The Conceptual Design of OpenNARS 3.1.0

Pei Wang, Patrick Hammer, Peter Isaev, Xiang Li

October 19, 2020

NARS (Non-Axiomatic Reasoning System) can be analyzed in three parts [Wang, 1995, Wang, 2006, Wang, 2013]:

- its *language* (Narsese) decides what can be expressed as tasks and knowledge,
- its *logic* (NAL) decides what tasks and knowledge can be derived from the available ones,
- its *control* mechanism decides which possible derivations will be actually carried out among the existing possibilities.

At the current stage, the first two parts (as described in [Wang, 2013]) have become relatively stable, while the control mechanism is still under study and development. There have been publications on the conceptual design of the control part [Wang, 1995, Wang, 2004, Wang, 2006, Wang, 2009, Wang, 2012], as well as its open-source implementation OpenNARS[1] [Hammer et al., 2016, Wang et al., 2020, Isaev and Hammer, 2020].

The overall architecture and working cycle of NARS is a major aspect of the control mechanism, and their design has mainly remained the same since OpenNARS 1.0.0, except minor changes. The major motivation of the new architecture in OpenNARS 3.1.0 is to support multiple I/O channels of various modality, including both external and internal experiences without human involvements. This extension also provides the basis for a more efficient and stable control mechanism to be gradually established in the future versions of OpenNARS.

This report provides a comprehensive description of the conceptual design of the new version of OpenNARS with a focus on the control mechanism, and explains the design decisions according to the basic principles behind NARS, that is, adaptation under the Assumption of Insufficient Knowledge and Resources (AIKR).

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

1 Data Items and Structures

1.1 Tasks and knowledge

All activities of NARS are task processing. There are four types of tasks, corresponding to the four types of sentence in Narsese: *judgment*, *goal*, *question*, and *query*.

The tasks are handled using the system's *knowledge*, which includes

- a collection of *beliefs*, each of which is a statement with a truth-value, and summarizes a fragment of past experience,
- a collection of *desires*, each of which is a statement with a desire-value, and hopes a fragment of future experience.

In NARS, desire is a special type of belief, just like query is a special type of question.

Each task is processed in a sequence of inference steps, in each step it interacts with a belief (or desire) so as to become partially achieved, as well as to generate derived tasks, which are expected to lead to its further achieving.

The *achieving level* of a task is a real number h in [0, 1], defined separately for each type:

Judgment: If the expectation of its truth-value is e, and the system already has another belief (with identical content but distinct evidence base) with expectation e', $h = |e - e'|$. If there is no previous belief, $h = |e - 0.5|$. It measures the extent the content of the judgment has been known to the system.

Goal: If the expectation of its desire-value is e, and the system has a matching belief with expectation e', $h = |e - e'|$. If there is no matching belief, $h = |e - 0.5|$. It measures the extent the desired statement is already realized.

Question: If t is the truth-value of a matching belief, h equals the *expectation* of t if the question contains query variables, otherwise h equals the *confidence* of t. It measures the extent the requested truth-value has been obtained.

Query: If d is the desire-value of a matching desire, h equals the *expectation* of d if the query contains query variables, otherwise h equals the *confidence* of d. It measures the extent the requested desire-value has been obtained.

Each task records its current achieving level and refers to the belief/desire providing h, the best solution found so far.

Therefore, what NARS does when running is to repeatedly select a task, then a belief or desire to process the task. After the two are selected, the inference rules decide the extent to which the task has been achieved, as well as what tasks to derive from them as means to further achieve the task. The job of

the control mechanism is to decide how to make the selections for each working cycle.

Under AIKR, task processing in NARS has the following features that distinguish it from the reasoning and problem-solving processes in conventional systems:

- Multi-tasking: new tasks are accepted or derived when the old ones are still under processing;
- No task-level algorithm: the processing of a task may have no available or feasible algorithm to follow;
- Open-ended: few task can be processed "to the end" by taking all relevant knowledge into account under the time restriction;
- Controlled concurrently: the tasks need to be processed in parallel, rather than one after another sequentially;
- Tolerance to under-achieved tasks: the processing of a task may be paused after any number of steps, due to the appearing of other more urgent tasks, no matter what is the achieving level of this task. As extreme cases, the system may have no idea about how to achieve certain tasks, though will not be trapped or crushed by them.

Under this condition, the overall objective of the system is not to reach any specific achieving level on any single task, but to achieve all of them as much as possible, meaning to raise a weighted sum of their achieving levels, where higher-priority tasks tend to get achieved to a higher degree. A control strategy is a concrete approach toward this objective in all conceivable situations.

Since different strategies will be suitable for different situations, and the future situations are unknown even in a statistical sense, there is no "optimal strategy" to talk about, though it does not mean that any design decision is equally justifiable as "adaptive under AIKR." In general, many designs can be considered as "reasonable," each with a different "personality" so is favored in certain (but not all) situations. What can be expected in the following discussion is to find such reasonable strategies.

Since it is impossible to use all knowledge on every relevant task, and the order matters (because some postponed possibilities may not be considered again as the situation changes), the major function of the control mechanism of NARS is to select a task to process in each step, and to select knowledge to process the task. These two selections will jointly decide the applicable inference rules in a data-driven manner. Such an inference step must be carried out in a short time to meet the requirement of real-time processing.

1.2 Budget and bag

Because the objective is to use limited resources on all the tasks, and each of them normally needs multiple steps, a *selection* problem becomes a *distribution*

problem, that is, instead of processing the tasks one after another, each task gets a "budget" indicating its relative share in the system's overall resource supply in the near future. This budget is determined by summarizing all the factors under consideration, including the (relative) urgency, salience, potential, etc. of the task, as well as its relevance to the context. Similarly, each belief/desire also has different accessibility that reflects its estimated contribution to the processing of the tasks. Conceptually, the control problem is not taken as sequential searching a state space, but as asynchronous parallel exploration of the paths, or *controlled concurrency* [Wang, 1995].

Because the present computer hardware does not directly support such a mode of processing, the resource distribution is implemented as priority-biased selections. Here the idea directly come from the resource management mechanism of operating systems, especially the time-sharing of the processor and dynamical allocation of primary storage. However, in NARS these mechanisms are further extended to allow anytime, real-time, and open-ended processing.

In NARS, a *budget* is given to any data items demanding (potentially unbounded) *repeated processing or accessing*. Each budget consists of a *priority* value (p) and a *durability* value (d), indicating the current rank among the items competing for the same resource and how long the rank will be maintained, respectively. The second factor is introduced to reflect the changing in the environment and within the system. This initial definition [Wang, 1995] was later augmented to include a third value *quality* (q), which summarizes the innate features of the item to evaluate its value [Hammer et al., 2016]. Quality decides the item's "base priority," after the context-relevant factor has completely decayed. The "quality-factor" in priority is $q * Q$, where Q is a system parameter *QUALITY_TO_PRIORITY*. Overall, the *priority* value is a function of time that decreases exponentially with a derivative determined by the *durability*, until it reaches a minimum value that is proportional to the *quality* value. A rough diagram of the roles of the three factors is Figure 1(a).

Roughly speaking, *priority* is about the (short-term) context, *durability* the (long-term) history, while *quality* is usually timeless. It is assumed that the current context is not exactly the same as the accumulative history, though they are closely related. Furthermore, different contexts have different time spans. With these independent dimensions, various budget can be specified.

In principle, it is possible to summarize the three into a single measurement (or two measurements), though to calculate, use, and adjust the three separately is conceptually easier, at least at the current stage. In situations where the three factors must be combined to summarize the item's "total budget," a justifiable measurement is $t = d(p + q)/2$, intuitively taking the budget as the area of a trapezoid, with p and q as two sides and d the height, as Figure 1(b). This value does not correspond to the time actually spent on the item (which depends on when it will be removed), but allows different budget values to be compared.

