Modeling mind state transitions based on the thesis:
Discrete state of consciousness

Radbout Universiteit Nijmegen
Department intelligent systems
Anthony mahabir (3517527)
Supervisor: Henk Barendregt
24-11-2014

# Content

# Introduction

This is an experimental project with the goal of creating models of mind state transitions. This means that the models are not based on experimentally determined parameters, nor transition rules of existing models. However, pre-defined categorical states are used from inspiration that already exists in relevant research fields. Specifically this project is confined to mind state transitions in a meditative context. Fields of interest that relate to meditation, include neuroscience, psychology and artificial intelligence.

The essence of free modelling is based on mathematical relations that are represented by mathematical functions. The core of the model, that will be described, consists of a functional program that encodes these mathematical functions. A functional state in the program is represented by pre-defined states and its concomitant parameter-values. To visualize the ongoing dynamics of mind state transitions the parameter-values are translated to a visual representation in the form of a dynamic flowchart. Here, the states are visualized as nodes and the transitions in the form of a moving agent, while most of the parameter values remain implicit in the code.

One advantage of these models is the possibility to study mathematical phenomena. Because the core of the model is expressed by mathematical relations, one can abstract from definite parameter values, and instead experiment with variable parameters while preserving relational properties. For example, it can be analyzed whether there are any bifurcation values forming the transition between going towards different eventual outcomes. These values could then possibly be expressed in a mathematical way, depending on the used transition functions, to draw areas in phase-space from which the eventual outcome is determined or strange attractors to which the eventual outcome evolves.

Worth to mention is also the fact that in the history of physics mathematical models sometimes predated realistic ones. Take for example the many non-Euclidean geometries of which some where useful for relativity; or the Hilbert spaces with operators of which some where useful for quantum mechanics. This is not to say that mathematical models could always predate realistic ones, nor to exclude mathematical modelling for reasons of validity. Besides, in mho it seems doubtful that so much is known about brain-processes that the tendencies and refined behavior that leads to certain eventual outcomes can accurately be described, leave alone reliably be predicted. That is to say there is no absolute superior golden standard at the moment that one must follow.

Before discussing the mathematical model, first the relevant background information is given, that is about the mental dynamics, which is intended to be represented in the model. Here, the mind is compared to a Turing process that has state and input. It provides an extensive description of the mental processes that are involved in meditation and how they relate to a Turing process. Basically this sets the context of the model, which is based on a multi-step computational algorithm, that is the functional code. Once the context is established, the model and its parameters will be discussed and how they relate to the context. Thereafter the implementation of the model in a functional code is described, which is supplemented with a discussion how the code is build up incrementally. Finally there is a discussion how the computational process can be manipulated by functions. This is illustrated with dynamic flowchart representations of computational processes that are specifically manipulated.

# Background

The brain can be considered as an information processing system, where a brain state represents information content and relations. This system can register new information defined as input that can influence the information state of the system, thereby producing a new state. This resembles a Turing machine process that can be described as a discrete dynamic process involving state, input and a set of transition rules that specify the transition steps that are associated with the input-state configurations (Barendregt, Raffone 2013). Likewise, any moment x in time the mind can be described by state and input. Basically the input-state interaction determines the next state.

*Attention as a multi-step computational process*

Attention operates as a gateway that determines the stream of information that is registered by the system as new input. According to Lamme (2003) attentional selection is a manifestation of adapted sensory processing that is a function of the interaction between long-term memory, innate wiring, experience and recent events. These are forms of memory that are inherent in the state of the neural network and influence the regulation of attention. According to Dehaene and Changeux (2011) attentional selection refers to: "the separation of relevant versus irrelevant information, isolation of an object or spatial location, based on its saliency or relevance to current goals, and amplification of its sensory attributes." Indicating that the attention process is partly influenced by internal milieu variables, like current needs, goals and contextual cues that selectively lead and facilitate corresponding sensory pathways via feedback loops. In other words, evidence accumulation by the neural network evolves towards a memory state that biases competition for attentional resources. Likewise, stimulus parameters such as emotional value can bias attentional processing. Thus, attention is regulated by both state and input. Therefore the regulation of attention represents the input-state interaction in the Turing machine that determines the next state, so that the transition step is reflected in the regulation of the attention process.

The mechanism of attention regulation can be considered as a serial multi-step computational process that directs discrete mental state transitions. Therefore, the attentional bias tendencies that are present in human individuals can function as predicates that determine subsequent mental state transitions. These bias tendencies are evolved as a function of both dispositional and experiential factors (karma). Attentional bias reflects the uneven distribution of attentional resources, thereby shaping perception in relation to the allocation of attentional resources. For example, emotional stimuli can be real or imagined, which can trigger distortions in initial attention allocation in momentary experience or subsequent information processing, thereby influencing the (un)conscious decision-making process.

In the presence of a stimulating environment, a selection is made to process those stimuli that are contextually relevant and prioritize these. Currently the predominating model for selection is "biased competition" (Desimone, Duncan 1995). This model states that allocation of attention is a function of bottom-up processes, top-down processes and stimuli characteristics. Meaning that representations of stimuli compete with each other in a bottom up fashion, while also being influenced by higher order systems.

There is a relation between emotional value and attentional processing, such that the former influences the latter. The other way around, attentional resource allocation relates to depth of processing, but not necessarily emotional attribution. Emotional attributes can strongly direct bottom-up processing, even to the degree of preconscious processing, which highly potentiates conscious access. Meaning that little top-down attentional amplification is needed for an object to be selected in a foreground access consciousness (Dehaene et al. 2006).

These automatic attentional processes depend on the interplay between memory, emotional charge and vigilance that determine the amount of bottom-up processing. Weak bottom-up activation is defined as subliminal processing and strong bottom-up activation is defined as preconscious processing (Dehaene et al. 2006). These processes are evolved according to values and criteria that serve the biological program (i.e. the DNA). Meaning that perception is biologically shaped , since information is processed selectively from both a physiological perspective (e.g. adapted sensory mechanics), as well as from a psychological perspective (e.g. innate emotional attribution) (Baars, Franklin & Ramsoy 2013).

Attentional processing is influenced by the perceptual window, such that attentional orientation determines the accessibility of stimulus dimensions. For example, spatial orientation involves the direction of attention to a place in the environment, which subsequently creates a new mental window where new stimuli can arise (or that exists already). The same holds for temporal orientation, only here the mental window can move into the imagined past or future. And finally there is the orientation on particular stimulus dimensions (e.g. global or local features of an object). The act of orientation implies that stimuli, which arise in the mental window, are amplified by either endogenous (i.e. internally directed) or exogenous (externally directed) means. Subsequently a response is triggered, depending on the strength of the activated stimulus-response mappings, or simply non-reactive attending.

*Attentional engagement*

Engagement implies the evidence of selection and facilitated processing of a given stimulus, while disengagement is defined as the process by which selection and facilitated processing of a given stimulus is withdrawn or inhibited (Yiend 2010). Control is closely related to disengagement as it refers to the ability to alter or stop a process once it is initiated. The difference is that control is always with respect to a proximal goal i.e. a goal that is momentarily relevant (e.g. shift attention from the object of distraction). These goals may subliminally guide mental behavior, which is a combination of both innate and acquired constraints. It is thought that some of these dispositions are lost in the transition from subclinical pathology to clinical disorder. For example if healthy constraints on extensive processing of emotional stimuli are lost, the propensity for rumination increases.

*Rumination*

Rumination involves compulsively focused attention on the causes and consequences of one's symptoms of distress without engagement in active coping or problem solving to alleviate dysphoric mood (Koster, De Lissnyder & De Raedt 2013). Neurobiologically rumination is associated with abnormal "circuit breaker activity." This is a system that breaks into the current attentional settings and captures attentional control whenever there is an event outside of the focus of attention that has sufficient potential biological relevance. Circuit breaker activity is predominantly right hemispheric, located at the

temporal-parietal junction and the inferior frontal gyrus. Moreover, depressed mood and pathological depression have been associated with excessive right hemisphere dominance, resulting in excessive circuit breaker activity and excessive sensitivity to anything that might possibly be novel, unexpected, or threatening (Kenemans, Bocker & Ramsey ).

The combination of excessive circuit breaker activity and excessive sensitivity to any stimuli that is emotion congruent (i.e. increasing the match with individuals' concerns increases attentional prioritization) interferes with the individual's ability to concentrate. Meaning that rumination can be described as successive trains of engagement, rather than attention shifting without strong facilitation of attentional processing (Yiend 2010). Shifting is defined as the relocation of attention across sensory fields, which essentially is a sub-step involved in successive bouts of engagement. Note, mental orientation can involve real or imagined threats, thereby perpetuating a hyper-vigilant state of excessive sensitivity by supplying imagined (i.e. emotion congruent amplification) stimuli as novel, unexpected or threatening.

Pathological mechanisms provide insight into the nature of attentional and emotional biases that are a more subtle and difficult to detect in subclinical individuals, and even more so in meditation practitioners. The underlying mechanisms in pathological conditions sometimes totally deviate from mechanisms in healthy conditions. Nevertheless, partial overlap of a general mechanism can also be identified, in which both the pathologically and healthy categorized representations exist within a linear spectrum (Vago, Silbersweig 2012). The meditator has to cope with transient mood states, while some personality-predisposed individuals have subclinical propensities (traits). On the extreme end (e.g. clinical disorder) the processes can be considered more intense and fluctuating compared to the other side, which is more equipoised. However it is not excluded that additional emergent cognitive processes arise in pathological conditions, other than a linear sum up of natural propensities.

According to empirical data attentional bias is not caused in the same way across different all disorders. On the contrary there is evidence of a double dissociation between anxiety and depression, where anxiety is more characterized by attentional bias (exogenous) than biased recall (endogenous), while the reverse is associated with depression (Yiend 2010). This implies that disorders can have different cognitive-processing biases, instead of every disorder acting as a "scalar", that affects the whole range of cognitive processing mechanisms linearly. This means that different spectra could be associated, that can be extreme in one disorder, but don't generalize to other disorders.

