A General Intelligence Oriented Architecture for Embodied Natural Language Processing

Ben Goertzel[1] **& Cassio Pennachin**[1] **& Samir Araujo**[1] **& Fabricio Silva**[1]
& Murilo Queiroz[1] & Ruiting Lian[2] & Welter Silva[1] &Michael Ross & Linas Vepstas[1] & Andre Senna[1]
[1] Novamente LLC
1405 Bernerd Place, Rockville MD 20851
[2] Artificial Brain Laboratory
Xiamen University, Xiamen, China

Abstract

A software architecture is described which enables a virtual agent in an online virtual world to carry out simple English language interactions grounded in its perceptions and actions. The use of perceptions to guide anaphor resolution is discussed, along with the use of natural language generation to answer simple questions about the observed world. This architecture has been implemented within the larger PetBrain system, which is built on the OpenCog open-source AI software framework and architected based on the OpenCogPrime design for integrative AGI, and has previously been used for nonlinguistic intelligent behaviors such as imitation and reinforcement learning.

Introduction

One key requirement of a humanlike AGI is the ability to communicate linguistically about the world it experiences, and to use its world-experience to understand the language others produce. We describe here an approach to achieving these abilities, which has been implemented within the PetBrain software system. After reviewing the current implementation and its capabilities, we discuss a pathway to extending it into a more powerful AGI system.

The PetBrain implements a portion of OpenCogPrime (Goe09), an integrated conceptual and software design aimed at achieving roughly humanlike AGI at the human level and ultimately beyond. It is implemented within the OpenCog open-source AI software framework (GH08); and its purpose is to control a virtual pet (currently a dog) in the Multiverse or RealXTend virtual worlds. Previous papers have described the PetBrain's capability for imitative and reinforcement learning (BG08b) and its personality and emotion model (BG08a); here we focus on its capability for linguistic interaction, with a focus on the way its virtual embodiment allows it to resolve linguistic ambiguity and answer questions about itself and its life. The paper is best read in conjunction with two online videos illustrating the phenomena described [1], [2].

[1] http://novamente.net/example/grab_ball.html
[2] http://novamente.net/example/nlp.html

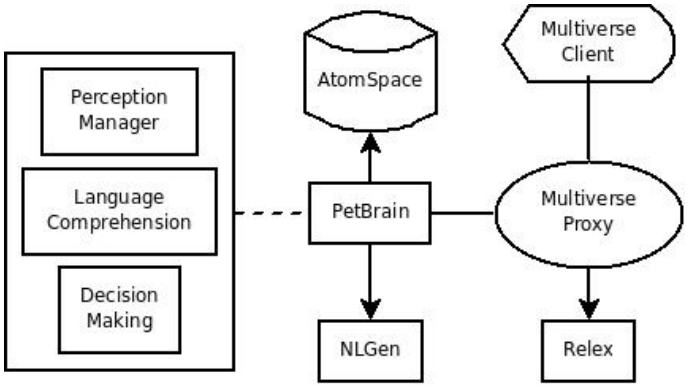

Figure 1: High-level overview of PetBrain software architecture

OpenCog and the PetBrain

OpenCogPrime is a cognitive architecture intended for implementation within the OpenCog AGI software framework, motivated by human cognitive science and overlapping significantly with Stan Franklin's LIDA (Goe08a) and Joscha Bach's MicroPsi (Bac09) architectures. The architecture consists of a division into a number of interconnected functional units corresponding to different specialized capabilities such as perception, motor control and language, and also an attentional focus unit corresponding to intensive integrative processing. Within each functional unit, knowledge representation is enabled via an Atomspace software object that contains nodes and links (collectively called Atoms) of various types representing declarative, procedural and episodic knowledge both symbolically and subsymbolically. (For a description of the node and link types typically utilized in OpenCog, the reader is referred to (GP06); here we will mention a few node and link types in passing, assuming the essential semantics will be clear from context.) Each unit also contains a collection of MindAgent objects implementing cognitive, perception or action processes that act on this Atomspace, and/or interact with the outside world.

The PetBrain, roughly depicted in Figure 1, is a sub-

set of the OpenCogPrime architecture, implemented within OpenCog. Currently it is used to control virtual pets, but in fact it could be used more generally to control various intelligent virtual agents; and work is underway to customize it to control humanoid robots as well (GdG98). The PetBrain stores all its knowledge inside the Atomspace. Part of this knowledge is produced by the agent's (pet's) sensors (exteroceptive and proprioceptive) and handled by the Perception Manager component. The agent's knowledge about the whole environment is used by the Language Comprehension component to link the elements, mentioned in the sentences heard by the agent, to the objects observed by the agent in the virtual world. An agent can recognize and execute commands requested by another agent/avatar, besides answering questions.

