**A natural language understanding approach to AGI**

Zoltan Foris, [foris64@gmail.com](mailto:foris64@gmail.com)

**Motivation**

We try to build a software system with thinking capabilities equivalent to human-level intelligence. Our final goal is a system that is able to *learn* just like people (often from a single example or instruction), *adapt* to new tasks like people, produce output based on *insufficient knowledge* and computing *resource*, possess cognitive capabilities present in children early in development (*common sense*), be able to *re-use* anything it learned like people when facing a new task.

We selected natural language as single input and output. Obviously, this is an ambitious selection. But while being ambitious, it makes a cruicial part of the project much easier: feeding the system with information to be learned (including common sense knowledge), is by far the least resource intensive if the information is provided in natural language – especially because we aim to cover **all** domains of language use relevant at the age of 6.

Our current goal is to implement a system with text comprehension capabilities comparable with a 6-years-old child, to be tested by the ability of the system to learn from information provided in natural language, and answer questions in natural language. This should be benchmarked against children`s performance, in all domains relevant at this age. Such system cannot work without a multi-domain common sense knowledge (intuitive theories, cognitive capabilities present early in children), which is fed to the system in natural language. The first version of this system is ready, it can understand many kinds of English sentences and answer questions, and it can be expanded to cover all types of language use at the age of 6.

The idea to proceed through phases of child development is very similar to the one in (*B. Goertzel,**S.V. Bugaj, AGI preschool: A Framework for Evaluating Early-Stage Human-like AGIs* in *AGI-2009*), but narrowed to natural language input/output.

**Summary of results**

The achievements delivered in this project so far:

1. The implementation of the concept network, which is a general-purpose knowledge representation methodology.
2. A language understanding and question-answering application written in Python. This works on top of the concept network, is capable of extracting knowledge from natural language (English) text and answering questions.
3. Demonstration how common sense knowledge can be added to the system using natural language English sentences. A methodology to measure gradual progress towards the goal of human level natural language usage.

**Our approach to Artificial General Intelligence**

We think this project is an effort towards artificial general intelligence. We see a few advantages of our aproach, and a number of challeges as well. Advantages over some other AI/AGI approaches include:

1. We have a well-defined way to measure progress. The final goal is language understanding on adult level, as measured by **standard text comprehension tests designed for humans**. The way we will measure progress is text comprehension tests designed for 6,7,8, … years old children, evaluated against human performance. Within each milestone, a more granual measurement is also available, see the next section.
2. The final goal is widely recognized as a major step towards AGI among the general public, which is important to gain credibility and funding for the field.
3. The final goal, if realized, is very valuable commercially. Literally, millions of jobs can be filled by such an AGI, even if it is limited to langauge understanding and production.
4. The project is a rather limited version of AGI. There is no visual module, no body, no need for a real or virtual environment etc. This makes it more plausible to achieve success, while opens the door to use the results in an integrated AGI approach.
5. The project is broken down into small steps (stepping from 6 years performance to 7 etc).
6. We have a viable vision, with a demonstration already working, how to provide common sense knowledge to the system, with much less effort than other projects, like Cyc. The amount of common sense (and any kind of special) knowledge fed to the system may be large, but the effort is still manageable, as we can reuse much of the existing natural language texts humans have produced in the world so far.
7. The project can become a useful component of an integrative AGI system, like Novamente (see *Artificial General Intelligence (editor B. Goertzel, Springer 2007), pages 77-127).*

Challenges:

1. The project might be considered to be too difficult; the first step, language understanding and production on 6-years level, too ambitious. We think this is a valid concern, and we mitigate the risk associated with this issue by a granular measurement of progress described below.
2. The project may work for the 6-years level, but may become unfeasible to make further progress up to the adult level. Some narrow AI techniques may ensure that the project delivers results on the lower levels but these will fail to scale up. The measurements may not be a good evidence of real progress, because the results might be attributable to more simple, narrow techniques, not to a genuine progress towards AGI. Our response to this is that this is a valid concern, and researchers should pay attention to avoid this trap (if possible). The measurement described below may help in this.

**Measurement of progress**

Within a milestone (like stepping from 6-years to 7-years performance), and especially within the first big milestone to achieve 6-years performance level, we propose a granular measurement that indicates incremental progress fairly well.

The idea is very simple. We take a piece of input text, like the one at the end of this document in the Appendix. The input has sentences (grouped in paragraphs) to provide information for the system, and a few questions for testing. A *development round* is the following: The developer feeds a few paragraphs and related questions to the system (on top of all the information covered in earlier rounds), and checks the answers. If the answers are acceptable, she moves on to the next round. If some answers are bad, she finds the reasons, fixes them and moves on. The developer keeps track of each development round, how many sentences and questions it involved, and how much effort was needed to have this round completed. Currently, a round of 10 sentences and 5 questions needs 2-3 weeks of work to be completed. Incremental progress means that the effort for one round gets always smaller as more and more rounds are already completed.

