foodCentered$, $foodLeft$, $foodRight$, $wallCentered$, $wallLeft$, or $wallRight$. The only outcome positive of interest is the agent colliding with food, but of course to achieve this the agent also has to learn to avoid obstacles in order not to get stuck. In this experiment, and with the same parameters as in Space Invaders, both techniques performed comparably well, with a slight leap of ONA in collected items on average, but overall similar values with the chosen hyper-parameters. This example can easily be extended to multi-objective scenarios and scenarios with changing objectives, showing merits in these areas will be part of our future work. Overall, as Table 4.1 suggests, except of the issue with the Markov property of states and rewards being violated for Q-Learning which demanded a simplification of the Pong example to get similar results, both techniques were comparable in performance on average. Hence the reasoning-based approach provides a viable alternative for such problems, while performing better whenever the Markov property is violated, since it does not explicitly depend on this property.

Table 4.1: End performance results of all experiments. (after 10000 iterations)

Success measure	**ONA**	**Q-Learner**	**Number of trials**
Space invaders	0.86	0.85	20 (10 each)
Pong (non-Markovian)	0.80	0.61	20 (10 each)
Pong (simplified)	0.98	0.97	20 (10 each)
Grid robot	91	87	20 (10 each)

4.2 Tests in cognitive psychology

This chapter is especially concerned about testing higher-level cognitive functioning of the overall system in isolation. For this purpose, multiple tests have been carried out, including:

1. Acquiring object permanence
2. Property association
3. Stanford Marshmallow test
4. Identity Matching
5. Active question answering
6. Disambiguation of nouns
7. Story understanding
8. Knowledge transferability

For all tests, motorbabbling has been disabled to ensure that the actions output by the system are made according to its best available evidence and hence representative of the system's functionality.

4.2.1 Acquiring object permanence

Object permanence is only barely handled by today's robotic solutions. For static objects the task is relatively easy as it can be done by marking objects in a map resulting from Simultaneous Localization and Mapping, for instance based on feature detection in visual input *Taketomi et al.* (2017). Dynamic objects however don't just stay in one place, and once out of sight, re-appear based on various different criteria. A person going to the restroom usually does not re-appear in less than a minute, an animal hiding behind a bush might only appear when a shot is fired, and so on. Of course it also depends on whether the object was observed going into the hideout in the first place, the past. Hence, to handle object permanence correctly, the context under which objects appear after hiding needs to be learned, it cannot be handled by a module unable to learn arbitrary contexts under which other events do appear. For ONA however, this capability is part of its basic temporal reasoning ability.

We will illustrate this with an example where a toy which can either be to the left or right, gets then hidden by a cup. The p̂ick operation is able to remove the cup to make it visible again. The four cases are:

```
//left toy gets hidden by left cup then revealed by picking the left cup
<toy --> [left]>. :|:
<cup --> [left]>. :|:
<({SELF} * left) --> ^pick>. :|:
<toy --> [left]>. :|:

//right toy gets hidden by right cup then revealed by picking the right cup
<toy --> [right]>. :|:
<cup --> [right]>. :|:
<({SELF} * right) --> ^pick>. :|:
<toy --> [right]>. :|:
```

There is a cup on the right but nothing beneath it, so nothing happens after pick: (repeated 13 times)

```
<cup --> [right]>. :|:
<({SELF} * right) --> ^pick>. :|:
```

There is a cup on the left but nothing beneath it, so nothing happens after pick: (repeated 13 times)

```
<cup --> [left]>. :|:
<({SELF} * left) --> ^pick>. :|:
```

The competing hypothesis in this case are:

```
<(<cup --> [left]> &/ <({SELF} * left) --> ^pick>)
    =/> <toy --> [left]>>.
Truth: frequency=0.863665, confidence=0.269194
<(<cup --> [right]> &/ <({SELF} * right) --> ^pick>)
    =/> <toy --> [right]>>.
Truth: frequency=0.863665, confidence=0.269194
<(<cup --> [$1]> &/ <({SELF} * $1) --> ^pick>)
    =/> <toy --> [$1]>>.
Truth: frequency=0.863665, confidence=0.424196
<((<toy --> [left]> &/ <cup --> [left]>) &/ <({SELF} * left) --> ^pick>)
    =/> <toy --> [left]>>.
Truth: frequency=1.000000, confidence=0.200929
<((<toy --> [right]> &/ <cup --> [right]>) &/ <({SELF} * right) --> ^pick>)
    =/> <toy --> [right]>>.
Truth: frequency=1.000000, confidence=0.200929
<((<toy --> [$1]> &/ <cup --> [$1]>) &/ <({SELF} * $1) --> ^pick>)
    =/> <toy --> [$1]>>.
Truth: frequency=0.994886, confidence=0.335766
```

whereby the last, most general hypothesis which works for any location, and also takes the past cue of where the toy was seen last into account, has gained the highest truth expectation. Hence the system has not only generalized the hypotheses across the locations the objects was hidden at, but also identified the last visible location as crucial.

4.2.2 Property association

In this task the system is supposed to form categories from labelled example instances of ducks and swans. Example of duck which isn't a swan:

```
<{instA} --> [yellow quacks]>.
<{instA} --> duck>.
<{instA} --> swan>. {0.0 0.9}
<{instA} --> bird>.
```

Example of a swan which isn't a duck:

```
<{instB} --> [white quacks]>.
<{instB} --> swan>.
<{instB} --> duck>. {0.0 0.9}
<{instB} --> bird>.
```

Four such instances were given to the system to induce the hypotheses:

```
<swan --> [white quacks]>.
<duck --> [yellow quacks]>.
<swan --> bird>.
```

From there on, 12 instances of swans and ducks were given to the system with overlapping properties. Whenever the instance was white and quacked, it was correctly associated with swan with highest truth expectation, even when it possessed

additional properties. Also, whenever the instance was yellow and quacked (plus other properties), it was correctly associated with duck. Also, in both cases, the instance was inferred to be a bird.

To show the robustness of this ability, 40 events unrelated events were input in the system between the training and test examples. The concept usefulness output led to:

```
{i=33} <duck --> bird>:
{ "usefulness": 0.336000, "useCount": 84, "lastUsed": 18536 }
{i=164} <duck --> [yellow quacks]>:
{ "usefulness": 0.223810, "useCount": 47, "lastUsed": 18539 }
{i=195} <swan --> [white]>:
{ "usefulness": 0.110619, "useCount": 25, "lastUsed": 18501
{i=172} <duck --> [yellow]>:
{ ""usefulness": 0.208738, "useCount": 43, "lastUsed": 18539
{i=281} <swan --> [white quacks]>:
{ "usefulness": 0.065116, "useCount": 14, "lastUsed": 18501 }
{i=1184} <swan --> bird>:
{ "usefulness": 0.000575, "useCount": 9, "lastUsed": 3062 }
```

meaning all relevant knowledge was among the ten percent of most useful concepts (16K concepts in total).

4.2.3 Stanford Marshmallow test

This test is about delaying gratification, which is often seen as a strong sign of self-control ability. We will use atomic terms here, since this experiment was also tried successfully with our Q-Learning implementation which accepts this input format:

```
//Training example1:
//System getting immediate repletion from eating the marshmallow
```

```
marshmallow1. :|:
^eat. :|:
replete. :|: {0.6 0.9}
//Training example2: The system getting another marshmallow after leaving it be,
//then getting a lot of repletion from eating
marshmallow1. :|:
^leave. :|:
marshmallow2. :|:
^eat. :|:
replete. :|: {0.9 0.9}

//Test example: System should have learned to leave the first marshmallow alone:
marshmallow1. :|:
replete! :|:
//expected: ^leave executed with args
```

4.2.4 Arbitrarily Applicable Relational Responding

Arbitrarily Applicable Relational Responding is a notion in Relational Frame theory, for a cognitive system to be able react to react to arbitrary relations present between the simuli it is presented with. Experiment in this topic have been tried partly in collaboration with Professor Robert Johansson, some of which he summarized in *Johansson* (2019). He tested several capabilities, but Identity Matching on which we collaborated on has not been described there. Identity Matching allows an agent to learn to choose between left and right side, dependent on whether the example presented on either side shares a characteristic common property with the example presented in the middle. In this case, mere pattern similarity between the examples is not sufficient, and neither is it sufficient to just learn the case for a par-

ticular set of properties. Many mammals, such as sea lions *Kastak and Schusterman* (1994), are equipped with this capability. In ONA, this was encoded in the following way:

```
<{right} --> [prop2]>. :|:
<{left} --> [prop1]>. :|:
<{middle} --> [prop1]>. :|:
^left. :|:
success. :|:
```

And a second example where the right side shares a property:

```
<{left} --> [prop20]>. :|:
<{right} --> [prop10]>. :|:
<{middle} --> [prop10]>. :|:
^right. :|:
success. :|:
```

From this experience it was able to derive both relevant hypothesis where the property was abstracted away, which drive its behaviors when examples with new properties are presented to the system:

```
<((<{right} --> [#1]> &/ <{middle} --> [#1]>) &/ ^right) =/> success>.
    Priority=0.148340 Truth: frequency=1.000000, confidence=0.282230
<((<{left} --> [#1]> &/ <{middle} --> [#1]>) &/ ^left) =/> success>.
    Priority=0.148340 Truth: frequency=1.000000, confidence=0.282230
```

4.2.5 Active question answering

In reasoning systems, questions are usually treated passively, that is: they trigger inference which is meant to answer the question, but do not cause any decisions to be made by the system. Active questions however often demand the system to perform

actions to find an answer. In simplest case, this might be a a robot which needs to turn around to see something which is not currently in sight, which often also demands the ability to handle object permanence to work properly. However, central here is the ability to handle goals with variables, so that any variant of the event which has the variable instantiated can fulfill the goal event. This makes sure the system is not trying to replicate a specific goal event (e.g. trying to put the cat into the living room when asked about where the cat is), but is merely trying to find the answer and is satisfied after observing it.

```
<corridor --> in>. :|:
<({SELF} * kitchen) --> ^go>. :|:
<({cat} * kitchen) --> in>. :|:
120
<corridor --> in>. :|:
<({SELF} * bedroom) --> ^go>. :|:
<({cat} * bedroom) --> in>. :|:
120
<corridor --> in>. :|:
<({SELF} * livingroom) --> ^go>. :|:
//no cat this time, it doesn't like the livingroom :)
120
<corridor --> in>. :|:
<({SELF} * bedroom) --> ^go>. :|:
<({cat} * bedroom) --> in>. :|:
120
//Ok you are in corridor now
<corridor --> in>. :|:
//NARS, where is the cat?
//Passive question <({cat} * ?where) --> in>? :|: wouldn't trigger a decision
```

```
//Active question however does:
<(<({cat} * #where) --> in> &/ <({SELF} * #where) --> ^say>) =/> G>.
G! :|:
120
//expected: ^go executed with args ({SELF} * bedroom)

//ok, feedback of NARS going to the bedroom, the cat is there!
<({cat} * bedroom) --> in>. :|:
G! :|:
10
//expected: ^say executed with args ({SELF} * bedroom)
```

4.2.6 Disambiguation of nouns

The same words often mean very different things in different contexts. This experiment shows how ONA is able to disambiguate the meaning of concepts based on other recently mentioned concepts that were part of the conversation. Existing tools such as **?** make use of a knowledge base in the background, and demand the input to be fully specified before disambiguation can be performed properly. ONA however is able to resolve ambiguities in an ongoing conversation.

To showcase this, we will utilize the English-to-Narsese translator and will use the german word "bank" which carries the meaning of the English word "bank" but additionally can mean "bench".

At first, we introduce to the system the german variant of "bank", as a furniture people sit on:

```
people sit on a bank
a bank is a furniture
```

and tell it that people sit on a wooden bench:

```
people sit on a wooden bench
```

Now, of course, the system should be able to answer that a wooden bench, which is abduced to be a special case of a bank, is hence also a furniture:

```
a wooden bench is a furniture?
//Answer: <([wooden] & bench) --> furniture>. :|:
          Truth: frequency=0.990960, confidence=0.253355
```

Additionally, we will now give the English interpretation of "bank", as a financial institute people withdraw money in:

```
people withdraw money in banks
a bank is a financial institute
```

Last, we will give "Volksbank" (an Austrian bank), as an example of something people withdraw money from:

```
people withdraw money in volksbank
```

When now asked about whether Volksbank is a financial institute, it's the first answer ONA generates:

```
volksbank is a furniture?
volksbank is a financial institute?
//Answer: <volksbank --> ([financial] & institute)>. :|:
          Truth: frequency=0.990960, confidence=0.253355
```

This is because last time in this conversation we talked about banks as a financial institute, which made the corresponding concept $< bank \rightarrow ([financial]\ \&\ institute) >$ gain more priority than concept $< bank \rightarrow furniture >$.