Anxiety and depression can be explained by different cognitive processing mechanisms. Anxiety could be marked by stronger priming, which is defined as an early automatic activation of an internal representation of a stimulus, thereby temporally enhancing its accessibility. There is some emphasis on 'automatic', indicating the involuntary and unconscious nature of priming. This can partly explain why negative priming is a strong force in keeping an individual locked in a certain mental frame that is strongly emotion congruent. In contrast, depression is marked by elaboration that is defined as a later strategic process (focused reflection on existing representations), which creates and strengthens interconnections between representations. This may involve deletion, distortion and generalization in order to preserve a coherent structure and reducing cognitive dissonance with the implication of causing recall biases due to infatuated retrieval. An anxious state can be considered as a high entropic

state of restlessness, while depression can be considered as a state of fixation that is strongly absorbed by the negative.

*Attention and emotion*

Attention has a limited capacity, which necessitates prioritization in the deployment of attentional resources, thereby imposing an uneven distribution on the allocation attentional resources. Capacity refers to the amount of resources that can be used for attentional processing. These resources can be flexibly allocated according to stimuli priorities. The process of rumination is demanding a high load of attentional resources, which is reflected by decrements in performance and interference with the ability to employ top-down control. Moreover, emotional material places heavier demands on attentional resources compared to neutral material, which can be attributed to an enhanced perceptual distinctiveness that is acquired or an innate biological preparedness. This causes relative salience of emotional stimuli leading to bottom up attentional prioritization.

Attentional prioritization is also influenced by factors like environmental context, past experience, prior knowledge, goal setting and intention. This functions as a mental frame (e.g. rumination involves the activation of an emotion congruent denominator) that influences the attentional engagement process in relation to certain emotional stimuli that are relevant according to this frame. Mental framing is closely associated with mental orientation, however the latter isn't necessarily involved with emotional processing.

Effective states of mind that can flexibly interact and adapt to the environment involve a present-centered attention with little self-focus, while pathological states of mind, that are associated with clinical predispositions and personality traits, are more involved with the deployment of attentional resources to emotion congruent stimuli and recurrent self-focus (e.g. priming in anxiety and elaboration in depression). Emotion congruence refers to the connection between the individual's concerns and related objects with strong emotional attribution, which biases attention processes. Emotion congruence is determined by conditioned mechanisms that have been evolved (i.e. innate) or acquired (e.g. classical conditioning). The principle of classical conditioning implies that neutral stimuli can be attributed with emotional value, after these stimuli have been paired through experience with aversive or attractive stimuli.

Emotion congruence is independent of valence attribution, as long as it associates with the individual's concerns. One study by (Mathews, MacLeod 2002) demonstrated that negative and positive stimuli that are related to the individuals negative concerns elicit similar attentional biases. For example stimuli that represent health and stimuli that represent disease can trigger the individual's concerns to the same degree, because they share the same association web, which is linked to negative dysfunctional schemata (i.e. the set of related beliefs and attitudes about the self, the environment, past and future). Schemata can be considered as complex chains of mental actions that involve the concatenation of multiple conditioned stimulus-response relations. Valence-independent association functions as a mechanism that connects current stimuli (i.e. both neutral and valenced) with mental stimuli (e.g. beliefs and concepts) that have a high emotional value. Depending on the emotional energy of the activated concepts, an emotional state can be triggered that is either positive or negative that can escalate in mental cascades.

*The self-concept*

Let's say at moment t in time there exists in an individual a mental frame with a negative emotional value. In this state there's the propensity to induce stimuli (i.e. thoughts) from an association web that is primed by the overarching negative theme. Still, the possibility of positive association exists, but against the presence of a strong negative competitive dynamics that draws attention toward the individual's concerns. Positive association is considered as not directly related to the individual's concerns, therefore it does not include the set of all positively valenced objects, as these can possibly be shared in the web of concerns too.

Positive association can be severely restricted by strong coherent frames that bind a certain negative theme. This can be very rigidly structured schemata, that are strongly entangled with a narrative self-concept, and can possibly associate to many different stimuli (Vago, Silbersweig 2012). Awareness of self can function in relation to the activation of schemata that are associated to the objects of attention. For example, the repetitive observation of the self in negative mood reinforces coherent dysfunctional schemata (i.e. the set of generalized negative beliefs and attitudes) and concurrently weave the 'concept of self' into schemata. Restriction of positive association would be the consequence of counteracting cognitive dissonance, in order to avoid incongruence with a stable belief system (even if it is dysfunctional).

The relation of self with conditioned schemata is not only involved in emotion congruent engagement processes, but also in withdrawing resources from the presence, and thereby preventing evidence accumulation and phenomenal processing depth (Vago, Silbersweig 2012). This property of withdrawal is mostly indirect, because resources are drawn towards negative elaboration and strong identification of the self with dysfunctional schemata. The rigidity of this identification determines the capacity to associate outside a dysfunctional web, however the interconnection of self with schemata imposes an authoritative value on the perception of self as absolute. Meaning that the perception of self is continuously compared against a moment to moment observation of reality and perpetuates itself by generalization, distortion and avoidance. The concept of self consists of beliefs, episodic memory and imaginations of the past and the future that are referenced to in the appraisal of present states of affairs.

*Meditation*

In focused attention meditation the attention is sustained on the intended object, which generally is the rise and fall of the breath. It involves the monitoring process of noticing distraction and non-judgmental redirection of the attention on the intended object. It is a concentrative practice with the aim of developing an internal witnessing observer that is non-reactive. In this way, mental objects can present itself in the background without being drawn in an explicit foreground consciousness, while the attention is gently sustained on the intended object. The mental objects that present itself in the background can be considered as phenomenal pre-conscious assemblies that are characterized by strong bottom-up activation without engagement. The maintenance of an ongoing open-field of endogenous attention, that is present-centered and non-judgemental, increases the capacity to represent or match fast phenomenal fluctuations, thereby enhancing phenomenal representation of

experience. In other words, this present-centered attention allows the experience and realization of reality by the cultivation of attentional acuity (Raffone, Srinivasan 2009).

The contrast between background and foreground awareness is reflected in the composition of state. A state consists of an explicit conscious representation together with an implicit memory system. New information is registered as input that interacts with the state to produce a new state. The brain is an information processing system that is based on memory and evolves towards discrete states of memory that can interact with new input. The implicit system represents the ongoing evidence accumulation in the form of bottom-up processes in a biased competition (Baars, Franklin & Ramsoy 2013). These processes are associated with unconscious stimuli that involve a transient memory (i.e. iconic memory) and constitute the ongoing phenomenal background. An explicit conscious representation can be described as a foreground access-consciousness, that involves a strong selective bias or focused attention, which includes facilitated processing of the respected object. This can be considered as a conscious working memory representation in the form of a distinct internal space that is buffered from fast fluctuations in phenomenal representation (Dehaene et al. 2006).

Thus in any moment t there exists a discrete mental state that consists of a background phenomenal awareness and an access-based conscious awareness. Earlier, it was stated that the transition step is reflected in the regulation of the attention process, that is dependent on the interaction of state parameters and stimulus parameters. The state parameters can be considered as internal milieu variables in the form of implicit memory, that is represented by the implicit system, which endogenously influence the attention process. Stimulus parameters function as the exogenous influence on the attention process. Thus the transition step lies in the establishment of a conscious working memory representation, which is determined by the input-state interaction of the attention regulation process.

Past experience is wired in long-term memory, just as innate properties, as the product of evolution and goal setting are basically a function of memory (Lamme 2003). In addition, conditioned patterns, that dictate stimulus-response mappings, are consolidated in memory, often in strong connection to emotion circuits. Therefore the mental frame, as the joint expression of all internal milieu parameters, is realized by the implicit memory system, which influences the attention regulation process. This also applies to the concept of self, that is stored in memory in relation to internal milieu parameters, including the representations of goals and their association webs. The same is true for dysfunctional schemata, that are wired in memory by a strong association web, which includes the individual's concerns and related objects with strong emotional attribution.

When the concept of self in its web of dysfunctional schemata is projected on the experience, reality becomes fixated in this frame. Here, the observation of self in the present moment becomes a conditioned interpretation based on a transient glimpse of ‘what phenomenally is’, instead of a sustained open-presence that allows a non-reactive internal witnessing observer. Sudden glimpses of phenomenal experience are immediately judged and compared against the concept of self, thereby reinforcing identification of an artificial concept against reality.

For example the phenomenal experience of a depressive state, that includes the mind and body, can involve strong emotional experience that is accompanied by mental avoidance (e.g. rumination). The attentional presence that accommodates momentary phenomenal experience is short-lived, so that

conscious awareness is almost impossible, and instead a reactive mind is triggered by conditioned engrams. These are the stored, emotionally charged concepts, that are associated with the self, and perpetuates itself when activated. This conditions an imagined frame that becomes highly associative, thereby resisting any form of self-dissonant association and reflexively directs attention to self-concerns.

Recurrent activation of conditioned memory traces reinforces the same stimulus-response mappings, thereby preventing the attention to center in the present. This counteracts the cultivation of attentional acuity and the experience of an enhanced phenomenal reality. Conscious awareness has an essential and functional role in learning by updating unconscious habituated processes in the reconsolidation of memory in order to develop a non-reactive introspective stance. The gradual experience of reality without conditioned projections stimulates the reconditioning process by adaptation to novelty (Baars, Franklin & Ramsoy 2013). This means that phenomenal experience is enhanced by the cultivation of an open-field of endogenous attention. On the other hand, the degree of free attentional resources, that are sustained on the present moment, decreases to the amount that the attentional resources are preoccupied with recurrent self-focus (Burgess, Dumontheil & Gilbert 2007). Self-generated thoughts, whether or not ruminative, that hijack attentional resources, can perpetuate a negative cycle of ever decreasing effective allocation of attentional resources to the object of intention.

