Chapter 6

From Knowledge to Reasoning, a Cognitive Perspective on Personal Knowledge Graphs

Dave Raggett[1]

Motivation

Work on personal knowledge graphs would greatly benefit from decades of progress in the cognitive sciences. Knowledge is about understanding information based upon past experience, where understanding enables reasoning and decision making. Today's applications, however, embed limited understanding in application code, as deductive logic isn't adequate for human-like reasoning. Humans are always learning and never attain perfect knowledge, and our reasoning out of necessity has to deal with uncertain, incomplete, imprecise and inconsistent knowledge. This chapter will introduce a cognitive approach to personal knowledge graphs, including plausible reasoning, System 1 for intuitive thinking, including effortlessly and rapidly generating coherent explanations, e.g., for natural language understanding, and System 2 for effortful slower deliberative thought, and how this can enable human-like memory, reasoning and decision making.

Introduction

Whilst today's knowledge graphs claim to capture knowledge there is little attention to automated reasoning with the exception of inheritance down class hierarchies. Application logic is instead embedded in application code, making it hard to understand and costly to update. Why should we continue to accept this state of affairs?

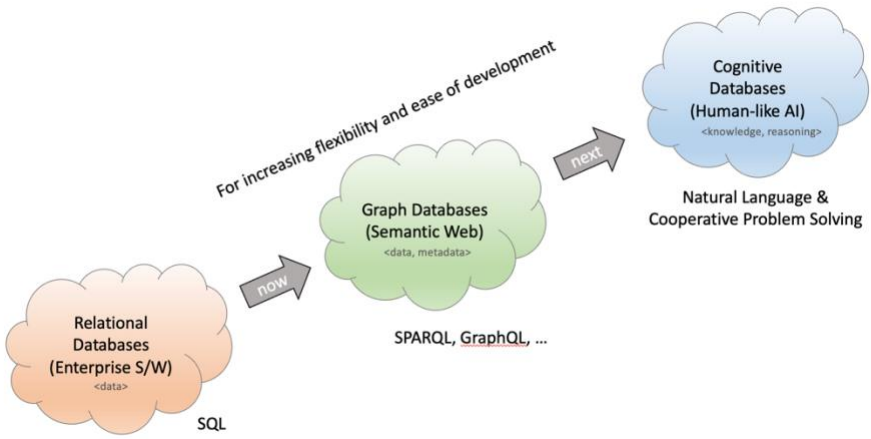
A starting point is to consider just what we mean by knowledge and its relationship to information and data. Data is essentially an unstructured collection of values, such as numbers, text strings and truth values. Information is structured data[2], such as tabular data labelled with column names. Knowledge is understanding how to

[1] W3C/ERCIM

[2] Unstructured data is often confusingly used as a term for information that doesn't follow specific data models, for example, text documents, where the structure is formed by characters, words, punctuation and paragraphs.

reason with information. Knowledge presumes reasoning and without it is just information. As such it is high time to focus on automated reasoning for human-machine cooperative work that boosts productivity and compensates for skill shortages.

The following figure depicts the evolution of databases from relational databases to graph databases, where the next stage is likely to be the emergence of cognitive databases featuring human-like reasoning. Relational databases are widely used in business, but there is growing interest in the greater flexibility of graph databases based on either RDF or Property Graphs.



Evolution of databases for greater flexibility

People have studied the principles of plausible arguments since the days of Ancient Greece, e.g., Carneades and his guidelines for effective argumentation. There has been a long line of philosophers working on this since then, including Locke, Bentham, Wigmore, Keynes, Wittgenstein, Pollock and many others.

Plausible reasoning is *everyday reasoning*, and the basis for legal, ethical and business decisions. Researchers in the 20th century were sidetracked by the seductive purity of mathematical logic, and more recently, by the amazing magic of deep learning. Traditional logic is a sterile dead end, elegant, but ultimately of limited utility! It is now time to exploit human-like plausible reasoning with imperfect knowledge for human-machine cooperative work using distributed knowledge graphs. This will enable computers to analyze, explain, justify, expand upon and argue in human-like ways.

In the real world, knowledge is distributed, imprecise and imperfect. We are learning all the time, and revising our beliefs and understanding as we interact with others. Imperfect is used here in the

sense of *uncertain*, *incomplete* and *inconsistent*. Imprecise concepts are those that lack a crisp definition, and needing to be interpreted in context, e.g., the color *red*, and a *young* person.

Conventional logic fails to cope with this challenge, and the same is true for statistical approaches, e.g., Bayesian inference, due to difficulties with gathering the required statistics. Evolution has equipped humans with the means to deal with this, though not everyone is rational, and some people lack sound judgement. Moreover, all of us are subject to various kinds of cognitive biases, as highlighted by Daniel Kahneman.

Plausible Knowledge

Consider the logical implication A ⇒ B. This means if A is true then B is true. If A is false then B may be true or false. If B is true, we still can't be sure that A is true, but if B is false then A must be false. Now consider a more concrete example: *if it is raining then it is cloudy.* This can be used in both directions: Rain is more likely if it is cloudy, likewise, if it is not raining, then it might be sunny, so it is less likely that it is cloudy, which makes use of our knowledge of weather.

In essence, plausible reasoning draws upon prior knowledge as well as on the role of analogies, and consideration of examples, including precedents. Mathematical proof is replaced by reasonable arguments, both for and against a premise, along with how these arguments are to be assessed. In court cases, arguments are laid out by the Prosecution and the Defence, the Judge decides which evidence is admissible, and the guilt is assessed by the Jury.

During the 1980's Alan Collins and co-workers developed a theory of plausible reasoning based upon recordings of how people reasoned. They discovered that:

- There are several categories of inference rules that people commonly use to answer questions.
- People weigh the evidence bearing on a question, both for and against, rather like in court proceedings.
- People are more or less certain depending on the certainty of the premises, the certainty of the inferences, and whether different inferences lead to the same or opposite conclusions.
- Facing a question for which there is an absence of directly applicable knowledge, people search for other knowledge that could help given potential inferences.

The Plausible Knowledge Notation

A convenient way to express such knowledge is the **Plausible Knowledge Notation** (PKN). This is at a higher level than RDF, and combines symbolic statements with qualitative metadata used to express certainty and conditional likelihood, etc. PKN supports a variety of statements including properties, relationships, dependencies, implications and ranges. Metadata can be provided at the end of statements as a comma separated list of name value pairs enclosed in round brackets.

Property statements declare the values of named properties, e.g.,

flowers of England includes daffodils, roses, tulips (certainty high)

Following the terminology introduced by Collins, *flowers* is the *descriptor*, i.e., the property name. *England* is the *argument*, i.e., the thing the property applies to. *includes* is the *operator*, with *excludes* as its antonym. The values are the *referents*, and either reference things or are literals such as numbers.

PKN statements optionally end with one or metadata parameters, e.g., *certainty* is an example of qualitative metadata, where the value of the parameter is from an enumerated range, e.g., *low*, *medium* and *high*.

Some concepts are context dependent, e.g., the meaning of young depends on whether you are referring to a child or an adult. The context can be stated with *for* as in the following examples:

age of young is birth, 12 for child

age of young is birth, 45 for adult

Relationship statements describe relationships between things, e.g.,

robin kind-of songbird
duck similar-to goose for habitat
duck dissimilar-to goose for neck-length
dingy is small for sailing-boat

Where *kind-of* describes the relationship between classes of things. *similar-to* indicates that one class has similar referents for a given descriptor, as named with *for*, whilst *dissimilar-to* has the opposite meaning. The *is* relationship is convenient for properties without descriptors. In this example, *small* is a term for describing the size of something relative to some context.

Relationships take the form *subject*, *type*, *obje*ct, so that in the first example above, the subject is *robin*, the relationship type is *kind-*

of and the object is *songbird*.