4.2.7 Story understanding

Story understanding demands to understand the meaning of the involved sentences:

```
*volume=0
A mouse is an animal.
It is dead if the cat eats it.
The cat is eating a mouse in the garden.
70
What is in the garden?
//expected: Answer: <(cat * garden) --> in>. :|:
            occurrenceTime=4 Truth: frequency=1.000000, confidence=0.900000
The cat is in the garden?
//expected: Answer: <(cat * garden) --> in>. :|:
            occurrenceTime=4 Truth: frequency=1.000000, confidence=0.900000
The mouse is dead?
//expected: Answer: <mouse --> [dead]>. :|:
            occurrenceTime=3 Truth: frequency=0.995047, confidence=0.425313
The cat is eating an animal?
//expected: Answer: <(cat * animal) --> eat>. :|:
            occurrenceTime=3 Truth: frequency=1.000000, confidence=0.286319
```

but also demands a system to understand the flow of events as described by a story, to such a degree that even hypothetical cases about possible extensions of the story could be answered:

```
*volume=0
the pizza is in the oven
the person takes the pizza
who took the pizza?
```

```
//expected: Answer: <(person * pizza) --> take>. :|:
            occurrenceTime=2 Truth: frequency=1.000000, confidence=0.900000
the person eats the pizza
the person is satisfied
100
the burger is in the oven
the person takes what?
//expected: Answer: <(person * burger) --> take>. :|:
            occurrenceTime=106 Truth: frequency=1.000000, confidence=0.228606
```

In this case the utilized hypothesis is:

```
it is in the oven leads to the person takes it
```

This is just a simple example which builds on the extracted hypothesis

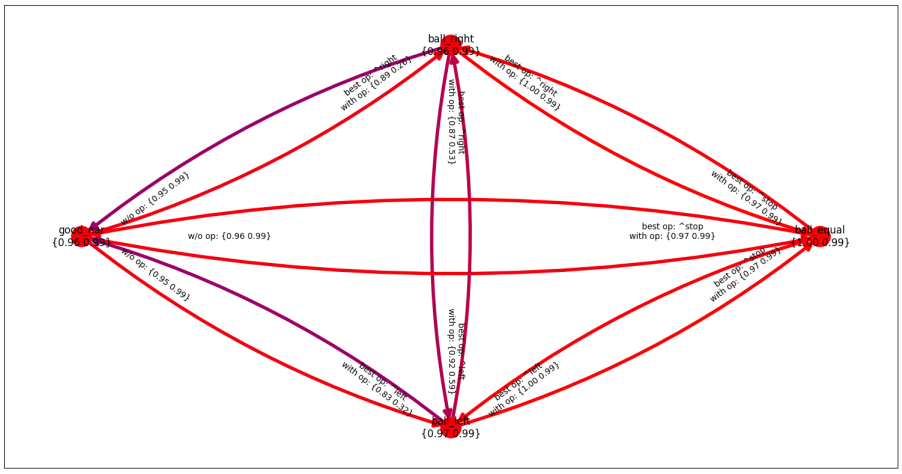
```
//When something is in the oven, the person will take it out.
 <<($1 * oven) --> in> =/> <(person * $1) --> take>>.
```

but with additional background knowledge, either extracted from similar stories, or given by a human, story understanding can go beyond that.

4.2.8 Knowledge transfer

In default configuration ONA uses up to 16384 concepts. What if we only filter the concepts of highest usefulness after processing some example? Expected is that the output will reflect the most precious knowledge. When this is done for the simple pong game , the 5 most useful concepts should include the procedure knowledge to play the game encoded as procedural links between "left, right, equal". If this knowledge is fed into a new ONA instance, the new instance should be able to play immediately.

First, let's look at the graph after 5000 game iterations:



It was generated with the command

```
./NAR pong2 5000 InspectionOnExit |
      python3 concept_usefulness_filter.py 4 > Best4Concepts
python3 concepts_to_graph.py NoTemporalLinks < Best4Concepts
```

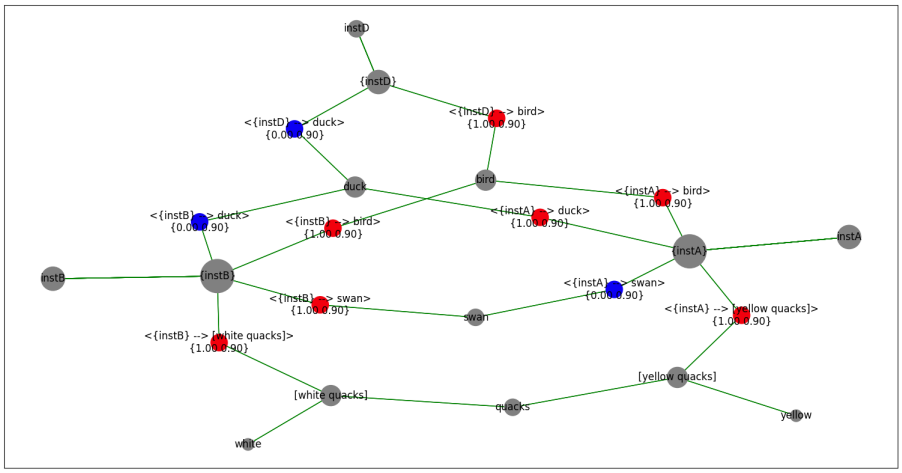
Note how this graph encodes the right behavior, that is, to move left if the ball is on the left to get it into the equal position (bat = ball position), to move right if the ball is right to get it into the equal position, and to stop if it is already in the equal position to achieve event *good_nar*. Also the "chunked" shortcut links are in the graph although with less truth expectation (violet), that is: if the ball is on the left, move to the left to achieve *good_nar* and if the ball is on the right, move to the right to achieve *good_nar*.

Hence when the best 4 concepts are put into a new instance, the new instance immediately plays perfectly, which can be done with:

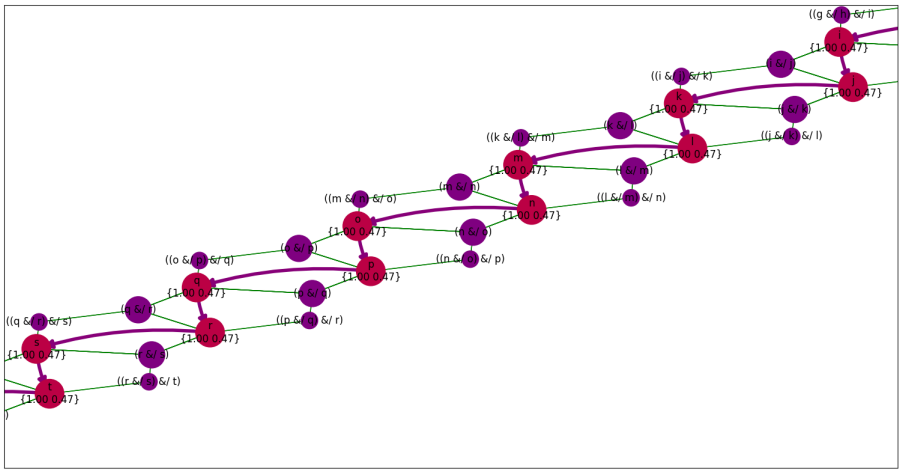
```
./NAR shell pong2 < Best4Concepts
```

The graph plotting can also deal with declarative knowledge, and can help to

increase the explainability of decisions the systems makes, and to explore how existing knowledge within the system is related to each other:



While it would be difficult to interpret the entire memory content, exploring the most used concepts is an insightful endeavour also to understand the functioning of ONA better. When for instance the letters of the alphabet are input in alphabetical order with 10 inference steps between events, the following graph results:



It shows the temporal order of the events but also the sequences the system composed of these input events, which are composed in a predictable manner by

the FIFO of the system corresponding to a sliding window-based pattern mining approach.

4.2.9 Other use cases

ONA can be used for to learn by observing other agents. This is important for agents operating in environments where other agents with overlapping capabilities are present. As an example, a ONA-controlled agent is supposed to be able to learn a behavior exhibited by another agent. Differently than in Reinforcement Learning, the other agents can pursue different goals yet their behavior can be modeled by ONA. This is possible since the knowledge ONA can learn from observation is not directly linked to a specific goal or reward outcome to achieve. We will now look at the key technological aspects which make learning from observation possible:

1. Different agents can be conceptualized as separate instances.
2. Knowledge to predict events can be acquired under absence of rewards.
3. Using the knowledge, intentions of other agents can be predicted when a similar behavior is exhibited in a similar context.
4. The logic allows to generalize a hypothesis about the action of another agent in such a way that the agent can try the hypothesis by itself.
5. The agent can communicate hypotheses, allowing to inform other agents.

To make this succeed, encodings do however matter. Please consider the following example:

```
//Agent1 is close to the fridge
<({agent1} * fridge) --> frontOf>. :|:
//Agent1 is opening the fridge
```

```
<({agent1} * fridge) --> ^open>. :|:
//the fridge is open
<fridge --> [open]>. :|:
//Agent1 picks a drink
<({agent1} * drink) --> ^pick>. :|:
//Agent1 is satisfied
<{agent1} --> [satisfied]>. :|:
```

Some relations, like $frontOf$ need to be built by the perception system, and in such a way, that observations for the agent itself lead to the same encoding. Also, the last event needs to come from the other agent through an innate form of communication. If this is the case, this example succeeds and leads to the formation of hypotheses such as:

```
<(<({agent1} * fridge) --> frontOf> &/ <({agent1} * fridge) --> ^open>)
=/> <fridge --> [open]>>.
<(<fridge --> [open]> &/ <({agent1} * drink) --> ^pick>)
=/> <{agent1} --> [satisfied]>>.
```

which can be generalised by ONA:

```
<(<({$1} * $2) --> frontOf> &/ <({$1} * $2) --> ^open>)
=/> <$2 --> [open]>>.
<(<fridge --> [open]> &/ <({$1} * drink) --> ^pick>) =/>
<{$1} --> [satisfied]>>.
```

4.3 Robotics experiments

4.3.1 Practical Reasoning and Learning

In this chapter we will see how ONA can be used to extend on the abilities of typical Practical Reasoning approaches *Ferrein et al.* (2012) *Bordini and Hübner*

(2005) by adding Learning at runtime to the picture and without relying on a separate learning technique or POMDP-based approach such as in *Amiri et al.* (2020). The strength of practical means-end reasoning lies in the ability to effectively utilize high-level (easily by human's expressible and communicable) knowledge for planning purposes. However, Practical Reasoning solutions such as *Bordini and Hübner* (2005) and *Ferrein et al.* (2012) lack robust learning ability or learning machinery altogether. Proper learning machinery would allow the agent to ideally deal with lack of knowledge and changing circumstances (and various forms of concept drift *Hu et al.* (2020)). We will see how ONA can both easily be fed with background knowledge while being able to learn new knowledge from observations at runtime as also the previous examples demanded. Differently than in previous collaboration work *Lanza et al.* (2020) with Professor Antonio Chella's team (especially Francesco Lanza) where NARS was combined with a BDI model implementation*Bordini and Hübner* (2005), ONA is both responsible for learning and decision making.

Encodings Before we continue, additional knowledge representation needs to be introduced beyond events, operations, sequences of events and operations, and implications. Previously we assumed input events just have a string or integer identifier attached to them. However, since ONA uses Non-Axiomatic Logic *Wang* (2013a), events can have richer internal (compound) term structure, which we will make use of for similar reasons predicates are common in First Order Predicate Logic-based Practical Reasoner applications and Logic Programming in general: they allow arbitrary relationships to be easily expressed. The following logical copulas are most important for this experiment. Please note that these can be arbitrarily nested, which makes the formal language ONA utilizes for knowledge representation very expressive.

- Inheritance statements. Inheritance $< A \rightarrow B >$ indicates A to be a special case B. For instance, that cats are animals can be encoded as an Inheritance statement $< cat \rightarrow animal >$.

- Terms referring to instances and properties, denoted by $\{instance\}$ and $[property]$ respectively. For instance, Garfield being orange in color can be expressed with $< \{garfield\} \rightarrow [orange] >$.

- Relational terms. This includes products $(a * b)$ to express anonymous relationships, these allow to express arbitrary relationships like that cats eat mice, $< (cat * mouse) \rightarrow eat >$. This is similar to predicates in predicate logic.

- Negation, expressed with $(\neg\, a)$, where a can be an arbitrary statement.

