NARS implementation in MeTTA

Peter Isaev

January 2024

1 Introduction

This document provides an overview of the basic principles behind Non-Axiomatic Reasoning System (NARS) implemented in MeTTa programming language and justifies the design decisions according to the specifics of the language. It also outlines architecture, structure and features of MeTTa-NARS demonstrating inference control and attention allocation.

2 Overall Architecture

The MeTTa implementation of NARS follows design paradigms borrowed from multiple previous NARS versions[1] as well as introduces own distinctive structure and design. MeTTa-NARS utilizes Non-Axiomatic Logic (NAL)[2], concept-centric semantic memory[3] and control mechanism, including declarative, temporal and procedural types of reasoning. The inference control framework is shown in Figure 1, whereby the control is mainly concerned with which premises to pick and which NAL inference rules to apply on a cyclic basis to derive knowledge, trigger a decision or lead to the derivations of subgoals.

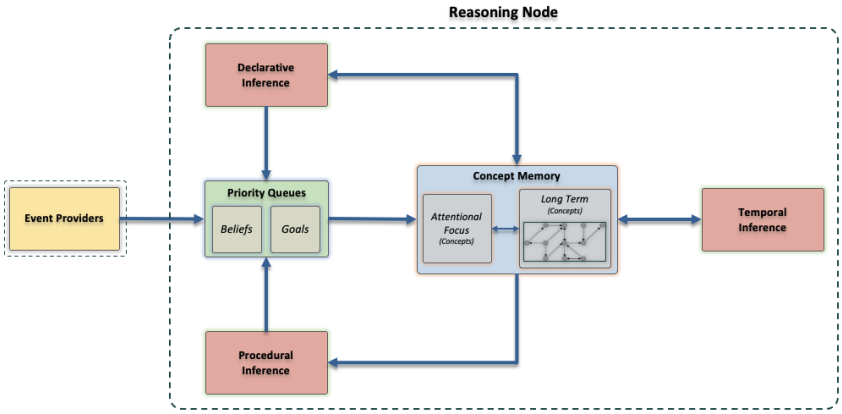

Figure 1: Overview of MeTTa-NARS

Describing the functionality and structure of the control components in detail is well beyond the scope of this document, however, we will recall the most relevant definitions of the components within the corresponding sections of the report to allow the reader to understand their principle functionalities.

3 Code Structure

The reasoning system consists of the following three (3) main components:

- **Logic**
- **Memory**
- **Control**

Whereby each of them is further subdivided into multiple parts as can be seen in code organization Figure 2.

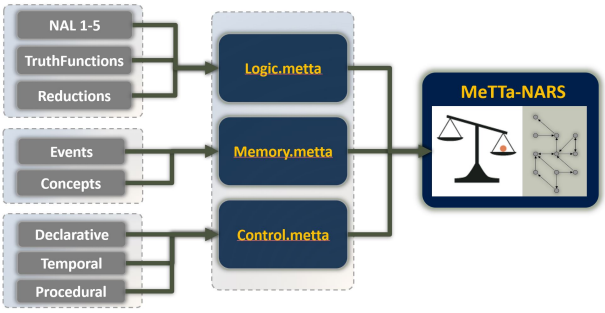

Figure 2: Code Structure Flowchart

Important to note that at the time of the creation of this document, the MeTTa language did not offer a reliable code importing process and therefore the simple concatenation of the source files was performed instead by using building scripts to construct each of the components from the corresponding smaller MeTTa files.

Hereby, the logic component is built using multiple smaller MeTTa files for NAL inference rules, NAL Truth Functions and Term Reductions, while the memory of the system is divided into Events and Concepts. Given the complexity of NARS reasoning, the control of the system is a more substantial entity consisting of the following sub-parts, each conforming to the type of reasoning:

- Declarative (deriving events and knowledge from events)
- Temporal (sequence and temporal implication formation)
- Procedural (decision making with subgoaling and planning)

The build script *build.sh* at the root of the directory performs the concatenation process of the initial source code files into 3 larger chunks and eventually into a single NARS reasoner *NARS.metta* as seen in Fig. 2.

4 Standard library extensions

In addition to the main three components discussed above, MeTTa-NARS features a small utility package within *util.metta* that defines the commonly used procedures as well as basic mathematical functions. Below are descriptions of the select procedures defined within *util.metta*.