The attention regulation practice becomes more efficient and automatic with practice, thereby promoting a minimally reactive attentional bias that allows unallocated resources to freely center in an open-field of distributed endogenous attention (Raffone, Srinivasan 2009). Automaticity in learning is established by creating a memory trace that represents the stimulus-response mappings that are acquired through repetitive training. Initially, learning involves effortful fine-tuning of the appropriate responses, which is followed by a natural flow through lower cognitive systems (Desmurget et al. 2009). As the meditator progresses the allocation of explicit cognitive resources decreases, because aspects of attentional processing are automatized and improve efficiency of engagement and disengagement processes.

As cognitive monitoring becomes automatic and effortless with experience, free attentional resources become available for a novelty-based adaptive consciousness as the phenomenal background comes to surface (Baars, Franklin & Ramsoy 2013). This allows for closer examination of subjective "I" states that resided in the "background" of a previously established state associated with its input, as subjective implicit content of the previous state together with its objective perceptual content becomes new input for the next bound as one and emerges as a "new" conscious core that contains the "Me" or explicit self. The cultivation of a non-reactive internal witnessing observer involves an ongoing meta-awareness in which objects can be observed without explicitly focusing on any of them (Raffone, Srinivasan 2009).

## The model

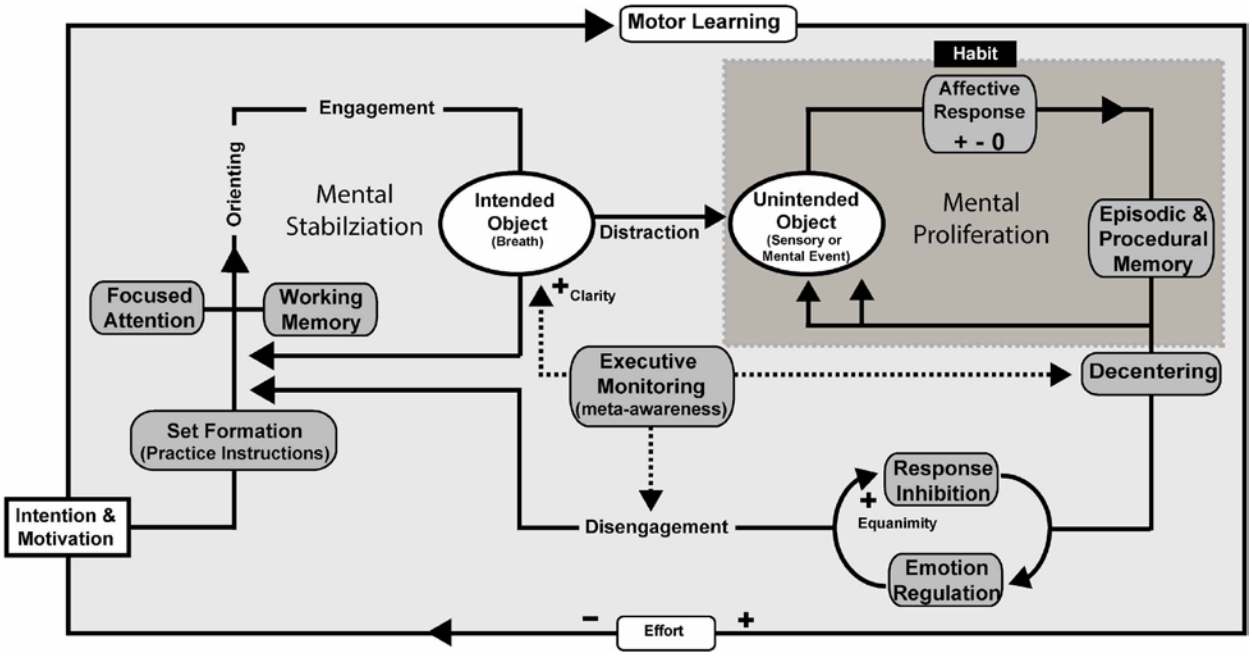



*Figure 1.* Flowchart representation of the attention regulation practice in focused attention meditation (Vago, Silbersweig 2012).

*Focused attention meditation*

The basic mind model is a minimal representation of the meditation process that is inspired by the mindfulness process model of concentrative practice (Vago, Silbersweig 2012). The goal of this practice is to focus the attention on the rise and fall of the breath. This requires the monitoring process of noticing distraction and the non-judgmental redirection of attention on the intended object. Figure 1 illustrates the attention regulation process. The focus of attention starts with the formation of an intention to establish an executive set that orients and focuses the attention on the intended object. The intention is supported by working memory, which facilitates sustained attention on the intended object. These are the processes that are involved in the circuit of mental stabilization.

However, the attention can be distracted by unintended objects, that includes thoughts and sensations, which are associated with an affective response that is either positive, negative or neutral. The affective response is dependent on stimulus parameters of the unintended object, which interacts with the state parameters that are represented as internal milieu variables in implicit memory. Both influence the attention process, which can lead to the establishment of a conscious working memory representation of a new unintended object. The processes that are involved in distraction are illustrated by the circuit of mental proliferation.

Mental proliferation refers to the recurrent activation of conditioned memory traces that dictate stimulus-response mappings. Often it involves the concatenation of multiple conditioned stimulus-response relations in schemata that can trigger mental cascades. Examples of such mental

cascades are mind-wandering and rumination. Rumination involves the activation of dysfunctional schemata, thereby limiting cognitive flexibility. In mind-wandering the attention is distracted by whatever thoughts present itself through association. It involves a freely associating mind that is not necessarily fixated on the self and the negative. However, mind-wandering prevents the development of an introspective stance due to attentional instability. This is even more so in rumination, because recurrent self-focus is more intensely concentrated , and thereby using more attentional resources. Thus, cycling in the circuit of mental proliferation increases reactive engagement, while decreasing the effective allocation of attentional resources on the present moment.

Monitoring distraction involves meta-awareness, which can only be realized if enough free resources are available to bind the "Me" as object. It can be described as distancing oneself from the subjective "I" experience (i.e. decentering). This is unlikely in the process of rumination, in which the mental capacity is overloaded with the preoccupation of recurrent self-focus that involves a strongly emotional congruent frame. However, if the process of mental proliferation becomes less emotional and less intense, the possibility for meta-awareness increases. Meta-awareness has an essential and functional role in the process of emotion regulation and response inhibition, by updating unconscious conditioned memory traces in the reconsolidation of memory in order to develop a non-reactive introspective stance (equanimity). This is illustrated in *figure 1* as the circuit of control and disengagement.

With training the attention regulation process becomes more efficient and automatic. Attentional acuity (clarity) increases as the monitoring process and a gently sustained attention on the object of intention require less attentional resources. This facilitates a novelty-based adaptive consciousness through enhanced phenomenal experience in an open-field of distributed attention that is sustained on the present (i.e. the meta-awareness or insight that is associated with a non-reactive internal witnessing observer). Now we will present a stylized version of the model.

*Discrete mind-state transitions*

*Figure* 1 is a schematic representation of the flowchart model that visualizes the mind-state transitions of the functional program. It illustrates the basic mind states that are involved in focused attention meditation, in contrast to the representation in *figure 1*, which illustrates the attention regulation process. The attention regulation process is implicit in the functional code, which is represented by parameter values that are updated in a discrete computational process. Before going into the details of the translation process, first there is a definition of the states and the possible transitions between the states, as represented in *figure 1*.

Making some abstraction, we get four states denoted by q0, q1, q2 and q3, which are categorically defined states that each are associated with a particular domain of the agent's state values. Directional edges that connect these nodes represent the accessibility relation between the nodes, such that the directional edges that originate from a particular node represent the possible routes by which other nodes can be reached.

The starting node q0 is defined as the default state. It corresponds with the circuit of mental proliferation in *figure 2*, that represents the attentional processes during distraction. There are three possible state transitions from q0. Continue cycling in q0, reflecting mental proliferation, or transitioning

to q1 or q2. If mental proliferation becomes emotionally intense, so that negative emotions such as fear, anxiety or anger become increasingly involved in projecting a mental frame, the mind can transitions to a state of rumination that is represented by q1.

In q1 rumination is defined as a form of mental proliferation that involves a strong emotion congruent frame that is highly reactive. The difference between q0 and q1 is the involvement of dysfunctional schemata in q1 that reinforce and lock intense negative states. It is characterized by a high mental load with little free attentional resources. Therefore, q1 can only continue to cycle in rumination, or it can transition to a state of emotionally less intense proliferation in q0. Transitioning to q1 can only via q0, because it is less intense and concentrated, allowing the possibility for meta-awareness.

State q2 is defined as initial application, which corresponds to the circuit of control and disengagement in *figure 1*. It involves the re-establishment of the executive set in order to re-engage the object of intention. This requires emotion regulation and response inhibition to the objects of distraction in the background, so that the attention can be directed to the object of intention in state q3.

State q3 is defined as sustained awareness, which corresponds to the circuit of mental stabilization in *figure 1*. Here the attention is gently stabilized on the rise and fall of the breath. This state is associated with the cultivation of attentional acuity and the absence of an explicit monitoring process.

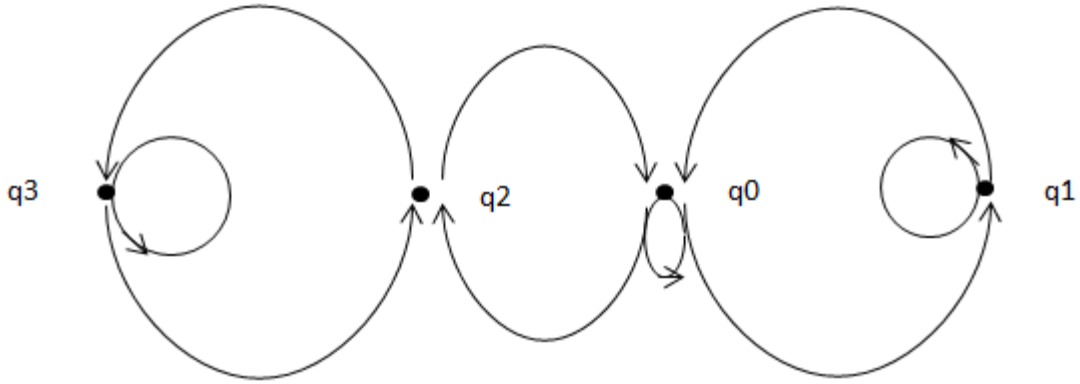



*Figure 2.* Schematic representation of the flow diagram. The *nodes* represent the possible states and the *arrows* represent directional edges that connect the nodes. The nodes correspond consecutively as q0: neutral, q1: rumination, q2: initial application, q3 sustained awareness.

