Università degli Studi di Padova

Department of Information Engineering

Master Thesis in Computer Engineering

A new scope for Open Cognition project: task and motion planning through its Neural-Symbolic Knowledge Store

Supervisor
Emanuele Menegatti
Università di Padova

Co-supervisor
Enrico Pagello
Elisa Tosello
Università di Padova

Master Candidate
Michele Thiella

Date
.. October 2021

Academic Year 2020 - 2021

Abstract

Contents

List of Figures

List of Tables

Road to agi.

1 Introduction

As industrial robots become faster, smarter, and cheaper, more and more companies are beginning to integrate this technology in conjunction with their workforce. Nowadays, robots have a range of applications that cover many different areas. These applications are no longer limited in structured environments, where the robot behavior could be directly specified by a human.

Where early robots blindly followed the same path, and later iterations used lasers or vision systems to detect the orientation of parts and materials, the latest generations of robots can integrate information from multiple sensors and adapt their movements in real time. They can also make use of more powerful computer technology and big data–style analysis.

Advances in artificial intelligence and sensor technologies allow robots to cope with a far greater degree of task-to-task variability. However, their ability to adapt to different tasks is still a long way from a general level of adaptability.

Until now, for example, every industrial robot is designed with a specific purpose. Therefore, a new robot is often needed to perform a new task. Even the most advanced robots, which exploit Neural Networks (NN) and Artificial Intelligence (AI), are unable to easily learn a new task.

It is necessary to design the AI architecture carefully and, to be successful, people must stop misusing the term AI. This term is used by generalizing those that are the three types of AI: Narrow-AI, General-AI, and Super-AI. The distinction between them is the key to technological progress, especially in the robotic field.

1.1 Artificial Intelligence

The following is a brief description of these three AI types, the Articles [1, 2, 3, 4, 5, 6] talk more about this.

1. Narrow-AI:
   Current AI technologies all fall under the Artificial Narrow Intelligence (ANI or Narrow-AI) category, which means they are very good at only one or a few closely related tasks. This type of AI has a limited range of abilities, specifically designed for a narrow use. It is able to reach a level of performance of a human, and even better, but only within this limited field that is its specialty. Examples of ANI include everything from Siri, Face ID and the Google Assistant, to self-driving cars and DeepMind's board game playing program. This is the only form of AI that has been developed so far.

2. General-AI:
   The next step after ANI is Artificial General Intelligence (AGI or General-AI), much more similar to human intelligence and not focused on specific tasks. It would be similar to a human mind and in theory, it should be able to think and function like it, being able to make sense of different content, understand issues and decide what is best in a complex situation. AGI hasn't been achieved yet. There isn't the technical capacity of producing something as complex yet, and there is also no certain knowledge of how the human brain actually works either. AGI is a relatively logical and rational future though, and it could be attained at some point if humans develop their knowledge and understanding, as well as technical skills to a high enough level.
   An overview of AGI, including important reflections, written by Ben Goertzel, can be found here: [7].

3. Super-AI:
   When AGI is achieved and computers are able to learn independently at a very quick rate, and exponentially improve on their own without human intervention or help, the final step that AI could hypothetically reach is Artificial Super Intelligence (ASI or Super-AI). At this stage AI would be capable of vastly outperforming the best human brains in practically every field. The evolution from AGI to ASI would in theory be fast, since AGI would allow computers to "think" and exponentially improve themselves once they are able to really learn from experience and by trial and error.

The most important differences, easily understood from the definitions above, are between ANI and AGI, partly because ASI will only be considered when AGI exists. The timing forecasts and the difficulties of that step can be found in [8].
These differences immediately lead into much wider contexts than industry, as AGI seems

focused on complex environments such as real-world and human-computer interaction. However, using an AGI system in an industrial setting can have many benefits:

- Having a general knowledge base for the robot, which can be shared with other ones.
- Use the robot(s) to cover multiple and more complex tasks.
- Make cooperation with humans natural and encourage learning from them.
- Robot(s) is robust to changes in the environment, its state and its task.
- Easier implementation of new modules and their merge into the system.
- Not only maintain, but rather exploit existing Narrow-AI systems as modules of the AGI system, in order to take advantage of their potential and allow interaction between them in a "single" large knowledge base.

For this project, one of the proto-AGI[1] systems currently under research is used, which proposes a concrete system that demonstrates the feasibility of the concepts just listed, and more.

There are several projects and researches that want to achieve the AGI goal. One of these is the Elon Musk and Sam Altman's OpenAI Project. Its mission is to ensure that AGI, intended as highly autonomous systems that outperform humans at most economically valuable work, benefits all of humanity.
OpenAI is focused on Deep Neural Networks for almost all projects such as OpenAI Codex, its AI system that translates natural language to code [9], CLIP (Contrastive Language-Image Pre-Training) a general-purpose vision system, which is a Neural Network trained on a variety of image-text pairs [10] and one of the most important: Generative Pre-trained Transformer 3 (GPT-3). This is an autoregressive language model that uses Deep Learning to produce human-like text [11]. With over 175 billion parameters, it is the largest Neural Network ever created [12].
Although OpenAI is well-funded and achieving excellent results, there is a second AGI-oriented project that is noteworthy: the Open Cognition (OpenCog) project. It is also the system on which this project is based.

[1] Any current theoretical or practical concept about AGI falls into a category that is called proto-AGI for now. However, in this paper the two terms are considered interchangeable.