TupleConcat:

TupleConcat concatenates two given tuples into a single one by explicitly using inbuilt *superpose*.

```
(= (TupleConcat $Ev1 $Ev2)
    (collapse (superpose ((superpose $Ev1) (superpose $Ev2)))))
```

Basic math functions:

```
(= (max $1 $2) (if (> $1 $2) $1 $2))
(= (min $1 $2) (if (< $1 $2) $1 $2))
(= (abs $x) (if (< $x 0) (- 0 $x) $x))
```

Sequential:

A wrapper for the sequential instructions evaluation and modifications. Currently using inbuilt *superpose* however might face certain concerns with the future MeTTa version as the execution order within *superpose* might not be respected given the possible multi-processing approach.

```
(: sequential (-> Expression %Undefined%))
(= (sequential $1) (superpose $1))
```

CollapseCardinality:

Given that MeTTa currently does not provide an easy inbuilt way of knowing the length of a tuple, *CollapseCardinality* is used to determine tuple size. To allow this happening, *BuildTupleCounts* creates a map by adding tuples of up-to the given size with their cardinality to the atom space in the background.

```
(= (BuildTupleCounts $TOld $C $N)
(let $T (collapse (superpose (1 (superpose $TOld))))
    (superpose ((add-atom &self (= (TupleCount $T) (+ $C 2)))
                (If (< $C $N) (BuildTupleCounts $T (+ $C 1) $N))))))
```

Do and If:

Do is trivial procedure returning an empty tuple, idea is to return *None*, when the output is not desired, thereby many functions within the control code are wrapped in *Do* procedure. Additionally, since MeTTa only provide *If-else* construct, more common definition of *if-else* statement is provided.

```
(: do (-> Expression %Undefined%))
(= (do $1) (case $1 ()))
(: If (-> Bool Atom Atom))
(= (If True $then) $then)
(= (If False $then) ())
(: If (-> Bool Atom Atom Atom))
(= (If $cond $then $else) (if $cond $then $else))
```

Please note, the type annotations are necessary to hamper it from evaluating the input expression.

5 The Logic

Non-Axiomatic Logic (NAL)[2] specifies which information can be derived from combinations of at most two premises with support of uncertainty estimation. It incorporates inference rules for learning from event streams, goal reasoning and decision making, as well as functions to estimate the uncertainty of statements based on evidential support. Hence, a means-end reasoner can be built to learn from experience using NAL.

NAL is defined using multiple layers (1-5) with corresponding truth functions for uncertainty estimation, whereby logic becomes more advanced with each layer. Inference Rules along with truth functions are being explicitly defined within the Logic component of MeTTa-NARS. Below are examples of basic inference rules representation following the convention of Hyperon-PLN which uses entails operator.

NAL-1 Inference Rules for Inheritance:

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

NAL-5 Inference Rules for Implication and Equivalence:

```
(= (|- (($A ==> $B) $T1) (($B ==> $C) $T2)) (($A ==> $C) (Truth_Deduction $T1 $T2)))
(= (|- (($A ==> $B) $T1) (($A ==> $C) $T2)) (($C ==> $B) (Truth_Induction $T1 $T2)))
(= (|- (($A ==> $C) $T1) (($B ==> $C) $T2)) (($B ==> $A) (Truth_Abduction $T1 $T2)))
(= (|- (($A ==> $B) $T1) (($B ==> $C) $T2)) (($C ==> $A) (Truth_Exemplification $T1 $T2)))
(= (|- (($S <=> $P) $T)) (($P <=> $S) (Truth_StructuralIntersection $T)))
(= (|- (($S ==> $P) $T1) (($P ==> $S) $T2)) (($S <=> $P) (Truth_Intersection $T1 $T2)))
(= (|- (($P ==> $M) $T1) (($S ==> $M) $T2)) (($S <=> $P) (Truth_Comparison $T1 $T2)))
(= (|- (($M ==> $P) $T1) (($M ==> $S) $T2)) (($S <=> $P) (Truth_Comparison $T1 $T2)))
(= (|- (($M ==> $P) $T1) (($S <=> $M) $T2)) (($S ==> $P) (Truth_Analogy $T1 $T2)))
(= (|- (($P ==> $M) $T1) (($S <=> $M) $T2)) (($P ==> $S) (Truth_Analogy $T1 $T2)))
(= (|- (($M <=> $P) $T1) (($S <=> $M) $T2)) (($S <=> $P) (Truth_Resemblance $T1 $T2)))
```