- Variables: Dependent and Independent Variables $\$name$ and $\#name$ respectively resembling all- and exists-quantified variables. These allow to make statements more abstract by allowing for variable binding and unification in inference to substitute a specific term as long as its structure matches and does not conflict with a previous assignment to the same variable.

Experiment setup In the experiment a NXT robot was used with the following components and software as also illustrated in Fig. 4.5.

- Sensors: Ultrasonic sensor, camera of attached smartphone.

- Actuators: Three motors. Two are for turning left, right and moving forward, while the third motor is responsible to pick up and drop objects.

- Software: A vision channel based on YOLOv4 *Bochkovskiy et al.* (2020) trained on ImageNet via the Darknet implementation

- Parameters: As in ONA v0.8.7, but with exploration rate 0.1.

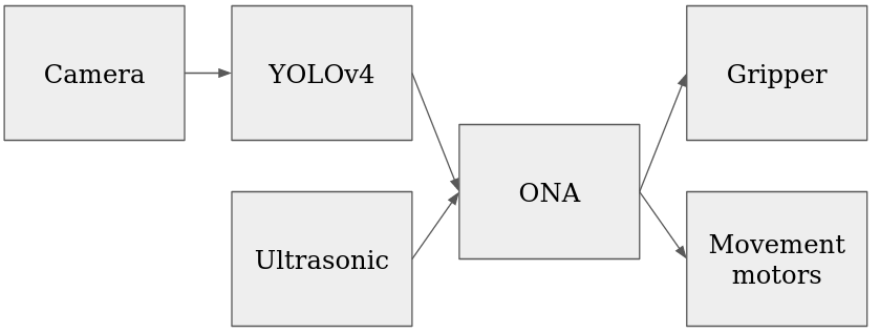


Figure 4.5: Components diagram

Due to the utilization of YOLOv4 trained on ImageNet, the vision channel can detect all object types in the dataset. The vision channel uses discretized relative location (relative to the center) information together with the outpiut label to form statements of the form:

$$< objectType \rightarrow [position] >$$

similar as in *Hammer et al.* (2020) but with relative location encoding where *position* for the X-axis is either $leftX$, $middleX$ or $rightX$, and for Y-axis is either $topY$, $middleY$ or $bottomY$. This encoding makes ONA aware of the detected object types and their position in the camera's field of view, as also needed to grasp objects (which is a common use for object detection *Jia et al.* (2018)). Additionally, the ultrasonic value is encoded into a statement

$$< obstacle \rightarrow [observed] >$$

if an obstacle is detected below 15 centimetres and

$$(\neg < obstacle \rightarrow [observed] >)$$

if above, which gives the robot a way to sense when in proximity to an obstacle. Additionally, movement of the robot is reported with an event which lets it sense whenever it moved forward successfully:

$$< \{SELF\} \rightarrow [moved] >$$

And last, the gripper provides events regarding whether it's open or closed:

$$< gripper \rightarrow [open] >, (\neg < gripper \rightarrow [open] >$$

.

Task 1: learning to maneuver around obstacles - The first task is about learning to explore while avoiding obstacles such as shown in 4.6. No background knowledge is given to the agent. There is only an innate goal to avoid low ultrasonic sensor values (below 15cm) encoded as $(\neg < obstacle \rightarrow [observed] >)!$ and a goal to keep moving, encoded as $< \{SELF\} \rightarrow [moved] >!$.

Figure 4.6: Robot in front of obstacles

In each run the robot was positioned at a different position inside of a closed

room. but as expected this experiment task did not depend on any particular starting position to succeed but simple on the occurrence of obstacles which the agent could learn to avoid. Across 10 trials after 2 encountered obstacles, the obstacle avoidance behavior was consistently learned, as directly checked by looking at the highest truth expectation implications the system has learned. The resulting statements are:

- When an obstacle is in front, rotating left leads to the obstacle to vanish (in 4 runs it learned the same with $\hat{r}ight$ instead, which of course is also fine):

$$< (< obstacle \rightarrow [observed] >, \hat{l}eft) \Rightarrow (\neg < obstacle \rightarrow [observed] >) >$$

- Invoking the forward operation without an obstacle in front leads to successful forward movement:

$$< ((\neg < obstacle \rightarrow [observed] >), \hat{f}orward) \Rightarrow < \{SELF\} \rightarrow [moved] >>$$

Then however, small objects the YOLOv4 model can detect, such as a red bottle, were placed in the room. This led to induction of the following relationships (via temporal induction as introduced earlier, but with variable induction based on common subterm between both events, as in *Wang* (2013a)) when the objects were encountered right or left to the robot and the corresponding operation was invoked:

- When an object is to the left, rotating to the left will make it appear in center, and the same for the right side:

$$< (< \$1 \rightarrow [left] >, \hat{l}eft) \Rightarrow < \$1 \rightarrow [centered] >>$$

$$< (< \$1 \rightarrow [right] >, \hat{r}ight) \Rightarrow < \$1 \rightarrow [centered] >>$$

Due the introduction of the variable at the place of the label term, these hypotheses are object-label-independent. They encode a spatial relationship which for objects generally holds, and encode behaviors the system can use to focus on arbitrary perceived objects through motor action. However, learning these hypotheses is more tricky than the more straightforward relationships for object avoidance behavior. Here, it's strongly dependent on the amount of detectable objects there are in its environment, and knocking over small objects can change their appearance and can sometimes make them undetectable. It took 6 object encounters on average (as expected as it can continue in current direction or rotate the wrong way with two thirds chance), 2 as minimum and 9 at most, across 10 attempted runs to learn both relationships. However, these implications can also be directly fed to the reasoner, bypassing the learning process. Most importantly, both representations represent sensorimotor contingencies that are independent from a particular object category or object instance even, so they gain evidence from encounters with all kind of detectable objects. Also, as we will now make use of, this generic procedure knowledge is automatically combined with more specific knowledge through the subgoaling process with variable binding.

Task 2: bottle collect mission - The second task shows the robot finding and retrieving a bottle as illustrated in Fig. 4.7. To succeed it it is fed with the knowledge learned from the previous task (move forward when no obstacle is observed, else rotate left) plus additional minimal background knowledge about the mission. The mission

knowledge only consists of picking a bottle if seen in front and dropping it if other bottles are seen in front:

- When the gripper is open, and a bottle is seen in the middle, the $\hat{p}ick$ operator should be invoked (to pick it up) in order to progress the mission, goal g:

$$< (< gripper \rightarrow [open] >, < bottle \rightarrow [centered] >, \hat{p}ick) \Rightarrow g >$$

- When the gripper is not open, and a bottle is seen in the middle, the $\hat{d}rop$ operator should be invoked (to drop the bottle to the other bottles), in order to progress the mission (again, encoded as goal g):

$$< ((\neg < gripper \rightarrow [open] >), < bottle \rightarrow [centered] >, \hat{d}rop) \Rightarrow g >$$

The experiment is designed to highlight three behaviors used in applicable circumstances: keep moving while avoiding obstacles, focus on and collect bottle if it appears in view, focus on other bottle to drop the held bottle to when it appears in view. This experiment has been repeated successfully 10 times with minor variations in starting angle and location, the movement trajectories have been visualized in Fig. 4.7. However, in more complex rooms more sophisticated exploration behavior is desired, in this case reliable long-term map memory, such as is usually captured with Visual Simultaneous Localization and Mapping (VSLAM) *Taketomi et al.* (2017) is beneficial. This would allow the agent to traverse the map in a more systematic way without visiting places multiple times if not necessary. Also, more recently neural map mechanisms such as *Parisotto and Salakhutdinov* (2017)) have been proposed, but have not yet reached the maturity needed for robotic application. While the purpose of this experiment is to show the ability to combine learned and given knowledge, our future work will also include VSLAM. A learned map would allow such tasks to

be performed more efficiently in more complex environments.

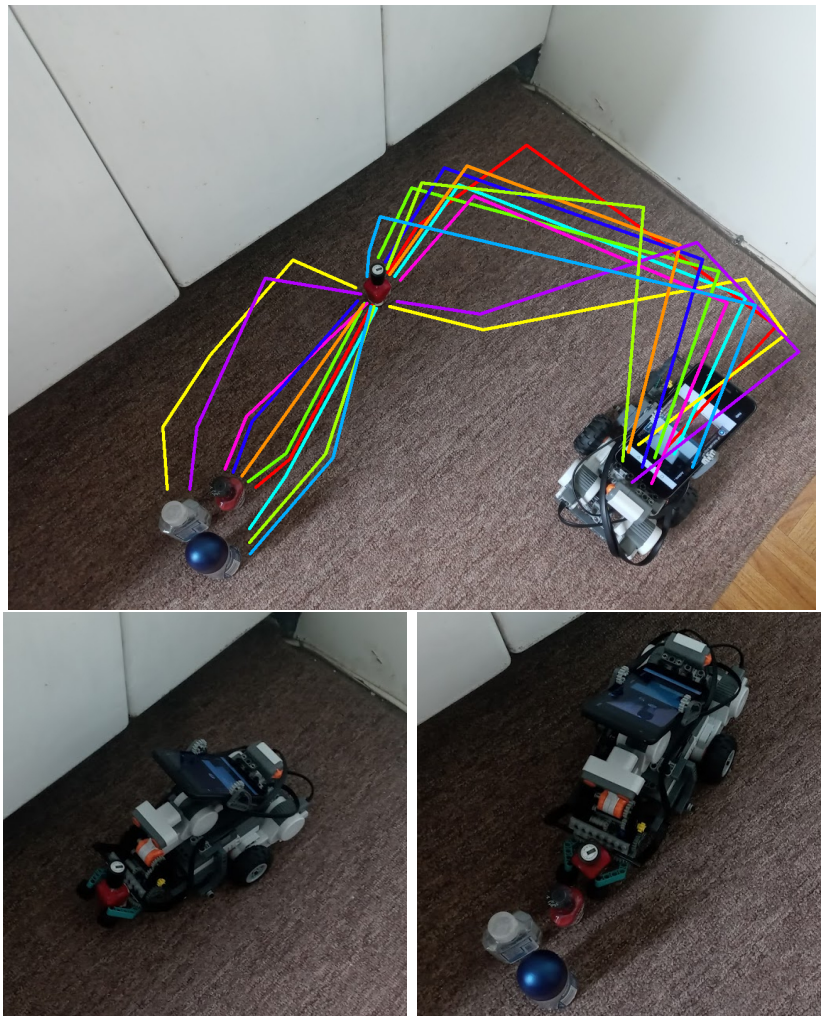


Figure 4.7: Bottle collect mission

In the current solution, the key mechanisms which allows to combine the learned and given knowledge is a common knowledge representation for both, and subgoal derivation. The latter allows to derive $< bottle \rightarrow [centered] >!$ as a subgoal to realize, which can then be used with the generic hypotheses described before (which is not restricted to bottles, it's a case of Transfer Learning *Zhuang et al.* (2020)) to focus on the bottle via right/left movement dependent on whether the bottle is left or right to the agent. Then, when the bottle is in the center, it's finally applying the mission knowledge to pick it up. Finally, it will unload it after the other bottles

have been found (via the movement and obstacle avoidance behavior) and brought to the center. Hence, the successful carrying out of the mission is a combination of learned and given knowledge. Also the ability to acquire lacking information (like how to avoid obstacles), or revise existing knowledge, at runtime, in a cumulative way *Thórisson et al.* (2019), is valuable. Through this, the agent has a chance to succeed even with simpler, partly incomplete or outdated knowledge of the mission. This concludes this experiment which is an attempt to show some initial merits of combining Reasoning and Learning for enhanced robot autonomy. And while the experiment is relatively simple, most autonomous robots do not possess real-time learning abilities to the demonstrated degree.

4.3.2 ONA-ROS Interface

Many available robotic platforms are running ROS (Robot Operating System), an unified platform which allows to develop code in a relatively hardware-agnostic manner and for multiple robots and robotic hardware. Having ONA available to run in ROS has become an important goal as it makes it easier for our collaborators and other robotics teams to utilize our technology.

Many modules are available for ROS, including modules for sensors and actuators, different AI algorithms for Planning, Machine Learning models, Computer Vision, Simultaneous Localization and Mapping, Multi-Agent Communication and so son. ONA can especially be used as a replacement for BDI models and Practical Reasoning modules, which are often used as the central decision making modules in autonomous robotic solutions. Since ONA can acquire and update knowledge at runtime, experiments like the previous become possible, providing a key advantage when autonomy is concerned.