The reason for this choice, the preference of OpenCog over OpenAI, lies in the approach to solving the AGI problem.
As mentioned above, OpenAI appears to be based on the general plan of starting from current Deep Neural Network tech, applying and extending it in various interesting and valuable ways, and in this way moving incrementally toward AGI without that much of a general plan or model of the whole AGI problem.
On the other hand, OpenCog is founded based on a comprehensive model of human-like general intelligence, and a comprehensive overall plan for getting from here to human-level AGI. Thus, it has an integrative approach in which multiple different sorts of AI algorithms (Deep Neural Networks, Probabilistic Logic Theorem Proving, Evolutionary Learning, Concept Blending, etc.) operate together on a common representational substrate.
It is not about building more accurate classification algorithms, or more efficient computer vision systems, or better language processing or boutique information retrieval algorithms, diagnosing diseases, answering trivial questions or driving a car, etc. It is concerned with generic intelligence and the inter-related cognitive processes it entails. It is about making software that perform specific tasks, using structures and processes that appear capable of being extended to more and more general tasks.
The OpenCog system is explained in detail later, in the Chapter **2**.

OpenAI and OpenCog are two distinct ways, both valid. However, we believe that the AGI architecture shouldn't be based on Neural Networks, for several reasons (in addition to classic general problems as in [13]):

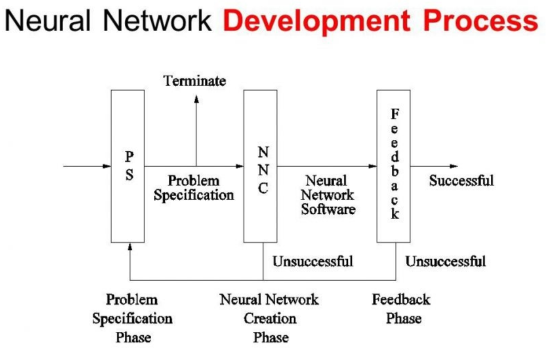
- A huge amount of data is required.
  Figure 1.1b shows annual data growth versus growth in CPU processing power. The gap leads one to think that as the data required for NNs increases, hardware technology must be improved.

- Adversarial Examples problem: it is a way to deceive almost all AI classifiers easily.
  Figure 1.1c is an illustration of machine learning adversarial examples. Studies have shown that by adding an imperceptibly small, but carefully designed perturbation, an attack can successfully lead the machine learning model to making a wrong prediction. For more information see [14, 15, 16, 17].

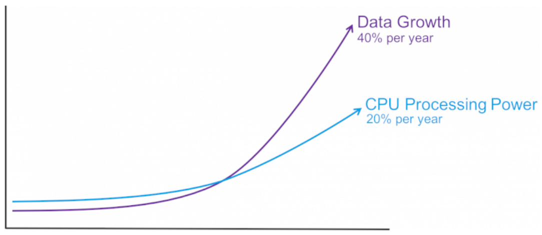
- The cost: training GPT-3, for example, would have an estimated cost between \$4.6 and \$12 million [18, 19].
  Figure 1.1a gives a general idea of a NN development process .

- Neural Networks as black box: the best-known disadvantage of Neural Networks is their "black box" nature. It means that, while it can approximate any function, studying its structure won't give any insights on the structure of the function being approximated. More details in [20].

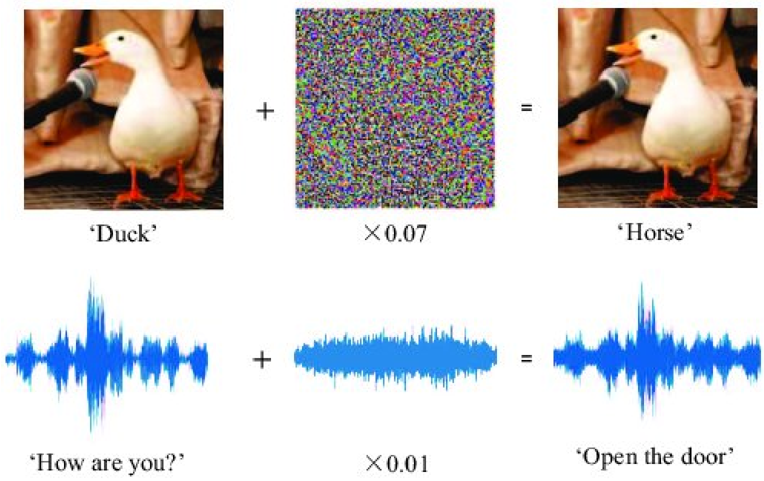
The disadvantages of an NN-based approach like OpenAI have been mentioned. Now, to understand the advantages of a human-like approach like OpenCog, it is necessary to explain the OpenCog system. Therefore, these advantages are postponed to the Section [sec:opencog advantages ...].



(a) Neural Network development process.
Image source: [21].



(b) Neural Network data process.
Image source: [22].



(c) Adversarial Examples.
Image source: [23].

Figure 1.1: Some disadvantages of Neural Networks.

2

OpenCog System

The OpenCog design aims to capture the spirit of the architecture and dynamics of the brain without imitating the details (which are largely unknown) via:

- Integrating together a carefully selected combination of cognitive algorithms acting on different kinds of knowledge.
- A scalable, robust and flexible C++ software architecture.
- A manner specifically designed:
  - To cooperate together with "cognitive synergy" for the scope of tasks characteristic of human intelligence.
  - To give rise to the emergence of an effectively functioning knowledge network in the AI system's mind, as it interacts with the world, including a self-updating hierarchical/heterarchical ontology and models of itself and others.

Following section, Section 2.1, elaborates on the new concepts introduced in these points.

2.1 Cognitive Synergy

OpenCog is a diverse assemblage of cognitive algorithms, each embodying its own innovations. The power of the overall architecture is its careful adherence to the principle of Cognitive Synergy.

The human brain consists of a host of subsystems that perform particular tasks, both specialized and general in nature, connected together in a manner enabling them to synergetically assist, rather than work against each other.
The essential principles of Cognitive Synergy Theory (CST) can be summarized in the following points, further explored in [24]:

1. Intelligence can be understood as the ability to achieve complex goals in a certain set of environments.

2. An intelligent system requires a "multi-memory" architecture, meaning the possession of a number of specialized yet interconnected knowledge types.