The priority of a task represents its relative urgency, rather than an absolute deadline, because

1. The actual processing time depends on the hardware/software host system

in which NARS is implemented,

2. For an adaptive system, a "soft deadline" (where the utility of its solution is a non-increasing function of time) is a more general form of time pressure, and a "hard deadline" is a special case where the utility drop from its maximum value to its minimum value at a certain moment.

3. Under AIKR, there is no guarantee to meet all deadlines,

As a consequence of AIKR, this need for dynamic resource allocation among items happens in many places in NARS. To handle them uniformly, the data structure "bag" has been introduced [Wang, 1995] as an abstract data structure of a constant capacity and can contain *items* of some type, each identified uniquely by a *key* and ranked by a *priority* (as part of its budget). The major operations defined on a bag are

put an item into the bag. If the bag already contains an item with the same key, the two will be merged, otherwise the item is added into the bag. If the number of items has reached the bag's maximum capacity (a system parameter), the item with the lowest priority is removed.

take an item from the bag. The item is selected probabilistically, and the chance for each item is positively correlated to its priority value.

There are other secondary operations, like access an item by key, etc.

In the current implementation, a bag is an array of lists, each for a range of priority value, plus a hashtable for key-based access, as shown in Figure 2.

Bags cannot be replaced by deterministic priority queues, partly because in this context "priority" does not indicating the *order* of processing, but the *share* of processing time an item obtains when competing with other items over a period of time. In this situation, the processing of each item cannot be finished before the next one starts, so a priority queue often causes starvation of low priority items, even when the priority values are based on insufficient knowledge. Bag realizes a balance between exploitation (of existing knowledge about an item) and exploration (of new knowledge about an item), as it still let the low-priority items be occasionally accessed, so as to given them the chance to increase their priority by proving their values.

1.3 Concept and memory

Since NAL is a term logic, every inference rule in it normally requires two premises containing a common term. Though this requirement looks restrictive, it actually enables the system to keep the premises and conclusions semantically related, as well as greatly reduces the scope of search in the selection of premises and rules in each inference step.

This property further suggests a "concept-centered" memory structure. In NARS, a "concept" is a data structure named by a term, and contains or links

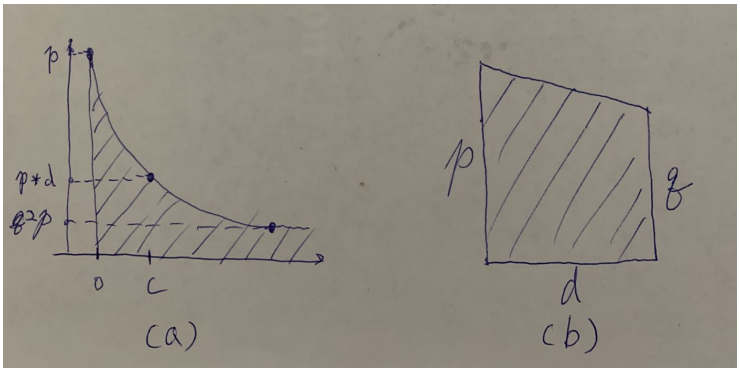

Figure 1: Factors in a budget value.

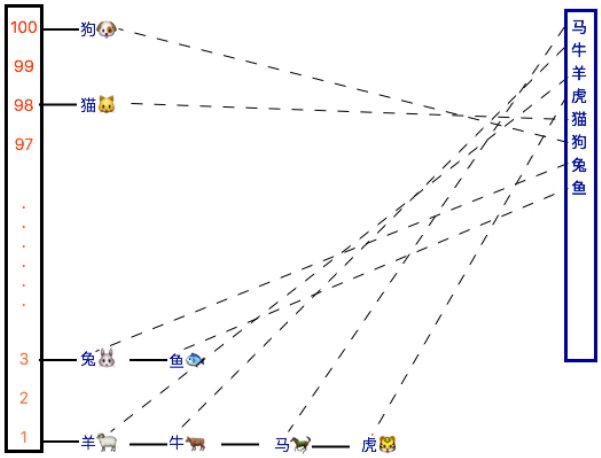

Figure 2: The structure of a bag.

to all the tasks and knowledge that can be directly used in an inference step with that term as the shared term.

For example, if statement $\langle S \to P \rangle$ is the content of a task, then it can and only can be accessed from the concepts named by terms S, P, $\langle S \to P \rangle$, and the compound terms with $\langle S \to P \rangle$ as a component, such as $\langle \langle S \to P \rangle \Rightarrow Q \rangle$. For the last cases, $\langle S \to P \rangle$ must be at a level of depth that it is directly identified in an inference rule. For example, statement $\langle \langle \langle S \to P \rangle \Rightarrow Q \rangle \to R \rangle$ can be directly accessed from the concepts named by R and $\langle \langle S \to P \rangle \Rightarrow Q \rangle$, but not directly from those named by S, P, and $\langle S \to P \rangle$, because each inference rule only parses the syntactic structures of its premises to a certain (usually 1 or 2) level of depth.

Because of this feature, a preliminary implementation of NARS [Wang, 1995] can take a concept as a node, and a task or belief as a link between two nodes (concepts). Figure 3 visualizes this memory, where each node is labeled by a term, each black link represents a belief, and each red link represents a task (either a judgment or a question). A concept is a node with its incoming and outgoing links. In the implementation, a term T names a concept C_T containing a bag of tasks and a bag of beliefs that can be directly used in inference with T as the shared term. The main memory is basically a bag of concepts representing a conceptual network with prioritized nodes and links.

The situation becomes more complicated when the logic is extended to include higher-order inference (NAL-5 and above). When a statement can be used as a term, the links in Figure 3 become nodes (concepts) themselves. In this situation, there are two possible treatments for tasks and beliefs, one is to store them as standalone entities linked from the relevant concepts, or as internal fields of a concept. In the current design, tasks are handled in the former way, and beliefs in the latter way.

One of the reasons of this treatment is that the content of certain tasks (goals and questions) may contain query variable (such as $\langle S \to ?x \rangle$), so is a "compound variable" whose meaning and truth-value can be decided only after being instantiated by a (constant) term. Therefore, such a term does not name a concept. On the contrary, beliefs are always judgments whose contents correspond to concepts. The variable components in a compound like $\langle \langle S \to \$x \rangle \Rightarrow \langle T \to \$x \rangle \rangle$ will not change the nature of this statement as referring to a concept, though its components $\langle S \to \$x \rangle$ and $\langle T \to \$x \rangle$ do not refer to any concept, and nor do they have truth-values, as they are (compound) variable terms, according to Definition 10.4 in [Wang, 2013]. In particular, all compounds containing query variables are variables themselves.

Consequently, a belief can be either contained in or linked from a concept, while some tasks cannot be contained in a concept corresponding to its content. A concept is an identifiable ingredient or pattern in the system's experience, and has experience-grounded meaning, or is "intrinsically meaningful" (no matter how rich its meaning is). On the contrary, variable terms, or certain compound terms containing variables, do not correspond to concepts.

Figure 4 shows a fragment of the memory with 5 concepts and 2 tasks. A task is stored outside any concept, and linked from all concepts where it can be

processed. The link from a concept to a task is a "task-link" and is stored in a bag in the concept. The syntactic relations among terms (between a compound term and its components) are represented by "term-links" which appear in pair, pointing to opposite directions. All the term-links in each concept are also maintained in a bag.[2]

As items in bags, all links have priorities, as marked in Figure 4. The task-links to the same task may have different priorities in different concepts, and opposite term-links may also have different priorities. This is because when the same data item is accessed from different places, its relative priority is usually different.

If a term is a statement, its corresponding concept will have a table of beliefs. Multiple beliefs of the same content may co-exist, as far as each of them has a different evidence base. Even after two beliefs are "merged" by the revision rule, the premises will still be kept, because the information in it is not completely merged into the revised version.[3]

The same treatment is used for desires. For instance, a desire about $\langle S \rightarrow P \rangle$ is stored in the desire table of the concept for $\langle S \rightarrow P \rangle$, which is linked with concepts for S and P by term-links. For a term that is not a statement (like *bird*), there is no belief or desire stored in its corresponding concept.

