'OpenNARS for Applications': Architecture and Control

Patrick Hammer[1] and Tony Lofthouse[2]

[1] Department of Computer & Information Sciences
College of Science and Technology
Temple University
Philadelphia PA 19122, USA
patrick.hammer@temple.edu
[2] Reasoning Systems Ltd (UK)
Tony.Lofthouse@Reasoning.Systems

Abstract. A pragmatic design for a general purpose reasoner incorporating the Non-Axiomatic Logic (NAL) and Non-Axiomatic Reasoning System (NARS) theory. The architecture and attentional control differ in many respects to the OpenNARS implementation. Key changes include; an event driven control process, separation of sensorimotor from semantic inference and a different handling of resource constraints.

Keywords: Non-Axiomatic Reasoning · Sensorimotor · Artificial General Intelligence · General Machine Intelligence · Procedure Learning · Autonomous Agent · Inference Control · Attention

1 Introduction

The Non-Axiomatic Reasoning System has been implemented several times ([10], [9], [6]). *OpenNARS*, now known as *OpenNARS for Research* (ONR), was used both as a platform for new research topics and an implementation for applications [5]. Not all ideas in ONR are complete, 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 proposed architecture, *OpenNARS for Applications* (ONA), has been developed to resolve ONR's limitations by combining the best results from our research projects. The logic and conceptual ideas of ONR [6], the sensorimotor capabilities of ANSNA [7] and the control model from ALANN [9] are combined in a general purpose reasoner ready to be applied.

ONA is a NARS as described by Non-Axiomatic Reasoning System theory [18]. 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 Non-Axiomatic Reasoning Systems have in common is the use of the Non-Axiomatic Logic (NAL) [18], 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.

Compared to BDI models [1] [3], plans and intentions are treated as beliefs, as procedure knowledge is learnable by NARS, instead of always provided 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 which is so critical for AGI. Reinforcement learning (see [15], [19] and [14]) captures the learning aspect and solves the Temporal Credit Assignment Problem, but does so just for a single signal (reward, a single outcome). NARS solves it for all events it can predict, some of which may correspond to goals to achieve. There is also multi objective reinforcement learning [8] [16], which however does not capture a changing utility function corresponding to changing goals. NARS does not learn a fixed state-action mapping, but instead its behaviors can change rapidly with the changing goals. Hence, NARS combines and extends the key aspects of both BDI and Reinforcement Learning without inheriting some of their limitations.

2 Architecture

A key driver of the architectural change is the nature of how concept, task and belief are selected for inference. In ONR the selection is based on a probabilistic choice from a data structure (Bag) and is concept centric [13]. ONA takes a different approach: an event is popped from a bounded priority queue. The event determines the concept to be selected, through a one-to-one mapping between event and concept terms. Then a subset of concepts 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 [6]). 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).

