ARTIFICIAL GENERAL INTELLIGENCE: ENGINEERING APPROACH

Mykola Rabchevskiy

March 28, 2010

1 AGI: WHAT'S THE DIFFERENCE

1.1 Immutable behavior systems as AGI predecessor

The simplest "smart system" is a kind of finite state automaton. A sensor subsystem provides automaton input; input evokes both system state change and correspondent action execution using actuators (effector subsystem). Such systems can be very complicated and very hard for reverse engineering but they can't use individual experience to change/improve behavior.

There is no clear border between such systems and AGI systems; each of the improwements described below makes the system more intelligent.

1.2 Event sequence

A finite state automaton keeps some information about past events in a state variable (or variables). However such information is very limited and represents past events implicitly.

The next level is represented by a stack-based automaton. Stack extends capability to remember past events sequence and allows ***explicit*** representation of past event sequence. However automaton stack is not intended for maintenance of such kind of information and a relatively small number of last events can actually be remembered, so the possibility of using individual experience for behavior improvement is still very limited.

AGI fundamentally expands the number of remembered past events: it remembers a much longer event sequence in a dedicated ***event sequeunce***. The event sequence is very similar to stack, referred to above, but it is intended for a long time data accumulation. Event sequence includes sensor data, internal state at correspondent time, and actions executed by system. The longer the event sequence is, the more intelligent behavior can be which uses this information.

1.3 Future prediction

AGI uses an event sequence to ***predict a possible future***. If an event sequence contains subsequences that are matched to the current tail of this sequence then a set of ***possible future events*** can be constructed.

Such a set generally contains variants of future events which started from some system action; such variants consist of three parts: events that preceded action (the same for all variants), performed action and subsequent events.

A collection of such variants can be checked for correlation between performed action and consequences; when correlation is detected then the most appropriate action can be selected for execution.

If "best choice action" is stable for a particular situation then this choice can be transformed into an explicit rule – it is a mechanism of ***conditional reflex***.

Such a mechanism provides the basis for self-learning and adaptivity. It works identically for external world and AGI system hardware. Hardware is actually part of outer world for the AGI "brain", so the AGI system adaptivity and self-learning are equally usable in case the environment changes and hardware changes.

1.4 Explorative behavior

The mechanism described above can only work if a system has a few different variants of performed actions for a particular situation in the past. To provide such variability an AGI system must sometimes perform a randomly selected action instead of the "best choice action" (some restrictions must be provided to protect the system from obvious failure). There is actually random ***experimentation*** which provides a base for behavior improvement (self-learning). Random choice can also be used in cases when a few different actions are equally good.

1.5 Emotions

To compare possible variants of future events, an AGI system requires some ***criteria***. These criteria are provided by sensor subsystem extension by using "generalized sensors" which reflect the current system state as a whole and conforms to human ***emotions***. Part of them is based on sensor data (low battery = "hungry"), the rest is history-based (danger/safe situation, ordinary/unusual situation and so on).

A hierarhical collection of ***emotions*** is used to reduce overall system state (using bottom up algorithm) to a single value on a scale "good – bad". This generalized criterion is used to select a preferred variant of future events and correspodent action to be executed. As is known, such reduction to a single criterion is voluntary by nature; different sets of emotions and different convolution algorithms produce different versions of AGI behavior.