Dependency statements are relationships that describe a coupling between a pair of properties, e.g.,

climate depends-on latitude
current increases-with voltage
pressure decreases-with altitude

Dependencies can describe a correlation as in the case of *increases-with*, or leave that unspecified, as for *depends-on*. If two locations have similar latitude, then the first statement above implies that they should have similar climates. Dependencies are useful for qualitative reasoning about physical processes.[3]

Implication statements are a form of if-then rules with locally scoped variables, for example, *?place*, as in:

climate of ?place includes hot and
rainfall of ?place includes heavy
  implies crops of ?place includes rice

Antecedents and consequents can be properties or relations, as in:

?adult is-a adult and age of ?adult less-than 25
  implies age of ?adult is very:young

A simplification is to constrain implications to use conjunctions of antecedents. Disjunctions can then be expressed using multiple implications. Negatives can be expressed using antonyms for relationships, operators and values. Where knowledge is uncertain, an implication may be held to weakly apply even if not all of its antecedents can be established against the knowledge base. This might be justified if there is a lack of information relating to a specific antecedent so that we don't know whether or not it applies.

Range statements are a form of properties used to describe the domain of scalar values, for example, *temperature*, *age* and *guilt*, in:

range of temperature includes cold, warm, hot

range of age includes infant, child, adult for person

range of guilt includes innocent, guilty (overlap none)

Fuzzy ranges allow values to overlap as needed, e.g., a temperature value that is considered intermediate between cold and warm. This corresponds to Zadeh's fuzzy sets, e.g. (cold 0.8, warm 0.2, hot 0). By

[3] The model may depend on the state, e.g., the effect of applying heat to water varies according to whether it is in the form of ice, liquid water or steam.

specifying *overlap none*, we declare that innocent and guilty are disjoint, so that guilt can be either innocent or guilty, but not both.

Ranges enable the value of different terms in a set to be justified by different lines of argument, for example, using different implication statements, such as:

```
temperature of room is cold implies fan-speed is stop
temperature of room is warm implies fan-speed is slow
temperature of room is hot implies fan-speed is fast
```

An explicit temperature measurement can be mapped to a fuzzy set with a certainty value for each qualitative term in the range. Likewise, the inferred fuzzy set for the fan speed can be mapped back to an explicit (i.e., numeric) fan speed.

The mapping can be specified using metadata that declare the lower and upper limits for each term, e.g., for fan speed we might want to map *stop*, *slow* and *fast* to increasing fan speeds. The shape of the transfer function for each term can be made explicit with additional parameters if so needed, for instance, when using trapezoidal or smoothly changing functions.

```
range of fan-speed is stop, slow, fast
fan-speed of stop is 0
fan-speed of slow is 0, 1200
fan-speed of fast is 1000, 1800
```

Statement Metadata

Most metadata parameters are qualitative rather than quantitative to reflect their lack of precision, as in many cases we have only a rough idea and are not in a position to give detailed and accurate statistics. These parameters are used to estimate the certainty of each inference. The following parameters are defined:

- **typicality** in respect to other group members, e.g., robins are typical song birds.
- **similarity** to peers, e.g., having a similar climate.
- **strength**, **inverse** – conditional likelihood for forward and inverse inferences for a given statement, e.g., whether rain predicts cloudy weather and vice versa.
- **frequency** – the proportion of children with a given property, e.g., most species of birds can fly, but not all.
- **dominance** – the relative importance in a given group, e.g., the size of a country's economy.

- **multiplicity** – the number of items in a given range, e.g., how many different kinds of flowers grow in England, remembering that parameters are qualitative not quantitative.
- **certainty** – this can be used to indicate the degree of confidence in a given statement being true.

Plausible Inferences

Inferences provide a means to infer a property or relation that isn't explicit in the knowledge base. There are two basic approaches:

1. Using implications where all of the rule's consequents are deemed likely if all of the antecedents can be established either directly or indirectly. An example given above implies that the weather is cloudy if it can be established that the weather is rainy.
2. Using relationships to infer that a relationship or property is likely to hold for the object of the relationship, if it can be established to hold directly or indirectly for the subject of that relationship. One example, is where it is known that birds have wings, so that if you know that ducks are a kind of bird, then you can reasonably infer that ducks have wings

Implications and relationships can also be used for inverse inferences with the corresponding likelihood depending on the statement metadata. For implications, you can infer that all of the antecedents are likely to hold if one of the consequents holds, for example, it is somewhat likely to be raining if it is cloudy.

Likewise for relationships, if robins are a typical kind of bird and you know that robins sing, then it is plausible that all birds sing.

When using a relationship for an inference, the likelihood of an inferred property or relationship will depend on the type of the relationship used for the inference. Here are a few examples:

Relationship Type	*Meaning*	*Reversibility*
kind-of	subclass of class	asymmetric
is-a	instance of class	asymmetric
similar-to	similar concepts	symmetric
depends-on	coupled concepts	asymmetric
antonym-for	opposite meaning	symmetric

The reasoner can be given a property or relationship as a premise that we want to find evidence for and against, for instance, here is a premise expressed as a property:

flowers of England includes daffodils

Its inverse would be:

flowers of England excludes daffodils

The reasoner first checks if the premise is a known fact, and if not looks for other ways to gather evidence. One tactic is to *generalize the property referent*, e.g., by replacing it with a variable as in the following:

flowers of England includes ?flower

The knowledge graph provides a matching property statement:

flowers of England includes temperate-flowers

We then look for ways to relate daffodils to temperate flowers, finding the following match:

daffodils kind-of temperate-flowers

So, we have inferred that daffodils grow in England. Another tactic is to *generalize the property argument* as in the following:

flowers of ?place includes daffodils

We can then look for ways to relate England to similar countries, for example:

Netherlands similar-to England for flowers

We find then a match, for example:

flowers of Netherlands includes daffodils, tulips, roses

Thus, providing us with a second way to infer that daffodils grow in England. The certainty depends on the parameters, and the *similarity* parameter in respect to the *similar-to* relation.

The above uses a property as a premise. You can also use relations as premises, for instance:

Peter is young

The reasoner will then look for relevant knowledge on whether Peter is, or is not, young, e.g., Peter's age, and whether that implies he is a child or an adult, based upon pertinent *range* statements and associated definitions for their values.

In a small knowledge graph, it is practical to exhaustively consider all potentially relevant inferences. This becomes increasingly costly as the size of the knowledge graph increases. One way to address this challenge is to prioritize inferences that seem more certain, and to ignore those that are deemed too weak. A useful approach to implementing this is to exploit spreading activation. Further details on this will be given later on in this chapter.

The JavaScript implementation used in the web-based demo works as follows: The starting point is when the reasoner is invoked with a premise, e.g., in the form of a property with the descriptor, argument, operator, and referent. The operator and referent are optional, and can be omitted when you want to find the values.

The reasoner adds the premise as an initial goal, and iteratively looks for direct evidence in the form of property statements or indirect evidence in the form of relations, dependencies or implications. Inferences may result in queuing new goals for arguments involving multiple steps.

A check is made to see if a goal has been previously considered, in order to avoid indefinite looping behavior. This mimics human recall, in that people generally realize that they have already worked on a goal and hence, there is no point in re-doing that work.

Reasoning works backwards from the premise towards the facts. This is recorded and used in reverse to compute the certainty of inferences by working forward from the facts. Finally, the record is used to generate the explanation.

An open question is when to curtail further search, for instance, immediately after finding direct evidence. A more sophisticated approach is to curtail search based upon diminishing return on effort, analogous to people getting bored and giving up. The web-based demo provides a checkbox that allows users to see the effects of trying harder as it were. It shows that trying harder may just show permutations on what was already discovered, with marginal extra value!

Computing Certainty

The estimated certainty of inferences is computed working forward from the recorded certainty for the facts in the knowledge base. The papers by Collins et al. do not define the algorithms, so what follows should be considered as preliminary and meriting work on a more principled analysis grounded in Bayesian statistics.

The use of qualitative parameters for plausible reasoning is due to the lack of detailed statistics, and as such represents best guesses. If certainty is modelled as a number between 0 and 1, we can map qualitative values to numbers, and use algorithms for estimating the numeric certainty, which can later be mapped back to qualitative terms as needed.

One such algorithm is where a goal directly matches multiple properties with non-exclusive values, which can then be combined as a set. The more matches, the greater the certainty for establishing the

goal.