Selected Truth Functions:

```
(= (Truth_Deduction ($f1 $c1) ($f2 $c2)) ((* $f1 $f2) (* (* $f1 $f2) (* $c1 $c2))))
(= (Truth_Abduction ($f1 $c1) ($f2 $c2)) ($f2 (Truth_w2c (* (* $f1 $c1) $c2))))
(= (Truth_Induction $T1 $T2) (Truth_Abduction $T2 $T1))
(= (Truth_Exemplification ($f1 $c1) ($f2 $c2)) (1.0 (Truth_w2c (* (* $f1 $f2) (* $c1 $c2)))))
```

Additionally, to allow structural term reduction during derivation process, term reduction rules are being featured in Logic component and applied at the time of deriving new or updating existing knowledge.

Selected NAL term reductions:

```
(= ($A & $A) $A)
(= ($A | $A) $A)
```

```
(= ($A && $A) $A)
(= ($A || $A) $A)
(= (({ $A }) | ({ $B })) ({ $A $B }))
(= (({ $A $B }) | ({ $C })) ({ ($A . $B) $C }))
(= (({ $C }) | ({ $A $B }) ) ({ $C ($A . $B) }))
```

6 Memory

Memory in NARS follows a concept-centric semantic memory structure in accordance with the NAL term logic the system uses. It can be viewed as a graph where concepts are represented as nodes and links designate relationships among them as seen within the memory component in Figure 1. Technically, NARS memory is a collection of concepts representing a conceptual network with prioritized nodes. **Concept** is a major entity, an identifiable unit of system's experience that has grounded meaning. It is also a data structure to hold various components of knowledge summarizing a fragment or pattern of the system's experience identified by a term in a formal language Narsese. For more details on functionality, memory types and semantics see [3].

In MeTTa-NARS implementation, the memory part is found within *Memory.metta* source file, which is being built using **Events** and **Concepts** located in *events.metta* and *concepts.metta* respectively. Below we provide explanations of the important procedures and data structures to allow the reader understand the functionality of the system.

6.1 Events

Events part of NARS Memory sets up two Atom spaces with related functions for *Beliefs* and *Goals* which together with *Attentional Focus* space, found within *concepts.metta*, serve as a working term memory of the system. In some NARS implementations the atom spaces for Beliefs and Goals are implemented using priority queues, prioritized by a priority (importance) value of items. Since at the current stage no efficient data structure is provided within MeTTa, we bound ourselves to use Atom Spaces and then linearly iterate over them retrieving the best candidate. Because of such limitation, the Atom Spaces have been currently limited to 100 elements. For details on budget and resource allocation of NARS, see [4], hereby we will provide an overview of procedures found within *events.metta*.

(= (ProcessBeliefEvent $Ev $t) ...)

ProcessBeliefEvent is called from *AddBeliefEvent* found in the temporal control, which is the entry point to system. The role is to add input event to the Belief space and create or update a concept for it (discussed in the *concept* section).

(= (SelectHighestPriorityEvent $collection $t) ...)

Currently it iteratively selects the highest priority item from either Belief or Goal spaces based on the event occurrence time and current time of the system measured in cycles (discussed in more details in 8).

(= (BoundEvents $collection $Threshold $Incr $TargetAmount $t) ...)

BoundEvents maintains the constant number of items ($TargetAmount$) within the Belief or Goal Atom spaces by recursively calling itself with incremented threshold until the collection has exactly $TargetAmount$ number of items (see 8 for more details).

Additionally, the Reasoner State is defined within *events.metta*. Reasoner state is a valuable concept that captures current cycle of the system and stamp id for derivations and inputs. Reasoner state is being constantly queried by system's control during reasoning for events creation, anticipation and evidential base setups.