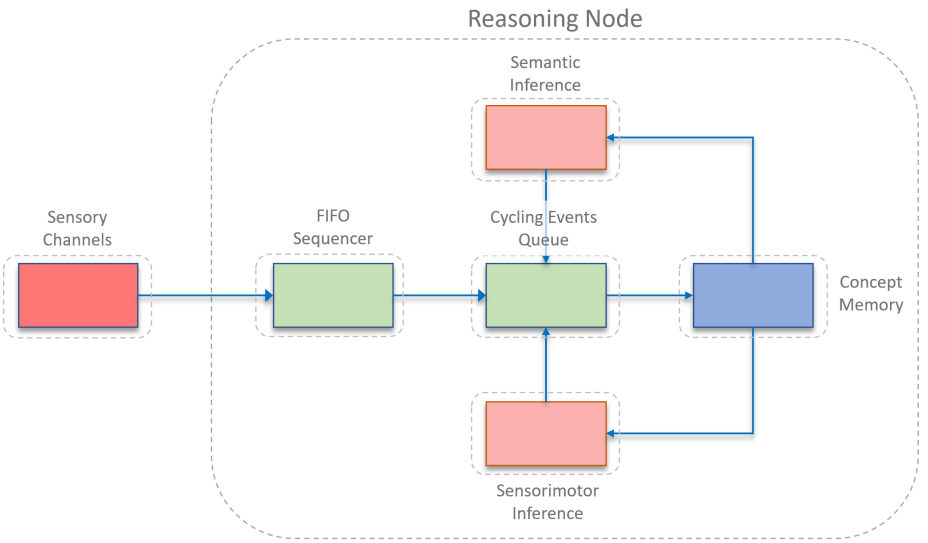

Fig. 1: High level architecture showing input sequencing and cycles for sensorimotor and semantic inference

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 [5].

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 1, 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 strengthening 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 [4].

Semantic Inference: All declarative reasoning using NAL layers 1-6 occurs here as described in [18], 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 [17], 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 is the case even when the meaning of some terms is not yet clear and needs to be enriched to become more useful.

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.

3 Data Structures

Data structures 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. Their structure is represented via a binary tree, where each node can either be a logical NAL copula or atomic.

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.

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. Plus an eternal belief (giving a summary of event truths), most recent event belief, and predicted event belief.

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 [2], 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 $usefulness = \frac{usefulnessRaw}{usefulnessRaw+1}$.

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, where $exp(f, c)$ is defined as $(c * (f - \frac{1}{2}) + \frac{1}{2})$, the confidence as $c = \frac{w}{w+1}$ where $w = w_+ + w_-$ is the total evidence, w_+ and w_- the positive and negative evidence respectively, and frequency is defined as $f = \frac{w^+}{w}$.

Attentional Control Functions

The high level requirements of the attentional control were explained in the Conceptual Foundation section. Here we cover the low level detailed functions for resource management which are enforced by the corresponding data structures:

Attention_forgetEvent: 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)

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

Attention_activateConcept: 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.

Attention_deriveEvent:

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.

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

4 Operating cycle

Following is an overview of the main operating cycle with a detailed breakdown of each component:

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, where 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 [14] and [15]). 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, with a monotonic decay function. 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, Prediction 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. Also, event a leads to the prediction of b via Deduction, assuming $a \Rightarrow b$ exists as implication in concept b.

Motor Babbling: 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 [11], [7] and [6]. 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.

5 Experiments & Comparisons

To demonstrate the reasoner's general purpose capabilities we tested with a variety of diverse examples using the same default system configuration. The following examples are all available at the project web site, see [20].

Real-time Q/A. In this example the reasoner needs to answer questions about drawn shapes in real time (see Fig. 2). Input events consist of shape instances, their types, and filled property as output by a Convolutional Neural Network. The shape's relative location is fed into the reasoner. Queries can be arbitrary queries such as "What is left of the unfilled circle?". In our experiment, the reasoner answered these questions correctly 80 percent of the time within 50 inference steps from 20 example inputs in 10 trials. This were 200 Narsese input events and 9 seconds per trial, fast enough for real time perception purposes.

Procedure Learning. In the toothbrush example knowledge about different objects, their properties and what they can be used for is provided (see Fig. 2). The goal is to unscrew a screw without a screwdriver. The solution is to melt a toothbrush and to reshape it into a form usable to unscrew the screw. The reasoner finds this solution consistently, within 30 inference steps, while ONR often needed 100K or more.

Generalisation. The goal of this experiment was to show that the reasoner could learn and then apply generalised procedural knowledge to examples not previously experienced. The test setup composed of: three switches, with different instance names and two operators, 'goto' and 'activate'. From 2 observations of the user activating switches, the reasoner should learn that the 'goto' operation applied from the start position, will lead to the agent reaching the switch position. It also learns that when the switch position was reached, and the 'activate' operation is called, the switch will be on. The third switch is then activated by the reasoner on its own as a solution to the user goal, by invoking 'goto' and 'activate' on the new switch instance, applying generalised behavior which the reasoner has learnt to be successful for the previously encountered instances.