In summary, a concept provides an intermediate-level structure (between the whole memory and the individual tasks and beliefs/desires) for storage and processing. It contains bags for task-links and term-links, and tables for beliefs and desires, as shown in Figure 5. There are also other concept-level attributes (budget, complexity of the term, etc).

Using the OOP terminology, each concept can be specified as an object of a certain type (the type of the term) that supports a fixed set of operations (including the relevant inference rules), and can be processed independently (even implemented distributively or using customized hardware). The concepts cooperate via message passing, where each message is a task.

This is why the knowledge representation and memory structure of NARS are "concept-centered." Here concepts can also correspond to sensors, operators, perceived patterns, words in human languages, etc., as special cases, as far as they have been experienced by the system. A fragment of such a memory is shown in Figure 6, where a term may correspond to a word or image, or merely be an internal identifier that cannot be directly communicated to another system.

---

[2] In the current implementation, each link has a "type" indicating the syntactic role played by the target term in the relation. The types of term-link are used to invoke corresponding inference rules. An alternative is to use multiple bags, each for a type of term-link. This design will eliminate the complexity introduced by link type, but add another level of selection among the link bags, which may introduce a bias among inference type, and make inference less data-driven.

[3] One possibility is to represent all statements first as links, and only some of them as nodes (concepts) whenever necessary, so as to achieve both capability and efficiency. However, it will make the implementation more complicated.

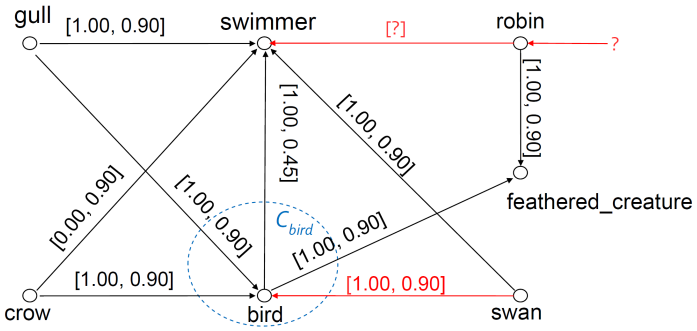

Figure 3: A simplified version of the memory.

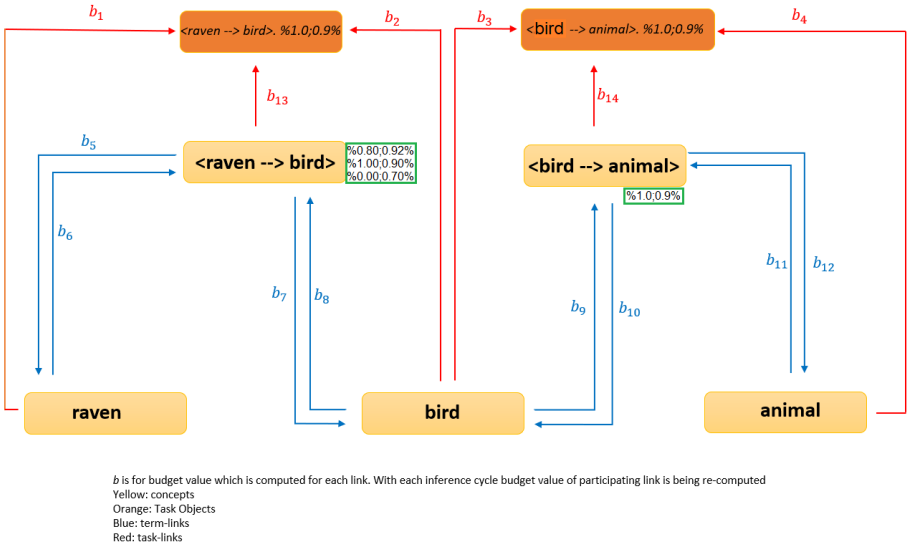

Figure 4: A refined version of the memory.

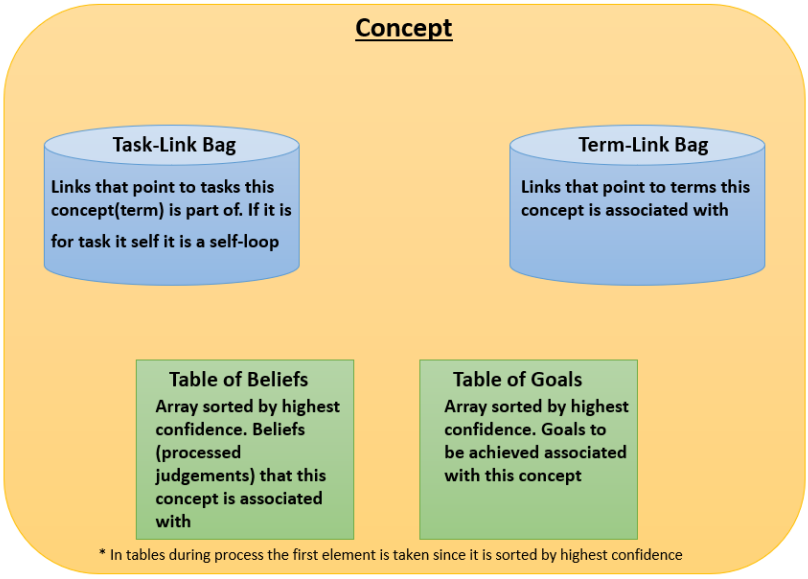

Figure 5: Data structures in a concept. [NOTE: "Table of Goals" should be "Table of Desires," and the selection in the tables is explained in the following, which is not by confidence alone.]

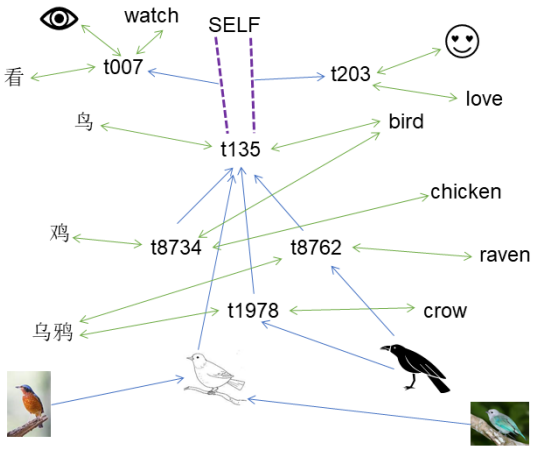

Figure 6: Multimodal memory.

2 Architecture and Units

2.1 System architecture

The previous architecture of NARS is shown in Figure 7, containing a memory, an inference engine, and a single task buffer.

The new version of NARS architecture is shown in Figure 8, where the major extension is the task buffer, while the memory and the inference engine are basically the same as before.

According to this architecture, the system has the following major processing units:

- the inference engine
- the main memory
- channels and buffers

The inference engine is invoked via a single routine *reason*, which normally accepts a task and a belief (or desire) as input. The input triggers some inference rules, and produces a number of derived tasks, which are put into the internal-experience buffer. The number of rules that can be triggered by each input is a constant, and the execution of each rule costs a roughly constant amount of time and space. Overall, the resource expense of each execution of the inference engine has a constant upper bound that is relatively low (at the level of millisecond). The inference engine only provides service, and does not store any local information.

The main memory of the system is a concept network plus linked tasks, as described previously. Its content summarizes the system's experience, and is constantly changing in a non-repetitive manner. Some of its contents (concepts, tasks, beliefs, and desires) will be kept for a long term, some others only a short term, and the two are relatively distinguished and form a continuum, so there is no separate long-term and short-term memories.

The memory has the following major public routines:

accept task: Accept a task from the overall buffer, and link it from all directly related concepts.

consider: Take a concept and carry out inference in it.

A concept has the following major routines:

accept task-link: Pre-process the task using the information local to the concept, then add the link into the task-link bag so as to process it repeatedly in the future.

ponder: A task is selected from the task-link bag, then relevant beliefs are accessed to carry out a limited number of inference steps on the task.