If c is the average certainty and n is the number of matches, the combined certainty is 1.0 - (1.0 - c)/n. If c = 0.5 and n = 1 we get 0.5. If n = 2, we get 0.75. If n = 4, we get 0.85, If n = 256, we get 0.998.

Collins et al. suggest a simplified approach to modelling the effects of the various metadata parameters in which each parameter boosts, weakens or has no effect on the estimated certainty, and using the same algorithm for all such parameters.

If the original certainty is zero, the parameters should have no effect, and likewise, the effect of a parameter should be smaller as the parameter's value tends to zero.

Treating the effect of a parameter as a multiplier m on the certainty, the number should be in the range 0 to 1/c, where c>0. If we want to boost c by 25% when c = 0.5, m = 1.25, but this should shrink to 1 when c = 1.

How much should m increase as c tends to 0? One idea is to use linear interpolation, i.e., 1.5 when c = 0, 1.25 when c = 0.5 and 1 when c = 1. This multiplier shrinks to 1 as the value of the parameter tends to 0 and when c tends to 0. Thus m = 1 + p/2 - p*c/2, where m is the multiplier, p is the parameter value (0 to 1) and c is the certainty (0 to 1).

We also need to deal with multiple lines of argument for and against the premise in question. If the arguments agree, we can aggregate their certainties using the first algorithm above. We are then left with a fuzzy set for the different conclusions, e.g. (true 0.8, false 0.2). Note that arguments may present multiple conclusions rather than true or false, as depending on the query posed to the reasoner.

Relationship to Fuzzy Logic

Plausible reasoning subsumes fuzzy logic as expounded by Lotfi Zadeh. Fuzzy logic includes four parts: fuzzification, fuzzy rules, fuzzy inference and defuzzification.

Fuzzification maps a numerical value, e.g., mapping a temperature value into a fuzzy set, where a given temperature could be modelled as 0% cold, 20% warm and 80% hot. This involves transfer functions for each term, and may use a linear ramp or some kind of smooth function for the upper and lower part of the term's range.

Fuzzy rules relate terms from different ranges, e.g., if it is hot, set the fan speed to fast, if it is warm, set the fan speed to slow. The rules can be applied to determine the desired fan speed as a fuzzy set, e.g., 0% stop, 20% slow and 80% fast. Defuzzification maps this back

to a numeric value.

Fuzzy logic works with fuzzy sets in a way that mimics Boolean logic in respect to the values associated with the terms in the fuzzy sets. Logical AND is mapped to selecting the minimum value, logical OR is mapped to selecting the maximum value, and logical NOT to one minus the value, assuming values are between zero and one.

Plausible reasoning expands on fuzzy logic to support a much broader range of inferences, including context dependent concepts, and the means to express fuzzy quantifiers and modifiers.

Richer Queries and Fuzzy Quantifiers

Here are some examples of richer queries:

```
all ?x where color of ?x includes red from ?x kind-of rose
few ?x where color of ?x includes yellow from ?x kind-of rose
which ?x where color of ?x includes yellow from ?x kind-of rose
most ?x where age of ?x greater-than 20 from ?x is-a person
any ?x where age of ?x less-than 15 from ?x is-a person
which ?x where ?x is-a person and age of ?x is very:old
which ?x where ?x is-a person and age of ?x slightly:younger-than 25
```

The first example searches the knowledge base to test if all roses are red. The *where* keyword is followed by a filter expressed as a conjunction of properties or relationships. The filter is applied to the set obtained by applying the criteria following the optional *from* keyword, and expressed as conjunction of properties or relationships.

In other words, we first find the set of things that are roses, and then test to see that they are all red. If the query omits *from*, the *where* filter applies to the entire knowledge base.

Traditional logic is limited to two quantifiers: *for all* and *there exists*. Plausible reasoning enables a much richer variety: *none, few most, many, all, which* and *count*. The meaning of *few*, *many* and *most* is interpreted by counting the number of elements obtained from the filter as compared to the size of the set they are drawn from. *which* returns the elements that pass the filter, whilst *count* returns the number of those elements.

The last two example queries involve modifiers that correspond to adverbs or adjectives in natural language. Multiple modifiers can be applied to a given concept, where each modifier is followed by a colon as a delimiter, e.g., *very:slightly:older-than*. This reflects Lotfi Zadeh's conception of fuzzy logic as a means to compute with words: "*small* can be multiplied by *a few* and added to *large*, or *colder* can be

added to *warmer* to get something in between."

The meaning of words is often context dependent, e.g., the same person may be considered old by a child, and young by an adult. To determine if someone is very old, we could query their age, and see where this fits in terms used for describing the age of people.

```
range of age is infant, child, adult for person
age of infant is 0, 4 for person
age of child is 5, 17 for person
age of adult is 18, age-at-death for person
```

If John is 63 and Pamela is 82, this implies they are both adults. We can then look at the terms used to describe adults.

```
range of age is young, middle-age, old, geriatric for adult
age of young is 18, 44 for adult
age of middle-age is 45, 65 for adult
age of old is 66, age-at-death for adult
age of geriatric is 78, age-at-death for adult
```

This implies that John is middle-aged and Pamela is geriatric. If we then define very old as synonymous with geriatric, we can finally infer that Pamela is very old. A complication is to how to compare a numerical age (in years) with the age at death. Essentially, a person's age increases until they die. One way to model that is as follows:

```
?person is-a person and age of ?person is ?age
    implies ?age less-or-equal age-at-death
```

We also need to define comparative concepts such as *younger than* as equivalent to *less than*, so we can determine if one person is younger than another person by comparing their respective ages. Modifiers such as *slightly* can be interpreted in respect to comparing the difference of ages with a person's age, for example if Mary is 23 and Jenny is 25, the age difference is 2 years so Mary can be considered as slightly younger than Jenny given that 2/25 is 8%.

Many such terms are inherently imprecise concepts that depend on the context and are debatable. Nonetheless, human communication abounds with imprecision, relying on the speaker and listener having roughly the same conceptual model.

Reasoning by Analogy

Analogies involve comparisons between things or objects where similarities in some respects can suggest the likelihood of similarities

in other respects, for instance, in their respective properties and relationships, or in ways to solve related problems. Gentner and Markman proposed a basis for modelling analogies in terms of establishing a structural alignment based on common relational structure for two representations, where the stronger the match, the better the analogy.

In some cases, the objects may have the same properties, in other cases, there is a systematic mapping between different properties, e.g., relating electric current to the flow of a liquid in a pipe, and correspondingly, voltage to pressure. If you know that flow increases with pressure, you can infer that current increases with voltage.

Analogical reasoning tests often take the form of A is to B as Y is to —, where students are asked to supply the missing term. This can be readily modelled using PKN, for example consider the query:

```
leaf:tree::petal:?     # leaf is to tree as petal is to what?
```

when applied to the knowledge:

```
leaf part-of tree
petal part-of flower
```

The reasoner finds that *leaf* is related to *tree* via the *part-of* relationship, and can then use that to look for a *part-of* relationship involving *petal*, yielding the result *flower*. Now consider

```
like:love::dislike:?
```

when applied to the knowledge:

```
love more-than like
hate more-than dislike
```

This gives the answer *hate* by matching the object of the relationship rather than the subject. Here is a more complex example showing the output from the reasoner:

```
Premise: mansion:shack::yacht:?
Found: mansion is large for building
Found: shack is small for building
So contrast is large, small
Found: yacht is large for sailing-boat
Found: dingy is small for sailing-boat

Therefore mansion:shack::yacht:dingy
```

Analogies involve a form of inductive reasoning, and is related to learning from examples, and being able to apply past experience to a new situation based upon noticing the similarities and reasoning about

the differences. Such reasoning can be simple as in the above examples or more complicated, for instance, when involving causal modelling.

Inferences vs Model Construction

Philip Johnson-Laird (1980, 1983) argues that the best account for human reasoning is not in terms of systematic rules or inference patterns, but rather in terms of the manipulation of mental models. Alan Collins by contrast talks about plausible reasoning in terms of inference patterns. How can both approaches be combined?