6.2 Concepts

The main purpose of the *concepts.metta* is to create, update and maintain system's concepts, revise beliefs and maintain attentional focus of the system. There are two Atom spaces used for *attentional focus* of the system and *concept* memory, long-term system's experience storage. At the current moment, to make the system responsive, the system attentional focus is bounded to 100 elements which suffices for running procedural learning experiments and testings (including pong example), while the long-term concept memory is bound to 10000 elements. In addition, the *concept.metta* provides estimation of priority for the concepts. The subject of attention allocation and priority estimation will be discussed in the later section. The following select procedures are found within *concepts.metta*:

(= (RevisionAndChoice (Event $ev1) (Event $ev2)) ...)

RevisionAndChoice revises the belief of existing knowledge with the newly added or derived knowledge given its evidential bases do not overlap, i.e. the knowledge being derived from different evidential traces. In the case of both of the beliefs share one or more components within their evidential base, the system makes the choice regarding which knowledge to utilize based on the confidence of the belief.

(= (UpdateConcept $NewEvent $t) ...)

The given procedure creates new or updates an existing concept. In the case of the new event to the system, either an input or a derivation, the given procedure creates a concept and adds it to the *attentional focus* atom space. If a concept for a given event is **already present**, either in *attentional focus* or in *concept*

spaces, the procedures calls *RevisionAndChoice* to revise beliefs, updates the concept's priority and insert the concept into *attentional focus* space. It is important to note that the concept only exists within *attentional focus* or *concept* spaces but not in both, thus if a concept has lost the competition for resources it remains only in the *concept* space.

(= (BoundAttention $Threshold $Increment $TargetAmount $t) ...)

BoundAttention is being called from declarative control (discussed later), the purpose is to maintain the size of the *attentional focus* atom space. Concepts whose priority is less than the given threshold are being moved from *attentional focus* to *concepts* atom space. The procedure recursively calls itself until the size of *attentional focus* has reached provided target number.

7 Inference Control

The reasoning abilities of MeTTa-NARS are accomplished by the repeated application of inference rules of Non-Axiomatic Logic. This is realized in the framework of a reasoning system using effective Inference Control with the additional constraint to operate under Insufficient Knowledge and Resources which entails the following requirements:

- **Finite**: The memory and processing capacity of the system is strictly constant-bounded.
- **Open**: The reasoner should accept new information at any moment while it is operating.
- **Real-Time**: Reasoning steps can only take up to a maximum sub-second constant amount of time.

Inference Control specifies which premises to pick and which inference rules to apply at any moment on a cyclic basis, whereby the reasoning system is designed to favor selection of goal- and question-relevant knowledge for reasoning, which treats attention allocation as a resource allocation problem. In other words, inference control decides which possible derivations will be actually carried-out among all existing possibilities.

In MeTTa-NARS, the control feature three major parts, each conforming to the type of reasoning:

- Declarative (deriving events and knowledge from events)
- Temporal (sequence and temporal implication formation)
- Procedural (decision making with subgoaling and planning)

All of these are functioning together in the real-time during the cycling reasoning process and are defined within *control.metta* source file which is being

built using *declarative.metta*, *temporal.metta* and *procedural.metta* sources respectively. In the following sections, we provide a high-level description of each sub-component of Inference Control along with the explanations of corresponding important procedures.

7.1 Declarative Inference

Declarative or semantic type of inference is concerned with selection of an event from *belief_events* space and a concept from the *attentional focus*, and performing reasoning on the basis of a common term between the selections according to rules defined in NAL 1-5 (see section 5). This is summarized in the following declarative cycle:

1. Select the highest priority event from *belief_events* space.
2. Select ALL concepts from *attentional focus* of the system.
3. For-each concept, perform reasoning by applying ALL matching inference rules from LOGIC.metta between concept's *belief* and selected *event*.
4. Evaluate priority for each of the derived events.
5. Process derived events by putting them into corresponding spaces.
6. Maintain the defined sizes of the *belief_events* space and *attentional focus*.

Below are the procedures necessary to carry out declarative inference found within *declarative.metta*.

(= (BeliefCycle $t) ...)

BeliefCycle is the main procedure in the *declarative.metta* that selects the highest priority *event* from *belief_events* and calls *ReasonWithTask*. It also maintains the defined sizes of *belief_events* space and *attentional focus* because they are being modified during the reasoning process.

(= (ReasonWithTask (Event *$S1* (*$time1 $ev1 $prio1*)) *$t*) ...)