In ROS, Module I/O follows a Publish-Subscriber model. For ONA this is straightforward to utilize:

- **Input:** Narsese encodings
- **Output:** Operator executions

The ONA ROS module has been designed accordingly. Additionally, ROS modules for the vision channel consisting of the YOLOv4 darknet-ros module, and a YOLO-output-to-Narsese encoder module, has been created. This essentially wraps the vision channel which was also used in the SpongeBot experiments into a ROS module compatible with the ONA ROS module. In addition, an Narsese encoder for the visp_auto_tracker OCR code reader ROS module has been added, to allow for task-specific landmarks to be printed and identified as often done in robotics.

The module interaction structure is as follows (output via RosGui, Fig. 4.8):

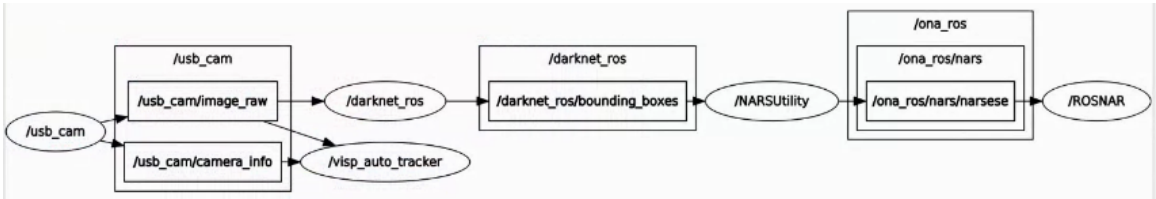


Figure 4.8: How the ROS modules are connected

Finally, RosGui was used for real-time monitoring to see if the modules are correctly working together and exchange messages properly as shown by Fig. 4.9:

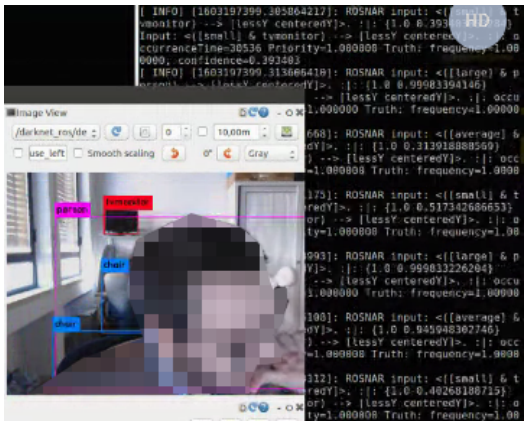


Figure 4.9: Running modules visualized by RosGui

Our ROS modules working properly entails that experiments with more sophisticated ROS-based robots are now possible as well. Experiments featuring such robots will be carried out in the future.

4.4 SmartCity Application

4.4.1 Introduction

Supervised Deep Learning has generated a lot of success in image classification, object detection and tracking. Integrating principles from neural networks and logical reasoning, opens up new possibilities for anomaly prediction and detection. In this collaboration project between Temple University and Cisco Systems in which I was involved, NARS is utilized, and receives tracklets provided by a Multi-Class Multi-Object Tracking (MC-MOT) system in real-time. The system infers spatial and temporal relations between events, describing their relative position and predicted path.

The multi-class and multi-object tracker was developed by Cisco Systems Inc, and the reasoning-learning component utilized is the OpenNARS implementation *Hammer et al.* (2016), and alternatively ONA. We tested our approach on a publicly available dataset, 'Street Scene', which is a typical scene in the Smart City domain, and later deployed it on real hardware in Australia. The dataset is video data obtained by a street cam mounted on a building. We have shown the system learning to classify scene regions, such as street or sidewalk, from the typical 'behaviour' of the tracked objects belonging to three classes, such as car, pedestrian, and bike. The system is shown to autonomously satisfy a range of goals to detect and inform the user of certain anomaly types. An anomaly ontology supports various anomaly classes; location based, relational based, velocity based, or vector based. From this anomaly ontology a range of specific anomalies can be detected; jaywalking, pedestrian in danger, cyclist

in danger, traffic entity too fast, traffic stopped, pedestrian loitering, vehicle against flow of traffic, etc. The current implementation supports the detection of 'street' and 'sidewalk' regions along with pedestrian in danger and jaywalking anomalies. The system learns in real-time and is capable of detecting anomalies after a cold start of few seconds of operation with no prior training. As well as the autonomous goal satisfying the system can also respond to user questions in real-time.

The effectiveness of learning mappings for question-answering has been demonstrated by *Mao et al.* (2019), at least for simple domains. However, their approach requires $(image, question, answer)$ triples, essentially requiring questions to be provided at training time. This is not feasible in cases where novel questions are input by an operator at any time and require a real-time response. Also their model has no time-dependence and no way to make use of background knowledge, while the system we introduce has a notion of time and has the ability to use background knowledge, allowing it to identify user-relevant situations, and to make predictions about possible future situations. The prediction capability also potentially allows the system to identify anomalies before they occur rather than detecting them after the event occurs, though our anomaly detection so far concentrates on recent input, on the current situation.

4.4.2 Architecture

Generally, our solution is closer to what is described as the first possibility in *Alirezaie et al.* (2018), namely the output of a sub-symbolic system (an object tracker in our case) is used as input to the reasoning component (NARS in our case). This differs from the second case proposed in *Alirezaie et al.* (2018) and followed in *Garcez et al.* (2019), to integrate the reasoning ability in the sub-symbolic architecture, such as within the hidden layers of a Convolutional Neural Network. We found the former approach easier to realize, as it allowed us to decouple the different requirements

between both components. Following this overall idea, the components are the following:

Multi-Class Multi-object Tracker is a main component of the distributed asynchronous real-time architecture, and allowed us to produce scalable applications. The overall system is made up of highly decoupled, single-purpose event processing components that asynchronously receive and process events. In particular, a video streamer based on GStreamer pipelines that receives video flows from multi-source streaming protocols (HLS, DASH, RTSP, etc.) and distributes video frames in compressed or raw formats, a multi-object Deep Neural network (DNN) detector based on YOLOv3 *Redmon and Farhadi* (2018), and a multi-class multi-objects tracker (MC-MOT) *Jo et al.* (2017). The outputs of the MC-MOT together with the input video frames are merged and sent in sync to the OpenNARS-based visual reasoner for further processing. Interprocess communication and data exchange within the different components is done using Redis, an in-memory data structure store configured as LRU cache for reliable online operations.

The tracking problem is about how to recognize individual objects in such a way, that they keep their ID while moving continuously in time. We used a tracking-by-detection approach for multi-objects (MOT) that has become increasingly popular in recent years, driven by the progress in object detection and Convolutional Neural Networks (CNN). The system performs predictions with a Kalman filter, and linear data association using the Hungarian algorithm, linking multiple object detections coming from state-of-the-art DNN detectors (e.g. YOLOv3) for the same class in a video sequence. The multi-class multi-object tracker extends this concept to tracking for objects belonging to multiple classes (MC-MOT) by forking a MOT for each class of interest. In this project we limit our solution to three classes: person, bike, and car. The CNN multi-object detector publishes the bounding boxes and object detection positions for the bounding box center (x, y), width and height

(w, h) respectively. The three MOTs subscribe to a specific object class and receive the corresponding object's detection. The output of each MOT is represented by a tracklet i.e., the fragment of trajectory followed by the moving objects. Each tracklet has a class ID, an instance ID, and a sequence of the previous five bounding box detections $(t_1, x_1, y_1, w_1, h_1), ..., (t_5, x_5, y_5, w_5, h_5)$, where t_i is the timestamp of the detection, x_i, y_i the location in pixel units on the X-axis and Y-axis, and w_i, h_i the width and height of the bounding box. The detection mAP is on average 80%, while it takes 30ms for objects detection on an NVIDIA Tesla P100 board, and 15ms for tracking. The object detection is based on Yolo v3 with network image size 768x768 for easy detection of small objects. It is worth noting that the model was re-trained on the Street Scene dataset (see *Ramachandra and Jones* (2020)) but then also tested on similar scenarios, using available webcams (like the ones our solution is deployed on), to ensure wider generalization abilities. These performances are acceptable for typical real-time smart city applications, since we can reliably send tracklets and frames at 15fps to the OpenNARS reasoner. Actually, detection and tracking performances can be controlled by the reasoner for a better street scene understanding, thanks to the richer symbolic description of objects behavior and their relationships described by the ontology. As an example, broken tracklets caused by occlusions or misdetections were merged by the reasoner and re-identified.

Tracklets to Narsese To map tracklets to NARS events, the numeric information encoded in each tracklet is discretized. This can happen in many ways. We used a fixed-sized grid that maps every detection tuple to the rectangle it belongs to, as shown in Fig. 4.10:

Figure 4.10: Spatial discretization by a grid

Additionally the class and instance ID is provided, which (currently) can be *Car*, *Bike* or *Pedestrian*. Now using the above, the following events can be built:

- Indicating the class of an instance:

  $\{InstanceID\} \rightarrow Class$

  Currently *Class* can either be *Car*, *Pedestrian* or *Bike*, and for regions *street*, *sidewalk*, *crosswalk* or *bikelane*.

- Indicating the direction of an instance:

  $(\{InstanceID\} \times \{Angle\}) \rightarrow directed$

- Indicating the position of an instance:

  $(\{InstanceID\} \times \{RectangleID\}) \rightarrow positioned$

- To reduce the amount of input events, also combinations are possible:

  $(\{InstanceID\} \times \{RectangleID\} \times \{Angle\} \times \{Class\}) \rightarrow Tracklet$

- The system learns to assign *street*, *sidewalk*, *bikelane* and *crosswalk* labels to the scene, based on the car and pedestrian activity. This is achieved through the use of implication statements:

  $((\{\#1\} \rightarrow Car); ((\{\#1\} \times \{\$2\}) \rightarrow positioned)) \not\Rightarrow (\{\$2\} \rightarrow street).$

$((\{\#1\} \rightarrow Pedestrian); ((\{\#1\} \times \{street\}) \rightarrow parallel); ((\{\#1\} \times \{\$2\}) \rightarrow positioned)) \not\Rightarrow (\{\$2\} \rightarrow sidewalk)$.

$((\{\#1\} \rightarrow Pedestrian); ((\{\#1\} \times \{street\}) \rightarrow orthogonal); ((\{\#1\} \times \{\$2\}) \rightarrow positioned)) \not\Rightarrow (\{\$2\} \rightarrow crosswalk)$.

where the additional *orthogonal* and *parallel* relation indicates whether the entity is orthogonal to the closest region labelled street, which the reasoner tracks by revising the directions of entities that appear on the related location, choosing the direction that has the highest truth expectation to decide the truth of the relationship.

Whenever the consequence is derived, it will be revised by summarizing the positive and negative evidence respectively, and then the label of highest truth expectation (either *street* or *sidewalk*) to answer the question $\{specificPosition\} \rightarrow ?X$ will be chosen.

- In addition, relative location relations R are provided by the system, including $leftOf$, $rightOf$, $topOf$, $belowOf$ and $closeTo$. These are encoded by $(\{InstanceID1\} \times \{InstanceID2\}) \rightarrow R$

  Please note that the closeTo relation is only input when the distance is smaller than some threshold defined by the system operator.

4.4.3 Results

Street Scene (see *Ramachandra and Jones* (2020)) is a dataset for anomaly detection in video data. It is data collected from a camera mounted on a building, watching a street. The dataset includes unusual cases that should be detected, such as jaywalking pedestrians, cars driving on the sidewalk, or other atypical situations. The tracker applied to the video dataset, outputs tracklets as introduced previously, which are then encoded into Narsese as described. NARS then can use the input

information to make predictions, satisfy goals, or to answer queries in real time. Also NARS can detect anomalies and classify them with a background ontology.

Reasoning-based annotation In the Street Scene dataset, our system is able to label a location (discretized according to the lattice) as street based on the tracklet activity of pedestrians and cars. This is done by the reasoner, which utilizes the related background knowledge expressed by the implication statements we have seen. The labelling usually happens in an one-shot way, but will be overridden or strengthened by further evidence, using Revision (see *Wang* (2013a)).

Question Answering Also in Street Scene, we tested the system's ability to answer questions about the current situation, in real time, demonstrating situational awareness about the current state of the street. Questions included $((\{?1\} \rightarrow Class); (\{?1\} \times \{LocationLabel\}) \rightarrow located)?$ where ?1 is a variable queried for, essentially asking for an instance of a specific class at a specific location such as *lane1*, which the system returns immediately when perceived, allowing a user, for instance, to ask for jaywalking pedestrians.

Anomaly Detection Often a system should not operate passively (answer queries), but automatically inform the user when specific cases occur. Our approach allows the usage of background knowledge to classify unusual situations, and to send the user a message, if desired.