Within a development round, to make the system work, these are the typical items the developer may need to add to the system:

1. Rules to translate natural language to the internal representation.
2. Additional input to make up for gaps in common sense knowledge.
3. New reasoning rules.
4. Python code to create or enhance some cognitive function.

Approximately 10 rounds (each having 10-20 sentences and questions) have been completed so far with 40 weeks effort (a single developer). As more and more rounds are completed, the information in all these rounds are cumulated in the system, rules needed for translation cover more and more sentences, cognitive functions get richer and richer.

After another 50-100 rounds we hope that the effort per round will decrease to 1-2 days (with many rounds not requiring any effort). The need for new reasoning rules and new python code should gradually go away. As the input approximates to cover all relevant domains, the effort per round is expected to decrease to human level. If this pattern is observed throughout the next 100 rounds, it is a reassuring indication that the 6-years performance level is achievable. The total effort needed depends very much on the total amount of input necessary (an optimistic estimation of total input is a few thousand pages). However, it is possible that this measurement will show very promising progress well before the total input is fed to the system.

The above development methodology does not seem to be a perfect fit when it comes to adding new cognitive capabilities and new reasoning rules, because it does not follow a predefined master plan. Therefore it is necessary to follow a high level design guideline, which we think can be obtained from the high level design of the Novamente system (see *Artificial General Intelligence (editor B. Goertzel, Springer 2007), pages 77-127).*

**Non-Axiomatic Reasoning System**

The system we implemented can be considered a “Non-Aximatic Reasoning System” (NARS) as described by P. Wang in *Artificial General Intelligence (editor B. Goertzel, Springer 2007).* Our project`s main achievement is a system which can be characterized by the key NARS attributes:

1. The system consumes new knowledge (provided in natural language) and updates existing knowledge in the context of new information.
2. All levels of knowledge is allowed to be unreliable and ambigous. There is no “true” knowledge, only stuff considered more or less “normal” or “probable”.
3. There is no guarantee that the rules are correct, complete, or sufficient to answer the questions. In case of insufficient rules/knowledge the system may still produce some response.
4. The system is capable to respond with little use of resources (memory, time) by ignoring a (potentially large) portion of rules and knowledge that might be applicable to the question. The system is also capable to invest more resource into a better response.

Here we note that using natural language as input creates an interesting situation. Natural language is the code by which humans most efficiently transfer information from one mind to the other, and by doing so, they explicitly encode a large amount of information. Using it as input for a software system means we have the opportunity to gain information from the input on two levels:

1. The system can “learn” by recognizing regularities in the input and generalizing them For example, the input given is the following: “Birds are animals. The eagle is a bird. The eagle has wings. The sparrow is an animal. The sparrow has wings.” The system may learn (with some uncertainty) that the sparrow is a bird. This is similar to an image recognition task.
2. The system can “understand” the information coded in the input and use it (like a child who takes a lesson on the sparrow). For example, the input given is the following: “The sparrow is a bird.” The system may learn (with some uncertainty) that the sparrow is a bird.

We have both capability. The second use of natural language (between people) often carries more information than the first one, plus this second use is a prerequisite for the first. Still, the first use is also very important and implementing it is inevitable (it gives rise to induction, abduction, analogical reasoning, using and understanding metaphors etc).

**The concept network representation (mentalese)**

Let us continue with the description of the representation used in our system. We use the term “concept network” or “mentalese” as synonyms. The concept network representation is based on a single data type, the *concept.*

By the term *concept* we understand the following:

* On one hand, the meaning of a word or an idiomatic phrase in English is a concept
* On the other hand, one or more concepts connected with a *relation* also result a concept. The concepts which are connected are the **parent concepts** of the emerging new concept. The emerging new concept is a **child concept** of its parents.

We use a predefined set of 20 relations, some of these are:

* C : membership in a class, IsA *A bird is a kind of animal = C(bird,animal)*
* F : feature, specification, HasProperty *Joe is tall = F(Joe,tall)*
* A : action acted by actor *Joe runs = A(Joe,run)*
* IM : implication, Causes *If Joe is adult, Joe is tall* *= IM(F(Joe,adult), F(Joe,tall))*
* NOT: negation *Joe is not tall = NOT(F(Joe,tall))* or *F(Joe,NOT(tall))*. This is also an example for ambiguity – the system is designed to manage it.