One solution is to base model construction on rule execution, where rule actions update working memory with new statements. This would involve an extension of the PKN rule syntax to represent actions as consequents. Such actions can also be used as a means to invoke external behavior, for instance, to direct a robot arm to pick something up, or to communicate with another agent. A cognitive architecture for perception, cognition and action, is presented in a later section of this chapter, along with a framework for chunks and rules. However, further work is needed to identify a higher level notation for manipulating mental models and for metacognition.

Metacognition

Metacognition is reasoning about reasoning, including, reasoning about reasoning strategies, for instance, reasoning about how to solve a given problem by comparing it to previous problems and adapting their solutions to match the new context.

Metacognition is applicable to deciding when the current approach isn't working well, so that it is time to try a different approach or to give up on the problem, at least for now, and perhaps to consider a different problem relating to the same overall goal.

Metacognition is likewise relevant to managing multiple concurrent goals, when it is necessary to dynamically switch focus between them, and to resource effort on them according to their priorities. Such goals can be at many different levels of abstraction.

Non-deductive Reasoning and Imagined Contexts

There are many forms of reasoning other than reasoning deductively from facts to conclusions. Inductive reasoning considers similarities and differences across a group of things. Abductive reasoning considers which explanations best suit a set of known facts.

This may involve reasoning with causal models that can be used forwards to predict a potential future, or backwards when seeking

plausible explanations or ways to achieve a desired outcome.

A common requirement is the need to represent things that are only true in a given context, whether past, present or an imagined situation. This is also needed to implement a theory of mind in which a cognitive agent represents and reasons about the likely beliefs of other agents (human or artificial). Story telling is central to human culture, and likewise involves the need to model knowledge specific to a fictional world.

Episodic contexts are those that relate to the passage of time, describing what was happening at different times, and involving the means to support temporal reasoning. This is key to continuous learning and further details are given in a later section.

PKN in relation to RDF and LPG

RDF (the resource description framework) is a suite of standards from W3C for symbolic graphs, the Semantic Web and Linked Data. These include SPARQL for querying RDF-based graphs, OWL as an ontology language and SHACL for expressing constraints on sub-graphs as a basis for validation.

RDF is based upon triples, i.e., directed labelled graph edges. RDF has several notations including XML, Turtle, N3[4] and JSON-LD. In RDF, graph vertices are URIs, blank nodes or literals, such as text strings, dates, numbers and truth values. Blank nodes are identifiers scoped to a serialization of particular graph. URIs are globally unique identifiers, such as HTTP based universal resource locators (URLs).

Linked Data makes use of HTTP as a basis for linking to further information about an RDF identifier, along with the means to download an RDF graph or a text description as appropriate. There is a rapidly growing number of open datasets and many ontologies.

LPG (labelled property graphs) allow both vertices and edges to be associated with sets of property-values. This is convenient when you want to annotate edges. One example is where the edge represents the relationship between an employer and an employee, where you want to indicate the employee's start date, his/her role, department and so forth.

PKN is at a higher level than RDF in that each PKN statement corresponds to multiple RDF triples. A PKN relationship corresponds to a single triple, but the statement metadata complicates matters, by requiring the relationship to be reified. Here is an example by a way

[4] Notation 3 (N3) is a superset of RDF

of explanation:

```
# PKN: John knows Mary (certainty high)
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix pkn: <http://example.com/pkn/> .
_:x rdf:type rdf:Statement .
_:x rdf:subject <#John> .
_:x rdf:predicate foaf:knows .
_:x rdf:object <#Mary> .
_:x pkn:certainty pkn:high .
```

Reification replaces an RDF triple by a set of triples for the subject, predicate and object. This example introduces a blank node (_:x) and an RDF triple that indicate it represents a reified triple. The example uses three RDF namespaces, one for the core RDF syntax, one for FOAF, the friend of a friend vocabulary, and another hypothetical vocabulary for the PKN core terms. The latter is used to annotate the relationship with the qualitative measure of certainty.

A similar approach can be used to represent PKN property statements. The argument is mapped to a subject node, the descriptor to the RDF predicate, and the referent to an object node. The PKN operator, e.g., *includes*, can be declared using another triple with *pkn:operator* as its predicate. One complication is that PKN allows properties to have a comma separated list of referents. These can be mapped to RDF collections, as in the following example:

```
# PKN: flowers of Netherlands includes daffodils, tulips
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix pkn: <http://example.com/pkn/> .
_:x rdf:type rdf:Statement .
_:x rdf:subject <#Netherlands> .
_:x rdf:predicate <#flowers> .
_:x pkn:operator pkn:includes .
_:x rdf:object _:y .
_:y rdf:type rdf:List
_:y rdf:first <#daffodils> .
_:y rdf:rest _:z .
_:z rdf:first <#tulips> .
_:z rdf:rest rdf:nil .
```

In principle, PKN could be easily extended to map referents to RDF identifiers in given namespaces as a basis for automating mapping PKN to RDF. One way to do this would be to borrow the context mechanism from JSON-LD. This would allow the RDF identifiers to be declared in separate files, linked from the PKN knowledge base.

PKN can be mapped to Labelled Property Graphs by mapping PKN properties to LPG node properties, and PKN relationships to LPG relationships. The details depend on the notation for property graphs which varies from one LPG vendor to the next.

Scaling and Graphs of Overlapping Graphs

Very large knowledge graphs are difficult to deal with. For instance, when visualizing the graph, it is intimidatingly complex when zoomed out, and lacks context when zoomed in. The lack of context is also a challenge for automated reasoning. One technique to address this is to transform very large graphs into overlapping smaller graphs that model individual contexts. The transformation can be applied either statically or dynamically according to criteria of interest.

In a large enterprise, subgraphs can be used for different business functions, offering different views for different departments, and dependency tracking for old and new uses given the enduring need to support a mix of new and old applications.

Contexts are also important for temporal reasoning, for handling counterfactuals when reasoning about explanations, and for imagined situations, e.g., when reasoning about plans. Contexts are important for natural language as a basis for grouping things that commonly occur in the same context.

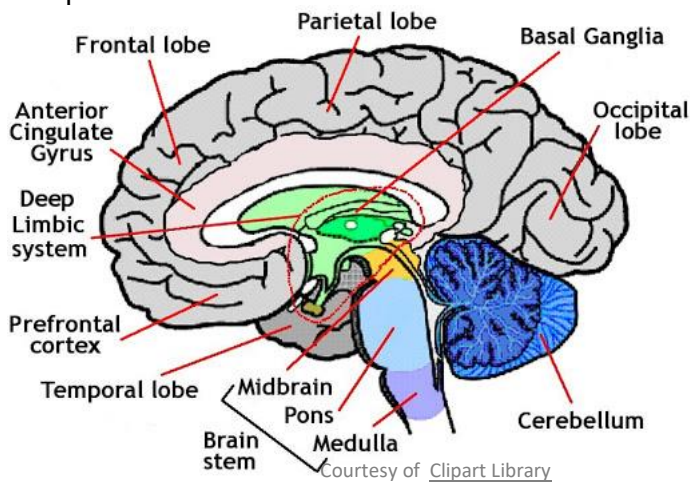
This is further related to Roger Shank's idea of scripts, which describe a stereotyped sequence of events in a given context, such as having a meal in a restaurant. Some events/actions have causal dependencies, whilst others have statistical correlations.

A complementary technique is to associate graph vertices with activations levels that are boosted when accessed or created, and otherwise decay over time, so that the activation level is a measure of how frequently a given vertex is used, mimicking the human forgetting curve. This can be combined with spreading activation as a way of priming related concepts. Each time a vertex is accessed, updated or added, a wave of activation spreads out along the connected graph edges. The more edges from a given vertex, the weaker the activation boost through each edge. A further weakening occurs at each vertex, to ensure that the wave rapidly decays.

Activation can be combined with stochastic recall to mimic other characteristics of human memory. If two edges have the same activation levels, they will be equally likely to be recalled. A further refinement is to mimic the spacing effect, which progressively reduces the boost when the time interval since the last boost is small compared to a given time window. The effect will be familiar to students in that the benefit of cramming revision into a short time is short lived compared to a more structured approach over a longer period of time.