ReasonWithTask accepts the highest priority *event* selected during *BeliefCycle* procedure, then selects all the concepts from attentional focus of the system and invokes reasoning (*Conclude* procedure) between the *event* and selected *belief* from each of the concept. Prior to the reasoning, the concept's *belief* is being temporarily aligned with the input event by applying the projection rule (see 7.2). Lastly, it processes all the derivations returned from *Conclude* procedure using *ProcessBeliefEvent* as discussed in section 6.

(= (Conclude *$Event1 $Event2 $CPrio $t*) ...)

Conclude performs the reasoning between the *event* ($Event1) and the *belief* ($Event2) given their evidential base does not overlap. Note that the reasoning happens in both directions by applying all the inference rules from 5 as follows:

$$superpose\ ((|-\ \$S1\ \$S2)\ (|-\ \$S2\ \$S1))$$

where $\$S1$ is the sentence from the *event* and $\$S2$ is the sentence from the *belief*. After the reasoning, priority values for the derivations are being computed as discussed in 8.

7.2 Temporal Inference

Temporal Inference is a type of reasoning that is concerned with temporal relation between the premises rather than a shared term. Temporal Inference features a special type of control requiring an Anticipation mechanism to deal with predictions and estimation of negative evidence for false predictions. In MeTTa-NARS each event is time dependent and utilizes a timestamp, whereby the handling of time is accomplished using derivation cycles, i.e. time is being incremented by 1 with each cycle.

Temporal control also builds and processes *Temporal Implications* which are the statements of the form $(A \Rightarrow B)$, and *Procedural Implications* (also called Procedural Hypothesis) that have a shape of $((A, Op) \Rightarrow B)$, where A and B are statements from events, and Op is an operation term referring to a procedure, typically motor-related, the system can invoke at any time. The former denotes that B will happen after A, and the latter that B will happen when Op is executed right after A has happened. Truth value is therefore being assigned based on temporal distance where *frequency* represents prediction success ratio of temporal implications and *confidence* encodes number of samples seen so far.

To control the process of forming temporal and procedural implications from events and to calculate their evidence, a sliding-window is utilized, a common approach in Data Stream Mining [5], which is implemented using a FIFO queue of size 128 that only holds the latest k events where k is a hyper-parameter. By iterating this structure, both the positive and the negative cases can be identified (for more details, see [6]), where the Induction rule is applied in each case. Hereby, if an induced temporal implication already exists in system memory, its truth value is being revised by applying the Revision Rule (section 6.2) adding up the evidences of the new and old implications. This ensures implications receive additional positive or negative evidence dependent on whether the consequent will happen when the antecedent has happened.

MeTTa-NARS Temporal Inference process can be summarized as follows:

1. Accept input event and put it into FIFO event queue
2. Trigger Anticipation procedure for k elements within the given FIFO event queue

3. Perform temporal compositions: create Sequences, Temporal and Procedural implications

4. Create or update concepts for implications

5. Process input event (section 6) and trigger declarative belief cycle (section 7.1)

In the following, we will discuss the select procedures from Temporal Control found in *temporal.metta*.

(= (AddBeliefEvent $Sentence) ...)

AddBeliefEvent procedure is the entry point to the system which serves as a "main" procedure for the entire MeTTa-NARS. *AddBeliefEvent* accepts a sentence as an input, converts it into the event by assigning time and default priority, and adds it to the FIFO event queue. Anticipation mechanism and procedures for creating temporal and procedural implications are then invoked. Lastly, *AddBeliefEvent* processes the input event as discussed in 6.1 and triggers *BeliefCycle* (see 7.1).

(= (Anticipate ($Pos $Op $Pre) $t) ...)

Anticipate procedure is the core to the anticipation mechanism of MeTTa-NARS. The idea is to observe whether an input event is being anticipated by the system or not, in other words, whether the system has experienced the same sequence of events in the past, and if such, the input event can be predicted with high confidence. If the system makes a prediction (anticipates an event) which is not as the observed one, the prediction being penalized by assigning negative evidence. Thereby, for a given observed event, a temporal implication is being built using events within FIFO queue where the observed event becomes the consequent of that implication. Entire memory of the system is then queried to return all temporal implications with the same antecedent as the one in the created implication from the FIFO. For all of the returned temporal implications whose consequent does not match the observation, *frequency* and *confidence* are being decreased to 0 and 0.1 respectively.