3. "Cognitive processes": a system must possess knowledge creation mechanisms corresponding to each of these memory types.

4. Each cognitive process must have the ability to recognize when it lacks information and thus, draw it from knowledge creation mechanisms related to other types of knowledge.

5. The Cognitive Synergy is, therefore, represented by the interaction between the knowledge creation mechanisms, which perform much more effectively in combination than non-interactive mode.

6. The activity of the different cognitive processes involved in an intelligent system can be modeled in terms of the schematic implication "Context & Procedure $\rightarrow$ Goal".

These points are implicit in the systems theory of mind given in [25], where more thorough characterizations of these ideas can be found.
Interactions as mentioned in Points 4 and 5 are the conceptual core of CST.
Most AI algorithms suffer from combinatorial explosions. In a "general intelligence" context, there is a lack of intrinsic constraint; consequently, the algorithms are unable to filter through all the possibilities (as opposed to a ANI problem like chessplaying, where the context is huge but constrained and hence restricts the scope of possible combinations that needs to be considered).
To decrease the severity of combinatorial explosions, one can use an AGI architecture based on CST, in which the different learning mechanisms dealing with a certain sort of knowledge, are designed to synergize with ones dealing with other sorts of knowledge.
It is necessary that each learning mechanism recognizes when it is "blocked" and then, it can ask for help to the other complementary cognitive mechanisms.

The Figure 2.1 is proposed to give a general visual idea of these concepts. It shows an overview of the most important cognitive dynamics considered in Cognitive Synergy Theory and describes the behavior of a system as it pursues a set of goals, which are then refined by inference (through a logic engine or as an emergent process resulting from the dynamics of an Neural Network system), aided by other processes.

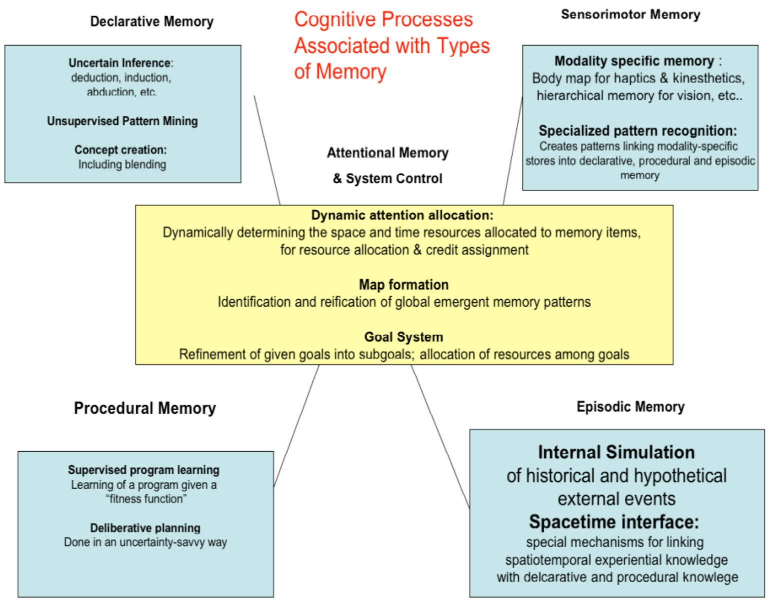


Figure 2.1: A high-level overview of the main types of cognitive process considered in Cognitive Synergy Theory, categorized according to the type of knowledge with which each process deals.

The detailed argument explaining how the cognitive algorithm selection and integration methods, chosen by the OpenCog team, will have the desired effect, is sketched on the OpenCog wiki site[1] and various previously-published conference papers. It has been presented more thoroughly in the 2014 books Engineering General Intelligence vol. 1 and 2 [26, 27].

2.2 OpenCog Architecture

There are several components that make up the basic architecture of OpenCog. In the following sections, they will be described, more or less independently, and then concatenated and contextualized into the problem considered in this project.

[1] https://wiki.opencog.org/w/Background_Publications

Currently, the OpenCog project is under strong development and some of the concepts presented here may be obsolete, improved or evolved, or being redesigned. However, much of the basic infrastructure and theory remains unchanged.

2.2.1 Atomspace

The AtomSpace is a platform for building Artificial General Intelligence (AGI) systems. It provides the central knowledge representation component for OpenCog. As such, it is a fairly mature component, on which a lot of other systems are built, and which depend on it for stable, correct operation in a day-to-day production environment.
It is a mashup of a large variety of concepts from mathematical logic, theorem proving, graph theory, database theory, type theory, model theory and knowledge representation.
More specifically, the OpenCog AtomSpace is an in-RAM knowledge representation database, an associated query engine and graph-re-writing system, and a rule-driven inferencing engine that can apply and manipulate sequences of rules to perform reasoning.
The best way to capture all of this, is a kind of in-RAM Generalized Hypergraph (Metagraph) Database.
On top of this, the Atomspace provides a variety of advanced features not available anywhere else. It is currently used to store natural language grammars, dictionaries and parsers, to store biochemical and biomedical data, robot control algorithms, machine learning algorithms, audio/video processing pipelines and deep learning neural networks.

2.2.1.1 Metagraph: a Generalized Hypergraph

Formally, a graph is:

- A set of vertexes $V = \{v_1, v_2, \cdots, v_M\}$
- A set of edges $E = \{e_1, e_2, \cdots, e_N\}$ where each edge e_k is an ordered pair of vertexes drawn from the set V.

Since edges are ordered pairs, it is conventional to denote them with arrows. In practice, one wishes to associate a label to each vertex, and also some additional attribute data (e.g. weight); likewise for the edges.