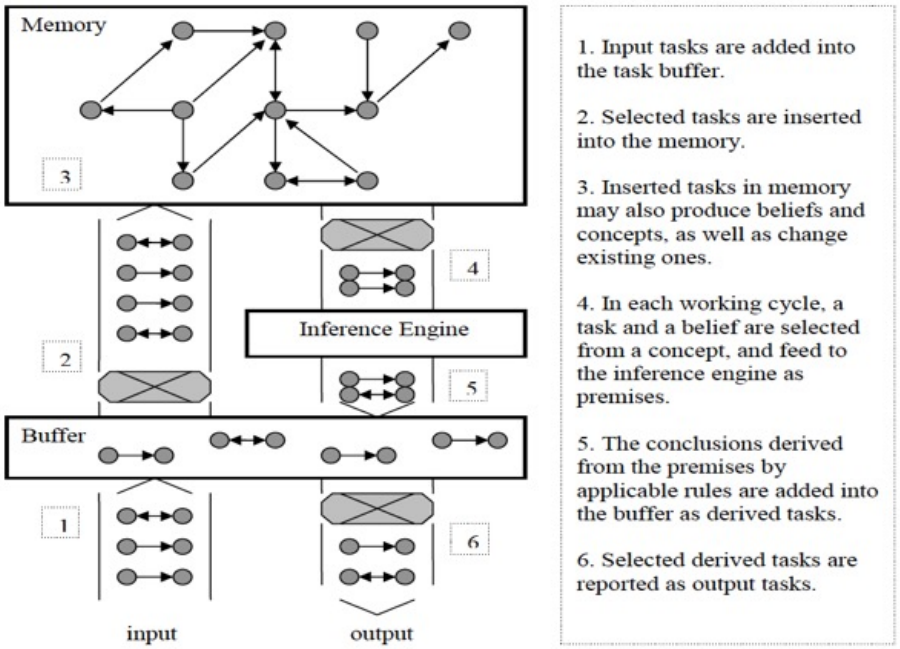

Figure 7: Previous architecture and working cycle of NARS.

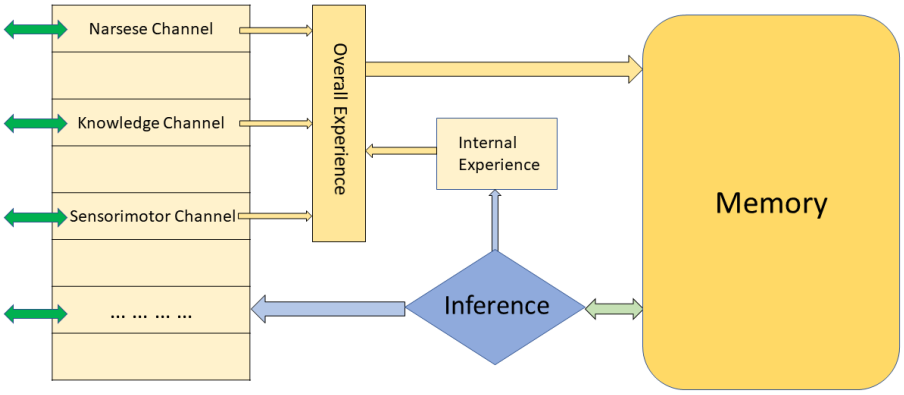

Figure 8: Current architecture of NARS.

Different from memory, the channels and buffers are purely short-term memory, and only hold the tasks to be processed.

A *buffer* is a time-restricted bag of tasks, and also carries out temporal composition in the *observe* method on the task that is taken out of the buffer.

A *channel* manages one type of interaction between the system and the environment. It supports all the routines of a buffer, plus the channel-specific operations that initiate processes outside NARS. A channel corresponding to a modality eventually sends the tasks it formed to the *overall-experience* buffer in Figure 8, where all input tasks from different sources and of different types are mixed with the output of the *internal-experience* buffer whose input are the derived tasks from the inference engine. In the overall-experience buffer, multi-modal compositions happen, and the results are added into the memory.

All of the unit routines have constant upper-bound for their running time. Among them, *put*, *accept*, and *execute* are "on-demand" routines, in the sense that each of them is called to accomplish a fixed service, and does not ask for more resources. On the contrary, *observe* and *consider* are "recurring" routines in the sense that they can be invoked repeatedly, so can consume as much time as available.

The NARS architecture allows the major processing units to run in parallel asynchronously and cooperate by passing tasks. Each channel, buffer, and the memory has its storage and independently defined routines, and the same is true for the concepts in the memory. Consequently, this architecture can be naturally implemented as a distributed system, though it is impossible to give each concept a processor, so some time-sharing is inevitable. Even when distributed implementation or multi-threads are possible, at the initial stage it is still desired to implement the system using time-sharing with a single thread (except for peripheral processes like event-handling in GUI), so as to guarantee the accurate repeatability of the system from the same initial memory and complete experience. Otherwise debugging and tuning will become even more complicated.

In a single-thread time-sharing implementation, a working cycle of the system can be seen as attention allocation among the processing units, that is, the buffers and channels (where *observe* is called), as well as the memory (where *consider* is called). Within the memory, attention is further allocated among concepts, tasks, and beliefs/desires. In principle, the unit selection can be carried out by the *take* operation defined in bag, though in implementation it is not necessary to actually put the units into a bag. Given that the units are predetermined and only a priority (rather than the 3-factor budget) is needed for this allocation, a simpler solution is to go through each unit, and call its recurring method a number of times according to the average priority of its recently processed tasks.

2.2 Buffer

In the new architecture, a *buffer* is a time-restricted bag containing new (input or derived) tasks.

In addition to the common bag operations and properties, every buffer has a time limit on how long a task can stay in it (in terms of the internal clock of the system), and expired tasks are removed without further processing. Consequently, each buffer serves as a filter of new tasks, so the system can focus on the most urgent and recent tasks. The time period allowed in each buffer will be a system parameter, as a factor multiplied to DURATION (which defines the width of the time window for "now"). By default, the factor is 2, so the buffer include the current moment and the previous moment.

It is possible, even necessary, for different buffers to have different duration values. In particular, the *overall experience buffer* should hold tasks for a longer time than the other buffers that passing tasks to it, since cross-modal compositions have greater spans than within-modal compositions. For example, consider the temporal induction of schema "$(cond \;\&/\; \uparrow oper) \;/\!\Rightarrow\; cons$."

Given its time-restricted nature, simple temporal compounds like $(a \;\&/\; b)$ and $(a \;/\!\Rightarrow\; b)$ from events a and b can be composed without demanding the premises to be semantically related. For this reason, temporal (and spatial) compositions of term normally happen in buffers, not in the memory, where semantic relations are required among premises and conclusions. Implication and equivalence statements generated in this way need to be marked in metadata, because only them can trigger anticipations that generate negative evidence when failed to be confirmed. On the contrary, predictions generated in other ways (such as by syllogistic rules) cannot expect to be confirmed by direct observation. This function can be turned off for buffers whether temporal composition is not needed (e.g., a knowledge base channel).

Therefore, a buffer has the following major routines:

put: As defined in bag.

take: As defined in bag, except that if the selected task is already expired, the selection will repeat up to a predetermined times. Also, in buffer this operation is not directly invoked from the outside, but from insider, as part of *observe*.

observe: If the buffer does not carry out temporal composition, this routine just call *take* to get a task, and return it. Otherwise it also uses the selected task and every other tasks to form tasks containing compounds events. The new tasks are put into the buffers. Given their high complexity, most of them will be removed. The remaining ones usually correspond to existing concepts in the memory or tasks in the buffer.