The relations we use mimic the grammar of the English language. This made it possible for us to develop a piece of software fairly easily which translates English into the concept network representation (which we call *mentalese*). At the same time, some of these relations are also a theoretical necessity for a non-aximatic reasoning system (like the C relation, the IM relation, and a few other). This is not a coincidence, but a reflection of the fact that both natural language and human intelligence are based on a set of foundations that are close to the features of NARS introduced by P. Wang.

As an example, let us take the English sentence “Money is in the bank”. In ConceptNet, for example, the representation is something like *location(money,bank).* In our mentalese, the representation is *F(money, R(in, Q(the,bank)))* . The relation Q is used to relate nouns (or noun phrases) with their articles or quantifiers. The relation R is used to relate a preposition with the following noun phrase. The meaning of the predicate *location* is carried by this portion of mentalese: *R(in* . We see that this translation to mentalese is simply following the grammatical structure of natural language.

In order to perform the natural language -> mentalese translation we make use of the SpaCy library to parse the sentences first. In the second step we take the parsed sentence as input and use rewrite rules developed by us to complete the translation. SpaCy does not provide information above the sentence level, but in our rewrite rules we can make connections between sentences to reflect the coherence on paragraph level.

We see this feature – the easy and straightfoward possibility to translate natural language text to mentalese – as the key differentiator to other AGI projects.

Mentalese is used throughout our implementation to store all kinds of information the thinking machine needs for its operation:

* Store input (after translation from English)
* Store knowledge
* Build models
* Define reasoning rules
* Store goals, scene descriptions, event sequences etc

In fact, the above types of information are not differentiated in our system. Input is knowledge, knowledge is model, model is rule, goals, descriptions etc., all these represented in mentalese.

Features of the concept network representation (mentalese):

1. Only 20 relations; only existing English words are used, no new labels are created.
2. Meaning is carried by the combination of (1) the original English word, (2) the position of this word in the hierarchy of nested concepts, and (3) the relations applied.
3. The resulting mentalese language is a rather mechanical copy of English, very much language-like. Therefore it is easy to translate from English to mentalese. (It will be more difficult to translate from mentalese to English - the implementation of this part has not been started yet).
4. English to mentalese translation requires rewrite rules, to be created manually.
5. We can add common sense knowledge to the system using natural language English sentences.
6. Ambiguity of English is transferred to the mentalese expression. Disambiguation happens in the reasoning module which processes the mentalese expressions.
7. Multiple meanings of a word are multiple concepts. Therefore, it is not a *word* that is a concept, but *one meaning of a word.* The mentalese representation has two stages. In the first stage, word meaning is not disambiguated, and we take the word as the concept. This is the output of the mentalese translator, and the input to the reasoning module. One task of reasoning is to replace the word with the proper meaning.
8. The recursive nature of the concept network enables to represent any level of knowledge in the same consistent fashion. A sentence of 10 words is usually a compound concept of 3-5 levels of depth. Such sentences can be easily organized into a deeper compound concept, representing a paragraph, and this can be continued.
9. Concepts have a gradual truth value, for example in the range 0-4. 0 means false, 1 means might, 3 means probably, 4 means true. Truth depends on the input (the “experience” of the system). It is being updated upon new relevant input.
   * The sentence “Joe is probably not tall” can be represented by *NOT(F(Joe,tall))* with truth=3. This is equivalent to “Joe might be tall”, *F(Joe,tall)* with truth=1 (for example). The reasoning module can perform this transformation.
10. Concepts have a number of scalar attributes, all of these are updated upon new input:
    * truth, can be considered as “normality” or “probability”
    * confidence level of the truth value
    * consistency of truth (being lower if exceptions or contradictions arrive)
    * relevance (how useful this concept is in question answering)
    * generality (used to differentiate a concrete instance of a concept from the general concept)

Meaning

The concept network representation provides a natural way how to interpret the “meaning” of a concept. Meaning is determined by the relationships between concepts: so it depends on the relations this concept participates in (these are the children of the concept) and on the relations its parents participate in. With new input, meaning can become richer, more elaborate. Meaning is not static, and may even change radically (sometimes) with new experience (=new input).

If a human makes a judgement on the extent to which the system *understands* the meaning of a certain concept, then this can be interpreted as a comparison between the meaning of the concept stored in the system, and the meaning stored in the human. This can be directly observed by someone looking at the memory of the system, or indirectly inferred from answers given to questions.

**Reasoning and question-answering application**

We have built a reasoning and question-answering module to process the mentalese representation of the input text. Reasoning takes mentalese as input, and produces new concepts in mentalese. Question-answering takes questions in mentalese as input, and provides answers in mentalese. These are implemented (though much work is still to be done).

