Multi-channel design in OpenNARS 3.1.3

Tangrui Li (tuo90515@temple.edu)

Introduction

In this version, multi-channel functionality is available, but in a limited scope. Direct Narsese input is temporally disabled, and all inputs are generated by sensorimotor channels (and buffers). This may lead to an unsatisfying result that there are no (or only a few) inheritances generated, all knowledge in NARS is about the temporal order of events happening. But currently, we acknowledge this problem, and we think it is reasonable, which is a deficiency under the conceptual design of OpenNARS 3.1.3 since it is the simplest multi-channel design.

The most important feature of OpenNARS 3.1.3 is about the introducing (and re-designing) of buffers which builds a bridge from the outside world and the internal experience of NARS. If the conceptual design of OpenNARS 3.1.3 is implemented successfully, reasonable temporal inference results will be generated and combined among different channels.

In the following sections, details about the implementation will be explained.

Basic working cycle

In previous OpenNARS versions, the knowledge (Narsese sentences) is input by the user which is not too different from some other symbolic reasoning systems. But in OpenNARS 3.1.3, we assume NARS is placed in an interactive environment and it is able to observe and change the world by some a priori design. For example, it's like an unmanned rover that uses its probes and equipment to explore the surrounding environment on the surface of Mars. But please do not expect a lot, since it is the simplest design, like Pavlov's dog, only some straightforward temporal relations are assumed to be found. For those deeper abstractions, we will leave it for the future versions.

Compound generation

Observations are generated by different channels. Take the vision channel as an example. While watching, we human beings usually have a focus point, and we may recognize the entity in the middle of the field of view easier compared with the objects in the peripheral area. But for computer vision, like using neural networks for object detection (e.g., using Yolo), all objects are detected simultaneously. Neural networks may give different objects different accuracy scores, but it does not have a focus. To reconstruct this focus, even different events are arrived at the same time, they have different priorities to let NARS know what should be focused on.

Of course, as a symbolic reasoning system, combinatorial problem should always be avoided, therefore, the first step in the event buffer is to generate compounds to increase the sparsity. Like in the following image, we tend to say it consists of 4 combinations of black and white blocks, instead of 16 blocks. Once a compound is generated, its components will receive less attentions. For those who are not familiar with non-axiomatic logic, don't worry that we have generated compounds incorrectly. Uncommon patterns will be gradually eliminated due to their rarity in the accumulation of experience. Currently, this combination generation process is determined by the

similarity of priority in concurrent events. Events with close priority will be combined. We may find some counterexamples to show the drawback of this approach, but it is not fully determined, if we find better approaches, we can make some ameliorations in the future.

It is easy to acknowledge that there is a necessity to make some compounds in concurrent events. Like in an image, we prefer to describe it with included objects but not pixels. But don't forget to pay attention to the history of events. For example, a red or blue light that is always on, and a police light that flashes between red and blue are indistinguishable in the same frame, so we need to make some combinations within a few time slices. Currently, this is achieved by considering the most prominent current event (could be a compound) and the most prominent events in the past. To make sure NARS's thinking considers the history and not stuck by it, currently we use a dynamic programming like style to decide what is the most prominent event in each time slice. In creating compounds among time slices, one time slice's event can choose whether to continue a previous temporal pattern, or to make a new episode.

Local evaluation

The second step in the event buffer is local evaluation. It has two sub-steps, one is prediction firing, another is anticipation checking. Like some popular natural language models based on the attention mechanism, NARS also makes predictions to build connections among events. For example, in our previous police light case, if the red light is recognized, the blue light is predicted to appear in the next time slice. Therefore, we will check all events and generated compounds in the first step with predictions (its generation process will be introduced in the 4^{th} step). A prediction that satisfies the precondition will create an anticipation at the appropriate time slice. After generating all anticipations, NARS will check whether there are some anticipations generated before in the current time slice. If we find the anticipation in events and compounds,

we say it is satisfied, otherwise it is not satisfied. Satisfied anticipations will make their parent predictions more credible, and vice versa. Note that we revise the observation with the satisfied anticipation, so if the anticipation is strong enough, the observation might be overturned. For example, typos. We have expectations of the word, but it is inconsistent with its spelling, however, we will hold our expectations. But it is not always the case, for example, before micro-physical theories were proposed, we would use macro-physical theories to explain phenomena. Although the facts did not match the calculated results, we did not forcefully believe that the facts were incorrectly recorded. Instead, we thought about whether our expectations were wrong. This is reflected in those unexpected events and compounds. NARS will think about why they are unexpected, so their priority will be increased for further thinking.

Memory-based evaluation

The third step is called memory-based evaluation, that is all events and compounds are check against the memory. When we are thinking about problems, we always want our attention to be focused. Reflected in NARS, we do not want the information provided from the event buffer to increase the number of concepts with the highest priority in memory, since this is equivalent to distract it. Therefore, we will check these events and compounds, if they have a higher priority in the memory, their priorities will be higher, otherwise will be lower. This may introduce a countereffect that NARS will become more and more focused on certain concepts. We are still thinking about this problem and some solutions will be applied in the future.

Prediction generation

The last step is called prediction generation. With the help of previous three steps, compounds are generated and checked against the memory, so NARS can dig deeper into the possible reason of the most prominent event (or compound). This will generate predictions with previous most

prominent event (or compound) as the precondition and the current most prominent one as the result. Note that there is a capacity for predictions, if it is max out, the prediction with the lowest priority value will be dropped.

For the internal experience buffer & overall experience buffer

These four above steps are finished in one time slice. After one time slice, the oldest slice will be dropped, and a new slice will be created, at the same time all time slices will “getting older”.

As in the conceptual design of OpenNARS 3.1.3, we have three types of buffers. It is described that the internal experience buffer should consider the feeling of NARS but this is not achieved in this implementation. The overall experience buffer is not too different from the event buffer, since it only introduces concurrent implication (=|>) additionally, which is implemented in compound generation. While generating concurrent compounds (&|), corresponding concurrent implication will also be generated.