Default time interval is used in the compounds formed in the buffers, and the occurrence time of a compound event is that of its first component, from which the occurrence times of the other components can be determined. This restriction do not limit the system's capability in temporal reasoning, because more complicated patterns, such as when the time interval between a and b is longer than the period allowed by a buffer, can still be formed by the inference engine with the premises selected from the memory, though in that case the two

need to be semantically related somehow, and the inference type is not limited to the above temporal compositions. The non-default time-intervals between two events is represented either by meta-data, or as terms representing special "timing" events.

2.3 Channel

A *channel* in NARS is responsible for a certain type of (input and output) interaction with a device or an environment outside NARS. Both input and output activities are carried out by operators defined on the channel, and the immediate results of the operations are input tasks added into a buffer in the channel.

At the next stage, there will be several types of channel:

Narsese channel accepts input tasks assigned by another system or device at any moment, as will as sends outgoing tasks as answers, questions, or demands to realize active learning and peer-to-peer communication. A Narsese parser will translate the plain text input into tasks. This type of channel will also support graphical user interface, and serve as a console that controls the running of the system.

Knowledge channel sends queries to a knowledge source, receives responses, and convert them from their native format into Narsese tasks using format-specific algorithms. This is how NARS acquires structured knowledge via active learning. This type of channel can also be used to run machine learning algorithms on data sets to get knowledge for NARS.

Sensorimotor channel manages sensors of a certain type. When there are multiple sensors with a stable spatial arrangement, their values at each moment are organized into a sensory term implemented as an array (which may have more than one dimension) [Wang and Hammer, 2018]. "Spatial composition" also happens to capture patterns of concurrent sensations like $(a \,\&|\, b)$ and $\langle a \Rrightarrow b \rangle$ where a and b are sensory terms.

One possible addition is an NLP Channel, which will use existing NLP tools rather than to do it within NARS. The backup database of Memory is not handled as a Channel, as it is controlled automatically, not via operations.

All channels are organized in a hashtable, and accessed by key. Each channel must be predefined, though can be turned on or off at the run time. Meta-data or higher-order statement can be used to remember the source of input tasks, so as to evaluate the quality of a channel, and adjust trust and attention accordingly.

Outgoing operations are usually channel-specific. For instance, a visual channel will control the sensors to move in two or three dimensions. An outside process does not have to finish within a short period (as the mental operations carried out by NARS itself). The feedback or reply of the operation is added into the buffer as tasks.

For certain modality (such as vision), there will be multiple levels of channels to gradually summarize the input. A sensorimotor channel may be realized by a

hierarchy of channels, so the input signal goes through multi-level categorization. This hierarchy is designed, not learned, and it is possible to be specified in the configuration file. On the other hand, compound terms formed in a channel can be a compound term coming out of an arbitrary number of composition, because the results of composition are added back into the buffer as components. This concept hierarchy is acquired from experience, unlike the channel hierarchy.

3 Resource Allocation

3.1 Resource competition

The resource competition among the items in their corresponding unit are summarized in the following:

1. **Tasks** in a system-level buffer: input tasks first compete in the channels, derived tasks in the internal-experience buffer, and then the selected ones in the overall buffer.

2. **Task-links** in a concept-level bag: The budget in a task-link is not necessarily the same as that of a task linked. The task is evaluated with respect to the whole system when its budget is calculated, but that of a task-link is from the perspective of a concept.

3. **Beliefs/desires** in a concept-level table: multiple versions of a belief with the same content but different truth-values compete each time a belief is requested. Bag is not used here because each request is one-time. Multiple versions of beliefs are kept because overlapping evidence may prevent the version with the highest confidence from being used. Desire table is used similarly to keep the desire values of a statement.

4. **Term-links** in a concept-level bag: These links eventually lead to beliefs kept in the target of the link. For example, in concept M a task with content $M \rightarrow P$ is directly linked, but a belief $S \rightarrow M$ $\langle f, c \rangle$ is stored in concept $\langle S \rightarrow M \rangle$ which is linked in M by a term-link.

5. **Concepts** in the system-level bag in memory: Though all inference happens between tasks and beliefs/desires, to take concept as a unit of representation and processing has profound consequences, even though this treatment makes the control mechanism more complicated.

6. **Processing units** in the system architecture: This level of resource allocation did not exist in the previous versions of NARS, where a single input buffer is used to handle input tasks expressed in Narsese, and the synchronized with the memory-based inference. With the addition of multiple I/O channels and internal experience, the system needs to distribute its attention among them according to the current situation.

Space competition also happens in the above cases except the last one. Currently every container has a fixed capacity specified by a system parameter, and when the capacity is exceeded, the item with the lowest priority is removed. This policy is implemented in bag and (belief/desire) table.

3.2 Relevant measurements

NARS is *adaptive* in the sense that it does not process each task according to a predetermined algorithm (for the task or a class of task it belongs to), but according to the system's history and context. Here both "history" and "context" are from the viewpoint of the system itself. For NARS, its the *history* can be recorded by an "experience file" of the system containing the complete input stream since it started with a initial memory; its *context* is provided by the current contents of the memory and buffers, as well as the priority distributions among them.

From the previous description about the architecture and working routines, it can be seen that the current context is formed by history, and decides the processing of the tasks. For the context to form and function properly, some measurements are needed to capture the relevant aspects.

The first group of measurements comes from the logic part of NARS, and summarizes the evidence of certain conceptual relations. Though the control mechanism of the system is not completely determined by the logic, they are nevertheless related, so these measurements are taken into account in resource allocation.

The *truth-value* (and its variant, *desire-value*) of a statement indicates the amount of evidence for its content (and its preferred consequences), and consists of a *frequency* f and a *confidence* c, as defined previously. Derived measurements include

- *ignorance* i is the complement to confidence, $i = 1 - c$;
- *expectation* e is an estimation of future frequency, $e = c(f - 0.5) + 0.5$;
- *sharpness* s indicates the closeness of the truth-value to binary, $s = 2 \times |expectation - 0.5|$.[4]

The *desire-value* measurement is extended from statements to all concepts (terms) to indicate the system's appraisal to the concept's association with the system's overall situation. It is a single value in [0, 1] used for control only, so should not be confused with the desire-table.

There are other measurements belong to the logic part that can be taken as input arguments of budget functions, such as

- *achieving-level* h of a task, as introduced previously;

[4]One alternative is to only use expectation. For a negative belief B, its negation (–, B) will be favored in this way, which will have lower priority than the positive ones by being less simple. However, this will only work among concepts, but not among beliefs, unless (–, B) and B are treated as one concept.

- *syntactic-simplicity* s of a term, defined as $s = m^{-r}$, where m is the number of atomic components within the term, and r a system parameter [Wang, 2013];

- *directness* n of a sentence from input, defined as $n = b^{-r'}$, where b is the length of the evidential base of the sentence, and r' another system parameter.

- *fondness* f of a concept indicates its correlation with the overall status of the system.

The core measurements dedicated to resource allocation are the factors in a *budget* value: *priority*, *durability*, and *quality*. As explained previously, they are defined on each task, (task or term) link, and concept, which are all items maintained in bags. They are both inputs and outputs of budget functions. Also mentioned previously, they are summarized into the *total-budget* measurement for comparisons.

There are also places where resource allocation is not bag-based, but using a single priority value:

- among beliefs and desires in a table within a concept,

- among channels, buffers, and the memory in the architecture.

At the system level, the global evaluations introduced in [Wang et al., 2016] will be realized:

- *satisfaction*: the extent to which the current situation meet the system's desires,

- *alertness*: the extent to which the system's knowledge is insufficient,

- *busyness*: the extent to which the system's time resource is insufficient,

- *well-being*: the extent to which the system's "body" functions as expected.

The current value of each of them can be accessed by a *feel* operator, and the results enter into the internal-experience buffer, as part of the system's self-awareness. They will also be directly taken into account by certain budget functions.

In a sense, these evaluations serve as "innate goals" guiding the system's actions. In that aspect, the desired values of *satisfaction* and *well-being* are 1.0, while the other two are in the neighbourhood of 0.5 (or a system parameter). In principle, *satisfaction* can cover the other three by taking "maintaining the evaluation to the desired level" as a goal. However, it may be easier to treat the four separately, both in conceptual design and in implementation.