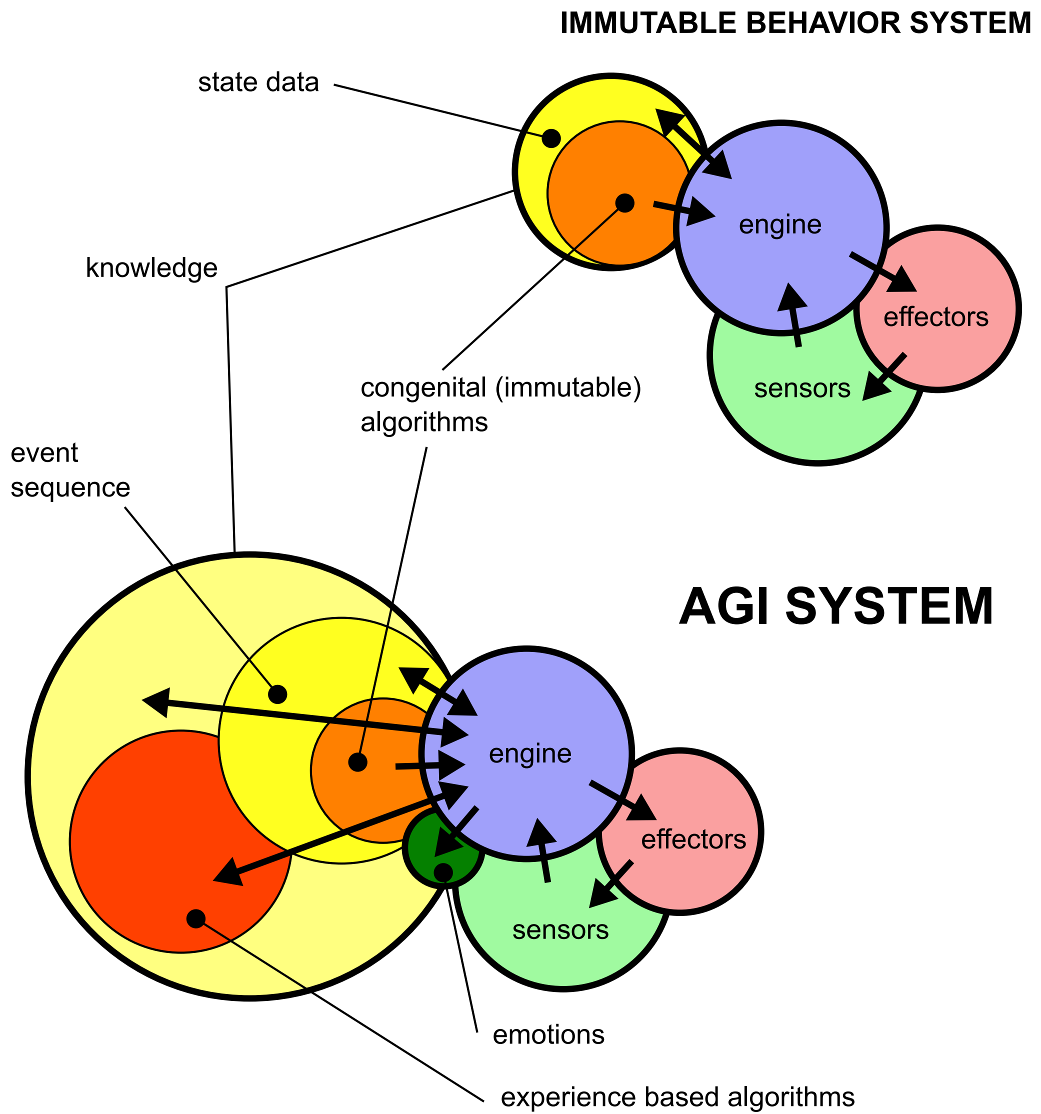


Figure 1: Comparison of immutable behavior system and AGI system

1.6 Generalization

To effectively use available memory resources an AGI system implements compact knowledge representation thanks to ***generalization***. Repeated (occurred at least twice) sub-

sequences of event sequence are detected and a new correspondent concept is created; all occurrences of subsequences are replaced by the newly created concept. Such restructurization activity is permanent (but can be performed at a time when the system is less loaded by external events processing).

1.7 Forgetting

Shortly after an AGI system is started, the whole available memory will be utilized (despite to permanent generalization), so newly added logical concepts must replace some currently stored ones. An algorithm of ***forgetting less usable concepts*** must be implemented. Some ***usability value*** associated with each logical entity reflects how frequently this entity has been used, how long it has been out of use and so on, and entities with the smallest usability value (or below some threshold) are forgotten and related system resources can be re-utilized.

Therefore, an AGI knowledge graph is continually modified: new events are contilually pushed into event sequence, newly discovered repeated fragments produce new concepts and less usable entities are discarded.

1.8 Intelligence level

Overall AGI system power and its specific behavior are dependent on total length of the remembered event sequence, size of event subsequences used at possible future events analysis, emotions subsystem design, generalization and forgetting algorithms.

Obviously, different AGI system characteristics are not coherently dependent on factors enumerated above, therefore AGI designs for high adaptivity systems will be different from designs for high robustness or high reactivity and so on.

1.9 Summary

Main AGI differences from simple system with immutable behavior are:

- AGI system keeps history knowledge explicitly
- AGI system has emotions
- AGI uses known past to analyze possible future
- AGI acts randomly to explore the world and themself
- AGI uses generalization to condense knowledge representation
- AGI permanently forgets less usable knowledge

2 KNOWLEDGE REPRESENTATION

2.1 Definition

Knowledge is a ***structured set of logical entities***.

What does ***structured*** mean? When some set of entities have *something in common*, this *something in common* engenders a concept which denotes this *something in common*. Of course this new concept *is a logical entity* as well. The new concept becomes a ***sign*** for each entity of the set having this *something in common*.