For instance, consider the case of a moving car getting too close to a pedestrian, putting the person in danger. This can easily be expressed in Narsese, using the previously mentioned relative spatial relations the system receives. Furthermore, it can be linked to a goal, such that the system will inform the user whenever it happens:

$(((\{\#1\} \rightarrow Car); (\{\#1\} \rightarrow [fast]); (\{\#2\} \rightarrow Pedestrian); ((\{\#1\} \times \{\#2\}) \rightarrow closeTo)), say(\#2, is_in_danger)) \not\Rightarrow (\{SELF\} \rightarrow [informative])$. which will let the system inform the user assuming the goal $(\{SELF\} \rightarrow [informative])!$ was given to the system. An example can be seen in Fig. 4.11:

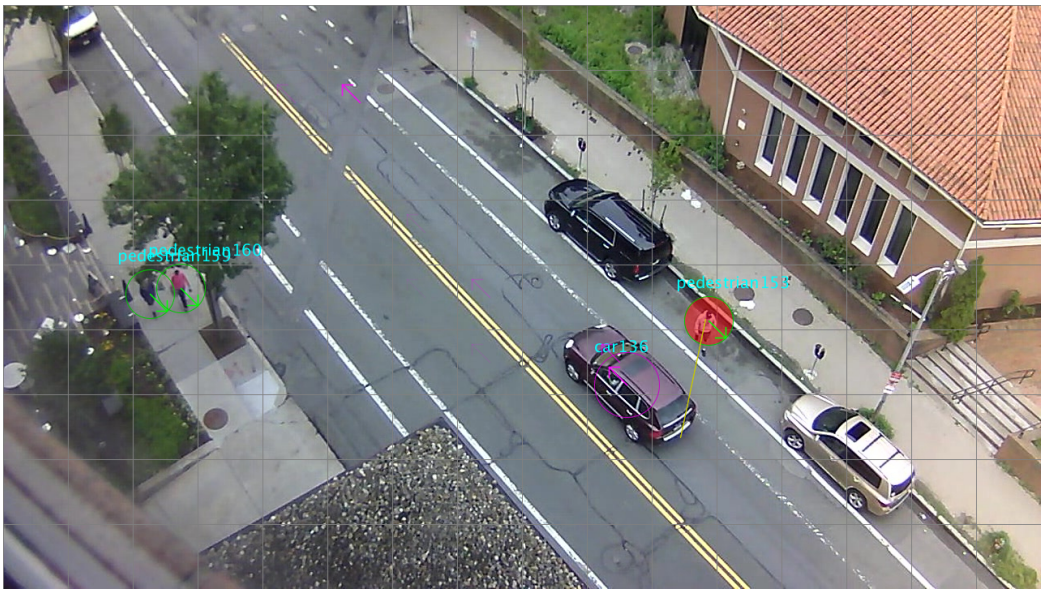


Figure 4.11: A pedestrian in danger due to close proximity to a quickly moving car

Also jaywalking pedestrians can be specified similarly, using $((((\{\#1\} \rightarrow Pedestrian); ((\{\#1\} \times \{street\}) \rightarrow at)), say(\#2, is_jaywalking))$ $/\!\!\Rightarrow (\{SELF\} \rightarrow [informative])$. An example can be seen in Fig. 4.12. As before, the anomaly is reported by the system through the invocation of the *say* operator with the instance name given as argument:

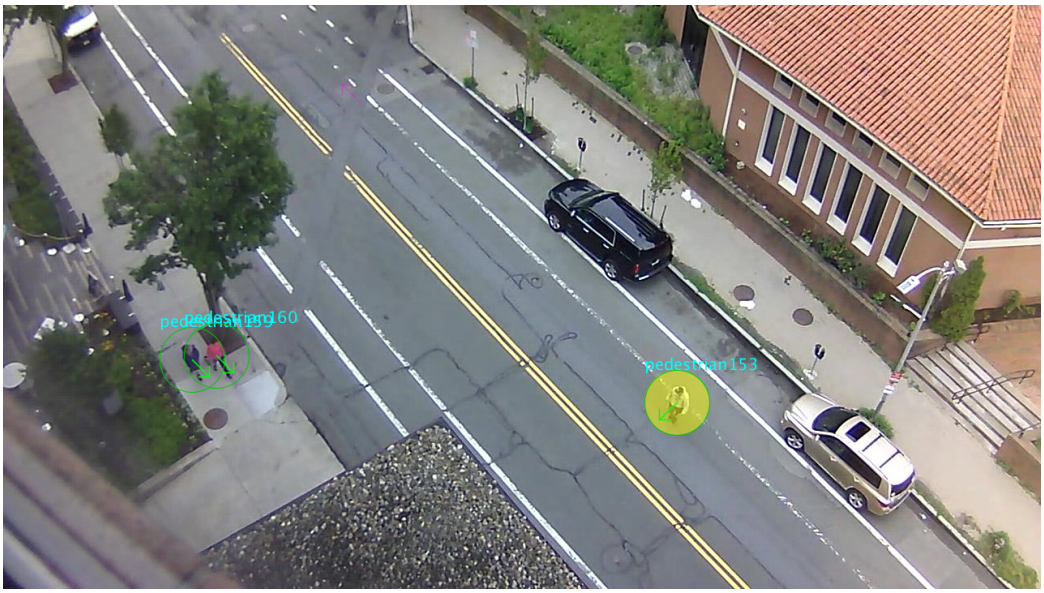


Figure 4.12: An example of Jaywalking

In our tests, the system reported 97 percent and 94 percent of the jaywalking and danger anomaly cases present in the Street Scene dataset after an initial runtime of 5 minutes to map the streets based on car tracklet behavior. The misdetections reported in Table 4.2 are mostly from grid resolution artifacts and corner cases where the ground-truth is questionable.

Table 4.2: Performance in Street Scene

Initial learning time	5 minutes
Detected jaywalking anomalies percentage	97%
Detected danger anomalies percentage	94%

4.5 TruePAL Application

4.5.1 Introduction

Trusted and explainable Artificial Intelligence for Saving Lives (TruePAL) is a collaboration project with NASA Jet Propulsion Laboratory, funded by the department of transportation, and part of the Next Generation First Responder (NGFR) Apex Program. The purpose is to assist and warn first responders by providing contextually relevant information when it is critical to do so. Hereby, information overload should be avoided, and warnings have to be explainable by the AI system. Hardware-wise, the First Responder (FR) vehicle is equipped with a large set of sensors, including camera, Radar, vehicle-to-vehicle communication, database access, and so on, which demands data fusion from many sensor sources to work properly. TruePAL faces similar challenges as autonomous driving solutions, though driving assistance is only a small part of the project, it's also able to assist with the mission and to give guidance regarding different types of incidents (such as electric vehicles). Besides that, it also works for first responders outside of vehicles, operating on their smartphones, and needs to connect the information from different first responders.

One tricky aspect about the project is that reasoning needs to happen in real-time, with sensor information constantly streaming in from multiple sensors. NARS, which was designed as a real-time system from the very beginning, and hence also the ONA implementation, work well under these working conditions. Additionally, the system's ability to learn in real-time can enhance the warning behavior of the system, which will be useful to reduce the generation of warnings which conditions they are triggered under need to be adjusted, as we will see. Please also note that the system can not only generate warnings but also explain why they were given, which is crucial for explainability.

4.5.2 Architecture

System diagram As visualized by the highly simplified diagram Fig. 4.13 NARS and related components of TruePAL consist of object detection, real-time object tracking, a database for vehicle-to-vehicle communication and querying of historic incident information, and ONA *Hammer and Lofthouse* (2020).

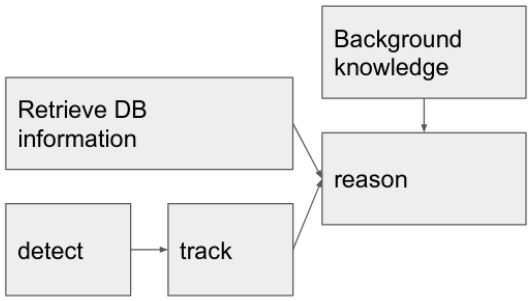


Figure 4.13: TruePAL Architecture

Like in the SmartCity project, YOLOv4 is used for object detection. The detections then enter the object tracker which estimates speed, movement velocity and angle of objects, and to assign an instance ID which remains stable across time / across multiple frames. Compared to the SmartCity project, the camera is mounted on the first responder vehicle, hence also the speed of the FR vehicle (to distinguish

own movement from object movement), and the distance (as estimated by Radar) is taken into account by the object tracker to reach higher accuracies. This tracking approach can be considered a direct extension of *Bewley et al.* (2016), using a Kalman filter for tracking, but with a linear dynamics model which also takes first responder vehicle speed and depth values (obtained by Radar and LIDAR) into account in addition to the bounding box coordinates. Here, matches with a depth difference above a threshold are rejected, which significantly improves the accuracy over a purely screen-space tracking approach.

Narsese encodings The inputs to NARS ("reason" block) result in the following event encodings, each of which are related to the instance which is paired with its class ID (car, pedestrian, first responder vehicle etc.).

- A property indicating if a certain instance is approaching the first responder vehicle or not.

  ```
  <(ClassID * InstanceID) --> [ApproachingOrLeaving]>.
  ```

- A property indicating the direction the instance is located at.

  ```
  <(ClassID * InstanceID) --> [Direction]>.
  ```

- A property which indicates the instance speed exceeds a certain number of interest.

  ```
  <(ClassID * InstanceID) --> [aboveNumberMPH]>
  ```

- Whether the instance drives within or outside of the lane. (where the FR vehicle is parked)

  ```
  <(ClassID * InstanceID) --> [InOrOutOfLane]>
  ```

- A property which indicates that the first responder is approaching an intersection with a certain trafficlight color.

```
<(intersection * color) --> [approaching]>
```

- A property which indicates that the first responder is approaching an intersection a different first responder is also expected to cross now.

```
<(intersection * color) --> [expectedFR]>
```

- General risk assessment of the intersection by the reasoner, based on historic accidents and reasoning which utilizes learned temporal relationships.

```
<(intersection * color) --> [risky]>
```

Background knowledge The background knowledge consists of critical situations under which a warning should be triggered:

- Generate a warning in case of a risky intersection to avoid collision:

```
<(<(intersection * #1) --> [risky]> &/ ^warning)
  =/> (! <{SELF} --> [collided]>)>.
```

- Generate a warning if another first responder is expected to cross the same intersection

```
<((<(intersection * #1) --> [approaching]> &/
   <(intersection * #1) --> [expectedFR]>) &/ ^warning)
     =/> (! <{SELF} --> [collided]>)>.
```

- Generate a warning if an entity is approaching us centered:

```
<(<(#entity * #id) --> [approaching centered]> &/ ^warning)
  =/> (! <{SELF} --> [collided]>)>.
```

- Roadside parking risk warning, from a vehicle which quickly approaches the first responder vehicle in the lane it is parked in:

```
<(<(car * #nr) --> [fast inLane]> &/ ^warning)
  =/> (! <{SELF} --> [collided]>)>.
```

If we are above KMPH approaching a red light intersection, a warning will lead the officer to go below KMPH:

```
<((<{SELF} --> [aboveKMPH]> &/
   <(intersection * red) --> [approaching]>) &/ ^warning)
     =/> <{SELF} --> [belowKMPH]>>.
```

4.5.3 Results

The latter hypothesis which we also gave is background knowledge with a reasonable initial value, is also learnable. This is, because the warning will be ignored by the first responder if the speed limit is given for a too low speed. So the implications with too low speed limit receives negative evidence from predictions which won't be confirmed (the first responder will still drive with above K miles per hour for a given K when the warning was given but ignored by the first responder), hence it won't be used after a few occasions. Also the previous implications are of course learnable and can be supplemented with additional observed relationships, though so far only the adaptation of the speeding warning behavior has been demonstrated. In this case multiple events such as

```
//the FR vehicle moves above 35MPH
<{SELF} --> [above35MPH]>. :|:
//We are approaching a red intersection
```

```
<(intersection * red) --> [approaching]>. :|:
//Generation of warning
^warning. :|:
```

enter the system in succession, in which case the system is expected to update the truth value of the hypothesis

```
<((<{SELF} --> [above35MPH]> &/
   <(intersection * red) --> [approaching]>) &/ ^warning)
     =/> <{SELF} --> [below35MPH]>>.
```

since the event to be below 35 MPH, is usually not observed when the warning is ignored, providing negative evidence, while when observed it provides positive evidence to the hypothesis.