The four evaluations are summarized into a single "pleasure value" p to indicate the overall status of the system, defined as conjunction or weighted average of the four $(s, w, not(|2a - 1|), not(|2b - 1|))$.

Even though the global evaluations will have impact on the system's decisions like the other goals, none of them will be taken as a "supergoal" to dominate the behaviors, in spite of the other goals. Instead, they have the same status as far as competing and conflicting among goals happen.

3.3 Budget functions

The budget functions specify the details of resource allocation in NARS, and therefore decide how each task will be processed, given the system's history.

The budget functions are designed according to the basic principles of NARS, with the objective of achieving all tasks as much as possible. Under AIKR, the system should use the limited resources in the most efficient way, judged at the moment according to the available information.

For a specific variable to be calculated, different formula will better serve different situations. Since the future is unknown, there is no optimal formula that guarantees the best results in all possible futures. Instead, each formula corresponds to a systematic preference or bias. In the design stage, what can be expected is to avoid violations of the basic principles, and to balance the relevant factors in a justifiable way. The choices are partially inspired by how the human mind handles similar situations, though NARS is not a descriptive (psychological) model that attempts to simulate human cognition in all details.

The budget functions are part of the system's intrinsic nature designed according to our current understanding about intelligence, and are not learned from the system's experience, though some parameters may be adjustable at run-time by the system itself. All parameters can be specified in the system's configuration, and collectively decides the "personality" of a specific implementation of NARS.

The input and output arguments of the functions have been summarized in the previous subsection. As the truth-value functions, each budget function is designed in the following procedure:

1. For the output, identify the relevant ones among the measurements listed in the previous section to be used as the input;

2. Analyze the relationship of the inputs and the output, and represent it as a function using Boolean functions (*and*, *or*, *not*) and other simple arithmetic functions (*minimum*, *maximum*, *weighted average*, etc.);

3. Rewrite the function as a real-number function by extending the Boolean functions to real number.

Since this design procedure depends on our current knowledge, all the functions are subject to future revisions coming from further theoretical analysis and empirical testing.

In the following, the functions are clustered according to where they are used in NARS.

[Bag]

This group of budget functions is shared by all items stored in bags (tasks, links, and concepts). The initial budget of an item is decided outside the Bag, though will be used and adjusted within the Bag by the following functions:

Merge: When an item is added into a bag where there is another one with the same key, the two will be merged, with their budget accumulated. In this process, the two *quality* values should be the same, and if not, the *max* operator is used. The two *priority* values are combined using *or*, so that the result will be no smaller than either of the two, while still remains in the [0, 1] range. For the same reason, *or* is used to combine the two *durability* values. Consequently, repeatedly appeared items will get more resources, both at the moment and in the near future.

Take: The rate for an item to be accessed is positively correlated with its *priority* value.[5] The default function is to let this correlation be a linear function, so the access rate of an item is approximately proportional to its *priority*. The access rate of each priority level is according to a constant distribution in the index array *DISTRIBUTOR*.[6] The priority of the selected item is used to update the busyness level of the bag.

Decay: The *priority* value of every item decays at a rate specified by its *durability*. Since modifying all items in a bag in every cycle would be too expensive, the adjustment is made each time an item is put back into the bag after processing ("lazy forgetting")[7]. If the budget at a certain moment is (p, d, q), then after a period of a constant length, the *priority* decays from p to $p * d$.[8] Since in this period the number of times the item is processed is roughly propositional to p, each time the priority should be updated to $p * d^{-(p*C)}$, where C is a parameter *FORGET_CYCLES* indicating the number of inference cycles taken as the above constant period for the durability to be fully applied. The larger this parameter is, the slower the decay (relative forgetting). This parameter should be separately defined for each type of bag (though not for each individual bag), and can be dynamically adjusted at run time (even for individual bag) according to the average priority of the items to avoid the items to

[5] A possible future implementation is to use GPU to carry out parallel composition in a buffer. However, even in that case, priority-biased selection will still be necessary.

[6] The effect of a nonlinear function may be obtained via forgetting, or by changing DISTRIBUTOR.

[7] Though "lazy forgetting" only roughly approximates the desired "continuous forgetting" in the long-run, it is probably the most feasible solution in the existing hardware. Even though parallel processing is possible to an extent, it cannot be expected that the priority of all data items decays constantly altogether. One variant of lazy forgetting is to implement the levels within a bag by a circular array where the top-level is increased after each put or take, so all items decay together, though their stored priority values remain unchanged until each budget is re-evaluated.

[8] One alternative is to replace the p here to $p - q * Q$, so that the result will not be lower than $q * Q$, the long-term priority determined by quality.

crowd at high or low levels. Since *quality* indicates long-term priority, this decay is applied until $p < q * T$, where T is another system parameter $QUALITY\_TO\_PRIORITY$. For example, of this parameter is 0.1, then item with *quality* 1 will not have a *priority* lower than 0.1.

Remove: When a bag is full, the item with the lowest priority is removed. This is the only place the space resource is maintained at run time. All the other considerations and restrictions on space are all specified as system parameters that normally cannot be modified when the system is running.

[**Task**]

The initial budget of a task has several alternative sources:

- An input task may come with a budget provided by a user or another system according to the definition of budget in NARS.[9]
- System parameters can be used to provide default budget for each type of task (for instance, *goals* and *questions* usually have larger budgets than *judgments*). These defaults can also be specified at the channel level.
- *The priority* and *durability* of a derived task is the conjunction (*and*) of the corresponding factors of its parents (task and belief), as well as a factor indicating the type of inference. The *quality* of the derived task comes from its content (complexity, truth-value, etc.).

When a task enters the overall buffer from a channel or the internal-experience buffer, its priority is adjusted according to the current context, as indicated by the global evaluators and the priority of the corresponding concept, so as to favor the tasks that are relevant when the system is satisfied, alert, idle, and feeling well. The function can be a weighted average of the previous task priority and the corresponding concept average, multiplied by the disjunction of the global evaluations.[10]

Within the memory, each time a task is directly processed by a concept with its local knowledge, its *achieving level* h is updated if necessary. Based on it, the *priority* of the task is decreased by multiplying $1 - h$, so that the relatively achieved tasks will get less resources. This adjustment is applied before the across-the-board decay in bag.

If the task is a judgment that provides a best-so-far solution to a pending goal or question, the *achieving level* h of the latter is used to increase the priority of the task from p to $or(p, h)$. This reward will facilitate forward inference invoked by backward inference, as it will use the belief associated with the goal or question, without a probabilistic selection of beliefs.

[9] At the current stage, an input budget will be accepted as it is, though in a future version channel-specific adjustments may be added as a *budget discount* by multiplying a factor in [0, 1] to the priority and/or durability value of each input task from the channel. The initial value of the factor can be specified when the channel is created, and revised according to the quality of the input.

[10] An alternative is to make this adjustment when the task enter the memory from the overall buffer, or to do it at both places, with different factors considered.

[Task-link]

The budget of a task-link is created when a task enters memory and is linked from the relevant concepts. Initially the budget of each link is the same as that of the task, and is adjusted by the *achieving level* in the same way as described above.

Even so, a link budget and the corresponding task budget will gradually become different, as the links compete at the concept level, while the tasks compete at the system level. Eventually, the links to the same task will get different budgets in different concepts, and the task will be accessed with different chances from different concepts.

[Belief and desire]

When a judgment task is added into memory, a belief is added into the belief table of the concept that corresponds to the content of the judgment. For example, the input task $\langle S \rightarrow P \rangle[t]$ will make $[t]$ and the related meta-data (evidential base, occurrence time, etc.) to be stored in the belief table of the concept $\langle S \rightarrow P \rangle$. Consequently, all beliefs in the table have the same content, but different truth-values coming from different evidential bases and time stamp. In particular, after a revision, the conclusion $st[t]$ and the premises $st[t_1]$ and $st[t_2]$ may coexist in the belief table of concept st.