Figure 2: Structured set of entities

Thus, a generated concept is associated with a *set of entities* which are similar in some way; each member of this set has the introduced concept as a ***sign***. Such set of entities will be called the ***explication***. For example, "circle" concept in the Fig.2 has one sign "shape" and an explication set of 3 entities; orange triangle has 2 signs, "orange" and "triangle", and an empty explication set.

2.2 Representation

Obviously, the structure described above can be represented by a ***directed graph*** where nodes(vertexes) are entities and arcs(edges) are directed from a particular entity to the concept which is a sign for this entity. Each edge of the knowledge graph represents an ***atomic statement*** about two connected entities:

square: shape;

A simple subgraph can be represented by the ***compound statement***:

x: orange triangle;

The left part of compound statement is some entity (or set of entities as described below) and the right part is a set of signs of the left-side entity. A more complicated subgraph can be represented by a set of such statements. There are no restrictions: a knowledge graph (as well as any subgraph) can have ***cycles*** and/or ***loops***.

Representation of set and hierarhical structure (like ontologies) are obvious: any concept represents some set of entities (explication) and any entity can be element of set.

2.3 Syndrome and anonymous entity

The next example shows another peculiarity of our approach: ***anonymous entities*** (concepts). An anonymous entity can be referred to using a set of it signs called a ***syndrome***:

(Mary sohn)

Because a syndrome is a ***set*** of signs, the order of signs in the parentheses is insignificant.

When a syndrome appears on the left part of statement it represent generally some set of entities. The following statement means "all John's cars are Fords"; the implied set of cars can be an empty set, a single car or a few cars:

(John car): Ford;

Actually, such a statement is a primitive form of rule.

A syndrome can be used recursively; when a syndrome appears on the right part of a statement it represents some unique entity:

((John family) car): (hybryd sedan);

Systematic use of syndroms make it unnecessary to label most of the knowledge graph vertexes; this proposed notation is also in some ways closer to simplified natural language.

2.4 Sequences, tuples, arrays and maps

Third (after *set* and *hierarchy*) most widely used data structure is ***sequence*** (and its derivatives like *stack* and *queue*): it is used to represent an event sequence, to keep an algorithm as a sequence of statements, and so on. A sequence can be represented by an ordered set of anonymous auxiliary entities; each auxiliary entity represents a particular ***sequence element*** and has the entity-***value of sequence element*** as a sign. Such a structure represents also a ***single linked list***.

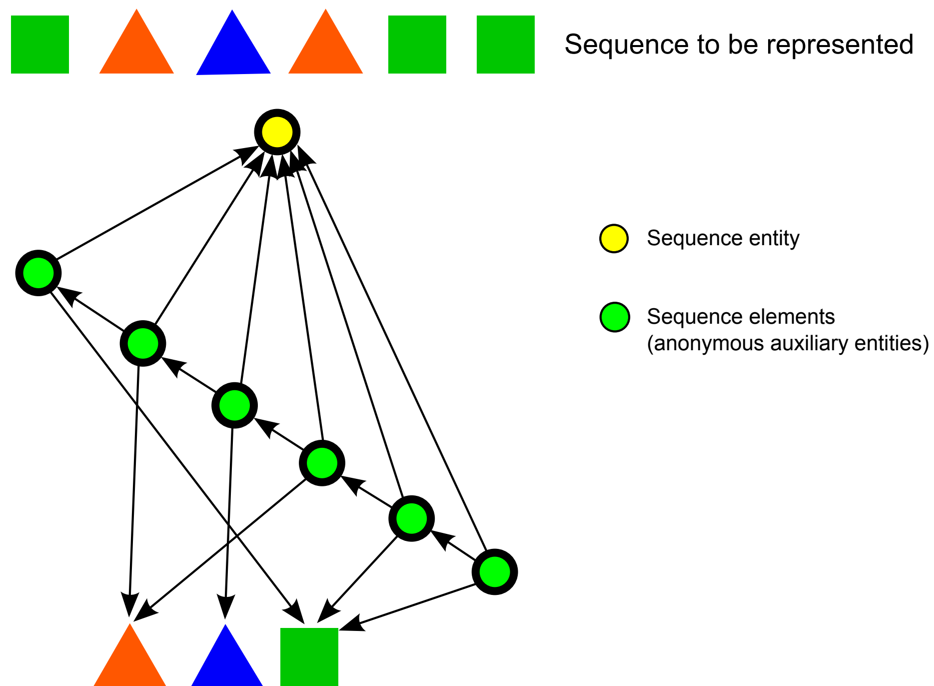


Figure 3: Sequence representation

Finally ***array***, ***tuple***, and ***map*** representations are based on a similar approach: each

collection element is represented by an anonymous auxiliary entity, and these auxiliary entities have a sign which is used as index, component name or key respectively.