A hypergraph is very similar to a graph, except for the edges. "Hyperedges" are defined as edges that can contain more than two vertices. That is, the hyperedge, rather than being an ordered pair of vertices, is an ordered list of vertices.
Formally, a hypergraph is:

- A set of vertexes $V = \{v_1, v_2, \cdots, v_M\}$
- A set of hyperedges $E = \{e_1, e_2, \cdots, e_N\}$ where each hyperedge e_k is an ordered list of vertexes drawn from the set V. This list may be empty, or have one, or two, or more members.

A good representation of a hypergraph is the one proposed in Figure 2.2. It was a straightforward extension of the edge table, to which a new column was added for each position and then, those columns were mashed together into one set.

vertex id	incoming-set	attr-data
v_1	$\{e_1, e_2, e_4\}$	...
v_2	$\{e_2, e_4\}$	...
v_3	$\{e_3, e_4\}$	...
v_4	$\{e_2\}$	

Figure 2.2: Hypergraph edge table

The vertex table looks like the edge table, but the vertex-list is an ordered list, while the incoming-set (the edge-set) really is a set. This is because a hypergraph is "almost" a bipartite graph, having the form of Figure 2.3, with the set E on the left being the set of hyperedges.

Before define a Metagraph, a change of terminology is useful: the basic objects are now called "Nodes" and "Links" instead of "vertexes" and "edges".
Thus, a Metagraph is:

- A set of nodes $V = \{v_1, v_2, \cdots, v_M\}$
- A set of links $E = \{e_1, e_2, \cdots, e_N\}$ where each hyperedge e_k is an ordered list of nodes, or other links, or a mixture. They are arranged to be acyclic (to form a directed acyclic graph).

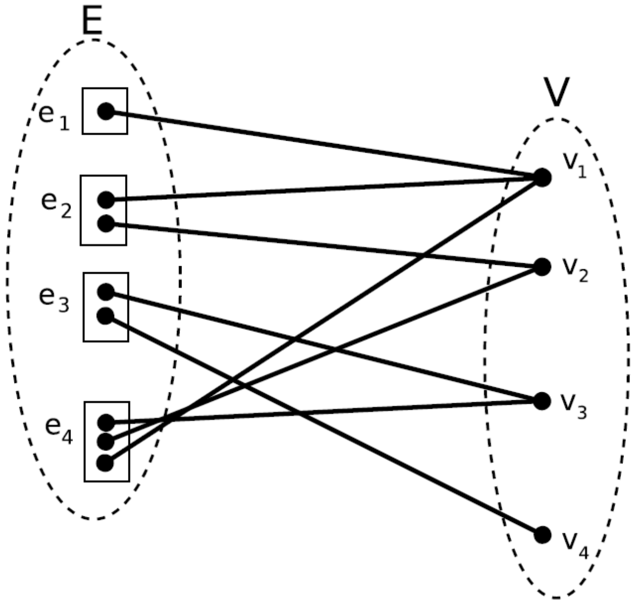


Figure 2.3: The E and V ellipses are the hyperedge and vertex tables. The boxes mean that the hyperedges are ordered lists.

The metagraph is a generalization of a hypergraph, in the sense that now a hyperedge (link) may contain either another vertex (node) or another link. Visually, it has the shape of a Directed Acyclic Graph (DAG), such as the one shown in Figure 2.4.

But in Metagraph, links are ordered lists, represented as boxes. Thus, to convert it in a DAG it is possible to collapse the boxes to single points or to dissolve the boxes entirely and replace a single arrow, from point-to-box, by many arrows, from point to each of the box elements.
Whereas the node table is the same as the vertex table for the hypergraph, nevertheless the link table now requires both an outgoing-atom list and an incoming-link set. Figure 2.5 shows the result.

For convenience, the name "Atom" is given to something that is either a node or a link. Links are then, sets of atoms.
These concepts are described in [28, 29], where RAM-usage considerations, reasons why metagraphs offer more efficient, more flexible and more powerful ways of representing graphs and reasons why a metagraph store is better than a graph store and much more, can also be found.

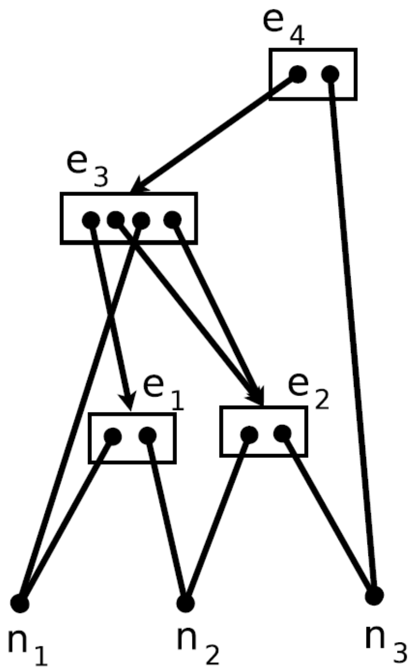


Figure 2.4: An example of a DAG.

Finally, some extensions of the metagraph are considered: Typed Metagraph (TMG) and Directed Typed Metagraph (DTMG).
Typed metagraphs are defined as hypergraphs with types assigned to hyperedges and their targets, and the potential to have targets of hyperedges connect to whole links, as well as targets. An example can be found in Figure 2.6.

