Synapse

Synapse is an emergence-oriented programming system. It attempts to *simulate* how wetware *computes* and *interacts with the environment,* and how *the environment* reacts to wetware computing and interacting within it.

How useful is it?

Synapse is not useful. *Yet.*

Programming for emergence is as new to programmers as programming is to cavemen.

Emergence is something truly *mastered* by life — and vice versa, of course, for life is also run on top of emergence. Interesting, don't you think?

Programming for emergence is also extremely ineffective, performance-wise. To-be emergence programmers find themselves in a world very much like that the pioneers of computing lived in — restricted by the speed (or, rather, slowness) of contemporary hardware solutions, and inadequacy (incompleteness or under-optimization) of software solutions.

Protocols

Analogous to genes, Synapse cells adhere to *protocols*. The definition of the word *protocol* in Synapse is as loose as the definition of the word *gene* in biology, and perhaps even looser. Despite this, protocols, together with chemistries, are *the* ideas at the basis of Synapse.

Adherence

Adherence attempts to simulate the complexity of *epigenetic factors*. Cell epigenetics is not limited by inheritance of epigenetic factors. Specific genes may be switched "on" or "off" due to activity in the environment, or activity inside the cell itself unrelated to reproduction.

In other words, epigenetics allows the current instance of *cellular runtime* (mainly the collection of all proteins within the cell) to alter the consequent instances. Note that in real life cells, there is no clear separation between such generations of *runtimes*.

Rather, the next runtime gradually *outpopulates* the current one, influenced greatly by the activity in the environment and inside the cell itself. The cell is running extremely rapid, iterative “evolution” over its “population” of proteins — with unneeded ones "dying", and ones “surviving” in the signal environment gaining higher and higher influence and “comfort” until they also degrade.

In Synapse, a cell is allowed to change which protocols it adheres to based on the information from the outside world or its internal state — effectively turning specific protocols "on" or "off" as a consequence of a computationally unrestricted (and perhaps nondeterministic) decision process partially driven by the environment. Expanding on the latter, the decision process is also greatly influenced by the other, neighboring or influential ("beacon") cells.

Genome and Phenome

The *genome* of a Synapse cell is the sum of the protocols it has *access* to. Its *phenome*, then, is the sum of the protocols it *adheres* to. A cell's phenome may change

during its lifetime. The cell is free to join or leave protocols from its *genome* during and due to computation.

The cell cannot change its *genome*. However, it can *reproduce*, for reproduction allows to restrict (or, rather, specify) the genome of the child cell. The parent cell can then commit suicide, emitting cues to the child cell if need be.

Replication and reproduction

Analogous to living cells, Synapse cells can *replicate* (and *reproduce*). The parent cell is allowed to alter the *genome* of the offspring: control the protocols that are going to be accessible for the offspring cell, modify a specific protocol by removing or adding new *rules*, or both. If this happens, then the process is known as *reproduction*.

Birth

Reproduction and creation of cells by GUI means also *sends* a special kind of *message* called the *birth message* to the offspring, which may or may not be *reacted to* by the offspring cell through the *expression* of its *birth rule*.

Imagine the above as if the parent cell has left a number of "you're born" messages inside the offspring cell, and then pressed the "Deploy and Run!" button.

Rules

Rules are named bits of computer code.

Genes come to action via the transcription-translation-active protein route. The transcription-translation part is obviously missing in Synapse, but rules, *at runtime* (in Synapse terms, *when expressed*), are semantically similar to proteins.

In crude comparison to life, before their *expression* rules are like *functionally distinct* substrings of genes, or sometimes entire genes.

To reiterate, there is most of the time no clear division between genes in the real world. Similarly, there is none between Synapse protocols.

Expression

Synapse cells *express* rules in response to signals coming from the environment. These signals are called *messages*, and the cell that receives them is called simply *the receiver cell*.

Now let me repeat the above using clearer terms. *In reaction to a message, the receiver cell expresses a rule*.

Rules and messages are in a lock-key kind of relationship. When a message (our "key") is *acknowledged* and matches a rule (our "lock"), then the rule is *expressed* (the "lock" is opened). In computer terms, the rule is executed, or evaluated.

Intuitively enough, if there are no "locks" matching the given "key", the system won't open anything — no rule is going to be expressed.