Real-time Reasoning. As presented in [5], ONR, was successfully used to autonomously label regions and to identify jaywalking pedestrians based on a very minimal background ontology, without scene-specific information, across a large variety of Streetcams, using a Multi Class Multi Object tracker. A similar example (capturing key reasoning aspects) is included in the release of ONA, the new reasoner will replace ONR in future deployments.

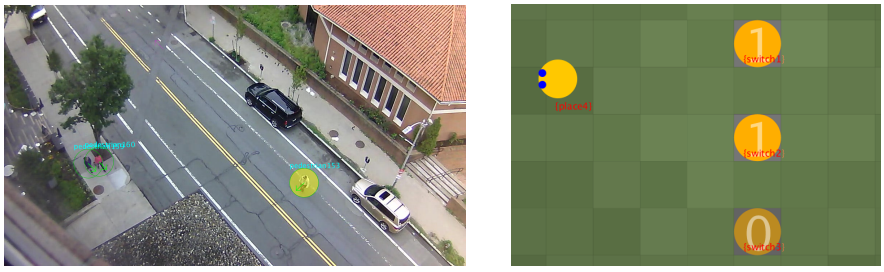

Fig. 2: Using minimal scene-independent background knowledge to detect jaywalking (left), learning to reach and activate switches from observations. (right)

Procedure Execution. Previously, a 24hr reliability test of OpenNARS v3.0.2 (ONR) was carried out with the Pong test case. The system ran reliably for the 24hr period with a hit/miss ratio of 2.5 with a learning time of two minutes and some minor fluctuation in capability in the first 3 hrs.

In comparison, OpenNARS for Applications v0.8.1 (ONA) ran reliably for the 24hr period with a hit/miss ratio of 156.6 with a learning time of <10 secs and no negative fluctuation. It should be noted that the pong test was not identical for each test scenario. The test for ONA was more difficult with 3 operations

(compared to left/right operations only for ONR Pong, it didn't include stop) and approximately 2x faster ball speed, demanding quicker reaction times.

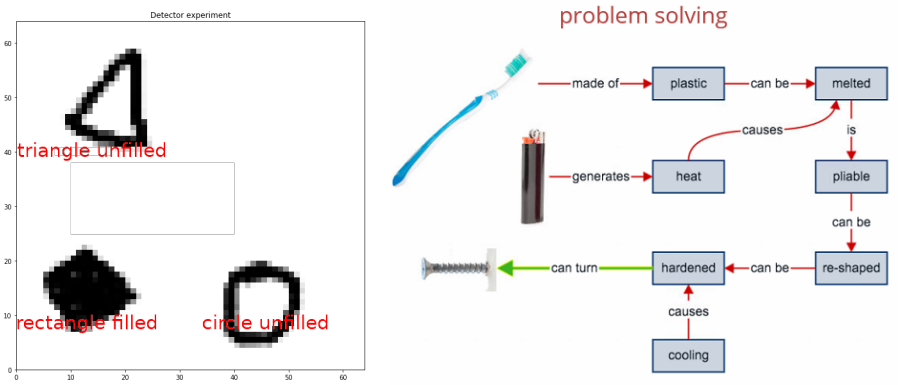

Fig. 3: Q&A about detected shapes (left), Toothbrush problem solving (right)

6 Conclusion

The decision to take a pragmatic approach to the architecture has proven to be a worthwhile investment. The change to an event driven control model has removed much of the complexity of the prior control system. The separation of semantic and sensorimotor inference has highlighted the key issues of both aspects whilst avoiding the complexity of a unified handling. The reduction in complexity has led to many benefits including: simplified parameter tuning, separation of concerns, and clear attentional focus boundaries.