Also other observations, many corresponding to common-sense knowledge are easily learnable by the system from observation, which includes knowledge such as

```
<(<(intersection * red) --> [approaching]> &/
   <(<($type * $id) --> [approaching left]>))
     =/> <($type * $id) --> [leaving right]>>.
```

which encodes that entities (such as cars and trucks) can pass from the left to the right when we encounter a red light intersection. (since for them, the traffic light is green) This capability is especially useful for QA, as the system can be asked about the behavior of specific vehicles, which adds to the explainability of the system: the ability to list warning-relevant conditions when they are observed with sufficient certainty.

Other than investigating use cases for learning, such as to adapt to the behaviors of the first responders and other vehicles, the main focus so far was to reliably detect in real-time the following risky situations which are already detected by TruePAL:

1. Ability to predict collision in intersections. This demands the detection and tracking of the instances based on visual input (via YOLOv4) and Radar input.

Figure 4.14: TruePAL collision detection

2. Ability to warn when other first responders approach the same intersection based on vehicle-to-vehicle communication. For this to work, ONA is querying a database where the telemetry of each first responder vehicle is streamed to in real-time.

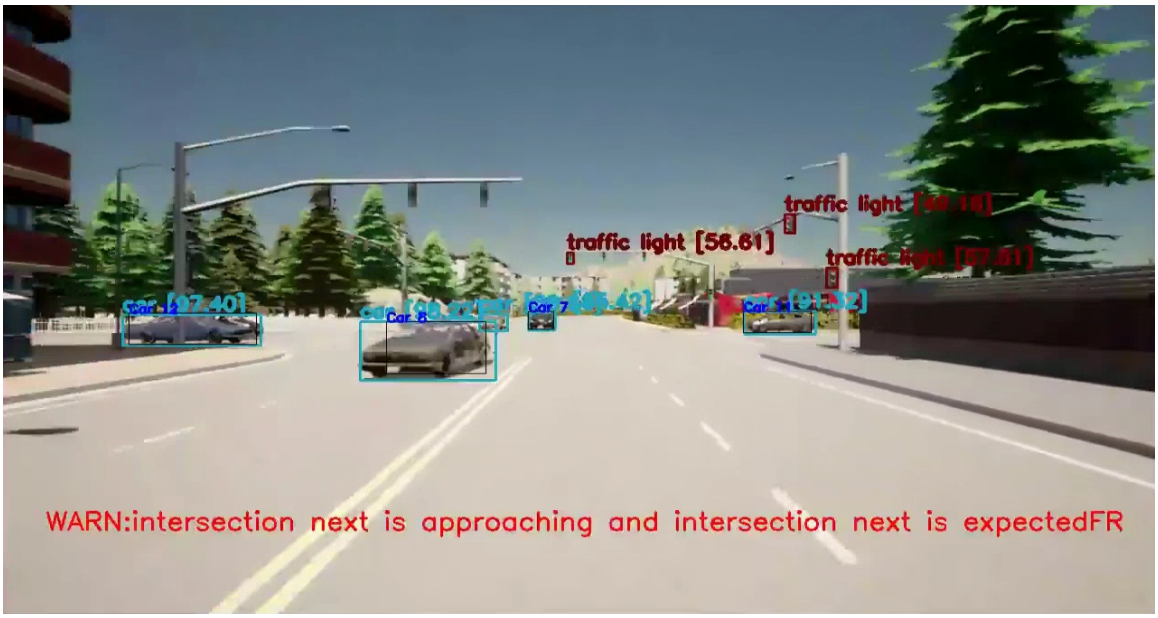


Figure 4.15: TruePAL expected first responder

3. Ability to warn of risky streets based on a database of past accidents. Such databases are widely available and are queried by the reasoner.

Figure 4.16: TruePAL risky street

4. Speeding warning when approaching a red-light intersection with too high speed. The speed of the own vehicle is known directly from the speedometer and also

from GPS coordinates.

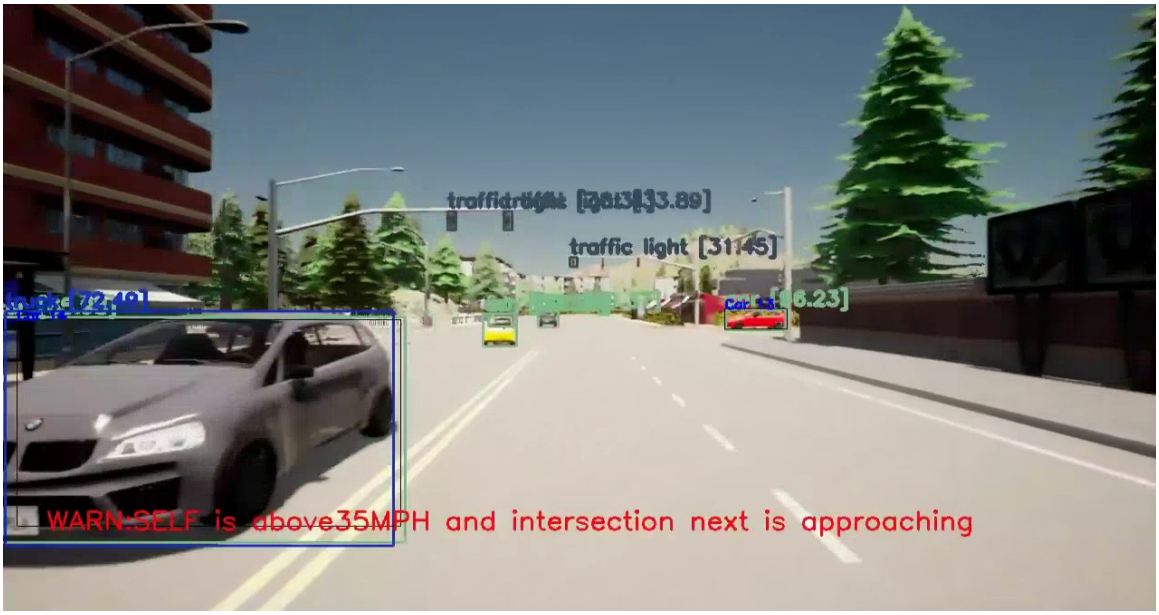


Figure 4.17: TruePAL speed warning

5. Roadside safety, warning from speeding vehicles in same lane or outside of lane, utilizing Radar and visual detection of lanes. If the vehicle switches to the inner lane in time, no warning is generated. The detection of lanes currently happens via canny edge detection and best-match fitting of two lines via OpenCV, which will be supplemented with a semantic segmentation approach in the future.

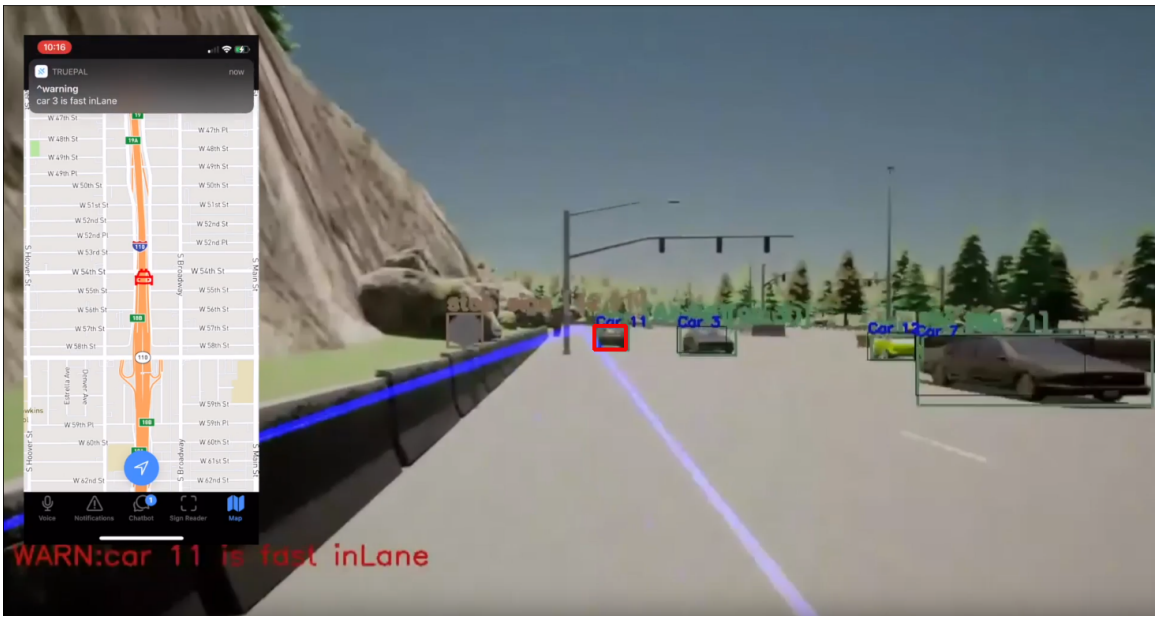


Figure 4.18: TruePAL roadside safety

TruePAL is an ongoing project for which, so far, only basic testing on constructed CARLA *Dosovitskiy et al.* (2017) scenarios has been performed. In these five scenarios, the above capabilities have been convincingly demonstrated, not missing warnings except for the roadside safety and collision detection case which are more tricky. There, the warning rates were however at least above 80 percent in the preliminary tests, which seems promising, and the given explanations for the warning consistently matched the situation. However, detailed performance metrics have yet to be developed and evaluated, which will be crucial once the project stabilizes and moves to real hardware. So far, the shown use cases work reliably in the constructed CARLA test cases, though improvements of various components are underway. Most importantly, the system can not only generate warnings but also explain why they were given, which makes the system more trustable.

CHAPTER 5

Discussion

5.1 Conclusion

In this thesis, an attempt to design an efficient real-time learning system for enhanced agent autonomy was made. To be effective, it has to solve the Temporal Credit Assignment Problem for arbitrary outcomes comparably well like Reinforcement Learning techniques do for a single reward signal, which a comparison on similar examples reveal. This capability is the basis to allow goal-directed behavior learning at runtime to happen, an ability which is especially beneficial for robot control which traditionally, when a Practical Reasoning approach (BDI with planning, typically) is taken, demand all relevant background knowledge to be given by the designer. These real-time learning abilities are also beneficial to other practical applications, such as driving assistance and traffic surveillance. Generally speaking they matter whenever the system's knowledge is insufficient, which can happen in novel or changing circumstances, and especially in complex environments such as the real world. These are cases where the designer cannot prepare the system, in advance, for all situations it will possibly encounter, hence adaptation at runtime is required.

To achieve these goals, the ONA system has been created, utilizing Non-Axiomatic Reasoning System theory and system design considerations which have been the result of years of research on NARS and experimentation with OpenNARS. ONA's biggest strength and for reasoners rare ability is to learn in real-time (differently than *Muggleton* (1991) and *Muggleton and De Raedt* (1994)), while using for reasoners common principles like goal derivation and planning based on goal-independent

representations.

It has been shown that on the particular set of tasks (Pong, Space Invaders, and grid robot), 'OpenNARS for Applications' *Hammer and Lofthouse* (2020), on average, performs comparably well like Q-Learning with Eligibility traces *Sutton* (1988) *Sutton and Barto* (2018), while also working when the Markov property for states and rewards is violated. This suggests the utilization of uncertainty reasoning (Non-Axiomatic Reasoning *Wang* (2013a) in particular) to be an alternative approach, and sometimes an improvement, to deal with various Reinforcement Learning problems. Most importantly it addresses some of the inherent limitations in common Reinforcement Learning techniques. For instance the Markov property of states and rewards, as we saw, can be easily violated even in relatively simple environments. This also makes it slightly easier to apply the technique, as the designer doesn't need to worry as much about the details of the environment and interface to the agent. Then, we have shown, and backed up with a real-world robotics use case which utilizes a state-of-the-art object detection model *Bochkovskiy et al.* (2020), that the reasoning-based approach naturally allows for the flexibility means-end Reasoning (Practical Reasoning *Bratman et al.* (1987) *Georgeff et al.* (1998)) approaches are typically known and valued for. This includes the ability to change behaviors immediately when goals change, to be able to plan to reach outcomes which have not necessarily been observed before in the same manner, to pursue multiple goals, and to take easily by human understandable and communicable background knowledge into account effectively. Additionally, we have seen potential of the system to be used as driving assistance for First Responders, and traffic surveillance using reasoning-based anomaly detection.