Short and long term memory

Inputs into the system are stored as mentalese concepts in two storages: short term memory and long term memory.

* In short term memory, we store the recent input. Short term memory is regularly cleared and written into long term memory.
* In long term memory, we use an activation mechanism to limit the scope of concepts used for the current processing. We do not delete anything from long term memory. The “relevance” value of concepts can be used to make more or less probable that a given concept gets activated.
  + Concepts being the same on general level but referring to different specific entities (instances) are not merged with each other, because they are not treated as being the same. So the system may remember individual instances, if those were written to long term memory.

Activation

Reasoning uses an activation mechanism. Activation is a binary flag on the concept (could be gradual in the future). Only activated concepts and rules take part in reasoning. This is used to avoid the exponential explosion of runtime. This is the main way to limit the resources consumed by the system when producing response.

The activation mechanism enables a simple way to make the meaning of concepts sensitive to the *context.* The set of activated concepts at any given time is the current context. For example, if the system has to decide whether a spoon of salt is “much salt”, and the current set of activated concepts has a person eating a single egg by using up this amount of salt, the system will be able to correctly imply that in this context “much salt” is true.

Reasoning rules

The mentalese concepts entertained by the system represent causal models of the world. (The “world” is the set of inputs provided to the system.) Some concepts give rise to others in causal relationships. We call this process reasoning.

Reasoning is driven by two types of mentalese concepts: (1) a small number of general rules, and (2) a large number of specific rules. Both rule types take the form of concepts, and in fact specific rules are often easy to express in English (for general rules the English version may seem odd or philosophical, as we are not used to verbalize these basic aspects of thinking).

An example of a general rule is the following. If the latest input is the concept *NOT(F(Joe,tall))*, then the system may infer the concept *F(Joe,tall)* to its short term memory, with a suitable truth value *.* This reasoning action is not hard wired, but initiated by detecting the general rule *IM(NOT(%1),%1)*. *%1* is a wildcard that matches any concept. This rule is saying that “from *NOT(something)* it follows that *something”.* The appropriate truth value of the implication is derived from the truth of the premises using a table that can be specific to the rule or shared with other rules.

So the form of a rule is always IM(A,B) where A is the antecedent and B is the consequent. In the general rule only wildcards can be found, no specific word meaning. Special rules work in the same way but they include specific word meanings (they are often used to explain something about those meanings). They are only used when activated.

Most general rules, and a portion of the special rules are added to the system directly in mentalese. This is the portion of the input that is not natural language. How many such rules might be needed? We can probably estimate this by the number of English words in use, which means that on 6-years-old level we may need a few thousand of these rules, or probably less (because many of them will be normal English sentences).

Managing ambiguity

Ambiguity is inherent to natural language. In our model we manage ambiguity on many levels. Examples are: multiple meanings of words; multiple meanings of a natural language sentence; multiple possibilities to which an anaphor can refer; multiple meanings of a mentalese concept; multiple ways to express the same meaning in mentalese; multiple ways to translate the same mentalese to natural language. The “multiple meaning” type ambiguities are managed by allowing the evaluation of many possibilities in parallel, and selecting the best ones (this is now implemented for anaphor resolution). The “multiple ways” type ambiguities are managed currently by allowing the reasoning of several concepts with the same meaning (restrained by activation).

Exceptions, contradictions

Natural language texts are full of contradictions, which come in many forms. Two cases are currently managed in our system.

We can handle exceptions – concepts that contradict to some general knowledge. For example, the system knows that “people can talk”. Later the system learns that “people less than 1 year old usually cannot talk”. This is resolved by noting which concept is a special case of a more general concept. “people less than 1 year old” is more special than “people”, therefore reasoning with it will invalidate the reasoning with the more general concept.

The system also encounters contradictions or reinforcements not involving a generality-specificity relation. In such cases some properties of the concept (like truth value, consistency of truth value) get adjusted.

Inductive and abductive reasoning, pattern recognition

Rules are usable to support some forms of inductive and abductive reasoning. For example, when reading about a specific cat and learning that it has a tail, the system can induce that cats in general have tails.

However more work is needed to make this really useful, and to introduce further forms of inference, for example analogical inference.

We already have a pattern recognition capability which matches concepts to rules and finds out whether a rule is applicable (the concept matches the antecedent or a portion of it). This is not general enough, it cannot handle approximate matches. This is an example of a cognitive process that is not yet developed and will need a few weeks of work to have it implemented.