Though belief with the highest confidence or expectation is usually preferred (e.g., by the choice rule), multiple truth-values for the same statement are remembered, for the following reasons:

- the evidence base of one truth-value may overlap with that of the task,
- high confidence truth-values tend to be equivocal and fail to provide guidance,
- in certain inference (such as analogy and metaphor), one-sided beliefs are used deliberately,
- context dependency and implicit conditions also ask partial evidence.

Since the selections here have different criterion than the resource-distribution implemented by bag, belief table has been organized as a deterministic priority queue. The proposed changes are:

- keep the table small (compared to the bags),
- the priority value is a summary of *confidence*, *sharpness*, *directness*, as well as an indicator of the recentness of the truth-value,
- priority is mainly used for space competition, and all beliefs will be considered for each task,
- for each task, only sufficiently different beliefs (in expectation) will be passed to the inference engine,

- previously each task will remember the beliefs it has been used with, which will be changed to remember term-links, instead.

The same treatment should be used for the desire table, as desires are beliefs on certain consequences.

[Term-link]

A term-link is used to access the beliefs in the target concept identified by a term, so it is like the "belief-link" in the earlier versions of NARS, though in its current form the target of the link is not a belief, but a concept where the belief is stored.

When a belief is added into a concept, term-links are sent to all the relevant terms where it may be used for task processing, with a initial budget coming from the task that created the belief. When a concept accepts this link, it is added into its term-link bag, where it may be merged with other links coming from the same concept triggered by different tasks.

Each time after a task is selected in a concept for processing, a few beliefs are selected by following one of the term-links. To increase the context-sensitivity of the selection, one option is to take multiple links from the bag first, adjusting their priority values according to the priority of the target, then choose the link to follow according to the adjusted priorities.

After each time a belief is used in inference, the term-link accessing it will be rewarded by a value r that is the maximum of the priority of the derived task (or the achieving level of the processed task). This reward can be considered as indicating the usefulness of the belief, and is used to increase the quality of the link from q to $or(q, r)$, so the link will remain active longer. Another possibility is to relate q to the weighed average of the achieving levels of the tasks processed by the belief.

Unlike task-links that tend to exhaust their budgets sooner or later and get removed from the system, many stable term-links form the core meaning of the concepts they link to by keeping in the middle-range of the bag.

[Concept]

A new concept in NARS is introduced when a task adding into the memory contains a term without an existing concept. This novel term may happen from several sources:

- input from a Channel,
- composed/decomposed by a compositional rule,
- coined by the mental operator *compile* for a compound.

In all cases, the initial priority and durability values of the new concept are from the task (the first two cases) or the compound (the last case). The quality value is the simplicity of the term.

Obviously, the use of compositional rules brings the challenge of combinatorial explosion. The NARS solution is again the experience-grounded semantics. Here a new compound term is introduced as an attempt to efficiently summarize the system's experience, rather than to describe the world, not to mention all "possible worlds." Consequently, a new compound is formed only when there are evidence supporting such a summary, such as existing extension/intension, experienced temporal/spatial pattern, etc., and its budget is continuously adjusted according to relevant evidence.

In particular, the compound terms formed in sensorimotor may come mostly from decomposition than composition, i.e., the incoming data forms temporal/spatial patterns directly, which are later decomposed into parts, and those that turn out to be useful and frequently used become patters themselves. NARS does not search a space of possible compounds, neither randomly generates compounds than selects the good ones.

Since each concept is uniquely identified by a term, in the concept bag of the memory the *merge* function is never applied, and only the *decay* function works to constantly decrease the priority values. The increasing of a priority p is carried out by an *activate* function that is called each time a new task-link or term-link is added into the concept with a priority p', and the new priority is $or(p, p')$. It realizes the priming effect or activation spreading. This is also the place for the desire-value of the term/concept to be taken into account. Concepts associating with positive emotions will surely be activated. Whether the concepts associated with negative emotions are also activated or prohibited is still an issue.

The *durability* of a concept is the accumulation of the durability values of the links. If its current value is d and the value of the incoming link is d', the new value will be $u * d' + (1 - u) * d$, a weighed average of the two, with the updating rate u as a system parameter.

The factors deciding the *quality* of a concept include the syntactic simplicity of the term and the average sharpness or quality of the beliefs contributing to the meaning of the concept. The latter can be updated each time a belief is inserted or accessed, using a formula similar to that of the above durability.[11]

The *fondness* f of a concept indicates its correlation with the pleasure value of the system. After each time a concept is pondered, its f is adjusted to f' by the current pleasure value p as $f' = r \times p + (1 - r) \times f$, where the updating rate r is a system parameter. In general, concepts with extreme fondness will get more resources.

Concept-level resource allocation provides NARS with cognitive functions not available at the sentence (task and belief) level. For example, adding a task to a concept may trigger the processing of a dormant task in the concept, and cause the latter to be processed in an unexpected way. Such an effect provides an explanation of "inspiration."

[11] Another consideration is to favor the "basic-level" concepts, that is, whose with a relatively balanced extension and intension. However, this factor may be implied by the usefulness of a concept, so does not need to be considered separately.

One future addition is swapping between primary storage and secondary storage, so as to only keep the active concepts in the former (RAM). There will be two thresholds in the priority of concepts, one for swapping and the other for removing. The secondary storage (HD) will be used like a notebook, which is larger than the primary storage, but still has a constant capability, otherwise processing time will become unbounded. The thresholds are initially determined as part of configuration, then adjusted according to the busyness of the system. Similarly, concepts will be the units in distributed implementations and direct hardware/firmware implementations of NARS.

[Processing units]

In each working cycle the system *observes* a channel/buffer or *considers* a concept in the memory. The relative frequencies of these operations indicate the top-level attention allocation strategy.

This frequency u will be positively correlated to the weighted average of the priority p of the recently accessed tasks, i.e., $u' = r \times p + (1 - r) \times u$, where r is a system parameter. Current value of this measurement is kept in a variable (call it "busyness" or "priority"?) of Buffer and Concept.

This allocation is managed by two complement mechanisms:

automatic: These processing unites are visited in a round-robin fashion. At each unit, the recurring operation is repeated, according to the value of u. For example, the number of repetition is 1 by default, but can be 0 if $u < 0.2$, 2 if $u > 0.7$, and 3 if $u > 0.9$.

deliberate: The related operations (*observe*, *consider*, *ponder*) can become goals and be invoked as mental operation by the system itself.

[Global evaluations]

The global evaluators are new additions to the NARS architecture. At this stage of the design, each of them, g, will be an accumulation of certain local measurement, m, using a weighted average $g' = r \times m + (1 - r) \times g$, where the updating rate r is a system parameter multiplying the priority of the item on which m is defined. All the updates happen after a task is processed in a *reason* or *accept* operation. Their current values will be displayed approximately in the GUI.

The correspondence between global and local measurements is listed in the following:

satisfaction: achieving level of a goal,

alertness: (also known as novelty) achieving level of a task that is a judgment or a question,

busyness: priority of a task,

well-being: the extent a *feel* operator returns a value in the normal range, to be used in implementations where the system needs to manage resources other than time and space (such as energy) and its own body (such as temperature).

Initially these global evaluations will be used only when a new task is accepted into the system. Later they will be used to control various operations.

4 Self-control

Operations can be executed either by NARS itself or a device or system outside NARS. The former is called *mental operations*, and the latter *physical operations*. Examples of physical operations include commends to sensors and actuators in a robot, which are device-specific and not discussed in this writing.

As complementation and augmentation of the automatic control mechanism described above, NARS can use mental operations to control its own working processes.

In principle, all the working routines listed previously for the unites can be "*operationalized*," i.e., being turned into operations executable by the system as a decision made in reasoning, though in practice only some of them should be handled in this way.[12] At the current stage, the way to go is to associate mental operations to the corresponding methods in the OpenNARS implementation of NARS, so the execution of such a method will cause a task to be added into the system's internal experience buffer as a record of the event. On the other hand, when a mental operation becomes a goal, it will be executed by invoking the method. The operator and routine name may or may not be the same.