A natural extension of a TMG is the Probabilistically TMG, based on probabilistic dependent types. Thus, one can assign a probability (or an entire probability distribution) to each connection between edges, thanks to the probabilistic type inheritance relations. In this way, it is possible to obtain a KB that can work with probabilistic logic, fuzzy logic, make uncertain inferences and much more.
Atomic Directed Typed Metagraphs (atomic DTMG) are introduced via partitioning the targets of each edge in a typed metagraph into input, output and lateral sets; one can then look at "metapaths" in which edges' output-sets are linked to other edges' input-sets (Figure 2.7). Thus, a DTMG is generally defined as a TMG composed by connecting DTMGs via metapaths, a recursive definition that bottoms out on the definition of atomic DTMGs.
For the whole theoretical formalism concerning TMG and DTMG refer to [30]. That pa-

link id	outgoing-list	incoming-set	attr-data
e_1	(n_1, n_2)	$\{e_3\}$	...
e_2	(n_1, n_3)	$\{e_3\}$	...
e_3	(e_1, e_2, n_1, e_2)	$\{e_4\}$	...
e_4	(e_3, n_3)	$\{\cdot\}$	

Figure 2.5: Metagraph link table

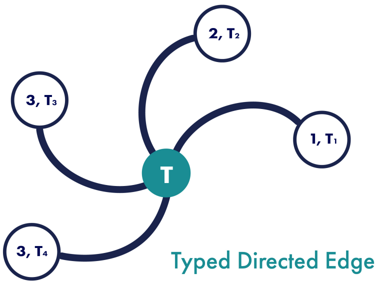


Figure 2.6: Example of a typed, directed edge. This one has 4 targets. T is the type of the edge itself. The third and fourth targets are unordered relative to each other.

per concludes by also describing useful types of morphisms that can be defined on a DTMG (catamorphisms, anamorphisms, histomorphisms, futumorphisms, hylomorphisms, chronomorphisms, metamorphisms and metachronomorphisms). They allow to formulate a wide variety of operations on metagraphs, which will not be described here.

The important thing to keep in mind is that the KB is used for AGI, so there will be many metagraphs and very large ones. Morphisms allow you to obtain simple and complex results by mutating/transforming/stretching/compressing the metagraph quickly and cleanly.

Lastly, it is also useful to associate metagraph edges E_i with nite lists V_i of Values, each of which may be integer, floating-point or more complex in structure. The reason for these Values will be mentioned later.

2.2.1.2 Atoms

The vertices (nodes) and edges (links) of a graph (metagraph), known as Atoms, are used to represent not only "data", but also "procedures" and then, many metagraphs are executable

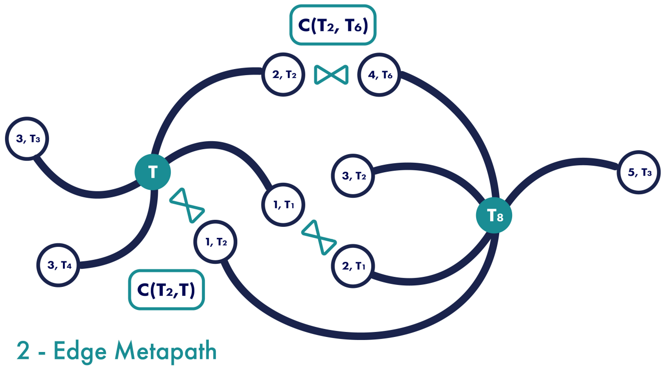


Figure 2.7: A short metapath, formed by connecting two directed edges.

programs as well as data structures.
Atoms are one of the main components of the AtomSpace. Atoms, together with Values are what the AtomSpace stores.
The two primary types of Atoms are Nodes and Links. They are used to represent anything that resembles a graph-theoretical graph.
From the TMG definition above, Atoms are typed (in the sense of Type Theory) and thus, they can be used to store a large variety of information. Values are used to assign "valuations" to Atoms, to indicate the truth or likelihood of that Atom, or to hold other kinds of transient data. Every Atom has a key-value database attached to it, that can store any kind of information about that Atom. The distinction between the "shape of the graph", and its related "data" is central for allowing high-speed graph traversal and generalized graph query.
Note that, the name "Atom" was chosen because of the resemblance of OpenCog Atoms to the concept of atoms in mathematical logic. Moreover, in Ruby and Prolog programming languages, symbols are literally called atoms; in both LISP and Guile, allow parameters to be attached to symbols, as Values can be attached to Atoms.

When an atom are placed in the AtomSpace, it becomes unique: it gets a single, unique ID. Thus, in comparison to other programming languages[2], Atoms can be understood to be the same thing as symbols. The AtomSpace is essentially a symbol table, which commonly has the unique-symbols property.
The atom ID is the string name of a Node and the outgoing set of a Link. Specifically: Nodes are identified by their names or labels, that is the only property that they have. Links are iden-

[2] Section 2.2.2 explains why it is possible to talk about programming languages

tified by their contents, which are ordered or unordered sets of other atoms. Links do not have any name or label other than their contents: a link is uniquely identified by it's type and it's contents.
A TruthValue (a certain type of Values) gives each Atom a valuation or an interpretation and conseguently, all Atoms in a given, fixed AtomSpace always carry a default valuation/interpretation (i.e. a SimpleTruthValue) along with them. Naturally, an Atom may have one of several different kinds of TruthValues.
All TruthValues (tv) expose at least two parameters, representative of the SimpleTruthValue (stv):

1. Strength: it represents a probability estimate of the true unknown probability. It is a floating-point value ranging from 0 to 1, with 0 denoting the classical Boolean false, and 1.0 denoting true.
2. Confidence: it captures the spread of the second order distribution over the true unknown probability. It is a floating point value ranging from 0 to 1, expressing the certainty of the strength, with 0 denoting completely uncertain, and 1 denoting completely confident.

The types form a type hierarchy: all atoms inherit from the type "Atom", and the type Atom itself inherits from ProtoAtom. The ProtoAtom is itself the base type for Values as well as Atoms. The atom type category site[3] lists all documented theorized, proposed, currently in use, deprecated and obsolete atoms.
It is not a complete list: new types are easily invented, and the various OpenCog books mention Atom types that are not implemented or have been implemented in a different way. This reflects a more general point: the specific collection of Atom types in an OpenCog system is bound to change as the system is developed and experimented with. In the current system, it does not necessarily have a profound and lasting significance.