Running the reasoning and question answering application

**Reasoning** runs in the following cycle:

1. the system takes the next mentalese input (into short term memory), activates it
2. in case of ambiguity, the system creates branches (new instances of short term memory), corresponding to each hypothesis. A goodness score is being updated for each branch.
3. if the new input is a question, go to **question answering**
4. the system activates concepts and rules being close to the input
   1. use *relevance* to decide whether to activate a concept in the vicinity of the input
5. the system checks whether the new input is same as some of the activated concepts, and updates scores of occurrence counter, truth, confidence, consistency etc.
6. the system compares the input with rules, selects rules that are applicable
   1. for specific rules, the comparison is extended to *activated* rules only
   2. the system finds out for each applicable rule whether a conclusion can be drawn
      1. if yes, the conclusion concept gets generated and stored
      2. the reasoned concept is taken as new input, start this **reasoning** cycle again
7. after all hypothesis branches are completed, the one (or few) with the best goodness score is kept, thereby disambiguating ambiguity
8. if we are at the end of some text section (like a paragraph) the short term memory gets partially cleared, activations (partially) nullified, and long term memory updated.

**Question answering** works in the following way:

1. the system checks in short and long term memory for the concept formulated in the question, in order to find out whether the answer is already stored
   1. if yes, the answer is found, we are done
2. if no, the system activates further concepts and rules being close to the question
3. the system reasons using these activated concepts (run **reasoning** from step 5)
   1. after each concept being used for reasoning, the system checks again whether we have the answer now, and decides whether to stop
4. if answers are good enough with respect to resource use, the system presents the answers on the output (or answers “I don`t know”)
   * 1. if more resource can be invested, the system returns to step 2.

**Future work**

In order to achieve the goal of domain independent language understanding on 6-years-old level, we must provide common sense knowledge for the system. This needs to be manually selected from existing pieces of text, or even written, as there is no organized resource available for domain independent children level common sense. The effort can be significant , however we try to write the first few dozen pages in the next 12 months for testing, and we already have the first few pages ready and working.

**APPENDIX**

Below is an example of input for the topic of penguins. This example is a mixture of descriptive and expository type of text. Obviously we need texts of narrative and argumentative types as well.

Penguins are a strange type of bird.

They are very different than most birds because they do not fly.

Penguins swim.

They are birds of the water.

Penguins are great at living in the ocean.

Can you find penguins in the mountains?

Why are penguins strange?

How many legs do penguins have?

Do penguins lay eggs?

Can penguins move in the air?

Can penguins move on land?

How do penguins move?

Instead of wings, penguins have flippers.

They use these flippers to push themselves through the water.

Their feet are webbed like a duck’s and they use them to kick and steer.

How do penguins move?

What does penguins help to move?

Do penguins use their legs in the water?

You see a penguin.

This penguin is not in the water.

Where is this penguin?

How does this penguin move on land?

Where is the flipper of this penguin?

Does this penguin lend his flipper to another penguin?

In order to be light enough to fly, most birds have hollow bones.

The bones of a penguin are solid.

This makes it easier for them to dive deep into the water.

Does it help penguins that their bones are solid?

How?

Why do other birds have hollow bones?

Why is this not important for penguins?

Penguins have dark backs and white bellies.

This is a form of camouflage.

Camouflage helps animals hide.

Penguin camouflage is called countershading.

What is the colour of a penguin like?

Does the penguin want to hide?

Why?

Does the dark colour of penguins` backs help them in any way?

Penguins waterproof their feathers by coating them with oil.

Penguins have a gland that lets out oil near their tail.

By touching their beak to the gland they can spread the oil to their other feathers.

Penguins need healthy feathers to survive.

When Penguins take care of their feathers it is called preening.

They use their beaks to clean their feathers and keep them healthy.

Penguin’s feathers go through a large amount of wear and tear.

To prevent them from wearing out, most penguins molt once a year.

The old feathers fall out as new feathers grow to replace them.

While penguins are molting, they are unable to hunt.

Their soft down feathers are exposed and these are not waterproof.

Because of this, penguins eat a lot before they molt.

Penguins eat seafood.

They hunt squid, shellfish, and other fish.

Being good swimmers helps them hunt and have food to eat.

When they are young, the thicker outer layer of feathers has not grown in yet.

This means their parents have to provide food for them.

They do this by regurgitating their food.

The parent hunts and catches food.

They then throw it back up into the chick’s beak.

Why is it important for penguins to be a good swimmer?

Where do penguins hunt?

Why do parents need to feed young penguins?

Why don`t young penguins hunt?

What are penguins?

What are penguins like?

How are penguins different from other birds?

How are they the same?