In the broader picture, and within the history of AI, it was long believed that reasoning is the pinnacle of natural intelligence, yet the entirely deduction-based symbolic systems have fundamentally failed to explain or replicate the adaptation abilities and ultimately autonomy of animals with high levels of cognition and real-

time learning abilities. I hope I have convinced the reader that a broader notion of reasoning which includes Induction and Abduction, namely Non-Axiomatic Reasoning, can be a viable direction towards this goal when crucial design challenges are addressed, like it was attempted in ONA. Hereby, also the combination with state-of-the-art techniques in Deep Learning, especially for sensor processing (vision in particular), has turned out to be fruitful.

5.2 Future works

Overall, this technology can be used to more effectively build systems which reach higher degrees of autonomy through robust, evidence-driven, reasoning and learning at runtime. May this technology open the doors for the next generation of autonomous systems, and allow to incrementally reach crucial to autonomy related goals:

- Allow the agent to explore its environment. (exploration)
- Allow it to learn from observations and interactions. (adaptation)
- Allow it to effectively remember places, objects, other agents and their properties as a special case. (modeling of the world)
- Allow it to physically manipulate objects with actuators. (play, tool usage)
- Demand it to be able to operate also without a human. (independence)
- Allow it to extract useful information out of human speech, to answer questions, and to use language in other ways. (communication)

Making progress towards this goals will benefit many kinds of intelligent agents, especially in the field of robotics: robotic explorers, robots as a housemaid, robots for automated mobility, and other robots which need to interact with their environments and sometimes with people. A reasoner which can deal with knowledge insufficiency

and learn from experience can allow a robot to adapt not only to unknown environments but also to the specifics of the interactions with other agents (such as humans). This includes an ability to adopt behavior exhibited by other agents, to predict their intentions, to make corresponding suggestions, and to interpret user input as a mission to complete.

This thesis has only scratched the surface of the quest towards greater autonomy of intelligent agents via real-time learning, and more research work in this direction is planned by the author for the future, together with more design and development work to get rid of known limitations. I also hope this work gives inspirations towards finding ways to combine the strengths of Machine Learning models and Reasoning in the NARS sense, with the ultimate goal being that research on this will bring into existence powerful techniques which have key features and core strengths of both. Techniques which hopefully will be less brittle when operating in real-world environments, especially in these in which data-efficient learning at runtime (as also emphasized by *Kunze et al.* (2018)) is often crucial for an agent to succeed, or at least helpful in exceptional circumstances.

Our experiments can be fully replicated with the code in the ONA open source code repository *Hammer* (2020a), also detailed building instructions for the in the experiment utilized robot do exist in a repository *Hammer* (2020b).

APPENDIX A

Grid

A.1 ONA: Sensorimotor inference rules

Table A.1: Truth functions used for sensorimotor reasoning

Truth_w2c(w)	w / (w + TRUTH_EVIDENTAL_HORIZON)
Truth_c2w(c)	TRUTH_EVIDENTAL_HORIZON * c / (1 - c)
Truth_Expectation((f,c))	c * (f - 0.5) + 0.5
Truth_Revision((f1,c1),(f2,c2))	(MIN(1.0, (w1 * f1 + w2 * f2) / w), MIN(MAX_CONFIDENCE, MAX(MAX(Truth_w2c(w), c1), c2)))
Truth_Deduction((f1,c1),(f2,c2))	(f1 * f2, c1 * c2 * f1 * c2)
Truth_Abduction((f1,c1),(f2,c2))	(f2, Truth_w2c(f1 * c1 * c2))
Truth_Induction((f1,c1),(f2,c2))	Truth_Abduction((f2,c2),(f1,c1))
Truth_Intersection((f1,c1),(f2,c2))	(f1 * f2, c1 * c2)
Truth_Eternalize((f1,c1))	(f1,Truth_w2c(c1))
Truth_Projection((f1,c1),t1,t2)	(f1, c1 * pow(DECAY,abs(t1 - t2)))

```
{Event a.} |- Event a. Truth_Projection (projecting to current time)

{Event a., Event b.} |- Event (a &/ b). Truth_Intersection (after projecting a to b)

{Event a., Event b.} |- Implication <a =/> b>. Truth_Eternalize(Truth_Induction) (after projecting a to b)

{Implication <a =/> b>., <a =/> b>.} |- Implication <a =/> b>. Truth_Revision

{Event b!, Implication <a =/> b>.} |- Event a! Truth_Deduction

{Event (a &/ b)!, Event a.} |- Event b! Truth_Deduction

{Event a!, Event a!} |- Event a! Truth_Revision or Choice (dependent on evidental overlap)

{Event a., Implication <a =/> b>.} |- Event b.  Truth_Deduction
```

Table A.2: Additional truth functions used for declarative reasoning

or(a,b)	1.0 - (1.0 - a) * (1.0 - b)
Truth_Exemplification((f1,c1),(f2,c2))	(1.0, Truth_w2c(f1 * f2 * c1 * c2))
Truth_Comparison((f1,c1),(f2,c2))	((if (f0 == 0.0) then 0.0 else ((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 * or(f1, f2))
Truth_Union((f1,c1),(f2,c2))	(or(f1, f2), c1 * c2)
Truth_Difference((f1,c1),(f2,c2))	(f1 * (1.0 - f2), c1 * c2)
Truth_Conversion((f1,c1))	(1.0, Truth_w2c(f1 * c1))
Truth_Negation((f1,c1))	(1-f1, c1)
STRUCTURAL_TRUTH	(1.0, 0.9)
Truth_StructuralDeduction((f1,c1))	Truth_Deduction((f1,c1), STRUCTURAL_TRUTH)
Truth_StructuralDeductionNegated(v1)	Truth_Negation(Truth_StructuralDeduction(v1))
Truth_DecomposePNN((f1,c1), (f2,c2))	(1.0 - (f1 * (1.0 - f2)), (f1 * (1.0 - f2)) * c1 * c2)
Truth_DecomposeNPP((f1,c1), (f2,c2))	(((1.0 - f1) * f2), ((1.0 - f1) * f2) * c1 * c2)
Truth_DecomposePNP((f1,c1), (f2,c2))	((f1 * (1.0 - f2)), (f1 * (1.0 - f2)) * c1 * c2)
Truth_DecomposePPP(v1, v2)	Truth_DecomposeNPP(Truth_Negation(v1, v2), v2)
Truth_DecomposeNNN(v1, v2)	(1.0 - ((1.0 - f1) * (1.0 - f2)), ((1.0 - f1) * (1.0 - f2)) * c1 * c2)

A.2 ONA: Semantic inference rules

The following are all the semantic inference rules ONA uses:

- NAL-1: Inference rules regarding inheritance:

```
{ (S --> M), (M --> P) } |-, (S --> P), Truth_Deduction
{ (A --> B), (A --> C) } |-, (C --> B), Truth_Induction
{ (A --> C), (B --> C) } |-, (B --> A), Truth_Abduction
{ (A --> B), (B --> C) } |-, (C --> A), Truth_Exemplification
```

- NAL-2 Inference rules regarding similarity:

```
{ (S <-> P) } |- (P <-> S), Truth_StructuralDeduction
{ (S --> {P}) } |- (S <-> {P}), Truth_StructuralDeduction
{ ([S] --> P) } |- ([S] <-> P), Truth_StructuralDeduction
{ (P --> M), (S --> M) } |- (S <-> P), Truth_Comparison
{ (M --> P), (M --> S) } |- (S <-> P), Truth_Comparison
{ (M --> P), (S <-> M) } |- (S --> P), Truth_Analogy
{ (P --> M), (S <-> M) } |- (P --> S), Truth_Analogy
```

```
{ ({M} --> P), (S <-> M) } |- ({S} --> P), Truth_Analogy
{ (P --> [M]), (S <-> M) } |- (P --> [S]), Truth_Analogy
{ (M <-> P), (S <-> M) } |- (S <-> P), Truth_Resemblance
{ ({A} <-> {B}) } |- (A <-> B), Truth_StructuralDeduction
{ ([A] <-> [B]) } |- (A <-> B), Truth_StructuralDeduction
```

- NAL-3 Inference rules regarding extensional intersection, intensional intersection, extensional and intensional sets and differences:

```
{ ({A B} --> M) } |- <{A} --> M>, Truth_StructuralDeduction
{ ({A B} --> M) } |- <{B} --> M>, Truth_StructuralDeduction
{ (M --> [A B]) } |- <M --> [A]>, Truth_StructuralDeduction
{ (M --> [A B]) } |- <M --> [B]>, Truth_StructuralDeduction
{ ((S | P) --> M) } |- (S --> M), Truth_StructuralDeduction
{ (M --> (S & P)) } |- (M --> S), Truth_StructuralDeduction
{ ((S | P) --> M) } |- (P --> M), Truth_StructuralDeduction
{ (M --> (S & P)) } |- (M --> P), Truth_StructuralDeduction
{ ((A ~ S) --> M) } |- (A --> M), Truth_StructuralDeduction
{ (M --> (B - S)) } |- (M --> B), Truth_StructuralDeduction
{ ((A ~ S) --> M) } |- (S --> M), Truth_StructuralDeductionNegated
{ (M --> (B - S)) } |- (M --> S), Truth_StructuralDeductionNegated
{ (P --> M), (S --> M) } |- ((P | S) --> M), Truth_Intersection
{ (P --> M), (S --> M) } |- ((P & S) --> M), Truth_Union
{ (P --> M), (S --> M) } |- ((P ~ S) --> M), Truth_Difference
{ (M --> P), (M --> S) } |- (M --> (P & S)), Truth_Intersection
{ (M --> P), (M --> S) } |- (M --> (P | S)), Truth_Union
{ (M --> P), (M --> S) } |- (M --> (P - S)), Truth_Difference
{ (S --> M), ((S | P) --> M) } |- (P --> M), Truth_DecomposePNN
{ (P --> M), ((S | P) --> M) } |- (S --> M), Truth_DecomposePNN
{ (S --> M), ((S & P) --> M) } |- (P --> M), Truth_DecomposeNPP
{ (P --> M), ((S & P) --> M) } |- (S --> M), Truth_DecomposeNPP
{ (S --> M), ((S ~ P) --> M) } |- (P --> M), Truth_DecomposePNP
{ (S --> M), ((P ~ S) --> M) } |- (P --> M), Truth_DecomposeNNN
{ (M --> S), (M --> (S & P)) } |- (M --> P), Truth_DecomposePNN
{ (M --> P), (M --> (S & P)) } |- (M --> S), Truth_DecomposePNN
{ (M --> S), (M --> (S | P)) } |- (M --> P), Truth_DecomposeNPP
{ (M --> P), (M --> (S | P)) } |- (M --> S), Truth_DecomposeNPP
{ (M --> S), (M --> (S - P)) } |- (M --> P), Truth_DecomposePNP
{ (M --> S), (M --> (P - S)) } |- (M --> P), Truth_DecomposeNNN
```

- NAL-4 Inference rules about transformations between relations and images

```
{ ((A * B) --> R) } -|- (A --> (R /1 B)), Truth_StructuralDeduction
{ ((A * B) --> R) } -|- (B --> (R /2 A)), Truth_StructuralDeduction
{ (R --> (A * B)) } -|- ((R \\1 B) --> A), Truth_StructuralDeduction
{ (R --> (A * B)) } -|- ((R \\2 A) --> B), Truth_StructuralDeduction
```

- only a few very basic NAL-5 rules

```
{ (! A) } |- A, Truth_Negation
{ (A && B) } |- A, Truth_StructuralDeduction
{ (A && B) } |- B, Truth_StructuralDeduction
```

  and related term reductions (not complete):

```
//Extensional intersection, union, conjunction
(A & A) = A
(A | A) = A
(A && A) = A
//Extensional set
({A} | {B}) = {A B}
//Intensional set
([A] & [B]) = [A B]
```

  which correspond to proven NAL theorems, see *Wang* (2013a).

ONA does not yet implement all existing inference rules and term reductions, but more tend to be added with each release whenever they turn out to be necessary.

BIBLIOGRAPHY

Ahmad, S., and J. Hawkins (2017), Untangling sequences: Behavior vs. external causes, *BioRxiv*, p. 190678.

Alirezaie, M., M. Laengkvist, M. Sioutis, and A. Loutfi (2018), Reasoning upon learning: a generic neural-symbolic approach, in *Thirteenth International Workshop on Neural-Symbolic Learning and Reasoning*.

Amiri, S., M. S. Shirazi, and S. Zhang (2020), Learning and reasoning for robot sequential decision making under uncertainty, in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 2726–2733.

Aradi, S. (2020), Survey of deep reinforcement learning for motion planning of autonomous vehicles, *IEEE Transactions on Intelligent Transportation Systems*.

Badue, C., et al. (2020), Self-driving cars: A survey, *Expert Systems with Applications*, p. 113816.