Cognitive Architecture for Artificial Minds

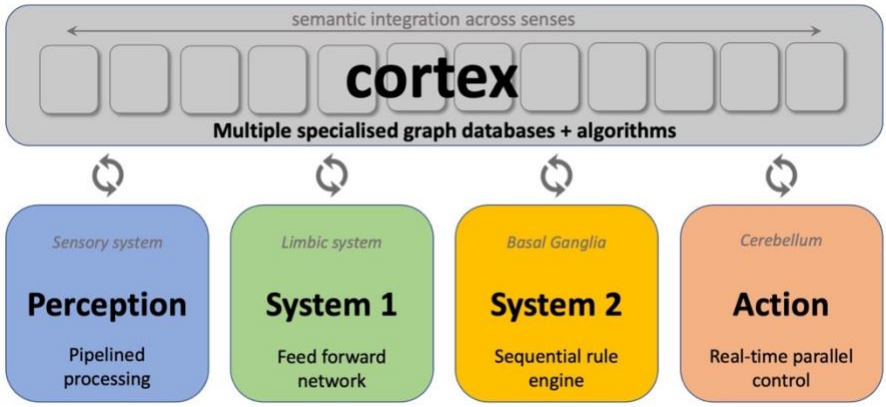
Cognitive architecture provides a functional model at a level above that of implementation choices. Consider the human brain as a source of inspiration:



Anterior temporal lobe as hub for integration across senses

This suggests the following functional architecture where the cortex is modelled as a collection of specialized graph databases along with associated algorithms that execute local to the data. The inner part of the brain is mapped to a set of cognitive circuits, analogous to blackboard systems, as each circuit has access to multiple cognitive modules in the cortex. This supports sharing of information along with invocation of cortical functions, including graph algorithms.

In the human brain, different senses are mapped to different cortical lobes. As an example, when we see a lemon, we perceive its shape, size, color, texture and smell. The Anterior Temporal Lobe acts as a hub for relating semantic multimodal models to unimodal models in the different lobes. The same hub and spoke model can be applied to semantic integration across the senses for artificial minds.



Cognitive Architecture

The architectural components have the following functions:

Cortex supports memory and parallel computation. Recall is stochastic, reflecting which memories have been found to be useful in past experience. Spreading activation and activation decay mimics human memory with semantic priming, the forgetting curve and spacing effect.

Perception interprets sensory data and places the resulting models into the cortex. Cognitive rules can set the context for perception, and direct attention as needed. Events are signaled by queuing chunks to cognitive buffers to trigger rules describing the appropriate behavior. A prioritized first-in first-out queue is used to avoid missing closely spaced events.

System 1 covers intuitive/emotional thought, cognitive control and prioritizing what's important. The limbic system provides rapid assessment of past, present and imagined situations. Emotions are perceived as positive or negative, and associated with passive or active responses, involving actual and perceived threats, goal-directed drives and soothing/nurturing behaviors.

System 2 is slower and more deliberate thought, involving sequential execution of rules to carry out particular tasks, including the means to invoke graph algorithms in the cortex, and to invoke operations involving other cognitive systems. Thought can be expressed at many different levels of abstraction, and is subject to control through metacognition, emotional drives, internal and external threats.

Action is about carrying out actions initiated under conscious control, leaving the mind free to work on other things. An example is playing

a musical instrument where muscle memory is needed to control your finger placements as thinking explicitly about each finger would be far too slow. The cerebellum provides real-time coordination of muscle activation guided by perception. Of note is the fact that the human cerebellum has over three times the number of neurons compared to the cortex, packed into a much smaller space.

System 1 and 2

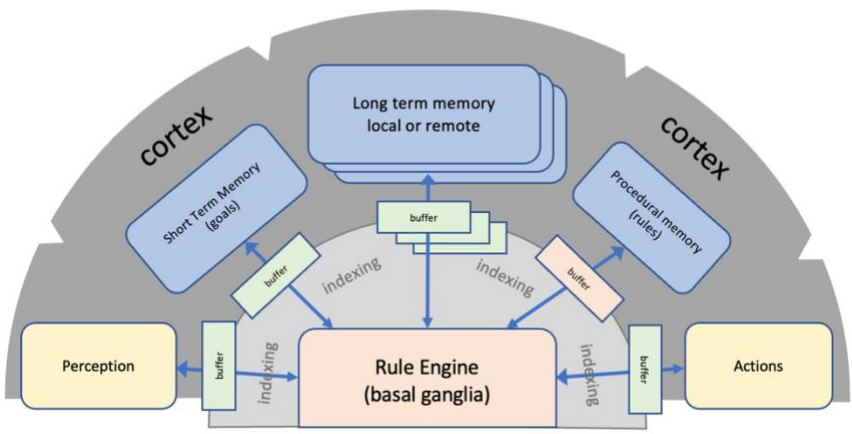
The idea of System 1 and 2 were popularized by Daniel Kahneman in his book "Thinking Fast and Slow". System 1 is fast, intuitive and apparently effortless, yet opaque as we aren't aware how we came to a conclusion. System 1 is subject to many cognitive biases and sometimes wrong.

Natural language understanding is largely handled via System 1, as we understand in real-time what people are saying by constructing a coherent explanation that hides the ambiguity of language. This involves semantic priming together with everyday knowledge that fills in the gaps in what we heard.

System 1 and 2 work in cooperation, with System 2's analysis overriding System 1's intuition as needed. System 2 is accessible to introspection and much slower. It also feels effortful, making thinking hard work and quite exhausting!

Modelling the Cortico-Basal Ganglia Circuit

The following diagram illustrates a functional model of the cortico-basal ganglia circuit as an asynchronous sequential rule engine, inspired by John Anderson's work on ACT-R:



Model of cortico-basal ganglia cortical circuit

The rule engine is connected to different cortical modules via buffers that each hold a single chunk – a collection of name/value pairs. These buffers correspond to a bundle of nerve fibers whose concurrent activation levels can be interpreted as a vector in a noisy high dimension space. Chris Eliasmith uses the term semantic pointers, and has shown how chunks can be encoded and decoded using circular convolution.

Perception dynamically updates the chunks to reflect the model of the current state of the world. Events update the chunk buffers to trigger the appropriate behavioral responses.

Procedural knowledge is encoded as collections of rules. The rule engine determines which rules match the current state of the buffers, and stochastically selects a matching rule to execute. Each rule has one or more antecedents and one or more consequents. A consequent may directly update a buffer, or do so indirectly via invoking a cortical operation on a module. These operations are asynchronous and can be loosely compared to the Web's hypertext transfer protocol (HTTP), with methods to retrieve (GET) and update data (PUT), as well as to invoke server-side operations (POST).

Chunks and Rules

The W3C Cognitive AI Community Group has devised a simple notation for chunks and rules, along with a variety of built-in operations, and an open-source implementation with a variety of web-based demos. Each chunk is associated with a chunk identifier, a chunk type and a set of name/value pairs for its properties. The chunk type is convenient way of grouping chunks, rather than having a predefined meaning.

The chunk notation minimises the need for punctuation, and you can choose between using a line break or a semicolon as a property separator. Here are examples with both conventions:

```
dog d25 {name molly; breed terrier; gender female; age 6}
dog d26 {
   name butch
   breed bulldog
   gender male
   age 3
}
```

Property values are chunk identifiers, string literals, URLs, ISO8601

dates, numbers, Booleans and comma separated sequences thereof. Note that using @ as a prefix is reserved for system identifiers. The chunk identifier (e.g., d25 in the above example) is optional and will be automatically assigned if missing.

Chunk Rules

Chunk rules are composed from chunks and have one or more conditions, and one or more actions. Conditions and actions are associated with the module they apply to, defaulting to the goal module.

```
count {state start; from ?num1; to ?num2}
  => count {state counting},
    increment {@module facts; @do get; number ?num1}
```

This example has one condition and two actions. The condition is matched to the *goal* module buffer. The first action updates that buffer changing the *start* property from *start* to *counting*. The second action applies to the *facts* module and initiates a *get* operation for the given *number*. Rule variables are prefixed with a question mark and scoped to the rule.

Module operations are invoked with *@do*. The above example invokes *get* to asynchronously retrieve a chunk that matches the properties given in the action, excluding the ones prefixed with @. The recalled chunk is put into the named buffer, triggering a fresh round of rule selection.