(= (TemporalImplicationInduction ($Cons $Op $Prec)) ...)

TemporalImplicationInduction procedure is essential for creating temporal compositions (Temporal Sequencing, Temporal and Procedural Implications) using events from the FIFO event queue. Temporal Sequence and Temporal Implications are being created by applying *Truth_Intersection* and *Truth_Induction* respectively to calculate the new truth value of a temporal composition.

(= (Projection $Sentence ($Time $Evidence $EPrio)) $TargetTime) ...)

Projection plays an important role in Temporal Inference. Its function is to decrease *confidence* with increased temporal distance between the events. The *projection* is being used in multiple places within MeTTa-NARS. For example, it is utilized during creation of Temporal Implications because it is important to account for the temporal distance and temporally align events while making temporal compositions, it is also employed during declarative reasoning 7.1 and decision making 7.3 because of the importance knowing the relevance of the selected premise to the current context of the system. *Projection* uses the following formula for *confidence* decay:

$$confidence * min(1,\ \frac{1}{|eventTime - targetTime|})$$

7.3 Procedural Inference

Procedural Inference is an essential part of Inference Control that is responsible for decision making, planning, and goal realization and satisfaction. In MeTTa-NARS, decision making model is concerned with realizing a goal by executing an operation which most likely and sufficiently likely leads to its fulfillment.

Goals G are represented as a temporal implication $(G \Rightarrow D)$ where D is implicitly present and stands for "desired state", and their desire value is defined as the truth value of this implication. In each system cycle, a goal (if existent) is selected to either trigger a decision (execution of an operation) or lead to the derivations of subgoals. For this purpose, the existing procedural implications of the form $((A_i, Op_j) \Rightarrow G)$ (see 7.2) are iterated over. When they have a sufficiently high truth value and event A_i recently happened, it will generate a high desire value for the reasoner to execute Op_j. The desire expectations of these operations are compared, and the operation from the candidate which leads to the highest desire value expectation will be executed if above the decision threshold (a hyperparameter usually around 0.5). Alternatively, all the preconditions (all A_i) of the considered implications will be derived as subgoals, competing for attention and processing in the *goal_events* space as discussed in 6.1.

To determine desire value of an operation (Op_j), a some calculations are needed along with the truth value of a precondition (A_i). This corresponds to the following inference rule:

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

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

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

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

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

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

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

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

for the operation subgoal to potentially execute if A_i has happened with f_{ded} being (as in [2]):

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

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

$$desire(A_i) = f_{struct\_ded}(desire((A_i, Op_j)))$$

where f_{struct_ded} defined as:

$$f_{struct\_ded}(f, c) = (f, 0.9 * f * c)$$

Such approach is similar to backward planning from a goal to current circumstances, while preferring to process more attainable goals by taking uncertainties of events and implications into consideration. Alg. 1 summarizes the goal processing in MeTTa-NARS.

Algorithm 1: Decision and subgoaling

```
Input: Goal G Result: Execution of Op, or subgoaling
subgoals = {}, bestDesire = 0.0
 forall ((X, Op) ⇒ G) ∈ memory do
 |  subgoals = subgoals ∪ {X}
 |   if desire(Op) > bestDesire then
 |   |  bestDesire = desire(Op), bestOp = Op
 |  end
end
if bestDesire > DECISION_THRESHOLD then
 |  execute(bestOp)
else
 |  forall s ∈ subgoals do
 |   |  derive s (for potential selection in next inference step)
 |  end
end
```

To allow effective processing, the procedural implications should be indexed by their consequent, where only a constant amount of implications is allowed for each consequent. One way of achieving is by ranking them according to their truth expectation, such that too weak and wrongly predicting implications are removed while those predicting successfully are kept, thus keeping the resource demands bounded. In MeTTa-NARS, however, inbuilt *match* function is used to select all implications with corresponding consequent from both attention focus and concept memory of the system.

Below we will describe select procedures from *procedural.metta* that are viable for carrying out the goal processing discussed above.

(= (AddGoalEvent $Sentence) ...)

AddGoalEvent takes a sentence as an input, converts it into goal event, and adds to the *goal_events* space followed by invocation of *GoalCycle*.