Following a descriptive list of the main types of atoms used for this project:

- **ConceptNode**: a Node representing any concept.
  Its TruthValue, composed of at least a strength and a confidence value, has the strength that indicates the occurence of a concept within the context of experience and the confidence that indicates how sure the agent or system is of this value.

[3] https://wiki.opencog.org/w/Category:Atom_Types

For example, imagine an empty AtomSpace and a newborn agent that begins observing the world for the first time. If the first two things it sees are a man and a cat, it may define the following concepts:

Code in Scheme programming language notation: ConceptNode example.

```
ConceptNode "man" (stv 0.5 0.001)
ConceptNode "cat" (stv 0.5 0.001)
```

Since the agent has only observed two concepts in its universe, it will split in two the universe (and Strength accordingly): half (0.5) consists of men and the other half (0.5) consists of cats. Because the agent has not made a lot of observations yet, it may assign a low Confidence to these values.

- **InheritanceLink**: a Link specify is-a relationships.
  In the OpenCog system, basic InheritanceLinks are used to specify both intensional (is-a) and extensional (is-an-instance-of) relationships.

  Code in Scheme programming language notation: InheritanceLink example, which specifies that a cat is an animal.

  ```
  InheritanceLink
          ConceptNode "cat"
          ConceptNode "animal"
  ```

  In this case, the TruthValue associated to InheritanceLink should be 0, if it is considered as extensional inheritance (inheritance between sets based on their members) and a value greater than 0 in the case of intensional inheritance (inheritance between entity-types based on their properties); obviously the value depends on how the agent assigns it.

- **PredicateNode**: it names the predicate of a relation.
  Predicates are functions that have arguments, and produce a truth value as output. Predicates in OpenCog roughly resemble the predicate of first-order logic, but more general. It is very similar to a characteristic function in probability theory, which helps assign a floating-point truth value to a declaration. Its usege will be understood at later points.
  Two interesting type of Atom that are derived from PredicateNode are the **DefinedPredicateNode** and the **GroundedPredicateNode**.
  The first is a single atom that is attached to a more complex definition. During the evaluation of the predicate, its definition (typically some formula or other complex expression, constituted entirely of Atoms) is looked up and is used during evaluation.
  The second specifies a predicate whose truth value is updated by the evaluation of a scheme, python or C++ code snippet. It is a "black box" from the point of view of

logical inference and knowledge representation. It is impossible to reason with it, as opposed to DefinedPredicateNodes, which are "clear boxes" whose inner workings are visible to inference, learning and analysis algorithms.

- **ListLink**: a Link used for grouping Atoms for some purpose, typically to specify a set of arguments to some function or relation.
  The ListLink is best understood as a Cartesian product[4], because all of its uses involve passing an ordered sequence of arguments to some function or predicate. Ordered sequences can be naturally understood as Cartesian products, which is extremely general, cutting across all branches of mathematics.

- **EvaluationLink**: provides a way for specifying the truth value of a predicate on a set of arguments.
  The EvaluationLink is the most central and important atom type in OpenCog, as it is how OpenCog implements knowledge representation.

  Code in Scheme programming language notation: EvaluationLink general structure, followed by a pratical example.

```
EvaluationLink <tv>
        PredicateNode some_p
        ListLink
                SomeAtom val_1
                OtherAtom val_2
```

  This indicates that the predicate *some_p*, applied to arguments *val*_1 and *val*_2, has the TruthValue $tv = some\_p(val\_1, val\_2)$.
  Pratical example, $3 < 42$ is true:

```
EvaluationLink <true_tv>
        PredicateNode "LessThan"
        ListLink
                NumberNode 3
                NumberNode 42
```

There are other atom types used here. A more general list of Opencog atom types, which can be divided into categories, is this one:

- Arithmetic Atom Types: EqualLink, GreaterThanLink, NumberNode, IntervalLink, etc.

- Boolean Atom Types: AndLink, OrLink, NotLink, etc.

[4] https://github.com/opencog/atomspace/issues/1490#issuecomment-352961503

- All atom types used for Natural Language Processing (NLP)
- Atom types for the OpenCog Probabilist Logic Network (PLN)
- Atom types to perform Pattern Matching (PresentLink, AbsentLink, VariableList/-VariableSet/VariableNode, etc.)
- Atom types for the threading
- Set-theoretical Atom Types: types having some form of set-theoretical signficance (MemberLink, SetLink, SubsetLink, etc.)
- Atom types to perform searches over the AtomSpace (JoinLink, QueryLink, MeetLink, etc.)

In conclusion, there is a type system for working with atom types: that is, the type of an atom can itself be specified using other atoms, which also applies to the polymorphic types. The type system allows for basic error checking, such as with the type checker, and more generally, it allows type-logical reasoning and inference to be performed on atom signatures.

2.2.2 Atomese

Because of these many and varied Atom types, constructing graphs to represent knowledge looks like a kind of "programming".
The programming language is informally named "Atomese".
Atomese is the concept of writing programs with Atoms. It vaguely resembles a strange mash-up of SQL, due to queriability, Prolog/Datalog, due to the logic and reasoning components, Lisp/Scheme, due to lambda expressions, Haskell/CaML, due to the type system, and rule engines, due to the graph rewriting and forward/backward chaining inference systems.

Atomese is not, and was never intended to be, a programming language with which humans would write source code, like Java, Python, LISP or Haskell. Rather, it is a language that computer algorithms, such as genetic programming systems, term rewriting systems and rule engines, would be able to manipulate. It is a graph database that pattern mining algorithms could search, manipulate and transform and is designed for easy self-introspection.

In its current form, Atomese was primarily designed to allow the generalized manipulation of large networks of probabilistic data by means of rules and inferences and reasoning systems. It extends the idea of probabilistic logic networks to a generalized system for algorithmically manipulating and managing data. The current, actual Atomese design has been heavily influenced by practical experience with natural-language processing, question answering, inferencing and the specific needs of robot control.