Bewley, A., Z. Ge, L. Ott, F. Ramos, and B. Upcroft (2016), Simple online and real-time tracking, in *2016 IEEE international conference on image processing (ICIP)*, pp. 3464–3468, IEEE.

Bochkovskiy, A., C.-Y. Wang, and H.-Y. M. Liao (2020), Yolov4: Optimal speed and accuracy of object detection, *arXiv preprint arXiv:2004.10934*.

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.

Bratman, M., et al. (1987), *Intention, plans, and practical reason*, vol. 10, Harvard University Press Cambridge, MA.

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.

Chaslot, G., S. Bakkes, I. Szita, and P. Spronck (2008), Monte-carlo tree search: A new framework for game ai., in *AIIDE*.

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.

DeKeyser, R., B. VanPatten, and J. Williams (2007), Skill acquisition theory, *Theories in second language acquisition: An introduction*, *97113*.

Do, T.-D., M.-T. Duong, Q.-V. Dang, and M.-H. Le (2018), Real-time self-driving car navigation using deep neural network, in *2018 4th International Conference on Green Technology and Sustainable Development (GTSD)*, pp. 7–12, IEEE.

Dosovitskiy, A., G. Ros, F. Codevilla, A. Lopez, and V. Koltun (2017), Carla: An open urban driving simulator, in *Conference on robot learning*, pp. 1–16, PMLR.

Drescher, G. (1989), A constructivist approach to artificial intelligence, *Ph. D. Thesis*.

Drescher, G. L. (1986), Genetic ai: translating piaget into lisp, *Instructional Science*, *14*(3-4), 357–380.

Drescher, G. L. (1993), The schema mechanism, in *Machine Learning: From Theory to Applications*, pp. 125–138, Springer.

Eagleman, D. M., and T. J. Sejnowski (2000), Motion integration and postdiction in visual awareness, *Science*, *287*(5460), 2036–2038.

Eberding, L. M., K. R. Thórisson, A. Sheikhlar, and S. P. Andrason (2020), Sage: task-environment platform for evaluating a broad range of ai learners, in *International Conference on Artificial General Intelligence*, pp. 72–82, Springer.

Farag, W. (2018), Recognition of traffic signs by convolutional neural nets for self-driving vehicles, *International Journal of Knowledge-based and Intelligent Engineering Systems*, *22*(3), 205–214.

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

Gaon, M., and R. Brafman (2020), Reinforcement learning with non-markovian rewards, in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 3980–3987.

Garcez, A. d., M. Gori, L. C. Lamb, L. Serafini, M. Spranger, and S. N. Tran (2019), Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning, *arXiv preprint arXiv:1905.06088*.

Georgeff, M., B. Pell, M. Pollack, M. Tambe, and M. Wooldridge (1998), The belief-desire-intention model of agency, in *International workshop on agent theories, architectures, and languages*, pp. 1–10, Springer.

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

Hammer, P. (2019), Adaptive neuro-symbolic network agent, in *International Conference on Artificial General Intelligence*, pp. 80–90, Springer.

Hammer, P. (2020a), OpenNARS for Applications software repository, `https://github.com/opennars/OpenNARS-for-Applications`, [Online; last accessed May 1, 2021].

Hammer, P. (2020b), SpongeBot robot building instructions, `https://github.com/patham9/SpongeBot`, [Online; last accessed May 1, 2021].

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 (2020), A reasoning based model for anomaly detection in the smart city domain, in *Proceedings of SAI Intelligent Systems Conference*, pp. 144–159, Springer.

Hipp, J., U. Güntzer, and G. Nakhaeizadeh (2000), Algorithms for association rule mining—a general survey and comparison, *ACM sigkdd explorations newsletter*, *2*(1), 58–64.

Hu, H., M. Kantardzic, and T. S. Sethi (2020), No free lunch theorem for concept drift detection in streaming data classification: A review, *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *10*(2), e1327.

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.

Jia, Q., J. Cai, Z. Cao, Y. Wu, X. Zhao, and J. Yu (2018), Deep learning for object detection and grasping: A survey, in *2018 IEEE International Conference on Information and Automation (ICIA)*, pp. 427–432, IEEE.

Jo, K., J. Im, J. Kim, and D.-S. Kim (2017), A real-time multi-class multi-object tracker using yolov2, in *2017 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*, pp. 507–511, IEEE.

Johansson, R. (2019), Arbitrarily applicable relational responding in nars, in *Artificial General Intelligence (AGI-2019), Shenzhen, China*.

Kanerva, P. (1992), *Sparse distributed memory and related models*, vol. 92, NASA Ames Research Center, Research Institute for Advanced Computer Science.

Kanerva, P. (2009), Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors, *Cognitive computation*, *1*(2), 139–159.

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.

Karpas, E., and D. Magazzeni (2020), Automated planning for robotics, *Annual Review of Control, Robotics, and Autonomous Systems*, *3*, 417–439.

Kastak, D., and R. J. Schusterman (1994), Transfer of visual identity matching-to-sample in two california sea lions (zalophus californianus), *Animal Learning & Behavior*, *22*(4), 427–435.

Khan, A., A. Sohail, U. Zahoora, and A. S. Qureshi (2020), A survey of the recent architectures of deep convolutional neural networks, *Artificial Intelligence Review*, *53*(8), 5455–5516.

Kunze, L., N. Hawes, T. Duckett, M. Hanheide, and T. Krajník (2018), Artificial intelligence for long-term robot autonomy: A survey, *IEEE Robotics and Automation Letters*, *3*(4), 4023–4030.

Laird, J. E. (2019), *The Soar cognitive architecture*, MIT press.

Laird, J. E., P. S. Rosenbloom, and A. Newell (1986), Chunking in soar: The anatomy of a general learning mechanism, *Machine learning*, *1*(1), 11–46.

Lanza, F., P. Hammer, V. Seidita, P. Wang, and A. Chella (2020), Agents in dynamic contexts, a system for learning plans, in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, pp. 823–825.

Latombe, J.-C. (2012), *Robot motion planning*, vol. 124, Springer Science & Business Media.

Lofthouse, T. (2019), Alann: An event driven control mechanism for a nonaxiomatic reasoning system (nars).

Lucas, P., and L. Van Der Gaag (1991), Principles of expert systems.

Mao, J., C. Gan, P. Kohli, J. B. Tenenbaum, and J. Wu (2019), The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision, *arXiv preprint arXiv:1904.12584*.

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*.

Muggleton, S. (1991), Inductive logic programming, *New generation computing*, *8*(4), 295–318.

Muggleton, S., and L. De Raedt (1994), Inductive logic programming: Theory and methods, *The Journal of Logic Programming*, *19*, 629–679.

Nachum, O., S. Gu, H. Lee, and S. Levine (2018), Data-efficient hierarchical reinforcement learning, *arXiv preprint arXiv:1805.08296*.

Nath, S. (2012), Ace: Exploiting correlation for energy-efficient and continuous context sensing, in *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pp. 29–42.

Nivel, E., K. R. Thórisson, H. Dindo, G. Pezzulo, M. Rodriguez, et al. (2013), Autocatalytic endogenous reflective architecture.

Nugraha, B. T., S.-F. Su, et al. (2017), Towards self-driving car using convolutional neural network and road lane detector, in *2017 2nd International Conference on Automation, Cognitive Science, Optics, Micro Electro-Mechanical System, and Information Technology (ICACOMIT)*, pp. 65–69, IEEE.

Parisotto, E., and R. Salakhutdinov (2017), Neural map: Structured memory for deep reinforcement learning, *arXiv preprint arXiv:1702.08360*.

Pitti, A., R. Braud, S. Mahé, M. Quoy, and P. Gaussier (2013), Neural model for learning-to-learn of novel task sets in the motor domain, *Frontiers in psychology*, *4*, 771.

Purang, K., D. Purushothaman, D. Traum, C. Andersen, and D. Perlis (1999), Practical reasoning and plan execution with active logic, in *Proceedings of the IJCAI-99 Workshop on Practical Reasoning and Rationality*, pp. 30–38.

Purdy, S. (2016), Encoding data for htm systems, *arXiv preprint arXiv:1602.05925*.

Ramachandra, B., and M. Jones (2020), Street scene: A new dataset and evaluation protocol for video anomaly detection, in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 2569–2578.

Redmon, J., and A. Farhadi (2018), Yolov3: An incremental improvement, *arXiv preprint arXiv:1804.02767*.

Redmon, J., S. Divvala, R. Girshick, and A. Farhadi (2016), You only look once: Unified, real-time object detection, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788.

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, IEEE.

Riedmiller, M., R. Hafner, T. Lampe, M. Neunert, J. Degrave, T. Wiele, V. Mnih, N. Heess, and J. T. Springenberg (2018), Learning by playing solving sparse reward tasks from scratch, in *International Conference on Machine Learning*, pp. 4344–4353, PMLR.

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

Schrijver, A. (1998), *Theory of linear and integer programming*, John Wiley & Sons.

Schrittwieser, J., et al. (2020), Mastering atari, go, chess and shogi by planning with a learned model, *Nature*, *588*(7839), 604–609.

Shams, Z., M. De Vos, J. Padget, and W. W. Vasconcelos (2017), Practical reasoning with norms for autonomous software agents, *Engineering Applications of Artificial Intelligence*, *65*, 388–399.

Shanahan, J. G., and L. Dai (2019), Realtime object detection via deep learning-based

pipelines, in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 2977–2978.

Spaan, M. T. (2012), Partially observable markov decision processes, in *Reinforcement Learning*, pp. 387–414, Springer.

Srinivasan, V., S. Moghaddam, A. Mukherji, K. K. Rachuri, C. Xu, and E. M. Tapia (2014), Mobileminer: Mining your frequent patterns on your phone, in *Proceedings of the 2014 ACM international joint conference on pervasive and ubiquitous computing*, pp. 389–400.

Staddon, J. E., and D. T. Cerutti (2003), Operant conditioning, *Annual review of psychology*, *54*(1), 115–144.

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.

Taketomi, T., H. Uchiyama, and S. Ikeda (2017), Visual slam algorithms: a survey from 2010 to 2016, *IPSJ Transactions on Computer Vision and Applications*, *9*(1), 1–11.

Thórisson, K. R., J. Bieger, X. Li, and P. Wang (2019), Cumulative learning, in *International Conference on Artificial General Intelligence*, pp. 198–208, Springer.

Van Moffaert, K., M. M. Drugan, and A. Nowé (2013), Scalarized multi-objective reinforcement learning: Novel design techniques, in *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pp. 191–199, IEEE.

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

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., and P. Hammer (2015a), Assumptions of decision-making models in agi, in *International conference on artificial general intelligence*, pp. 197–207, Springer.

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

Wang, P., and P. Hammer (2018), Perception from an agi perspective, in *International Conference on Artificial General Intelligence*, pp. 259–269, Springer.

Wang, W., and M. Sebag (2012), Multi-objective monte-carlo tree search, in *Asian conference on machine learning*, pp. 507–522, PMLR.

Watkins, C. J. C. H. (1989), Learning from delayed rewards.

Wooldridge, M. (1996), Practical reasoning with procedural knowledge, in *International Conference on Formal and Applied Practical Reasoning*, pp. 663–678, Springer.

Yang, R., X. Sun, and K. Narasimhan (2019), A generalized algorithm for multi-objective reinforcement learning and policy adaptation, *arXiv preprint arXiv:1908.08342*.

You, C., J. Lu, D. Filev, and P. Tsiotras (2019), Autonomous planning and control for intelligent vehicles in traffic, *IEEE Transactions on Intelligent Transportation Systems*, *21*(6), 2339–2349.

Zadeh, L. A. (1988), Fuzzy logic, *Computer*, *21*(4), 83–93.

Zhang, H., X. Lan, S. Bai, L. Wan, C. Yang, and N. Zheng (2019), A multi-task convolutional neural network for autonomous robotic grasping in object stacking scenes, in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6435–6442, IEEE.

Zhou, Y., E.-J. van Kampen, and Q. Chu (2019), Hybrid hierarchical reinforcement learning for online guidance and navigation with partial observability, *Neurocomputing*, *331*, 443–457.

Zhuang, F., Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He (2020), A comprehensive survey on transfer learning, *Proceedings of the IEEE*, *109*(1), 43–76.

Zintgraf, L. M., T. V. Kanters, D. M. Roijers, F. Oliehoek, and P. Beau (2015), Quality assessment of morl algorithms: A utility-based approach, in *Benelearn 2015: Proceedings of the 24th Annual Machine Learning Conference of Belgium and the Netherlands*.