Alternatively, if there are several "locks" matching a single "key", the system will open all of them — all matching rules are going to be expressed.

Zygotic bootstrapping — the proto-cell

The only way for a system as a whole to have access to all available protocols (e.g. for self-healing) is to undergo zygotic development — there is no other way around it. In this sense in large systems there must exist the *proto-cell* from which the whole system arose.

Stem cells

Akin to stem cells, a system may develop cells that can be further specialized on demand, and which otherwise follow a simple periodic replication protocol.

Provided cells can crash due to a computational error (e.g. division by zero) — as in express a “death” message and commit suicide, — a management (“rescue”) subsystem may be imagined where *rescuer cells* (or even nearby cells adhering to *rescue protocols*) monitor the environment for “death” messages and send/relay an “allocate” signal to a local stem cell bank, followed by a “specialize” signal to the allocated stem cell(s).

Complex cue emission is imaginable, where nearby cells guide the now-specialized stem cell to the place where the previous cell resided.

Chemistries

Chemistries are the second most important idea in Synapse. Chemistries describe how the environment should react to cells computing and cooperating by the way of messages.

The name, *chemistries*, is simply that — a name. There isn't a clear analogy to Synapse chemistries in the real world. Real world chemistry operates at a much lower scale than chemistries in Synapse. However, there is always the possibility that I am simply unaware of such an analogy.

Vesicles

A Synapse cell — the *sender cell* — simply *declares* a message send when it wants to *send a message*. What follows is done behind the scenes by the Synapse system.

Namely, the system creates a number of copies of the message. The number is obtained using a special formula based on the desired strength. Then the system packs these copies into so-called *vesicles*, one vesicle per copy, and *distributes* them.

The way it distributes them matters, of course. Generally, though, the vesicles are simply *exploded* around the sender cell, just a bit outside of its radius.

Collisions

Like all other physical entities in the environment (including cells), vesicles can *collide* with each other and with cells. The amount of entities participating in a collision is not limited by the environment, but may be limited computationally.

Acknowledgement

When a vesicle collides with a cell, the cell *acknowledges* the message contained in the vesicle. Nothing happens to the vesicle, and whatever the cell does in response is unimportant to the vesicle nor to the chemistries involved.

Refer to the **Protocol** section for details.

Vicinity ring

Vesicles have small virtual *vicinity rings* around them. The vesicle at hand is aware of other vesicles in its vicinity ring, and vice versa.

Synapse chemistries work mainly thanks to vicinity rings, so that vesicles are able to react to and with each other not only by the way of bumping into each other, but also by simply being in the *vicinity* of each other.

Reactions

A *reaction* is a specification of a vesicle-vesicle(s) interaction in the following form:

1. For a specific *kind* of vesicle, let's say for vesicles of the X kind,
2. *what should happen* when vesicles of the A, B, C, D, etc. kind (the number of *participants* in a reaction is unlimited)
3. are either
    - in X's vicinity ring R (provided the *strength of the vicinity ring*), or
    - colliding with X

The majority of the above three parts are expressed using computer code.

What CAN happen?

The complexity of Synapse reactions is computationally unbounded. The *outcome* of the reaction may or may not be deterministic.

Despite this, several *kinds* of outcomes are readily distinguishable:

- Attraction — vesicle X moves toward A, B, C, D, etc., similar to the *attack* device in Synapse cells. Somewhat similar to how magnets work.
- Repulsion — vesicle X moves away from A, B, C, D, etc., similar to the *evasion* device in Synapse cells. Also like magnets, only with two same poles facing each other.
- Promotion — vesicle X "replicates". For instance, when the chemistry "sees" that there aren't enough Xs in the vicinity of an X, it may *promote* the X — create a second, perhaps just a bit weaker, copy.

- Inhibition — vesicle X commits suicide. I can imagine a population control situation when the chemistry "measures" the concentration of Xs in the vicinity of an X, and if this concentration is greater than some threshold (i.e., too crowded), then the X is *inhibited*.
- Transmutation — vesicle X changes its message content. Pretty straight-forward, but note that in this case, as the name suggests, X *mutates* rather than replicates. The old message content of the X subject to transmutation is lost forever.
- Emission — vesicle X replicates, offspring vesicles have different message(s).
- Ignorance — vesicle X does nothing in response to the reaction.