(= (GoalCycle $t) ...)

GoalCycle procedure, similarly to *BeliefCycle* in Declarative Inference, selects the highest priority *goal* from *goal_events*, calls *BestDecision* procedure and maintains the defined sizes of *goal_events* space because it is being modified while processing subgoals.

(= (BestDecision $t (Event ($Term $DV) ($GTime $EvBase $GPrio))) ...)

BestDecision is the central piece of Procedural Inference, it mainly follows the Alg. 1 where estimations of desire values for operations and preconditions are performed given the system has learned procedural hypotheses from past experience. The operation with the highest desire value is selected and executed if its truth expectation is greater than 0.5. Alternatively, the preconditions are treated as subgoals and placed in *goal_events* for further processing.

8 Attention Allocation

The general role of Attention Allocation or Attentional Control is to decide which premises should be selected for inference in the real-time during system's reasoning cycle, which itself needs to finish roughly in a constant time, thereby the data structures should be bounded in size with eviction strategies to maintain this constraint.

In MeTTa-NARS, Attention Allocation is realized implicitly through the use of data structures and explicit estimation of priority values for the events and concepts.

8.1 Data Structures and Corresponding Spaces

There are three (3) atom spaces which participate in Attentional Control of the system and are maintained on the basis of *priority* of items stored whereby priority can be seen as an importance of the item to the current context of the system. The spaces are summarized as following:

- *belief_events*: a collection to hold input and derived events
- *goal_events*: a collection that holds input goals and derived sub-goals
- *attentional_focus*: a collection holding highest priority concepts

All of the above spaces are bound in size by system's hyper-parameter and defined in *memory.metta*.

Belief_events and *goal_events* serve the purpose of the priority queue where items should be sorted according to their priority value and only the highest priority event or goal to be selected during the cycle. The role of *attentional_focus* is to hold most relevant concepts to the current context or experience of the system where all of the concepts within its attentional focus are selected to participate during reasoning cycle as discussed in 7.1. Because of no data structure exists in MeTTa to handle the stated requirements at the time of creating this report generic atom space is utilized, whereby instead of presorting, item selection is performed using the states and bounding of the spaces is accomplished by thresholding.

(= (BestCandidate $tuple $evaluateCandidateFunction $t) ...)

BestCandidate is a procedure that selects the best candidate item from a collection based on evaluation functions (both are input parameters). It either returns the highest priority event based on *PriorityOf* evaluation function or an operation with the highest truth expectation based on *OpExpectation*. *PriorityOf* evaluation function makes use of *EventPriorityNow* helper procedure discussed in 8.2. The selection is made in an iterative manner using *superpose* over the collection and supplementary state to hold the best item so far.

BestCandidate procedure is used for *belief_events* and *goal_events* spaces, while for *attentional_focus* the similar procedure is utilized with the difference of moving all the items that violate the space constraints of *attentional_focus* to the system's long term memory, *concepts* space, instead of being removed and hard forgotten.

(= (BoundEvents $collection $Threshold $Incr $TargetAmount $t) ...)

BoundEvents is a recursive procedure that maintains desired space constraint (*$TargetAmount*) for a given atom space (*$collection*) using a threshold (*$Threshold*). It does so by iteratively comparing priority of an event using *EventPriorityNow* with the threshold. In MeTTa-NARS priority values ranging between 0 and 1, and therefore the threshold is deliberately chosen to be 0 as a starting point. With each recursive call it slowly increasing the bound by ($Incr) factor removing all the items below that bound until the space only contains the desired maximum number of items.

8.2 Priority Estimation

In MeTTa-NARS priority value measures the impotence of an item to the current context and experience of the system such that it will get more attention and expedited processing as a result. Priority consist of a pair of elements (t, p) where t is the timestamp (see 7.2) at which the priority value has been assigned and p is the actual priority. Priority value ranges from 0 to 1 with max priority of 1 reserved for input events and goals only. Priority estimation is therefore

assigns priority values to events, derivations, goals, subgoals and concepts in the real-time during system cycle.