Both conditions and actions can use *@id* to bind to a chunk identifier and *@type* to bind to the chunk type. The following actions are built-in, with *update* as the default action:

- **@do update** to directly update the module's buffer
- **@do queue** to push a chunk to the queue for the module
- **@do clear** to clear the module's buffer and pop the queue
- **@do get** to recall a chunk with matching type and properties
- **@do put** to save the buffer as a new chunk in the module's graph
- **@do patch** to use the buffer to patch chunk in the module's graph
- **@do delete** to forget chunks with matching type and properties
- **@do next** to load the next matching chunk in an implementation dependent order
- **@do properties** to iterate over the set of properties in a buffer
- **@for** to iterate over the items in a comma separated list

Applications may define additional operations, e.g., for controlling a robot arm. Apart from *clear*, *update* and *queue*, all actions are asynchronous, and when complete set the buffer status to reflect their

outcome. Rules can query the status using *@status*. The value can be *pending*, *okay*, *forbidden*, *nomatch* and *failed*.

This is loosely analogous to the hypertext transfer protocol (HTTP) and allows rule engines to work with remote cognitive databases. To relate particular request and response pairs, use *@tag* in the action to pass an identifier to the subsequent asynchronous response where it can be accessed via *@tag* in a rule condition.

You can define rules that match a buffer when a module operation hasn't succeeded. To do this place an exclamation mark before the chunk type of the condition.

Actions can be used in combination with *@id* to specify the chunk identifier, for instance, to recall a chunk with a given identifier. Additional operations are supported for operations over property values that are comma separated lists of items, see below. The default action is *@do update*, which just updates the properties given in the action, leaving another properties in the buffer unchanged. You can use *@do clear* to clear all properties in the buffer.

Whilst *@do update* allows you to switch to a new goal, sometimes you want rules to propose multiple sub-goals. You can set a sub-goal using *@do queue* which pushes the chunk specified by an action to the queue for the module's buffer.

You can use *@priority* to specify the priority as an integer in the range 1 to 10 with 10 the highest priority. The default priority is 5. The buffer is automatically cleared (*@do clear*) when none of the buffers matched in a rule have been updated by that rule. This pops the queue if it is not already empty.

Actions that directly update the buffer do so in the order that the action appears in the rule. In other words, if multiple actions update the same property, the property will have the value set by the last such action.

The *@do get* action copies the chunk into the buffer. Changing the values of properties in the buffer won't alter the graph until you use *@do put* or *@do patch* to save the buffer to the graph. Put creates a new chunk, or completely overwrites an existing one with the same identifier as set with *@id*. Patch, by contrast will just overwrite the properties designated in the action.

Applications can define additional operations when initialising a module. This is used in the example demos, e.g., to allow rules to command a robot to move its arm, by passing it the desired position and direction of the robot's hand. Operations can be defined to allow messages to be spoken aloud or to support complex graph algorithms, e.g., for data analytics and machine learning. Applications cannot

replace the built-in actions listed above.

Note if you add to, or remove matching chunks during an iteration, then you are not guaranteed to visit all matching chunks. A further consideration is that chunks are associated with statistical weights reflecting their expected utility based upon past experience. Chunks that are very rarely used may become inaccessible.

Iteration over properties

You can iterate over each of the properties in a buffer by using *@do properties* in an action for that buffer. The following example first sets the facts buffer to *foo {a 1; c 2}* and then initiates an iteration over all of the buffer's properties that don't begin with '@':

```
run {}  =>
   foo {@module facts; a 1; c 2}, # set facts buffer to foo {a 1; c 2}
   bar {@module facts; @do properties; step 8; @to goal} # launch
iteration
# this rule is invoked with the name and value for each property
# note that 'step 8' is copied over from the initiating chunk
bar {step 8; name ?name; value ?value}  =>
   console {@do log; message ?name, is, ?value},
   bar {@do next}  # to load the next instance from the iteration
```

Each property is mapped to a new chunk with the same type as the action (in this case *bar*). The action's properties are copied over (in this example *step 8*), and *name* and *value* properties are used to pass the property name and value respectively. The *@more* property is given the value *true* unless this is the final chunk in the iteration, in which case *@more* is given the value *false*.

By default, the iteration is written to the same module's buffer as designated by the action that initiated it. However, you can designate a different module with the *@to* property. In the example, this is used to direct the iteration to the goal buffer. By setting additional properties in the initiating action, you can ensure that the rules used to process the property name and value are distinct from other such iterations.

Operations on comma separated lists

You can iterate over the values in a comma separated list with the *@for*. This has the effect of loading the module's buffer with the first item in the list. You can optionally specify the index range with *@from* and *@to*, where the first item in the list has index 0, just

like JavaScript.

```
# a chunk in the facts module
person {name Wendy; friends Michael, Suzy, Janet, John}
# after having recalled the person chunk, the
# following rule iterates over the friends
person {@module facts; friends ?friends}
  => item {@module goal; @for ?friends; @from 1; @to 2}
```

which will iterate over Suzy and Janet, updating the module buffer by setting properties for the item's value and its index, e.g.

```
item {value Suzy; @index 1; @more true}
```

The action's properties are copied over apart from those starting with an '@'. The item index in the list is copied into the chunk as *@index*. You can then use *@do next* in an action to load the next item into the buffer. The *@more* property is set to *true* in the buffer if there is more to come, and *false* for the last property in the iteration. Action chunks should use either *@do* or *@for*, but not both. Neither implies *@do update*.

You can append a value to a property using *@push* with the value, and *@to* with the name of the property, e.g.

```
person {name Wendy} => person {@push Emma; @to friends}
```

which will push Emma to the end of the list of friends in the goal buffer.

```
person {name Wendy} => person {@pop friends; @to ?friend}
```

will pop the last item in the list of friends to the variable *?friend*. Similarly, you can prepend a value to a property using *@unshift* with the value, and *@to* with the name of the property, e.g.

```
person {name Wendy} => person {@unshift Emma; @to friends}
```

will push Emma to the start of the list of friends in the goal buffer.

```
person {name Wendy} => person {@shift friends; @to ?friend}
```

will pop the first item in the list of friends to the variable *?friend*.

Named Contexts

It is sometimes necessary to represent knowledge that is only true in a specific context, for example, when modelling another agent's knowledge, or when reasoning about counterfactuals during abductive reasoning. This is supported using *@context*. In its absence, the default context applies. Here is an example adapted from John Sowa.

```
# Tom believes that Mary wants to marry a sailor
believes s1 {@subject Tom; proposition s2}
wants s3 {@context s2; person Mary; situation s4}
married-to s5 {@context s4; @subject Mary; @object s6}
a s6 {@context s4; isa person; profession sailor}
```

More complex queries

Modules may provide support for more complex queries that are specified as chunks in the module's graph, and either apply an operation to matching chunks, or generate a result set of chunks in the graph and pass this to the module buffer for rules to iterate over. In this manner, chunk rules can have access to complex queries capable of set operations over many chunks, analogous to RDF's SPARQL query language. The specification of such a chunk query language is left to future work, and could build upon existing work on representing SPARQL queries directly in RDF.

A further opportunity would be to explore queries and rules where the conditions are expressed in terms of augmented transition networks (ATNs), which loosely speaking are analogous to RDF's SHACL graph constraint language. ATNs were developed in the 1970's for use with natural language and lend themselves to simple graphical representations. This has potential for rules that apply transformations to sets of sub-graphs rather than individual chunks.

Natural Language

Natural language will be key to enabling flexible dialogs between cognitive agents and humans, including commands, handling questions, providing answers, understanding and learning from text-based resources. The field of computational linguistics focuses on text processing, but not on representing and reasoning with the meaning of language. Traditional logic is inadequate when it comes to the flexibility of natural language, including uncertainty, imprecision and context sensitivity. This suggests that different approaches are needed that support plausible reasoning.

Large Language Models

Large language models such as GPT-3 and BLOOM are based upon neural network models trained to predict masked words using a vast corpus of text documents. BLOOM’s language model has 176 billion parameters covering 46 human languages. The model maps the

user supplied text to an opaque internal model of the meaning, referred to as the *latent semantics*. This can then be used stochastically to generate text continuations consistent with those semantics.