A list of built-in concepts will be created or loaded for the mental operators, as part of the initialization of the memory. The current "built-in terms" include (1) all mental operators, (2) the four global evaluators, (3) `SELF`. Each built-in term also has the corresponding concepts created, and will be handled as the acquired terms. Some mental operators may have innate knowledge. To get the knowledge about the preconditions and consequences of each mental operation, some experimental executions (*operation babbling*) may be necessary, though they bring risks and dangers, besides resources expense. It is still unclear whether operationalization can fully replace the need of random experiments.

Here is a list of operator–method associations:

believe(s), want(s), wonder(s), assess(s) associate with *acceptTask(t)* in memory, where the task t is generated from the sentence s for each of the four task type (*judgment, goal, question, quest*), respectively, with a high priority value. By executing such an operation, the task selection process

[12] In the future, when OpenNARS is completely re-implemented, this requirement should be carefully considered at the beginning, so the same code can be invoked both as an operation (consciously) and as a routine (unconsciously).

in the buffers will be skipped, though this task will still need to be added into the relevant concepts to interact with the beliefs there.[13]

tell(s), demand(s), ask(s), check(s) generate tasks like the previous group at a specific Channel, and send the task to the system or device connected to NARS using proper format.[14]

observe(channelID) associates with *channelID.observe()*, and carries out a working cycle at the specified channel, and returns a task to be added into the overall buffer.

remind(term) activates a concept, just like after accepting a task.[15]

compile(term) is invoked when a compound term have a large total budget and a low simplicity (e.g., measured by *and* of the two). This operation will coin an atomic term and build a similarity statement with truth-value $(1, c)$, where c is a systematic parameter, between the new term and the compound. In the compound, this term-link will be handled specially, so the compound will not be complied more than once. In the long run, the two terms (concepts) may develop different meanings, with the compound keeping the literal meaning, and the atomic the holistic meaning. Eventually, one of them may be forgot, while the other is still active.[16]

register(operator, command) associates the *operator* with a *command* of a system or device connected via a Channel when NARS is running.[17]

feel(v) associates with the updating of the global evaluators at the end of each working cycle. When the value is too high or too low (say, at the top or bottom quarter), an event (e.g., *feel(satisfaction)*) is created, with the value as frequency and a default confidence, then added into internal experience. It can also be called consciously to evaluate the measurement.

---

[13] It is probably not a good idea to have a mental operation that directly calls *reason(task, belief)* in the inference engine. Instead, the consequence can be expected by executing *believe(s)* for the belief and one of the above four for the task.

[14] To use these operations, NARS needs to do channel-specific high-order inference. This will be related to knowledge acquisition and active learning. In principle, an "incoming task" for the system itself to do becomes an "outgoing task" for someone else to do when the expected achieving-level are higher in the latter for goals and questions, while the outward judgments are answers to the input questions or goals, as well as preconditions for the related outgoing goals and questions – if the system needs the help of another system, the related background knowledge should be provided unless it can be assumed to be already known by the other.

[15] The operator *ponder(term)* is postponed to a future version. Hopefully *remind(term)* can already achieve the desired results.

[16] Two possibilities to be compared when coining a new term for a compound, such as $(a\&b)$: (1) like in Prolog, use an internal counter to get unique names, such as $term\_1204$, (2) keep the string of the compound, though omit the structure, such as $'a\&b'$, where the quotation mark indicates that the term is atomic. The two have no difference for the system, but suggest different meaning to human readers.

[17] The current system only allows built-in channel operations, though will allow run-time registration in a future version.

As a result of temporal composition, some mental operations will be recognized as causes or effects of certain events.

5 Future Works

Besides the issues mentioned in the above footnotes, there are issues to be addressed in the future versions.

5.1 Remaining issues

There are still remaining issues to be resolved in conceptual design before implementation is considered.

- By default, task derivation only generates tasks of the same type. However, there are exceptions. For instance, a judgment-task T may generate question-task $\langle ?x \Rightarrow T \rangle ?$ when T has a low achieving level and high priority. It is related to curiosity, explanation, active learning, etc. What needs to be decided is whether to add these derivations directly into the inference routines with built-in triggers, or to handle them using mental operators with learned conditions.

- The proposed operators *doubt, hesitate*: These operations demand a decrease of the confidence of a belief or desire, which cannot be handled as revision, forgetting, or temporal projection. One possibility is to call them in revision to penalize the contradicting premises. Another option is to leave the function to higher-order inference, rather than to mental operator. After all, deliberative reasoning, like in mathematics, does not fully depend on truth-values. Here the treatment should be consistent with how illusion and magic are handled.

- The proposed operator *assume* will introduce a belief without evidential base. Though such a "groundless belief" cannot be used to resolve a question or goal, nor to revise a grounded belief, it can be used in hypothetical inference. One possibility is to omit the truth-value of some existing beliefs to explore their implications. This issue is probably related to the "analytic truth" introduced in NAL-5, as well as to the deriving of questions of the form "$\langle T \Rightarrow ?x \rangle ?$" from "$T?$". One possibility is to leave hypothetical reasoning to explicit applying of learned inference rules, so there will not be a need for this mental operation.

- The proposed operators *count, compare, calculate*: They are easy to implement, but their relations with the working routines need to be studied. This is related to the role of mathematics in thinking.

It is still unclear in what sense a set of mental operators can be considered as "complete." Since the completeness of NAL is argued with respect to Narsese ("All Narsese expressible sentences are NAL derivable"), the completeness of

NARS probably should be something like "All NAL derivable processes are realizable under conscious control."

5.2 Additional topics

The following topics are also crucial for the control of NARS, and they roles cannot be replaced by the design discussed in this report:

- education materials and procedures,
- evaluation metrics,
- configuration for special needs.

In the future, the configuration can be the result of an evolution process, so the parameter values may move beyond the initial choices and what a system can learn within its life-cycle.

References

[Hammer et al., 2016] Hammer, P., Lofthouse, T., and Wang, P. (2016). The OpenNARS implementation of the Non-Axiomatic Reasoning System. In Steunebrink, B., Wang, P., and Goertzel, B., editors, *Proceedings of the Ninth Conference on Artificial General Intelligence*, pages 160–170.

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

[Wang, 1995] Wang, P. (1995). *Non-Axiomatic Reasoning System: Exploring the Essence of Intelligence*. PhD thesis, Indiana University.

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

[Wang, 2006] Wang, P. (2006). *Rigid Flexibility: The Logic of Intelligence*. Springer, Dordrecht.

[Wang, 2009] Wang, P. (2009). Case-by-case problem solving. In Goertzel, B., Hitzler, P., and Hutter, M., editors, *Proceedings of the Second Conference on Artificial General Intelligence*, pages 180–185.

[Wang, 2012] Wang, P. (2012). Solving a problem with or without a program. *Journal of Artificial General Intelligence*, 3(3):43–73.

[Wang, 2013] Wang, P. (2013). *Non-Axiomatic Logic: A Model of Intelligent Reasoning*. World Scientific, Singapore.

[Wang and Hammer, 2018] Wang, P. and Hammer, P. (2018). Perception from an AGI perspective. In Iklé, M., Franz, A., Rzepka, R., and Goertzel, B., editors, *Proceedings of the Eleventh Conference on Artificial General Intelligence*, pages 259–269.

[Wang et al., 2020] Wang, P., Hammer, P., and Wang, H. (2020). An architecture for real-time reasoning and learning. In *Proceedings of the Thirteenth Conference on Artificial General Intelligence*. To appear.

[Wang et al., 2016] Wang, P., Talanov, M., and Hammer, P. (2016). The emotional mechanisms in NARS. In Steunebrink, B., Wang, P., and Goertzel, B., editors, *Proceedings of the Ninth Conference on Artificial General Intelligence*, pages 150–159.