Most of our work with the PetBrain to date has involved the Multiverse [3] virtual world, though we have also worked with RealXTend. We've customized the Multiverse Server and created a Multiverse Proxy to mediate communication between the Virtual World and the PetBrain. The Multiverse Proxy sends perceptual data from Multiverse to the PetBrain, and in the current system it also sends linguistic relationship. The RelEx language comprehension system is used by the Multiverse Proxy to parse the sentences given by the user, via the Multiverse Client, and only the Relation objects produced by RelEx are then sent to the PetBrain. Conversely, the PetBrain sends Relation objects based on conceptual Atoms to the Multiverse Proxy, which invokes the NLGen system to transform these into English to be conveyed to the user.

The current PetBrain architecture is not intended as a human(or animal)-level AGI but rather as an interesting proto-AGI software system incorporating multiple components designed with human-level AGI in mind. Strategies for incrementally modifying the PetBrain into a human-level AGI will be discussed below.

Natural Language Processing with RelEx and NLGen

OpenCog's current Natural Language Processing subsystem (invoked by the PetBrain but also heavily used outside it) contains two main components: RelEx (GGP$^+$06), which is the natural language comprehension engine, takes sentences and maps them into abstract logical relations which can be represented in the OpenCog Atomspace; and NLGen, a natural language generation engine, that translates Atoms embodying logical relations into English sentences.

RelEx itself contains multiple components, only the largest or most pertinent of which will be reviewed here. RelEx carries out syntactic parsing via the open-source Link Parser created at Carnegie-Mellon University (ST91)[4]. It then contains semantic interpretation code that converts the Link Parser output to a feature structure representation (a directed graph), and uses a series of hand-coded rules ("sentence algorithms") to modify the feature structure. The modified feature structures are used to generate the RelEx semantic relationships corresponding to a sentence, which bear some resemblance to the output of the Stanford dependency parser [5], but often contain significant additional semantic information. Finally, the RelEx2Frame component uses additional hand-coded rules to map RelEx semantic relationships into sets of more abstract logical relationships, constructed utilizing the FrameNet (BFL98) ontology and other similar semantic resources (Goe08b); and the Frame2Atom component translates these relationships into OpenCogAtoms Thus, RelEx translates English into Atoms. Among the many details we have left out in this precis are the ranking of multiple outputs (parses and FrameNet interpretations are ranked using a combination of inference and heuristics) and the handling of ambiguity (e.g. anaphor resolution which will be discussed below).

NLGen is a sentence generation system which is used to generate sentences from RelEx semantic relationships – which in turn may be produced by applying a component called Frames2RelEx to appropriate Atoms in the OpenCog Atomspace. One of the core ideas underlying NLGen is that most language generation may be done by reversing previously executed language comprehension processes (GPI$^+$),(LGL$^+$). Given a set of interconnected Atoms to express, NLGen iterates through the predicates P in this set, and for each one it produces a graph G_P of associated RelEx semantic relationships, and then matches this graph against its memory (which stores previously perceived sentences and their semantic interpretations) via the SAGA matching algorithm (TMS$^+$07), looking for remembered sentences that gave rise to semantic relationship-sets similar to G_P. The results from doing this matching for different graphs G_P are then merged, and some rules are applied for handling phenomena like tense and determiners, ultimately yielding one or more sentences based on combining (instantiated abstractions of) pieces of the remembered sentences corresponding to selected predicates P. Finally, while this similarity matching approach has quite broad applicability, for dealing with complex sentences it must be augmented by additional mechanisms, and a prototype exists that uses an implementation of Chomsky's Merge operator for this purpose (operating on the same RelEx relationships as the primary NLGen system).

Embodiment-Based Anaphor Resolution

One of the two ways the PetBrain currently relates language processing to embodied experience is via using the latter to resolve anaphoric references in text produced by human-controlled avatars.

In our current work, the PetBrain controlled agent lives in a world with many objects, each one with their

[3] http://www.multiverse.net

[4] http://www.link.cs.cmu.edu/link/

[5] http://nlp.stanford.edu/software/lex-parser.shtml

own characteristics. For example, we can have multiple balls, with varying colors and sizes. We represent this in the OpenCog Atomspace via using multiple nodes: a single ConceptNode to represent the concept "ball", a WordNode associated with the word "ball", and numerous SemeNodes representing particular balls. There may of course also be ConceptNodes representing ball-related ideas not summarized in any natural language word, e.g. "big fat squishy balls," "balls that can usefully be hit with a bat", etc.