*The attention process*

Let us define 'sea' as the set of all possible objects that could arise in the mind. It is the set of objects that consists of the breath, which is the intended object, and the unintended objects, which are either positive, negative or neutral. Intention is also included as a mental object, because the activation of a working memory representation is important in the establishment of an executive set. The set of objects that constitutes the sea is functionally defined as the following datatype:

:: Object = Breath | Plus | Minus | Neutral | Intention

At any moment in time there exists a set of objects from the sea that are available at a particular mental state, which is defined as protospace. Thus at any moment in time there exists a protospace, which is a subset of the sea. For example, we stipulate that the breath is not available as the object of attention in the state of rumination, because it requires the establishment of an executive set through intention. We do this by introducing a function that gives the set of objects in protospace given a particular state and the sea, which is represented as a function:

$F_{proto}$ :: State Sea -> Proto

At any moment in time, each of the objects that are available in protospace has a certain emotional intensity, which is represented by a number. A high number means strong emotional intensity, which increases the probability of drawing attention to the associated object. The distribution of probabilities influences the attention selection process. This distribution is relative (i.e. in relation to the values of the other objects), which represents the competition for attentional resources. High values for the breath as object does not necessarily mean it is emotionally involved, but rather that attentional resources are available to engage the breath. Likewise neutral stimuli don't have to be emotionally involved, but can include strong non-emotional association (e.g. logical priming). Thus the numbers should be individually interpreted for each object, as each state transition involves the regulation of attention by endogenous and exogenous means. The attention selection process as a function of protospace is represented as a function:

$F_{att}$ :: Proto -> Object

Subsequently the object that is selected as the focus of attention together with the momentary state determines which state transition occurs. Thus the transition step is reflected in the regulation of the attention process, that is dependent on the interaction of state parameters and stimulus parameters, which is represented as a function:

$F_{transition}$ :: State Input -> State

The state is a form of memory that consists of stored parameter values. The state parameters represent the internal milieu variables that are part of the implicit memory system. The implicit system is associated with ongoing evidence accumulation in phenomenal processing, however the model is a basic representation of massive parallel processing in the brain. The program constitutes basic parameters that abstract from a complex parallel process, but it can be expanded to include more parameters to represent a more complex attention regulation process.

A system of massive parallel processing evolves towards discrete states of memory that can interact with new input. These processes are associated with unconscious stimuli that involve a

transient memory (i.e. iconic memory) and constitute the ongoing phenomenal background. In this model the sea can be compared with the implicit system, because at any moment in time the state parameters of the sea participate in the attention selection process. The sea evolves with every transition step as the parametervalues that are stored in memory are updated. The function that updates the sea is represented as a function:

$F_{update}$ :: State Sea Input -> Sea

In the program state is actually defined as a type that represents the categorical state that is used in the computational translation to a visual flowchart model. One final note on the interpretation of state parameter values. When the attention is sustained on the breath in state q3, the sea is updated accordingly. Because the sea is a relative distribution, the cultivation of attentional stability and acuity is reflected in the value that is associated with the breath as object. This seems counterintuitive, because the free resources that are associated with effortless sustained attention are represented by a higher value of the breath as object. However a high value should not be interpreted as a strong selective bias or high intensity focus that explicitly selects the object in a strongly segregated foreground. Instead, a high relative value for the object as breath indicates the availability of unallocated resources that freely center in an open-field of distributed endogenous attention.

This is implicit in the value of the breath as object, because the rise and fall of the breath represent the only real phenomenal experience, that do not form a projection of reality. The maintenance of an ongoing open-field of endogenous attention increases the capacity to represent or match the phenomenality of the rise and fall of the breath, which reflects the cultivation of attentional acuity. In this state of sustained awareness the potential for meta-awareness increases, which is associated with the possibility of a deep introspective stance. However, because the program constitutes a basic representation of the focused attention meditation process, it only shows the possibility for meta-awareness, not if it actually happens or not. Meta-awareness is only made explicit in state q2 that is characterized by decentering. Meaning that meta-awareness is reflected in the random distribution that determines the attention selection process (see the implementation). Thus in the transition from q0 to q2 there is explicit evidence of meta-awareness, which can lead to decentering in q2.

## The implementation

Functional programming is based on describing the relation between the input and output of processes. The program defines the way in which a result can be computed given certain arguments. Therefore the basis of functional programming consists of defining functions that can be applied to concrete values, like single numerical values, but also compound structures, like lists and partially applied functions. In this way, processes with a large structured input and output could be modelled.

A functional program is based on a type system. Every singular or structured input and output is defined by type, just as the functions that describe their relations. This facilitates syntactical correctness by allowing the compiler to type check the code before running the code. For example, it is incorrect to take the square of a list, however the square function can be mapped to the contents of a list, provided it consists of singular integers. Thus a function can also be used as an argument of other functions, like the map function, or be given as output. The possibility to define functions over functions allows the construction of higher order functions and structured complexity.

The implementation of the theoretical model described above is written in a code that consists of a module which is built on other modules. Often, a module contains a coherent piece of code that is aimed at implementing a particular function or collection of functions that deal with a specific input-output process. In the same way, a complex function is built of other functions. For example an important function for random transformation that is often used in the code is genRandDist. The genRandDist module contains all the code that is needed for a random generator to function properly. The genRandDist module uses pre-existing modules like StdEnv (clean standard environment) and Mersennetwister (a pseudo-random generator) chooses at a node in the flowchart which is the next node with a certain probability distribution.

*The random generator module*

The module genRandDist contains the code that is needed for the implementation of the randomgenerator function, which is the main function the program is built on. This function makes it possible to incorporate probability distributions into the code in order to inject random variability, thereby transforming the code into a random process. The probability distributions aren't necessarily a uniform distribution, because this distribution is influenced by multiple parameters. Essentially the probability distribution determines the expression of a semi-random process, which can sometimes be more predictable than other times.

Likewise the attention process is a semi-random process that is reflected in the properties of the neural network. Neurons express complex integration of signals by gradual behaviour in the form of graded potentials. In this way single cell assemblies can operate with variable probability distributions, and even with potential for adaptive signalling. This transforms the elemental seriality of the action potential with random variability in the neural network. The attention process is always a biased process, which is dependent on the state of the network. Therefore the probability distribution of the attention process is a function of massive parallel projection, which is represented by the protospace in the code.

Below is an overview of the genRandDist module that contains the code. There is a build in commentary marked by "//" that is in the color blue, which explains the function definitions and type definitions that are contained in this module. Basically the randomgenerator function takes a randomseed (randomnumber) and a list of integers, [n1,...,nk], such that it generates an infinite list of integers from the codomain of f = x+1 with domain [0,(k-1)] with a probability distribution [n1/sum,...,nk/sum], where sum = n1 + ... + nk

```
implementation module genRandDist

import StdEnv
import Math.Random

Start = take 30 (genRandDist 1 [1,2,0,3])
// Takes the first 30 elements of the infinite
// pseudo-random sequence (based on the one from
// the Mersenne Twister) with distribution
// [1,2,0,3]. That is, after normalization (dividing by 6
// which is the sum 1+2+0+3) obtaining chances
// [1/6,1/3,0,1/2] for the numbers {1,2,3,4} to occur.
// The first argument of genRandDist
// is a seed to make variations.

trans :: Int Int -> Int
trans k p
  | k==0 = p
  | otherwise = k
// Changes the set of numbers {0, 1, 2, ..., p-1}
// into {1, 2, ..., p}, keeping equality mod p.

mmod :: Int Int -> Int
mmod p n
  | n<0       = p - ((abs n) rem p)
  | otherwise = trans (n rem p) p
// Note that
// -12 rem 10 = 2.
// This is not what we want.
// Also the C compiler does rem this way.
// The algorithm behaves as follows:
// map (mmod 7) [16,14,-20,0] = [2,7,1,7]

partialSum :: Int [Int] -> Int
partialSum 0 list   = 0
partialSum 1 [ ]    = 0
partialSum 1 [x:xs] = x
partialSum n [ ]    = 0
partialSum n [x:xs] = x + (partialSum (n - 1) xs)
// partialSum 2 [n1,n2,n3,n4,..] = n1 + n2
// partialSum 3 [n1,n2] = n1 + n2

least :: (Int -> Bool) -> Int
least pred = leastFrom 0 pred
where
    leastFrom k pred
      | (pred k)  = k
      | otherwise = leastFrom (k+1) pred
// Standard way to program "the least" in lambda calculus.
// least (\n-> (5<n)) = 6.

node :: [Int] Int -> Int
node list n = least (\k->n<=(partialSum k list))
// node list transforms a distribution list
// like [2,0,1,2] into a map that
// transforms a number from {1,...,5} (5 is the sum of the given list)
// into one from {1,...,4} (4 is the number of items in the given list)
// that will be randomly chosen with the right frequency:
// node list n = least k such that n <= partialSum k list
// This is the (easy) mathematical core of the algorithm.

// We use the following pseudo random generator from the Mersenne Twister
// genRandInt :: Int -> [Int]
// Generates an infinite list of in [-(2^63), (2^63)-1] uniformly distributed
// signed integer pseudorandom numbers. There period is (2^19937)-1.
// Input any nonzero integer as seed value.

randomMod :: Int Int -> [Int]
randomMod s p = (map (mmod p) (cutoffMod p (genRandInt s)))
// transforms random list with seed s of Int
// by taking values modulo p (my way, see above) obtaining list over {1,...,p}.
// The 'cutoffMod p' (see below) takes care of incomplete rows after folding
// integers (mod p) at the MIN and MAX values of used integers. It could
// be omitted.
```

```
randomReady :: Int [Int] -> [Int]
randomReady s list = randomMod s (sum list)
// The input list, e.g. [2,0,1,2]
// indicates the distribution of the chosen items
// {1,2,3,4}, so that 2 is never chosen and 1 and 4 are
// chosen twice as often as 3.
// This function transforms the random list of integers
// to a random list from {1,2,3,4,5}, 5 being 2+0+1+2.

genRandDist :: Int [Int] -> [Int]
genRandDist k list = map (node list) (randomReady k list)
// "randomGenDist k [2,0,1,2]" generates an infinite list
// of elements from {1,2,3,4} with chance to be 1 is 2/5,
// chance to be 2 is 0, chance to be 3 is 1/5, and chance to be
// 4 is 2/5.

// The following is a refinement to take into account the
// edges at the near minN, maxN.

maxN :: Int
maxN = IF_INT_64_OR_32 ((2^63) - 1) ((2^31)-1)
//
minN :: Int
minN = IF_INT_64_OR_32 (-2^63) (-2^31)

pFoldCut :: Int Int -> Int
pFoldCut p a = a-(a rem p)

mstart :: Int -> Int
mstart p = pFoldCut p minN
//
mend :: Int -> Int
mend p = pFoldCut p maxN

cutoffMod :: Int [Int] -> [Int]
cutoffMod p list = filter (cutoffModPred p) list
where
  cutoffModPred p = \n-> (((mstart p)<=n) && (n<(mend p)))

// If list consists of elements from {a,a+1,a+2,...b}
// say from {-103,-102, ... ,76,77} and p=10, then
// cutoffMod 10 list
// is that list in which only
// those elements n with -100 <= n < 70 are kept, to make it
// a rectangualr block. Eg
// cutoffMod 10 [2,-101,-15,76,4,70,-100,69] -103 77 =
// [2,-15,4,-100,69].
// That is: the 'edges' [-103,-102,-101] and
// [70,71,72,73,74,75,76,77] are cut off to obtain a list
// from the 10-block
// ----------------
// -100,-99,...,-91,
// - 90,-89,...,-81,
// - 80,       ,-71,
// ....
//  0,   1, ... , 9,
// 10, ...      ,19,
// 20, ...      ,29,
// ....
// 60,61,       ,69
// ----------------
// leaving out the edges
//   -103,-102,-101
// and at the other end
// 70,71,72,73,74,75,76,77
// having width less than 10.
// The edges make the random distribution a bit skewed.
```