More complicated structures can be constructed from the described components; for example ***three*** is a ***hierarhy of tuples***.

2.5 Knowledge graph vs semantic network

Knowledge graph looks similar to a semantic network with the single "universal" relation ***is-a***. A classic semantic network splits all logical entities into two categories, ***objects*** and ***relations***. Objects are associated with vertexes and relations are associated with edges. Such a dichotomy creates a lot of problems because actually the same logical entity can be an object in one statement and a relation in another statement, so a single semantic graph generally can't represents an arbitrary collection of statements. Knowledge storage based on a semantik network is forced to use a *set of semantic graphs* (and some information about this set in addition) to represent a knowledge collection. Unlike a semantic network, our approach represents any knowledge collection (including any knowledge about the collection itself) as single knowledge graph (see Fig.4). Another advantage of this proposed knowledge representation is that each logical entity is presented in the knowledge graph exactly once.

So the main differences between our approach and a classical semantic networks are:

- any knowledge collection can be represented by a single knowledge graph
- each logical entity is presented in a knowledge graph exactly once

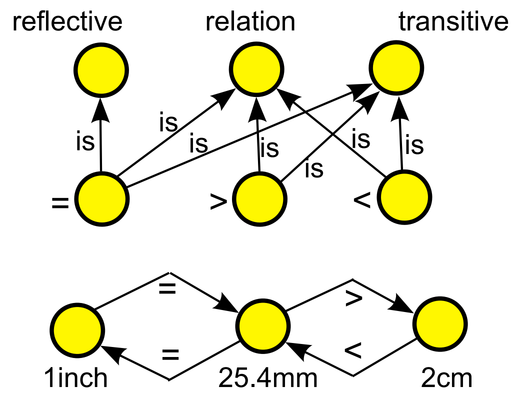
3 EMBODIMENT

The AGI system engine (Fig.1) consists of following components:

- knowledge graph loading/saving unit
- sensor data acquisition unit
- actuator controlling unit
- knowledge browsing unit
- knowledge processing unit

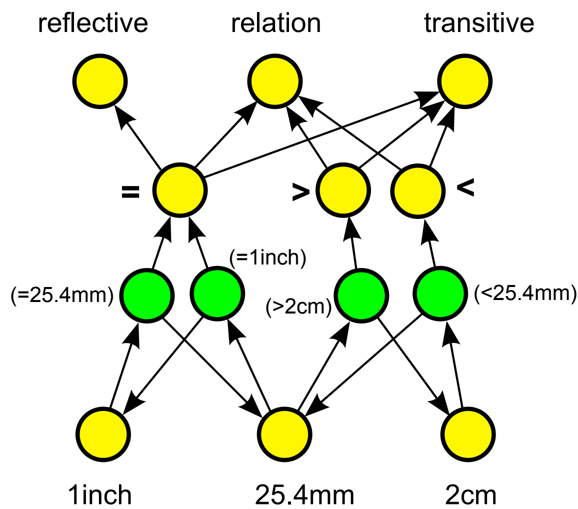
All components of the engine are hard coded and hardware dependent; there is really part of the system "body", as opposed to the knowledge graph which is analog of the central nervous system.

Loading/saving unit provides knowledge persistence.



SEMANTIC NETWORKS

Two graphs can not be merged
Logical entity can be presented many times



KNOWLEDGE GRAPH

Single graph
Each logical entity presented once

Auxiliary anonymous entity

Figure 4: Knowledge graph vs semantic network

Optional knowledge browsing unit provides debugger-like access to knowledge graph at runtime.

The knowledge processing unit is intended to execute both immutable (congenital) and mutable (individual experience based) algorithms. It makes sense to represent both kinds of algorithms in the same way; in particular, it simplifies the transformation of mutable algorithms (as part of individual experience) into immutable for another system version.

An algorithm is represented in a knowledge graph by a sequence of entities where each sequence element represents operand, operator or algorithm. Recursion produces hierarchical knowledge about actions.

The knowledge processing unit is actually an interpreter of a concatenative stack-based FORTH-like language. Low level language operators are predefined entities; they provide basic operations on the knowledge graph. The rest of the operators are subprograms. In contrast to FORTH this interpreter recognizes operators and operands using entity signs instead of traditional parser utilization, so textual representation of algorithms is not used by the interpreter.

Immutable (congenital) algorithms are part of congenital knowledge, so they are preloaded into the knowledge graph before the AGI system is started. Any non-protected (i.e. mutable) part of the knowledge graph including mutable programs can be modified by being stored into the knowledge graph program.

Such an approach provides clear separation between AGI engine coding and programming the AGI itself; the same AGI engine can be used for different AGI systems, and the same congenital programs can be used with different AGI engines and/or different hardware.