As the agent interacts with the world, it acquires information about the objects it finds, through perceptions. The perceptions associated with a given object are stored as other nodes linked to the node representing the specific object instance. All this information is represented in the Atomspace using FrameNet-style relationships (exemplified in the next section).

When the user says, e.g., "Grab the red ball", the agent needs to figure out which specific ball the user is referring to – i.e. it needs to invoke the Reference Resolution (RR) process. RR uses the information in the sentence to select instances and also a few heuristic rules. Broadly speaking, Reference Resolution maps nouns in the user's sentences to actual objects in the virtual world, based on world-knowledge obtained by the agent through perceptions.

In this example, first the brain selects the ConceptNodes related to the word "ball". Then it examines all individual instances associated with these concepts, using the determiners in the sentence along with other appropriate restrictions (in this example the determiner is the adjective "red"; and since the verb is "grab" it also looks for objects that can be fetched). If it finds more than one "fetchable red ball", an heuristic is used to select one (in this case, it chooses the nearest instance).

The agent also needs to map pronouns in the sentences to actual objects in the virtual world. For example, if the user says "I like the red ball. Grab it," the agent must map the pronoun "it" to a specific red ball. This process is done in two stages: first using anaphor resolution to associate the pronoun "it" with the previously heard noun "ball"; then using reference resolution to associate the noun "ball" with the actual object.

The subtlety of anaphor resolution is that there may be more than one plausible "candidate" noun corresponding to a given pronouns. RelEx's standard anaphor resolution system is based on the classical Hobbs algorithm(Hob78). Basically, when a pronoun (it, he, she, they and so on) is identified in a sentence, the Hobbs algorithm searches through recent sentences to find the nouns that fit this pronoun according to number, gender and other characteristics. The Hobbs algorithm is used to create a ranking of candidate nouns, ordered by time (most recently mentioned nouns come first).

We improve the Hobbs algorithm results by using the agent's world-knowledge to help choose the best candidate noun. Suppose the agent heard the sentences:

```
"The ball is red."
"The stick is brown."
```

and then it receives a third sentence

```
"Grab it.".
```

the anaphor resolver will build a list containing two options for the pronoun "it" of the third sentence: ball and stick. Given that the stick corresponds to the most recently mentioned noun, the agent will grab it instead of (as Hobbs would suggest) the ball.

Similarly, if the agent's history contains

```
"From here I can see a tree and a ball."
"Grab it."
```

Hobbs algorithm returns as candidate nouns "tree" and "ball", in this order. But using our integrative Reference Resolution process, the agent will conclude that a tree cannot be grabbed, so this candidate is discarded and "ball" is chosen.

Embodiment-Based Question Answering

Our agent is also capable of answering simple questions about its feelings/emotions (happiness, fear, etc.) and about the environment in which it lives. After a question is asked to the agent, it is parsed by RelEx and classified as either a truth question or a discursive one. After that, RelEx rewrites the given question as a list of Frames (based on FrameNet [6] with some customizations), which represent its semantic content. The Frames version of the question is then processed by the agent and the answer is also written in Frames. The answer Frames are then sent to a module that converts it back to the RelEx format. Finally the answer, in RelEx format, is processed by the NLGen module, that generates the text of the answer in English. We will discuss this process here in the context of the simple question "What is next to the tree?", which in an appropriate environment receives the answer "The red ball is next to the tree."

Question answering (QA) of course has a long history in AI (Zhe09), and our approach fits squarely into the tradition of "deep semantic QA systems"; however it is innovative in its combination of dependency parsing with FrameNet and most importantly in the manner of its integration of QA with an overall cognitive architecture for agent control.

Preparing/Matching Frames

In order to answer an incoming question, the agent tries to match the Frames list, created by RelEx, against the Frames stored in its own memory. In general these Frames could come from a variety of sources, including inference, concept creation and perception; but in the current PetBrain they primarily come from perception, and simple transformations of perceptions.