Large language models are surprisingly good at this, and can effectively mimic a wide range of topics and styles of text, e.g., BLOOM generated the italicized text below following the user supplied prompt shown in regular text:

John picked up his umbrella before stepping out of the door as he had heard the weather forecast on the radio *in the living room that said that heavy rain was forecasted and that there was a high chance of thunderstorms throughout the afternoon. The rain had become a little less torrential as he made his way to his sister's house, so that he didn't feel too drenched when he arrived*.

Large language models show that machine learning can be successfully applied to learning to generate latent representations of the meaning of everyday language, and that these representations can be used in reverse to generate plausible text continuations.

Unfortunately, we have yet to discover how to mimic human-like reasoning in terms of operations applied directly to the latent semantics embodied in large language models. Similarly, we can't yet map the latent semantics to symbolic graphs as another way to implement reasoning. Both challenges are exciting opportunities for research studies.

Natural Language and Common Sense

Another approach to natural language understanding is to combine conventional parsing with symbolic representations of meaning such as the plausible knowledge notation (PKN). Parsing isn't particularly difficult, provided that dealing with the ambiguity of natural language is largely delegated to semantic processing.

This involves tasks such as selecting the most appropriate sense of each word according to the context in which it appears, figuring out how to correctly attach prepositional phrases, creating semantic models and resolving references.

Finding a semantically coherent model of a given text utterance is challenging, despite being apparently effortless for humans. A promising approach is to combine spreading activation with models of everyday knowledge, as a basis for understanding what's implicit in the utterance.

As an example, consider the sentence "John opened the bottle and poured the wine." Human readers know that John is a male adult, and that the occasion is likely to be social event such as a dinner or a

party. The wine is a liquid for the guests to drink from their glasses. The bottle needs to be opened before it can be poured into the glasses. Pouring transfers liquid from one container to another, conserving the volume of liquid as it does so.

Human readers are aware of all of this, and don't need to spend effort consciously reasoning it out unless that becomes necessary, nonetheless, the background knowledge is key to efficiently finding a coherent model of the utterance.

Further work is now needed to look at how this can be mimicked by cognitive agents. This can start from a small set of examples where the background knowledge and typical inferences can be developed with a modest level of effort.

Natural language generation involves an understanding of aims of the communication, the knowledge that the listener or reader is likely to already have, and how to minimize what needs to be said. Grice's maxims describe the principles of cooperative dialogue in terms of quantity, quality, relation and manner.

- Try to be informative and give as much information as needed, but no more.
- Try to be truthful and avoid giving information that is false or not supported by the evidence.
- Try to be relevant and say things pertinent to the discussion.
- Try to be clear and orderly, avoiding obscurity and ambiguity.

How can cognitive agents be designed to implement these maxims? It is likely that cognitive agents can be trained to be effective by using machine learning techniques that pit a generator against an adversary that seeks to distinguish machine generated from human generated utterances in respect to a large training corpus of short dialogs.

Metaphors in Everyday Language

Lackoff and Johnson in their book "Metaphors we live by" (1980) showed just how much humans rely on metaphor in everyday thought and language. People will likewise expect cognitive agents to understand and to use metaphors in human-agent dialogs.

One such metaphor is ideas as food, e.g., *raw facts*, and *half-baked ideas*. People use many different metaphors, and much of this is culture dependent, i.e., learned from what we read and listen to. In principle, cognitive agents should be able to do likewise, understanding previously unheard metaphors in terms of reasoning about analogies. This is an example of continuous learning.

Continuous Learning

Many AI systems are trained up front, and have difficulties in coping when the statistics of the run-time data diverge from that of the training data. Continuous learning is a way to adapt as the data changes, and will be essential for resilient operation of cognitive agents.

There are different ways to learn. Syntagmatic learning deals with patterns in co-occurrence statistics. Paradigmatic learning deals with taxonomic abstractions. Skill compilation is the process of speeding reasoning by exploiting experience with previous solutions as a short cut, including the use of analogies. Meta-learning is the process of learning how to learn, including reasoning about strategies and tactics for how to reason.

Learning can be from direct experience by interacting with the world, and exploiting a sense of curiosity about how things work. However, there are issues of safety and cost for applying this in practice, e.g., with allowing robots to roam around freely. A work around is to use virtual worlds, but it is challenging to make those a sufficiently faithful representation of the real world.

Another approach is learning from observations and asking questions. Children are incredibly good at this. There is a huge opportunity to learn from large corpora of text documents, images and videos.

Finally, learning can be arranged in the form of lessons and assessments, i.e., courses designed specifically for AI agents. Enabling AI agents to devise their own knowledge representations as part of machine learning is likely to more scalable than direct hand authoring of knowledge, see the later section on scalable knowledge engineering.

Hive Minds, Knowledge Caching and Swarm Intelligence

Cognitive agents can be designed using shared access to remote cognitive databases, analogous to the lobes in the cerebral cortex. By duplicating agents in this way, as each agent learns, the updated knowledge and skills are immediately available to all of the agents, contributing to accelerated communal learning, in what can be thought of as a form of hive mind. Existing knowledge often helps when it comes to learning new knowledge, so it is likely that the speed up will be greater than linear in respect to the number of agents involved.

Resilience in the face of disrupted connectivity to remote cognitive databases can be achieved via mechanisms for selectively duplicating frequently used knowledge to local storage, as a form of

knowledge caching. This would come with the need for synching local updates to the cache with the remote cognitive databases when connectivity resumes.

In the face of the unexpected, a diversity of personalities, opinions, knowledge and skills can be invaluable. Rather than designing a hive of agents with identical clone minds, it makes much more sense to design the hive mind as a collective with shared knowledge and awareness, forming a collective consciousness, whilst allowing a diversity of different skills and perspectives to be deployed as appropriate to the challenges at hand. This would also allow human users to select the personality of the agents they interact with. The collective mind would be engineered to enforce confidentiality in respect to personal privacy.

This can be contrasted with swarm intelligence featuring the collective behavior of decentralized, self-organized agents. Swarm intelligence typically involves identical agents applying relatively simple rules, e.g., to prevent a collection of airborne drones from crashing into each other, and enabling them to fly around obstacles. In principle, the two approaches can be used together, where swarm intelligence is a comparatively low-level solution under the control of the higher-level collective mind.

Complementary Role of Artificial Neural Networks

Deep learning over large corpora has produced very impressive results. Text to image generators such as Stable Diffusion and DALL-E are trained against many millions of images associated with short text descriptions. The following example shows that this approach has captured knowledge about marble, the effects of light and shadow, the human form, and the styles of famous sculptors.

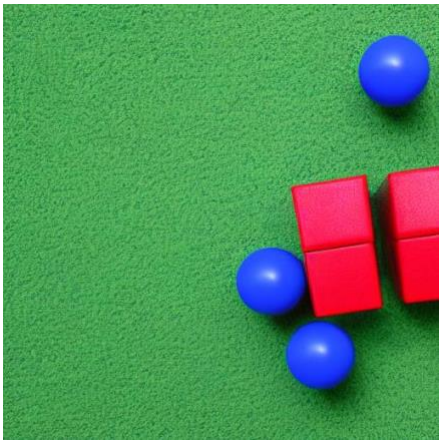
Marble sculpture of a young woman generated using Stable Diffusion

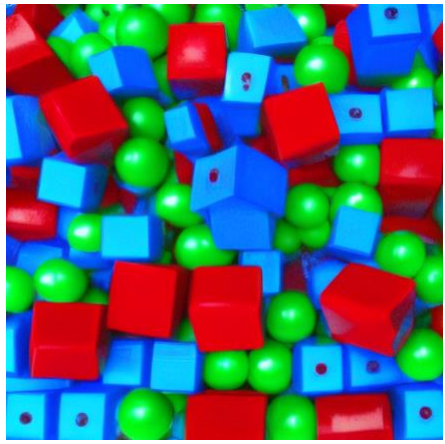
The breadth of the training data is reflected by the ability to generate images in a very wide range of styles. However, there are some obvious limitations.