There are two (2) helper procedures that are utilized during selection of best candidates (see 8.1) and priority estimation for events/goals and concepts during the derivation stage:

- (= (EventPriorityNow ($T $P) $t) (* $P (/ 1 (+ 1 (- $t $T)))))
- (= (ConceptPriorityNow ($T $P) $t) (* $P (/ 1 (+ 1 (- $t $T)))))

Given that no special data structures exist in MeTTa, priority of items is not adjusted within the spaces discussed above after an item enters a corresponding space, instead, during the selection and estimation of priority, time relevance of selected item to the current time is taken into consideration ensuring the priority decay rate proportional to the time distance as:

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

or as seen in the Fig 3

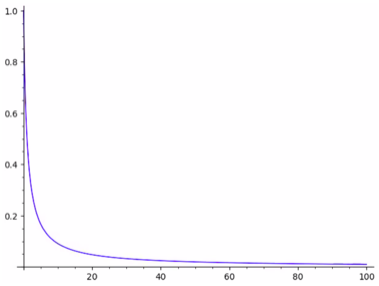

Figure 3: *EventPriorityNow* and *ConceptPriorityNow* priority decay

As of now *EventPriorityNow* and *ConceptPriorityNow* employ the same definitions, however it is made deliberately to introduce other metrics for concept selection at a later development stage.

The actual calculation of priority happens in two places:

- Derivations: for events being derived during declarative inference

  (= (ConclusionPriority $EPrio $CPrio $ConcTV) (* (* $EPrio $CPrio) (Truth_Exp $ConcTV)))

- Sub-goaling: during procedural reasoning when sub-goals are derived

  (= (SubgoalPriority $EPrio $ConcTV) (* $EPrio (Truth_Exp $ConcTV)))

For derived events, the priority is the parent event's priority (the first premise), multiplied with the belief concept's priority (where the second premise came from), multiplied with the truth expectation of the conclusion event. While for sub-goal events, there is no belief concept involved so that factor is omitted. Priority of a concept is initially being inherited from event when the concept is

being created, and during belief revision, concept's priority becomes the largest priority between concept's current priority and the priority of the event triggering the revision.

The long term memory or concept's space employs *forgetting* mechanism whereby during each cycle concepts are being removed on the basis of how recently and frequently they have been used in the past. Two metrics are utilized, namely *last recently used*, *most frequently used* to decide which concepts to evict given the concept' space is full.

9 Tests

There are the following test provided with MeTTa-NARS that thoroughly check the functionality of the system.

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

10 Future Research and Optimizations

While manipulating lists with even 100's of items recursively can take seconds in MeTTa at this stage, some operations on spaces are relatively fast. For instance lookup for a pattern is quick. This can be exploited for instance in evidental base overlap check by generating all the tuples up to a certain length. Furthermore, spaces can be used as sets to check for evidential base overlaps faster than recursive versions. MeTTa-Morph also provides an efficient, fast and deterministic execution of *superpose* statements with support of forking and multi-processing.

Future research and implementation will include more advanced attention allocation control with additional metrics, for example for attentional focus space, as well as enhanced temporal control with larger FIFO and wider sliding window. Given the successful development and application of MeTTa-Morph framework, future version will allow widely increase the current constraint on data structures to allow practical real-world reasoning and application. Lastly, new revolutionary procedural control module will be introduced for casual exploration, planning and decision making and all of it should fit into modular architecture of MeTTa-NARS.

References

[1] P. Hammer and T. Lofthouse, *'OpenNARS for Applications': Architecture and Control*, 07 2020, pp. 193–204.

[2] P. Wang, *Non-axiomatic logic: A model of intelligent reasoning.* World Scientific, 2013.

[3] P. Isaev and P. Hammer, "Memory system and memory types for real-time reasoning systems," in *Artificial General Intelligence*, P. Hammer, M. Alirezaie, and C. Strannegård, Eds. Cham: Springer Nature Switzerland, 2023, pp. 147–157.

[4] ——, "An attentional control mechanism for reasoning and learning," in *Artificial General Intelligence*, B. Goertzel, A. I. Panov, A. Potapov, and R. Yampolskiy, Eds. Cham: Springer International Publishing, 2020, pp. 221–230.

[5] S. Pramod and O. Vyas, "Data stream mining: A review on windowing approach," *Global Journal of Computer Science and Technology Software & Data Engineering*, vol. 12, no. 11, pp. 26–30, 2012.

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