However, the agent cannot use the incoming perceptual Frames in their original format because they lack

[6] http://framenet.icsi.berkeley.edu

grounding information (information that connects the mentioned elements to the real elements of the environment). So, two steps are then executed before trying to match the Frames: Reference Resolution (described above) and Frames Rewriting. Frames Rewriting is a process that changes the values of the incoming Frames elements into grounded values. Here is an example, using the standard Novamente/OpenCog indent notation described in (GP06) (in which indentation denotes the function-argument relation, as in Python, and RAB denotes the relation R with arguments A and B):

Incoming Frame (Generated by RelEx)

```
EvaluationLink
  DefinedFrameElementNode Color:Color
  WordInstanceNode "red@aaa"
EvaluationLink
  DefinedFrameElementNode Color:Entity
  WordInstanceNode "ball@bbb"
ReferenceLink
  WordInstanceNode "red@aaa"
  WordNode "red"
```

After Reference Resolution

```
ReferenceLink
  WordInstanceNode "ball@bbb"
  SemeNode "ball_99"
```

Grounded Frame (After Rewriting)

```
EvaluationLink
  DefinedFrameElementNode Color:Color
  ConceptNode "red"
EvaluationLink
  DefinedFrameElementNode Color:Entity
  SemeNode "ball_99"
```

Frame Rewriting serves to convert the incoming Frames to the same structure used by the Frames stored into the agent's memory. After Rewriting, the new Frames are then matched against the agent's memory and if all Frames were found in it, the answer is known by the agent, otherwise it is unknown.

In the current system, if a truth question was posed and all Frames were matched successfully, the answer will be "yes"; otherwise the answer is "no". Mapping of ambiguous matching results into ambiguous responses has been deferred till a later version.

If the question requires a discursive answer the process is slightly different. For known answers the matched Frames are converted into RelEx format by Frames2RelEx and then sent to NLGen, which prepares the final English text to be answered. There are two types of unknown answers. The first one is when at least one Frame cannot be matched against the agent's memory and the answer is "I don't know". And the second type of unknown answer occurs when all Frames were matched successfully they cannot be correctly converted into RelEx format or NLGen cannot identify the incoming relations. In this case the answer is "I know the answer, but I don't know how to say it".

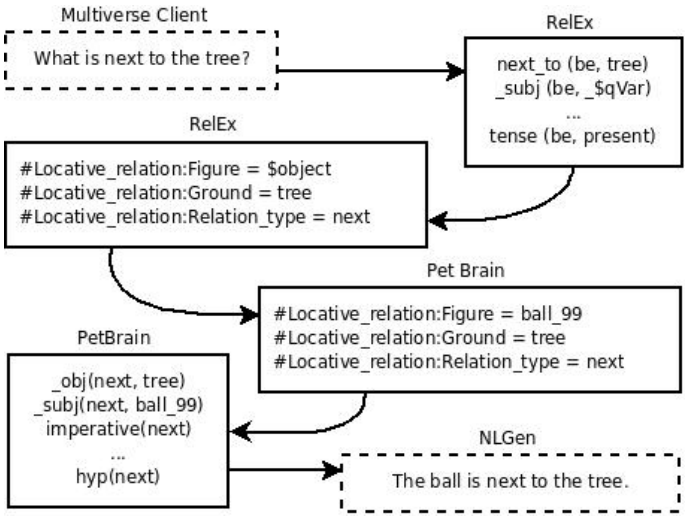

Figure 2: Overview of language comprehension process

Frames2RelEx

As mentioned above, this module is responsible for receiving a list of grounded Frames and returning another list containing the relations, in RelEx format, which represents the grammatical form of the sentence described by the given Frames. That is, the Frames list represents a sentence that the agent wants to say to another agent. NLGen needs an input in RelEx Format in order to generate an English version of the sentence; Frames2RelEx does this conversion.

Currently, Frames2RelEx is implemented as a rule-based system in which the preconditions are the required frames and the output is one or more RelEx relations e.g.

```
#Color(Entity,Color) =>
  present($2) .a($2) adj($2) _predadj($1, $2)
    definite($1) .n($1) noun($1) singular($1)
      .v(be) verb(be) punctuation(.) det(the)
```

where the precondition comes before the symbol => and $Color$ is a frame which has two elements: Entity and Color. Each element is interpreted as a variable $Entity$ = $1 and $Color$ = $2. The effect, or output of the rule, is a list of RelEx relations. As in the case of RelEx2Frame, the use of hand-coded rules is considered a stopgap, and for a powerful AGI system based on this framework such rules will need to be learned via experience (a topic beyond the scope of this paper).

Example of the Question Answering Pipeline

Turning to the example "What is next to the tree?", Figure illustrates the processes involved:

The question is parsed by RelEx, which creates the frames indicating that the sentence is a question regarding a location reference (next) relative to an object (tree). The frame that represents questions is called

Questioning and it contains the elements Manner that indicates the kind of question (truth-question, what, where, and so on), Message that indicates the main term of the question and Addressee that indicates the target of the question. To indicate that the question is related to a location, the *Locative_relation* frame is also created with a variable inserted in its element Figure, which represents the expected answer (in this specific case, the object that is next to the tree).