Painting of a woman generated with Stable Diffusion

The woman's hand has too many fingers! This reflects weaker knowledge about hands and fingers compared to that of the human face.

Another example is when the prompt is "one blue ball and two red cubes on a green floor". Stable Diffusion clearly lacks an understanding of how to count instances of things, as can be seen in the two examples below generated using that prompt.





The same applies to the BLOOM large language model as can be seen with the following prompt where BLOOM's text is in italics.

There is one blue ball, two green balls and five red balls. How many balls are there in total? *The first possible answer is '7' because there is one blue ball and six other balls. The second possible answer is '4', because the first ball is blue, two of the other six balls are green and the remaining four balls are red.*

Together, this suggests a huge potential for combining deep learning with techniques that support human-like reasoning and learning. Richer semantic knowledge would allow the software to understand text documents and images in the training set at a much deeper level, enabling faster learning from smaller data sets.

It would also enable artists to work collaboratively with AI's to iteratively improve a composition through suggestions on changes to particular aspects. One opportunity would be for generating richer experiences in the virtual reality Metaverse by exploiting learning from a vast corpora of text documents, images and videos.

More generally, AI's should be able to learn everyday knowledge and reasoning from understanding such corpora, just as young children do as they observe the world around them, applying their prior knowledge to reason about possible explanations, and by asking questions.

Scalable Knowledge Engineering

Hand crafted knowledge doesn't scale and is brittle when it comes to the unexpected, i.e., things that the developers haven't anticipated and designed for. Deep learning scales very effectively, but is similarly brittle, and requires vast datasets for training.

Humans are much better at generalizing from a few examples by

seeking causal explanations based upon prior knowledge. Humans are also good at reasoning using mental models and chains of plausible inferences, supported by metacognition.

We need research focused on extending artificial neural networks to support human-like learning and reasoning. At the same time, we should explore scalability of machine learning for symbolic representations of knowledge as a complementary technology, moreover, hand authoring for small scale experiments can help illustrate what's needed from more scalable approaches.

An open question is how to apply artificial neural networks to human-like reasoning and learning. It is likely that vector-space representations will prove to be very effective in respect to handling imprecise and context dependent concepts.

Application to Privacy centered personal assistants

Today's consumer Web is dominated by advertising-based business models that have a strong focus on gathering personal information for metrics and for targeting advertising through live auctions for space on Web pages. Consumers are habituated to clicking away annoying permission requests for enabling tracking.

It is time to make privacy a central part of ecosystems of services. One way to realize this is with personal assistants that act on their user's behalf in respect to providing services using ecosystems of third-party providers. This would support a privacy centered evolution away from dominance by Web search engines.

Personal assistants acting in this role can be thought of as digital guardian angels, and designed to apply and safeguard their user's values as learned from observing their behavior via privacy-protecting federated machine learning. Personal assistants select services matching their user's requests by using service metadata plus independent trust attestations and live auctions.

For this to work, the business models need to align costs and benefits for implementing and operating personal assistants, as well as for attestations based upon aggregating feedback from both users and personal assistants.

Personal assistants share pertinent personal information with the selected services, e.g., their user's travel plans and preferences, when seeking proposals for flights, hotels, local travel, restaurants, museums, etc. Some services are immediate, whilst others may take significant time to fulfill, and rely on smart notifications to alert users when ready.

Personal information is shared subject to contractual terms and conditions. In principle, this can include the role of personal assistants in downstream checks and permissions. This gets more complicated as data is progressively transformed and merged with other sources of information.

Practical personal assistants will be reliant on advances in human-like reasoning with everyday knowledge, as well as advances in natural language understanding and generation.

Summary

Plausible reasoning is a major paradigm shift for knowledge graphs, in that it embraces the uncertainty, imprecision, context sensitivity and inconsistencies in everyday knowledge and use of natural language. This addresses a key challenge for human-agent collaborative work in enabling the use of natural language dialogs.

This chapter has presented two contrasting approaches, one focusing on plausible inferences inspired by Allan Collins, and the other on a procedural approach inspired by John Anderson. Further work is needed on how metacognition can be used to combine reasoning with mental models as per Philip Johnson-Laird, and logical reasoning based upon plausible inferences.

Recent successes with large language models and image generation are very impressive, and demonstrate the practicality for applying machine learning to latent semantics, in essence, automating knowledge engineering. This is dramatically more scalable than hand authoring of knowledge graphs.

Open AI's ChatGPT is a large language model that supports follow-up questions, and can challenge incorrect premises and reject inappropriate requests. Nonetheless, it lacks continuous learning, and is limited to what was in its training dataset.

Google's Minerva is a large language model that was further trained on technical datasets. It correctly answers around a third of undergraduate level problems involving quantitative reasoning. However, it lacks a means to verify the correctness of the proposed solutions, as it is limited to intuitive reasoning.

New work is needed that operates directly on latent semantics to support plausible reasoning and model building, combining intuitive reasoning with deliberative analytic reasoning. In principle, this will enable cognitive agents to learn more efficiently by understanding training examples at a deeper level. A related challenge is to enable general purpose agents rather than agents limited to a single domain of

competence.

Work is also needed on symbolic graphs as a complementary approach to neural networks when it comes to scalable knowledge engineering, based upon advances in machine learning to circumvent the bottleneck of handcrafted knowledge. It is conceivable that this may offer benefits in respect to avoiding the huge energy costs involved in applying deep learning to vast corpora.

These advances will pave the way for developing digital guardian angels that are better at identifying inappropriate content, including fake news and social media posts that violate the terms of use. Guardian angels have plenty of other applications including enabling privacy-centric ecosystems of services, managing auctions for service providers and consumers, and orchestrating resources across the computing continuum from the far-edge to the cloud.

More generally, the emergence of cognitive agents that mimic human-reasoning and learning will help to boost productivity and counter the shrinking work force associated with ageing human populations across many countries.

The next generation of AI will probably seem like science fiction, and will grow to embrace intellectual and artistic skills better than most humans, based upon learning from vast corpora of text, images and videos, plus foundational courses designed for AIs and interaction with billions of people.

This should enable economies to break free of constraints on growth associated with a limited work force, but at the same time will necessitate attention to distributing benefits fairly across society and countering the risk of monopolistic control of AI technology and its exploitation.

Further Reading

Here are some suggestions if you want to read more.

Argument and Argumentation, Stanford Encyclopedia of Philosophy

Core theory of plausible reasoning, Allan Collins and Ryszard Michalski, Cognitive Science Volume 13, Issue 1, 1989

"Thinking Fast and Slow", Daniel Kahneman, 2011, see excerpt "Of 2 Minds: How Fast and Slow Thinking Shape Perception and Choice", Scientific American, June 15, 2012.

Lotfi Zadeh and the birth of Fuzzy Logic, IEEE Spectrum, June 1995

John R. Anderson – Biography, American Psychologist, April 1995,

see also the ACT-R Website, hosted by CMU

"Metaphors we live by", George Lackoff and Mark Johnson, University of Chicago Press, 1980, pp 276

Structure Mapping in Analogy and Similarity, Dedre Gentner and Arthur Markman, 1997, *American Psychologist*, 52 (1), 45–56

"Mental models or formal rules", Philipp Johnson-Laird and Ruth Byrne, 1993, *Behavioral and Brain Sciences* 16 (2), 368-380

"How to build a brain: A neural architecture for biological cognition", Chris Eliasmith, 2013, Oxford University Press

What we learn by doing, Roger Shank, 1995, Institute for the Learning Sciences Northwestern University

W3C Cognitive AI Community Group, which developed the Chunks & Rules specification, along with a suite of web-based demos.

Web-based demo for plausible reasoning and argumentation, Dave Raggett, 2022

Stable Diffusion web-based demo by Hugging Face

BLOOM (BigScience Large Open-science Open-access Multilingual Language Model) hosted by Hugging Face

Minerva, a large language model pretrained on general natural language data and further trained on technical content, by Google Research

ChatGPT, a conversational agent by Open AI

DALL·E 2, image generator by Open AI

The Semantic Web & Linked Data, Ruben Verborgh, Ghent University
Metadata and Discovery, University of Pittsburgh