*The function implementation module*

The implementation of the functions that are described in the section on the attention process are contained in the function implementation module. Because the programming language is functional, mathematical functions can easily be translated into a functional code. First, there is a type definition to represent the relevant input variables in the attention process. This allows the dissociation of state parameters from input parameters, which can be refined further into structured definition. For example, the Sea is defined as a list of weighted objects, where a weighted object represents an object together with its associated intensity in the form of a tuple.

The categorical state that is visualized in the flow diagram is represented by an integer in the program, which is defined as a node through a type synonym. The InnerState is defined as a compound structure that represents the explicit foreground (object of attention) and the implicit context (sea). The program is based on function application, thus the computational process begins with a start value as input. The start value for the function implementation consists of state and input, which is defined as sea0 and (Intention,1) in the defaultInnerState (A standard integer is defined to as control value).

The defaultInnerState is used as the first argument of the function nk, which represents the input-state interaction. The function nk represents a complex function composition that is recursively defined, which computes the node and InnerState at moment t. In order to accomplish this, it calls on the transition function and upd function. The transition function computes the transition step as a function of the object of attention and the Node. It integrates the outcome of state variables and input variables of the attention regulation process in a single transition step.

The upd function is also a complex function composition that calls on other function to update the InnerState. It takes the current InnerState, but also the Seed, the Moment and the Node that are forwarded to curried functions. The Seed is a random number, which is based on real-time, which is forwarded from the Java code. The InnerState consists of the sea and the object of attention x that are updated to sea2 and x2. The new object of attention x2 is determined by the probability distribution of the Objects in Protospace, which is an argument of the random generator function. The function OdN (Object-Distribution-Node) computes the Protospace based on the Node and object of attention. For example, in node 2 (rumination) the Breath as Object is not available, which is reflected as zero in the probability distribution.

The implicit memory is updated to sea2 based on the new object of attention x2 and the Node by the function updatew (update weight). This function takes the implicit memory sea as argument in order to update the stored values. It is defined as a case expression extended with pattern matching through guards. The distribution wobjects gives the default distribution. Behind the '//' possible function compositions are given that can manipulate the probability distribution according to certain predicates. When the Node and InnerState are updated they become the new input for the input-output computational process, which is infinite if no limit is specified.

The start expression that is defined with 'Start world = startEngine t world' that couples the functional process to a java code that is discussed in the next section. Below the start expression there are a few examples of functional expressions that give specific input and/or state parameter values. Below is an overview of the function implementation, including auxiliary functions.

```
module Function Implementation

import StdArray
from StdMisc import undef
import iTasks
import iTasks.API.Core.Client.Editlet
import iTasks.API.Core.Client.Interface
import genRandDist

Start world = startEngine t world
           // map (\x -> length (filter ((==)x) (map (\(a,x) -> fst (a,x)) (map (nk 1) [0..200])))) [1..4]
           // gives relative distribution of node positions
           // map (\(a,x) -> fst (a,x)) (map (nk 1) [0..1000])
           // numerical tracking of nodes
           // [(y) \\ (w,x,y,z) <- (iterate nodeI (1,1,1,(10,sea0,(Plus,1)))) | ((>)2000 x)]
           // List comprehension, tests complete pattern match
           // map (\(a,b,c) -> snd (select 2 (tmiddle (a,b,c)))) (map (\(a,x) -> snd (a,x)) (map (nk 1) [0..1000]))
           // numerical tracking of Plus weight

// Representation of directed graphs: Nodes & Edges
:: Node :== Int                                    // code for a location
:: Location :== (Int,Int)                          // locations on canvas
l :: Node -> Location                              // assigns location to node
l 1 = (300,600)  //mental proliferation
l 2 = (740,600)  //rumination
l 3 = (740,400)  //initial application
l 4 = (300,400)  //sustained awareness

:: InnerState :== (Int,[WO],WO)
:: Seed :== Int
:: Edge :== (Node,Node)
:: Moment :== Int
:: Dist :== [Int]

sea0 = [(Breath,1),(Plus,1),(Minus,1),(Neutral,1),(Intention,1)]
:: Attention :== WO //the object of attention
:: Sea     :== [WO] //The set of all possible objects that could arise in the mind

:: Object = Breath | Plus | Minus | Neutral | Intention // data constructors for objects
:: WO       :== (Object,W) // weighted object
:: W        :== Int        // weight

defaultInnerState :: InnerState // initial innerstate consisting of (Int,Sea,Attention)
defaultInnerState = (1,sea0,(Intention,1))

nk :: Seed Moment -> (Node,InnerState) //Gives a tuple of categorical state and innerstate
nk s 0 = (1,defaultInnerState)
nk s t = (n2, (upd s t n2 k1))
   where
     (n1, k1=:(i,sea`,x)) = nk s (t-1)
     n2 = transition n1 x

transition :: Node WO -> Node // next node based on current node and object of attention
transition node (object,w) = case node of
                             1
                                 | object == Plus        = 2
                                 | object == Minus       = 2
                                 | object == Neutral     = 1
                                 | object == Intention   = 3
                             2
                                 | object == Plus        = 2
                                 | object == Minus       = 2
                                 | object == Neutral     = 1
                             3
                                 | object == Plus        = 1
                                 | object == Minus       = 1
                                 | object == Breath      = 4
                             4
                                 | object == Breath      = 4
                                 | object == Plus        = 1
                                 | object == Minus       = 1
                                 | object == Neutral     = 1
```

```
upd :: Seed Moment Node InnerState -> InnerState
upd s t n (i,sea,x)
  |n == 2       = ((i+1),sea2,x2)
  |otherwise    = (i,sea2,x2)
  where
  sea2 = updatew sea x2 n   // updates all the weights of objects in the sea
  x2 = selector sea (select t (genRandDist s (OdN n sea)))  // attention selection in protospace


OdN :: Node [WO] -> [W] // gives distribution of protospace.
OdN node [(Breath,w1),(Plus,w2),(Minus,w3),(Neutral,w4),(Intention,w5)]
    = case node of
    1 = [0,w2,w3,w4,w5]
    2 = [0,w2,w3,w4,0]
    3 = [w1,w2,w3,0,0]
    4 = [w1,w2,w3,w4,0]

                  // updates the sea, including protospace
instance updatew WO // wobjects gives same distribution. Part after '//' gives alternative effects
where updatew [(Breath,w1),(Plus,w2),(Minus,w3),(Neutral,w4),(Intention,w5)] (object,w) x
                  = case object of
                  Breath
                      | x == 3 = wobjects//double (creset wobjects (Breath,w1) 100) (Breath,w1)
                      | x == 4 = wobjects//half (half (half wobjects (Minus,w3)) (Plus,w2))(Neutral,w4)
                  Plus
                      | x == 1 = wobjects//reset (double wobjects (Plus,w2)) (Intention,w5)
                      | x == 2 = wobjects//reset wobjects (Minus,w3)
                      | x == 3 = wobjects//reset wobjects (Plus,w2)
                      | x == 4 = wobjects//reset (double wobjects (Plus,w2)) (Intention,w5)
                  Minus
                      | x == 1 = wobjects//half wobjects (Intention,w5)
                      | x == 2 = wobjects//cdouble wobjects (Minus,w3) 40
                      | x == 3 = wobjects//double wobjects (Intention,w5)
                      | x == 4 = wobjects//cdouble wobjects (Minus,w1) 10
                  Neutral
                      | x == 1 = wobjects//double wobjects (Intention,w5)
                      | x == 2 = wobjects//half wobjects (Minus,w3)
                      | x == 4 = wobjects//half wobjects (Minus,w3)
                  Intention
                      | x == 1 = wobjects//double wobjects (Intention,w5)
               where wobjects = [(Breath,w1),(Plus,w2),(Minus,w3),(Neutral,w4),(Intention,w5)]

//---- auxiliary functions ----
select :: Int [a] -> a // selects n-th element of (infinite) list
select n [x:xs] // must exist already
  |  n==0      = x
  |  otherwise = select (n-1) xs

min :: Int Int -> Int //selects minimum
min n m
    | n<m = n
    | otherwise = m

instance == Object //Object type bool definition instance
where
    (==) :: !Object !Object -> Bool
    (==) Breath Breath       = True
    (==) Minus Minus         = True
    (==) Plus Plus           = True
    (==) Neutral Neutral     = True
    (==) Intention Intention = True
    (==) _ _                 = False


selector :: [WO] Int -> WO //translates Int list into WO
selector [(Breath,w1),(Plus,w2),(Minus,w3),(Neutral,w4),(Intention,w5)] number
    = case number of
    1 = (Breath,w1)
    2 = (Plus,w2)
    3 = (Minus,w3)
    4 = (Neutral,w4)
    5 = (Intention,w5)
```

```
class updatew a
where
    updatew :: [a] a Node-> [a]


output2 :: (Moment,Node,Attention,Sea) -> (Moment,Node,Attention,Sea) // function for list generation
output2 (t,place1,wo1,sea1) = (t+1,place2,wo2,sea2)
    where
    sea2 = updatew sea1 wo1 place1
    place2 = transition place1 wo1
    wo2    = selector sea2 (select t (genRandDist t (OdN place2 sea2)))


nodeI :: (Moment,Seed,Node,InnerState) -> (Moment,Seed,Node,InnerState) //embedded function for list generation
nodeI (t,s,node1,k1=:(i,sea,x)) = (t+1,s+1,node2,k2)
    where
    node2 = transition node1 x
    k2    = upd t s node2 k1


reset :: [WO] WO -> [WO] //resets value of specified object to 1
reset [] (b,z)              = []
reset [(a,x):ts] (b,z)
            |(a,x) == (b,z) = [(a,1): reset ts (b,z)]
            |otherwise      = [(a,x): reset ts (b,z)]

half :: [WO] WO -> [WO] //halfs value of specified object
half [] (b,z)               = []
half [(a,x):ts] (b,z)
            |(a,x) == (b,z) = [(a,x/2): half ts (b,z)]
            |otherwise      = [(a,x): half ts (b,z)]

double :: [WO] WO -> [WO] //doubles value of specified object
double [] (b,z)                 = []
double [(a,x):ts] (b,z)
            |(a,x) == (b,z) = [(a,x*2): double ts (b,z)]
            |otherwise      = [(a,x): double ts (b,z)]

cdouble :: [WO] WO Int -> [WO] // conditional double
cdouble [] (b,z) i              = []
cdouble [(a,x):ts] (b,z) i
         |i >= z
            |(a,x) == (b,z) = [(a,x*2): double ts (b,z)]
            |otherwise      = [(a,x): double ts (b,z)]
         |otherwise = [(a,x):ts]

creset :: [WO] WO Int -> [WO] // conditional reset
creset [] (b,z) i           = []
creset [(a,x):ts] (b,z) i
        |i <= z
            |(a,x) == (b,z) = [(a,1): creset ts (b,z) i]
            |otherwise      = [(a,x): creset ts (b,z) i]
        |otherwise  = [(a,x):ts]

chalf :: [WO] WO Int -> [WO] // conditional half
chalf [] (b,z) i                = []
chalf [(a,x):ts] (b,z) i
        |i <= z
            |(a,x) == (b,z) = [(a,x/2): chalf ts (b,z) i]
            |otherwise      = [(a,x): chalf ts (b,z) i]
        |otherwise = [(a,x):ts]
```