The use of the AtomSpace, and the operation and utility of Atomese, remains a topic of ongoing research (a re-evaluation of the OpenCog architecture is currently underway. A new OpenCog system called "Hyperon" is being studied [31], introducing upgrades from Atomese to Atomese 2.0 [32], Distributed AtomSpace architecture [33, 34, 35, 36] and much more) and design experimentation, as various AI and knowledge-processing subsystems are developed. These include machine learning, natural language processing, motion control and animation, deep-learning networks and vision processing, constraint solving and planning, pattern mining and data mining, question answering and common-sense systems, and emotional and behavioral psychological systems. Each of these impose sharply conflicting requirements on the AtomSpace architecture. The AtomSpace and Atomese are the current best-effort KR system for satisfying all these various needs in an integrated way.

2.2.3 Pattern Engine and Unified Rule Engine

Three more components of the OpenCog architecture are now introduced: Pattern Matcher, Unified Rule Engine and Relex2Logic.
Each of them is necessary for the following ones, in order of listing. Unified Rule Engine is mostly built on top of the Pattern Matcher and it is applicable to rules written in a Scheme/Atomese representation, such as for Relex2Logic and PLN [trova il modo di introdurre PLN nei Values].

2.2.3.1 Pattern Matcher

OpenCog has a Pattern Matcher (PM), or Query Engine or Variable Unifier, that can be used to search the AtomSpace for specific patterns or arrangements or 'templates' of atoms. The PM can be used from C++, Scheme or Python.
After specifying some (arbitrarily complex) arrangement of atoms, that is, a hypergraph consisting of Nodes and Links of several types, the PM can find all instances of that hypergraph in the AtomSpace. The pattern or template can have "holes" in it, locations that are variable, and so the pattern matcher can act to "fill in the blanks" when presented with a pattern that

has blanks in it (better known by the expression: 'grounding' a pattern).
For example, the VariableNode type are used to indicate these "blank spots".
For a good introduction to these concepts see [37]. Although, the PM implementation is considerably more sophisticated: it unifies multiple terms at once, automatically handles unordered terms (terms that can be in any order) and provides support for quotation, execution and evaluation.

For this project, patterns are specified using QueryLink type.
One creates the pattern-template that one wishes to search for. The pattern can be specified as a collection of trees, containing the VariableNodes, making up the graph. During the search, all matching graphs are found and the groundings are noted.
The QueryLink is used for graph-rewriting, thus these grounding are then pasted into a second pattern, to create a new graph.
In Chapter [applicazione...], practical applications will be presented to help understand these concepts as well.

2.2.3.2 Unified Rule Engine

The Unified Rule Engine (URE) is a generic OpenCog rule engine operating on the AtomSpace and it can be used to implement any logic.
Two chaining modes are currently supported, Forward Chaining and Backward Chaining.
The strengths of the URE are:

- Reads/writes knowledge directly from/to the AtomSpace
- It is generic, can be used to implement any logic, even higher order logics with some limitations
- Comes with a powerful control mechanism to speed up reasoning

It was used as a first approach to solve the problem faced in this project (set out in Chapter [...]). It was later abandoned due to conceptual and practical problems in favour of a new approach, although, in the end, good ideas emerged that solved these problems. See related Section [...].
For these reasons, any other information related to URE is referred to [38] and the webpages linked to that.

2.2.4 Reasons

[39]

3
Conclusione

Bibliography

[1] Leah Davidson. *Narrow vs. general ai: What's next for artificial intelligence?* Aug. 2019. URL: https://www.springboard.com/blog/ai-machine-learning/narrow-vs-general-ai/.

[2] *The three different types of artificial intelligence – ANI, AGI AND ASI*. Oct. 2019. URL: https://www.ediweekly.com/the-three-different-types-of-artificial-intelligence-ani-agi-and-asi/.

[3] Lawtomated. *AI for Legal: Ani, AGI and ASI*. Feb. 2020. URL: https://lawtomated.medium.com/ai-for-legal-ani-agi-and-asi-cea6e7a4079e.

[4] Sean Allan. *The three tiers of AI: Automating tomorrow with Agi, ASI and ANI*. Sept. 2018. URL: https://www.aware.co.th/three-tiers-ai-automating-tomorrow-agi-asi-ani/.

[5] Astute Solutions. *Artificial narrow intelligence and the customer experience*. Nov. 2020. URL: https://astutesolutions.com/ani/artificial-narrow-intelligence.

[6] Richa Bhatia. *AGI vs ANI and understanding the path towards machine intelligence*. Jan. 2018. URL: https://analyticsindiamag.com/agi-vs-ani-understanding-the-path-towards-machine-intelligence/.

[7] Ben Goertzel. *Artificial general intelligence*. URL: https://wiki.opencog.org/w/Artificial_General_Intelligence#:~:text=types%20of%20problems.-,Overview%20of%20AGI,-edit.

[8] Cem Dilmegani. *When will singularity happen? 995 experts' opinions on AGI*. Sept. 2021. URL: https://research.aimultiple.com/artificial-general-intelligence-singularity-timing/.

[9] Mark Chen et al. "Evaluating Large Language Models Trained on Code". In: *CoRR* abs/2107.03374 (2021). arXiv: 2107.03374. URL: https://arxiv.org/abs/2107.03374.

[10] Alec Radford et al. "Learning Transferable Visual Models From Natural Language Supervision". In: *CoRR* abs/2103.00020 (2021). arXiv: 2103.00020. URL: https://arxiv.org/abs/2103.00020.

[11] Tom B. Brown et al. "Language Models are Few-Shot Learners". In: *CoRR* abs/2005.14165 (2020). arXiv: 2005.14165. URL: https://arxiv.org/abs/2005.14165.