The use of the meta rule DSL [6] to represent the logic rules allows the reasoner to be configured for specific domains. Enabling subsets of inference rules for specific use cases avoids the processing of unnecessary inference rules and the resulting increase in non-relevant results.

From a software engineering perspective, the OpenNARS (ONR) codebase was well overdue a rewrite as the continuous incremental change had led to it being difficult to maintain and modify. The choice of portable C as the implementation language means the reasoner can be compiled on a broad range of platforms including embedded, mobile and all major OSs.

In summary, the new architecture and control has led to significant improvements in both efficiency and quality of results, especially in respect to procedure learning and attention allocation. Connecting to the reasoner via the shell or UDP protocol is straightforward and tuning the parameters and inference rules for specific use cases is now possible with minimal effort.

The project is open source, under the MIT license, and available in [20].

References

1. Bratman, M. E. (1987). Intention, Plans, and Practical Reason. CSLI Publications. ISBN 1-57586-192-5.
2. Drescher, G. L. (1993). The schema mechanism. In Machine Learning: From Theory to Applications (pp. 125-138). Springer, Berlin, Heidelberg.
3. Georgeff, M., Pell, B., Pollack, M., Tambe, M., & Wooldridge, M. (1998, July). The belief-desire-intention model of agency. In International workshop on agent theories, architectures, and languages (pp. 1-10). Springer, Berlin, Heidelberg.
4. Hammer, P., & Lofthouse, T. (2018, August). Goal-Directed Procedure Learning. In International Conference on Artificial General Intelligence (pp. 77-86). Springer, Cham.
5. Hammer, P., Lofthouse, T., Fenoglio, E., Latapie, H. (2020, in press). A reasoning based model for anomaly detection in the Smart City domain, Advances in Intelligent Systems and Computing.
6. Hammer, P., Lofthouse, T., & Wang, P. (2016, July). The OpenNARS implementation of the non-axiomatic reasoning system. In International conference on artificial general intelligence (pp. 160-170). Springer, Cham.
7. Hammer, P. (2019, August). Adaptive Neuro-Symbolic Network Agent. In International Conference on Artificial General Intelligence (pp. 80-90). Springer, Cham.
8. Liu, C., Xu, X., & Hu, D. (2014). Multiobjective reinforcement learning: A comprehensive overview. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 45(3), 385-398.
9. Lofthouse, T. (2019). ALANN: An event driven control mechanism for a non-axiomatic reasoning system (NARS). www.researchgate.net
10. Ivanović, M., Ivković, J., & Bădică, C. (2017, September). Role of Non-Axiomatic Logic in a Distributed Reasoning Environment. In International Conference on Computational Collective Intelligence (pp. 381-388). Springer, Cham.
11. Nivel, E., & Thórisson, K. R. (2013). Autocatalytic endogenous reflective architecture (AERA).
12. NLT.org, Python Natural Language Toolkit, https://www.nltk.org/book/, last accessed February 29, 2020.
13. Rehling, J., & Hofstadter, D. (1997, October). 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.
14. Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. Machine learning, 3(1), 9-44.
15. Sutton R. S., Barto A. G. Reinforcement Learning: An Introduction, The MIT Press Cambridge, Massachusetts, London, England (2012)
16. Van Moffaert, K., Drugan, M. M., & Nowé, A. (2013, April). Scalarized multi-objective reinforcement learning: Novel design techniques. In 2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL) (pp. 191-199). IEEE.
17. Wang, P., (2006). Rigid Flexibility, The Logic of Intelligence. Springer
18. Wang, P. (2013). Non-axiomatic logic: A model of intelligent reasoning. World Scientific.
19. Watkins, C.J.C.H. (1989). Learning from Delayed Rewards. PhD thesis, Cambridge University, Cambridge, England, 1989.
20. OpenNARS for Applications, https://github.com/opennars/OpenNARS-for-Applications, last accessed March 7, 2020.