*Visual translation in itasks-environment*

In order to visualize the input-output process, the clean code is coupled to an itasks-environment, which is a task-oriented programming framework that is used to design workflow systems for the web. This itasks environment is used to construct a dynamic flow chart, so that the computational process can be visualized in a web-based application. The code that is presented below translates the node positions and node transitions to a graphic representation of an agent that transitions between nodes by moving via their connecting edges. Because the web-based application is dynamically constructed, the process can be tracked in real-time. The random seed that is used by the random generator to generate random numbers is based on real-time. The code is flexible with an adaptive framework that can be easily expanded. Additional nodes and edges can be flexibly integrated in the code and also the shape of the edges can be adjusted. Colour, sound and speed are also integrated and could represent additional parameters. Currently they are congruent with state and don't represent additional independent values. Below is an overview of the Java-code.

```
//--- Java-part ----------------------------------------


//agent colour codes
blue = "#0000ff"
yellow = "#ffff00"
red = "#f00f0f"
green = "#00ff00"
turquoise = "#66ffff"

t = viewInformation "Pacman" [] pacman

pacman :: Editlet Void Void
pacman = Editlet Void
                { EditletServerDef
                | genUI   = \cid world -> (uiDef cid, world)
                , defVal  = Void
                , genDiff = genDiffServer
                , appDiff = appDiffServer
                }
                { EditletClientDef
                | updateUI = onUpdate
                , defVal   = Void
                , genDiff  = genDiffClient
                , appDiff  = appDiffClient
                }
where
// My DAG specific. The numbers '1900' etc represent milliseconds

  Tr (1,1) = rondje2 800 1 yellow        // Tr stands for 'Trajectory'.
  Tr (1,2) = StraightC 400 1 2 red
  Tr (1,3) = StraightC 1000 1 3 blue
  Tr (2,1) = StraightC 600 2 1 yellow
  Tr (2,2) = rondje2 200 2 red
  Tr (3,1) = StraightC 800 3 1 yellow
  Tr (3,4) = StraightC 1200 3 4 green
  Tr (4,1) = StraightC 800 4 1 yellow
  Tr (4,4) = rondje2 2000 4 green
  Tr _     = bounce 1000 1 blue   // Default pattern, when there is no pattern match.

  uiDef cid
    = { html          = CanvasTag [IdAttr (canvasId cid), WidthAttr "1024px", HeightAttr "768px"]  []
      , eventHandlers = []
      , width         = ExactSize 1024
      , height        = ExactSize 768
      }

  canvasId cid = "canvas_" +++ cid

  onUpdate cid Nothing clval world
    # (obj, world) = findObject "createjs" world
    | not (jsIsUndefined obj) = onLoad cid undef clval world
    # world = addJSFromUrl "easeljs-0.7.1.min.js" Nothing world
    # world = addJSFromUrl "soundjs-0.5.2.min.js" Nothing world
    # world = addJSFromUrl "tweenjs-0.5.1.combined.js" (Just (createEditletEventHandler onLoad cid)) world
    = (clval,world)

  onUpdate cid _ clval world
    = (clval, world)
  onLoad cid v clval world
    # (ap, world)     = findObject "createjs.WebAudioPlugin" world
    # (_, world)      = ("createjs.Sound.registerPlugins" .$ toJSArg [ap]) world
//    # (_, world)      = ("createjs.Sound.registerSound" .$ ("beep.wav", "beep")) world
    # (tweenC, world) = findObject "createjs.Tween" world
    # (_, world)      = ("createjs.MotionGuidePlugin.install" .$ tweenC) world
    # (_, world)      = ("createjs.CSSPlugin.install" .$ tweenC) world
    # (canvas, world) = getDomElement (canvasId cid) world
    # (stage, world)  = (new "createjs.Stage" canvas) world
    # (_, world)      = ("createjs.Ticker.addEventListener" .$ ("tick", stage)) world
    # world           = (stage .# "autoClear" .= True) world
    # (actor, world)  = mkShape world // (new "createjs.Shape" ()) world
    # world           = drawActor actor "#ff0000" world
    # (_, world)      = (stage .# "addChild" .$ actor) world
    # (now, world)    = ("Date.now" .$ ()) world
    # standVdMaan     = (jsValToInt now) //rem 20
```

```
//    # (clval, world)  = nextMove ([(1,2),(2,1),(1,3)]) actor cid v clval world
//    # (clval, world)  = nextMove (ES standVdMaan) actor cid v clval world
    # (clval, world)  = nextMove standVdMaan [(1,0,defaultInnerState)]  actor cid v clval world
    = (clval, world)

//  nextMove :: Seed [Edge] Shape String v cl *JSWorld -> *(cl, *JSWorld)
// nextMove [x:xs] actor cid v clval world
//    # world = Tr x actor xs cid v clval world
//    = (clval, world)
//  nextMove _ _ _ _ clval world = (clval, world)


  //nextMove :: Seed [(Int,Int,Int)] Shape String v cl *JSWorld -> *(cl, *JSWorld)
  nextMove seed [(n,t,is=:(i,sea`,x)):xs] actor cid v clval world
  # node = transition n x
  # world = Tr (n,node) actor [(node,t+1,(upd seed t node is))] cid v clval world
//            where
//               m = (select t (genRandDist seed (EdN n 0)))
  = (clval, world)
  nextMove _ _ _ _ _ clval world = (clval, world)

  clearActor :: Shape *JSWorld -> *JSWorld
  clearActor actor world
    # (ag, world) = getGraphics actor world
    = clearGraphics ag world

  drawActor :: Shape String *JSWorld -> *JSWorld
  drawActor actor fillColour world
    # (ag, world) = getGraphics actor world
    # (_, world)  = (ag .# "setStrokeStyle" .$ (1, "round", "round")) world
    # (_, world)  = (ag .# "beginStroke" .$ "#000000") world
    # (bf, world) = (ag .# "beginFill" .$ fillColour) world
    # (_, world)  = (bf .# "drawCircle" .$ (0, 0, 10)) world
    # (_, world)  = (ag .# "endStroke" .$ ()) world
    # (_, world)  = (ag .# "endFill" .$ ()) world
    = world

  changeActorColour :: Shape String *JSWorld -> *JSWorld
  changeActorColour actor fillColour world
    # world = clearActor actor world
    = drawActor actor fillColour world

  getActorTween :: Shape *JSWorld -> *(Tween, *JSWorld)
  getActorTween shape world = ("createjs.Tween.get" .$ (shape, defaultTweenFlags, True)) world

  mkShape :: *JSWorld -> *(Shape, *JSWorld)
  mkShape world = new "createjs.Shape" () world

  getGraphics :: Shape *JSWorld -> *(Graphics, *JSWorld)
  getGraphics sh world = .? (sh .# "graphics") world

  clearGraphics :: Graphics *JSWorld -> *JSWorld
  clearGraphics g world
    # (_, world) = (g .# "clear" .$ ()) world
    = world

  beep :: Int *JSWorld -> *JSWorld
  beep vol world
    # ( , world)  = ("createjs.Sound.play" .$ ("beep", {defaultAudioFlags & volume = vol})) world
    = world

  to :: Tween Guide Int *JSWorld -> *JSWorld
  to tween guide duration world
    # (_, world) = (tween .# "to" .$ (guide, duration)) world
    = world

  wait :: Tween Int *JSWorld -> *JSWorld
  wait tween duration world
    # (_, world) = (tween .# "wait" .$ duration) world
    = world

  call :: Tween (JSFun a) *JSWorld -> *JSWorld
  call tween cb world
    # (_, world) = (tween .# "call" .$ cb) world
    = world
```