Strength of the vicinity ring

An arbitrary number which is mapped by the Synapse system to the radius of the vicinity ring, using a special formula.

The Signal Environment

Living cells are not black boxes. Neither are Synapse cells. Inside Synapse cells lies *the signal environment*, which is populated by *actors* of rules and protocols.

This world is quite similar to the outside environment, and may even seem redundant (it certainly does to me considering how much code I will have to write!). But the longer you look at it, the longer you think about it — the more strikingly different it appears, enriching the system with even more capabilities when it comes to generating complex emergent behavior.

Actors enable rule-to-rule signaling, mass signaling, and network signaling powered by computationally unbounded code, all within a cell. Mass signaling in particular is absolutely essential in the outside Synapse environment. Turns out, it is useful inside the cell as well!

Rules are connected to one or more protocol(s); there are no standalone rules.

Like Synapse cells, rules (that is, rule actors) can replicate and reproduce.

Rule actors form a rigid(ish) structure, normally around the protocol they're part of. Practically, this means that if you try to move a single rule, the whole structure will move (if you pull hard and far, that is). This is the first major difference: rule actors (and protocol actors) can't move on their own.

Rule-to-rule signaling works like in the outside environment, in that multiple copies of a message are packed in vesicles and exploded with the specified strength.

The crucial difference is that cells support *local, controlled chemistries*. This means you can programmatically, from within one rule or another, regulate the chemistry in the cell's internal signal environment (by adding/removing reactions etc.) — compared to the outside environment, where chemistry is set in stone, for the entire environment, and rules cannot change other than through the UI.

That's cool, but what's the truly *new* , previously *unimaginable* stuff that signal environment brings about? Well, here it is: *Actor population transfer and subsequent interpretation.*

In Synapse, in the outside environment, you cannot simply "copy" and "send" large populations of cells using a message; you can't even *describe* something like that, there's no way to!

However, using the signal environment, you can, because "sending an actor population" means simply sending the corresponding protocol(s), which is perfectly possible in Synapse already.

This is similar to how living cells can "pack" and "send out" functionally distinct pieces of RNA and DNA, which can be transcribed/translated on the receiving side or even within the vesicle — except that in Synapse there's no transcription/translation, and therefore we can simply send *select actor populations*.

A living cell won't really be able to do something like that, at least with great precision (except through compartmentalization/isolation which is another story!). The "signal environment" or a living cell is a soup filled with all sorts of "actors" (predominantly proteins), and it can't *easily* control the contents of this "soup" in specific pockets.

The presence of local, controlled chemistries inside Synapse cells enables different interpretation of same (or at the very least similar) actor populations by different cells, namely of their actor-actor interactions, messaging, or both.

You can imagine two cells, A and B, actively exchanging (or sharing) actor populations through messages, by sending each other the corresponding protocols. Given that A and B are set to have different (or differently *controlled*) chemistries, their interpretation of actor-actor signaling will be different once the actor population is received. Therefore, the resulting (expressed) computation will differ as well, adding yet another "degree of freedom" to the system.

Special formulas

The Synapse system contains a number of "seen" and "unseen" formulas that map all sorts of numbers to all other sorts of numbers. This is done to carefully trade off performance, but more importantly to restrict what a cell or a reaction *does* and *does not* know about, what it can and cannot do.

This is somewhat analogous to how real-world living organisms can't replicate or heal by simply assembling atoms the right way — they just don't have the right set of "handles", "embassies" or "maniples" in the atomic or subatomic worlds.

Molecules are the “floor of control” to life. Similarly, the arbitrary numbers used in entities and reactions are their “floor of control” in the Synapse system. And, as life can control atomic and subatomic structure through chemistry — indirectly and imprecisely most of the time, but still, — entities in Synapse can control low-level emission properties (e.g. the amount of vesicles emitted or lifetimes of individual vesicles) through indirect properties like *strength, liveness,* and so on.

Even though the formulas are the "theory of everything" for the Synapse environment and entities in it, they should not be of importance to the emergence programmer.

That is to say she is mainly going to work with a significant amount of interacting cells and vesicle-vesicle(s) reactions. The more "significant" this amount of interacting cells is, the less influential are the special formulas.