[12] Alberto Romero. *GPT-3 - a complete overview*. May 2021. URL: https://towardsdatascience.com/gpt-3-a-complete-overview-190232eb25fd.

[13] V. Vemuri. "Main problems and issues in neural networks application". In: *1993 First New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems*. Los Alamitos, CA, USA: IEEE Computer Society, Nov. 1993, p. 226. DOI: 10.1109/ANNES.1993.323037. URL: https://doi.ieeecomputersociety.org/10.1109/ANNES.1993.323037.

[14] Christian Szegedy et al. "Intriguing properties of neural networks". In: *International Conference on Learning Representations*. 2014. URL: http://arxiv.org/abs/1312.6199.

[15] Adam Gleave et al. "Adversarial Policies: Attacking Deep Reinforcement Learning". In: *CoRR* abs/1905.10615 (2019). arXiv: 1905.10615. URL: http://arxiv.org/abs/1905.10615.

[16] Nicolas Papernot et al. "The Limitations of Deep Learning in Adversarial Settings". In: *2016 IEEE European Symposium on Security and Privacy (EuroS P)*. 2016, pp. 372–387. DOI: 10.1109/EuroSP.2016.36.

[17] Ian Goodfellow. *Attacking machine learning with adversarial examples*. Oct. 2020. URL: https://openai.com/blog/adversarial-example-research/.

[18] Ben Dickson. *The untold story of gpt-3 is the transformation of openai*. Sept. 2020. URL: https://bdtechtalks.com/2020/08/17/openai-gpt-3-commercial-ai/.

[19] Kyle Wiggers. *OpenAI's massive gpt-3 model is impressive, but size isn't everything*. May 2021. URL: https://venturebeat.com/2020/06/01/ai-machine-learning-openai-gpt-3-size-isnt-everything/.

[20] Vanessa Buhrmester, David Münch, and Michael Arens. "Analysis of Explainers of Black Box Deep Neural Networks for Computer Vision: A Survey". In: *CoRR* abs/1911.12116 (2019). arXiv: 1911.12116. URL: http://arxiv.org/abs/1911.12116.

[21] *Backpropagation networks*. URL: http://slideplayer.com/slide/6816119.

[22] Niklas Donges. *4 reasons why deep learning and neural networks aren't always the right choice*. URL: https://builtin.com/data-science/disadvantages-neural-networks.

[23] Yuan Gong and Christian Poellabauer. "An Overview of Vulnerabilities of Voice Controlled Systems". In: Mar. 2018.

[24] Ben Goertzel. "Cognitive synergy: A universal principle for feasible general intelligence". In: June 2009, pp. 464–468. DOI: 10.1109/COGINF.2009.5250694.

[25] Ben Goertzel. *The hidden pattern : a patternist philosophy of mind*. Boca Raton: Brown-Walker Press, 2006. ISBN: 1581129890.

[26] Ben Goertzel, Cassio Pennachin, and Nil Geisweiller. *Engineering General Intelligence, Part 1 - A Path to Advanced AGI via Embodied Learning and Cognitive Synergy*. Vol. 5. Atlantis Thinking Machines. Atlantis Press, 2014. ISBN: 978-94-6239-026-3. DOI: 10.2991/978-94-6239-027-0. URL: https://doi.org/10.2991/978-94-6239-027-0.

[27] Ben Goertzel, Cassio Pennachin, and Nil Geisweiller. *Engineering General Intelligence, Part 2 - The CogPrime Architecture for Integrative, Embodied AGI*. Vol. 6. Atlantis Thinking Machines. Atlantis Press, 2014. ISBN: 978-94-6239-029-4. DOI: 10.2991/978-94-6239-030-0. URL: https://doi.org/10.2991/978-94-6239-030-0.

[28] Linas Vepstas. *Graphs, metagraphs, ram, cpu*. 2020. URL: https://wiki.opencog.org/w/File:Ram-cpu.pdf.

[29] Amit Basu and Robert Blanning. *Metagraphs and Their Applications*. Jan. 2007. ISBN: 978-0-387-37233-4. DOI: 10.1007/978-0-387-37234-1.

[30] Ben Goertzel. "Folding and Unfolding on Metagraphs". In: *CoRR* abs/2012.01759 (2020). arXiv: 2012.01759. URL: https://arxiv.org/abs/2012.01759.

[31] Ben Goertzel et al. July 2020. URL: https://wiki.opencog.org/wikihome/images/7/77/OpenCog_Hyperon.pdf.

[32] Ben Goertzel. "What Kind of Programming Language Best Suits Integrative AGI?" In: *CoRR* abs/2004.05267 (2020). arXiv: 2004.05267. URL: https://arxiv.org/abs/2004.05267.

[33] July 2017. URL: https://wiki.opencog.org/wikihome/images/5/53/Distributed_Atom_Space_-_Requirements_and_Ideas_-_DRAFT.pdf.

[34] André Senna. Aug. 2018. URL: https://wiki.opencog.org/wikihome/images/f/fd/Distributed_Atom_Space_-_Architecture.pdf.

[35] Alexey Potapov. Mar. 2020. URL: https://wiki.opencog.org/wikihome/images/0/0d/Distributed_Atomspace_2.0_and_graph_DBs_differences.pdf.

[36] Alexey Potapov. Apr. 2020. URL: https://wiki.opencog.org/wikihome/images/a/ab/Pattern_Mining%2C_Indexing_and_Generalized_Hypergraph_Representation.pdf.

[37] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998. DOI: 10.1017/CBO9781139172752.

[38] Nil Geisweiller. *Unified rule engine*. Feb. 2019. URL: https://wiki.opencog.org/w/Unified_rule_engine.

[39] Ben Goertzel. *Probabilistic logic networks : a comprehensive framework for uncertain inference*. New York, NY: Springer, 2008. ISBN: 978-1441945785.