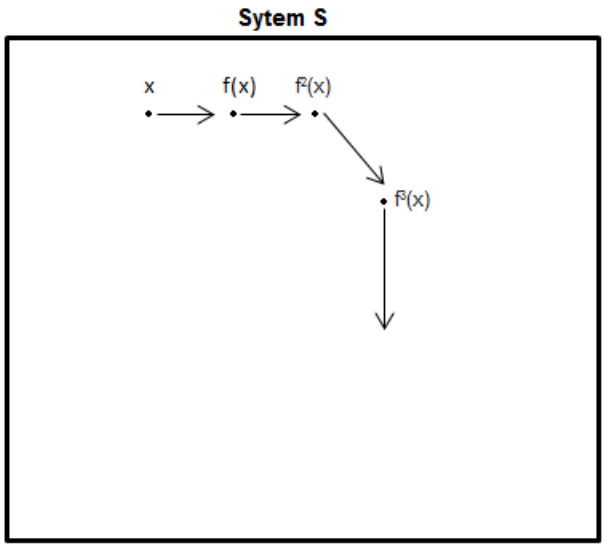
```
pad :: Location Location Location *JSWorld -> *(Guide, *JSWorld)
pad (p,q) (p1,q1) (p2,q2) world= mkGuide [p,q, p1,q1, p2,q2] world
pad5  (p1,q1) (p2,q2)  (p3,q3) (p4,q4)  (p5,q5) = mkGuide [p1,q1, p2,q2, p3,q3, p4,q4, p5,q5]

      // trajects java drawn path from n=(p,q) to n2 via n1.
straight(p1,q1) (p2,q2) = pad (p1,q1) ((p1+p2)/2,(q1+q2)/2) (p2,q2)
      // trajects straight line from A to B
boogjeO (p1,q1) (p2,q2) = pad (p1,q1) ((p1+p2)/2,(q1+q2)/2+90) (p2,q2)
      // with small down detour
boogjeN (p1,q1) (p2,q2) = pad (p1,q1) ((p1+p2)/2,(q1+q2)/2-90) (p2,q2)
      // with small up detour
stand n  = pad n n n // remains at node n

StraightC sec n1 n2  = Straight sec (l n1) (l n2)
 where
 Straight sec loc1 loc2 color actor xs cid v clval world
  # (at, world) = getActorTween actor world
  # (g0, world) = stand loc1 world
  # (g1, world) = straight loc1 loc2 world
  # (g2, world) = stand loc2 world
  # world       = changeActorColour actor color world
  # world       = to at g0 50 world
  # world       = to at g1 sec world
  # world       = to at g2 50 world
  # cb          = createEditletEventHandler (nextMove 1 xs actor) cid
  = call at cb world

bounce sec n color actor xs cid v clval world
  # (at, world) = getActorTween actor world
  # (g1, world) = pad (p,q) (p,q/2) (p,50) world
  # (g2, world) = mkGuide [p,50, p,(q/2), p,q] world
  # (g3, world) = stand (p,q) world
  # (g4, world) = stand (p,50) world
  # world       = changeActorColour actor color world
  # world       = to at g3 100 world
  # world       = to at g1 sec world
  # world       = to at g4 100 world
  # world       = changeActorColour actor turquoise world
  # world       = to at g4 100 world
  # world       = to at g2 (sec+1500) world
  # world       = wait at 90 world
  # cb          = createEditletEventHandler (nextMove 1 xs actor) cid
  = call at cb world
      where
           (p,q) = (l n)

rondje sec (p,q) color actor xs cid v clval world
  # (at, world) = getActorTween actor world
  # (g1, world) = boogjeO (p,q) (p+50,q-150) world
  # (g2, world) = boogjeN (p+50,q-150) (p,q-300) world
  # (g3, world) = boogjeN (p,q-300) (p-50,q-150) world
  # (g4, world) = boogjeO (p-50,q-150) (p,q) world
  # (g0, world) = stand (p,q) world
  # world       = changeActorColour actor color world
//   # world       = beep 1.0 world
  # world       = to at g1 (sec/4) world
  # world       = to at g2 (sec/4) world
  # world       = to at g3 (sec/4) world
  # world       = to at g4 (sec/4) world
  # cb          = createEditletEventHandler (nextMove 1 xs actor) cid
  = call at cb world

rondje2 sec n color actor xs cid v clval world
  # (at, world) = getActorTween actor world
  # (g1, world) = pad5 (p,q) (p+250,q-150) (p,q-300) (p-250,q-150) (p,q) world
  # (g4, world) = boogjeO (p-150,q-150) (p,q) world
  # (g0, world) = stand (p,q) world
  # world       = changeActorColour actor color world
  # world       = to at g1 (sec) world
  # cb          = createEditletEventHandler (nextMove 1 xs actor) cid
  = call at cb world
      where
           (p,q) = (l n)

genDiffClient clval1 clval2 = Nothing
```

```
  genDiffServer val1 val2 = Nothing

  appDiffClient diffs clval = clval

  appDiffServer diffs val = val

:: TweenFlags =
  { loop     :: Bool
  , override :: Bool
  }

:: AudioFlags =
  { interrupt :: String
  , volume    :: Real
  }

:: ShapePtr = ShapePtr
:: Shape :== JSObj ShapePtr

:: GraphicsPtr = GraphicsPtr
:: Graphics :== JSObj GraphicsPtr

:: TweenPtr = TweenPtr
:: Tween :== JSObj TweenPtr

:: GuidePtr = GuidePtr
:: Guide :== JSObj GuidePtr

defaultAudioFlags = { interrupt = "any", volume = 1.0 }
defaultTweenFlags = { loop = False, override = True }

mkGuide :: [Int] *JSWorld -> *(Guide, *JSWorld)
mkGuide gd world
  # (g, world) = jsEmptyObject world
  # (p, world) = jsEmptyObject world
  # world      = (p .# "path" .= toJSArg gd) world
  # world      = (g .# "guide" .= p) world
  = (g, world)
```

*Flowchart representation*

The flowchart visualizes the dynamics of the computation process. The nodes that represent the categorical states, which are defined in the function implementation correspond to specific locations on the canvas where the flowchart is displayed. The functional process that represents the attention process is implicit in the code, however the implicit state parameters and input parameters can be displayed by list comprehensions that can possibly be translated into a vector space. The benefit of the flowchart representation is the real-time visualization of a dynamic process, instead of analyzing a static vector-space. It provides a clear overview of the state transitions that are reflected in the functional computation, regardless of the complexity of the implicit calculation.

The computational process can be expanded with additional parameters and function integration to increase complexity and enhance combinatorial power. Function compositions can introduce complex mathematical relations with known bifurcation values or randomized with the purpose of analyzing eventual outcomes or strange attractors in the phase-space. Strange attractors may occur in a dynamical system that is based on multi-dimensional variables. Let's define a dynamical system as a set of values S and a function F = S -> S. Then the trace of x ∈ S is the sequence x -> f(x) -> $f^2$(x) -> $f^3$(x) ->…, which can be represented schematically as follows:




The functional model of focused attention meditation is an example of a dynamical system, in which the stochasticity of the system is implemented by the random generator module, such that the function F = S -> S is a stochastic operation. Specifically this a discrete dynamic model, contrasting it with a dynamical system based on continuity. This is inspired by the thesis on discrete states of conscious cognition, such that a discrete mind state is represented by a point in a dynamical system. The complexity of the innerstate has been expanded to a multi-variable computation that is defined by a different function that applies to the new dynamical system. The expansion of dynamical system $S_A$ with $S_B$ can be formally represented as: $S_A \times S_B = \{(a,b) \mid a \in A, b \in B\}$. The function composition of the new dynamical system can be represented as: $F = \{S \times I \mid S \in S_{A,} I \in \text{Innerstate}\}$.

A strange attractor is defined as a subset A of a system S, which has a neighborhood, such that the traces of every point in this neighborhood enter A. These are semi-stable patterns in a system that is based on an apparent random expression. The neighborhood or attractor space is the area in a phase-space model that can adopt a stable equilibrium. As the dynamical system moves towards a strange-attractor zone, it becomes drawn into a specific pattern. However the pattern does not have to follow specific spatial coordinates. So, a strange-attractor can be used to describe semi-stable patterns in a seemingly chaotic system.

*Figure 3* shows a snapshot of the dynamic flowchart representation of the basic model that is shown in *figure 2.* The agent dynamically transitions from node to node (represented by black dots) via the edges that are represented by colored lines. The flowchart representation is defined as a square for better overview in quadrants. The lower left node represents q0: mental proliferation, the lower right node represents q1: rumination, the upper right node represents q2: initial application, and the upper left node represents q3: sustained awareness. In this figure the color is congruent with the categorical state and also functions as a marker to reflect the direction in which the edge is traced. Currently the agent is marked in yellow, which indicates that the agent just got distracted and is transitioning to a state of mental proliferation. The computational process that is visualized is the default uniform random computation where the implicit state is not updated, so there that there is no convergence to any particular quadrant.

The computational process can be manipulated by defining the mathematical relations between the input and state parameters. The dynamic flowchart in *figure 4* displays a strange attractor in the field that represents sustained awareness, which is marked in green. The system dynamically converges to a state of sustained awareness by reinforcing values that propagate the process towards the strange-attractor zone. This is realized by introducing predicates in the form of functions that update the InnerState, reflecting biased attention (which is desirable in this case). The program is called 'equanimity' (see *figure 5)*, because the process eventually culminates in a state of sustained awareness.

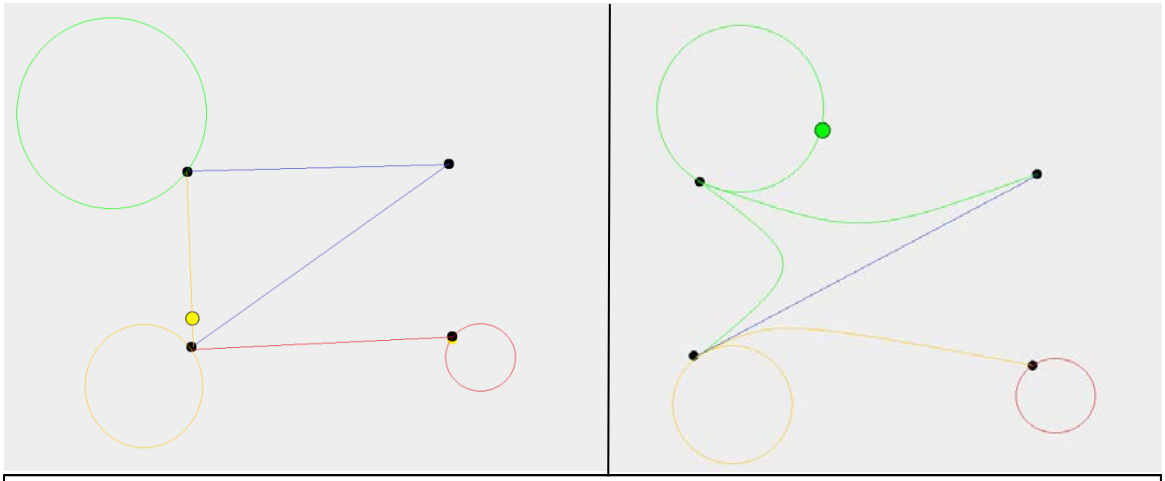


*Figure 3 (left).* Snapshot of basic model flowchart representation. The yellow ball represent the agent. Upper left and right black dots represent sustained awareness and initial application respectively. Lower left and right black dots represent mental proliferation and rumination respectively. Edges are represented by lines and circles. Color dynamically changes depending on the state of the process. *Figure 4 (right).* Equanimity. The process dynamically converges to a state of sustained awareness. The strange-attractor zone is marked in green.

*Figure 5* shows the program 're-contemplate'. The computational process displays a strange attractor that reinforces an equilibrium state until it reaches an upper limit from where it collapses. Because the memory that is contained in the parameters is reset, there cannot be a higher order pattern over and above the first semi-stable pattern that is rooted in sustained awareness. The lapse potentiates a transition to a state of rumination, which represents an undesirable state.

The program 'Trigger chaos' displays two strange attractors. One strange attractor can be triggered in q1, that results in adding differential randomness with a normal distribution around Minus objects. This increases the propensity to remain in the rumination state with a high equilibrium value. However, if the process exits the strange attractor, there is increased randomness due to Minus value reset. There is also a weak strange attractor in sustained awareness that is easily disturbed by high random values.

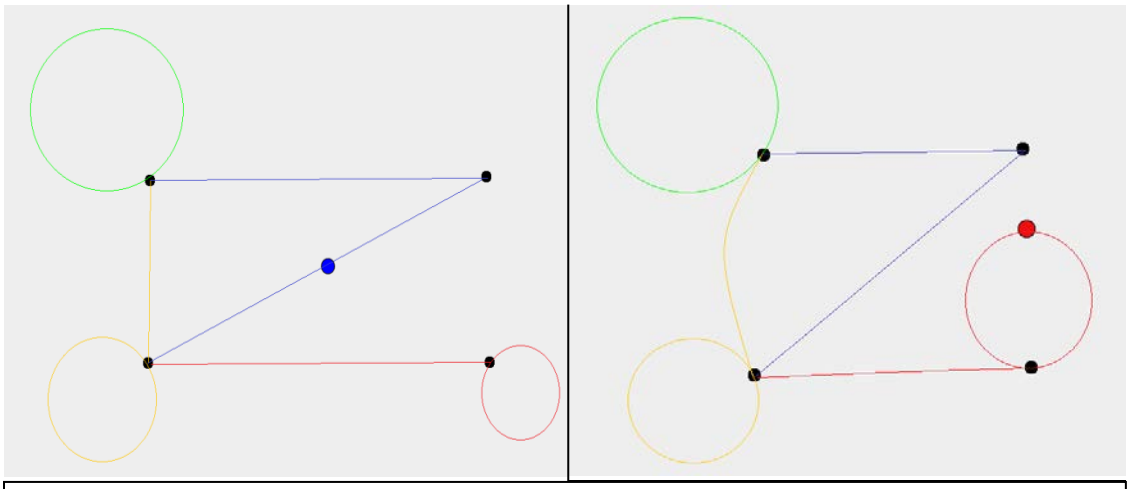


*Figure 5 (left).* Re-contemplate. Strange-attractor is marked in green. *Figure 6 (right).* Triggerchaos. Strong strange attractor is marked in red. Weak strange attractor is marked in green.

# Conclusion

This project focused on the creation of a functional model of the mind state transitions that are observed in focused attention meditation. The variables that are defined were inspired by existing models, especially in the fields of neuroscience and artificial intelligence. The adaptive workspace model provided an abstract mechanism of attention regulation and awareness, which was used to compare the mechanism of consciousness to a Turing process that could be implemented by the functional code. The flow chart model of the attention regulation process in focused attention meditation by Vago inspired the use of particular discrete mental states.

The process of experimental modelling created a basic functional model on a high abstraction level that operates smoothly and is easily modified. The strong abstract foundation of the model provides a stable functional backbone, which can be expanded in complexity. The functional structure of the code together with a sound type system allows flexible manipulation of the code and additional function and variable integration. This was demonstrated by the introduction of functional predicates that manipulated the computational process, which was expressed by different flowchart representations.

The essence of free modelling is based on the construction of mathematical functions. By adjusting these functions, one can analyze whether there exists certain bifurcation values in order to determine semi-stable patterns in a seemingly chaotic system. This is especially relevant in a complex process with high random variability, where it may predict the effect of particular parameter adjustments. In this project basic examples are given of simple strange attractors, which can be considered oversimplified. However complexity is built on a simplicity, such that this basic functional model is the foundation for a more complex model. As the model becomes more complex, there exists a higher capacity to represent a more detailed mechanism of attention regulation in focused attention meditation, in which functions that are acquired through neuroscientific research can be modelled and adjusted with higher predictive validity.

# Literature


Baars, B.J., Franklin, S. & Ramsoy, T.Z. 2013, "Global workspace dynamics: cortical "binding and propagation" enables conscious contents", *Frontiers in psychology,* vol. 4, pp. 200.

Barendregt, H.P. & Raffone, A. 2013, "Conscious cognition as a discrete, deterministic, and universal Turing Machine process" in [S.l.] : Elsevier, , pp. 92-92-96.

Burgess, P.W., Dumontheil, I. & Gilbert, S.J. 2007, "The gateway hypothesis of rostral prefrontal cortex (area 10) function", *Trends in cognitive sciences,* vol. 11, no. 7, pp. 290-298.

Dehaene, S. & Changeux, J.-. 2011, "Experimental and Theoretical Approaches to Conscious Processing", *Neuron,* vol. 70, no. 2, pp. 200-227.

Dehaene, S., Changeux, J.-., Naccache, L., Sackur, J. & Sergent, C. 2006, "Conscious, preconscious, and subliminal processing: a testable taxonomy", *Trends in cognitive sciences,* vol. 10, no. 5, pp. 204-211.

Desimone, R. & Duncan, J. 1995, "Neural mechanisms of selective visual attention", *Annual Review of Neuroscience,* vol. 18, no. 1, pp. 193-222.

Desmurget, M., Reilly, K.T., Richard, N., Szathmari, A., Mottolese, C. & Sirigu, A. 2009, "Movement intention after parietal cortex stimulation in humans", *Science,* vol. 324, no. 5928, pp. 811-813.

Kenemans, L., Bocker, K. & Ramsey, N. *Psychology in the Brain: Integrative Cognitive Neuroscience,* Palgrave Macmillan.

Koster, E.H.W., De Lissnyder, E. & De Raedt, R. 2013, "Rumination is characterized by valence-specific impairments in switching of attention", *Acta Psychologica,* vol. 144, no. 3, pp. 563-570.

Lamme, V.A.F. 2003, "Why visual attention and awareness are different", *Trends in cognitive sciences,* vol. 7, no. 1, pp. 12-18.

Mathews, A. & MacLeod, C. 2002, "Induced processing biases have causal effects on anxiety", *Cognition & Emotion,* vol. 16, no. 3, pp. 331-354.

Raffone, A. & Srinivasan, N. 2009, *An adaptive workspace hypothesis about the neural correlates of consciousness: insights from neuroscience and meditation studies*.

Vago, D.R. & Silbersweig, D.A. 2012, "Self-awareness, self-regulation, and self-transcendence (S-ART): a framework for understanding the neurobiological mechanisms of mindfulness", *Frontiers in Human Neuroscience,* vol. 6.

Yiend, J. 2010, "The effects of emotion on attention: A review of attentional processing of emotional information", *Cognition & Emotion,* vol. 24, no. 1, pp. 3 <last_page> 47.