The question-answer module tries to match the question frames in the Atomspace to fit the variable element. Suppose that the object that is next to the tree is the red ball. In this way, the module will match all the frames requested and realize that the answer is the value of the element Figure of the frame *Locative_relation* stored in the Atom Table. Then, it creates location frames indicating the red ball as the answer. These frames will be converted into RelEx format by the RelEx2Frames rule based system as described above, and NLGen will generate the expected sentence "the red ball is next to the tree".

Example of the Language Generation Pipeline

To illustrate the process of language generation using NLGen, as utilized in the context of query response, consider the sentence "The red ball is near the tree". When parsed by RelEx, this sentence is converted to:

```
_obj(near, tree)
_subj(near, ball)
imperative(near)
hyp(near)
definite(tree)
singular(tree)
_to-do(be, near)
_subj(be, ball)
present(be)
definite(ball)
singular(ball)
```

So, if sentences with this format are in the system's experience, these relations are stored by NLGen and will used to match future relations that must be converted into natural language. NLGen matches at an abstract level, so sentences like "The stick is next to the fountain" will also be matched even if the corpus contain only the sentence "The ball is near the tree".

If the agent wants to say that "The red ball is near the tree", it must invoke NLGen with the above RelEx contents as input. However, the knowledge that the red ball is near the tree is stored as frames, and not as RelEx format. More specifically, in this case the related frame stored is the *Locative_relation* one, containing the following elements and respective values: Figure → red ball, Ground → tree, *Relation_type* → near.

So we must convert these frames and their elements' values into the RelEx format accept by NLGen. For AGI purposes, a system must learn how to perform this conversion in a flexible and context-appropriate way. In our current system, however, we have implemented a temporary short-cut: a system of hand-coded rules, in which the preconditions are the required frames and the output is the corresponding RelEx format that will generate the sentence that represents the frames. The output of a rule may contains variables that must be replaced by the frame elements' values. For the example above, the output $\_subj(be, ball)$ is generated from the rule $output\_subj(be, \$var1)$ with the $\$var1$ replaced by the Figure element value.

Considering specifically question-answering (QA), the PetBrain's Language Comprehension module represents the answer to a question as a list of frames. In this case, we may have the following situations:

- The frames match a precondition and the RelEx output is correctly recognized by NLGen, which generates the expected sentence as the answer;
- The frames match a precondition, but NLGen did not recognize the RelEx output generated. In this case, the answer will be "I know the answer, but I don't know how to say it", which means that the question was answered correctly by the Language Comphrehension, but the NLGen could not generate the correct sentence;
- The frames didn't match any precondition; then the answer will also be "I know the answer, but I don't know how to say it" ;
- Finally, if no frames are generated as answer by the Language Comprehension module, the agent's answer will be "I don't know".

If the question is a truth-question, then NLGen is not required. In this case, the creation of frames as answer is considered as a "Yes", otherwise, the answer will be "No" because it was not possible to found the corresponding frames as the answer.

From Today's PetBrain to Tomorrow's Embodied AGI

The current PetBrain system displays a variety of interesting behaviors, but is not yet a powerful AGI system. It combines a simple framework for emotions and motivations with the ability to learn via imitation, reinforcement and exploration, and the ability to understand and produce simple English pertaining to its observations and experiences; and shortly it will be able to carry out simple inferences based on its observations and experiences. What needs to be done to transform the current PetBrain into an AGI system with, say, the rough general intelligence level of a young child? Of course the answer to this question must be speculative at this point; but we will give an answer here based on the assumption that the fundamental cognitive theory underlying the system is correct, and focused on NLP aspects.

Firstly, the current PetBrain QA system has one major and obvious shortcoming: it can only answer questions whose answer is directly given by its experience.

To answer questions whose answers are indirectly given by experience, some sort of inference process must be integrated into the system. OpenCog already has a probabilistic engine, PLN (Probabilistic Logic Networks), and one of our next steps will be to integrate it with the PetBrain. For instance, suppose the agent is in a scenario that has many balls in it, of different colors. Suppose it has previously been shown many objects, and has been told things like "The ball near the tree is Bob's", "the stick next to Jane is Jane's", etc. Suppose that from its previous experience, the agent has enough data to infer that Jane tends to own a lot of red things. Suppose finally that the agent is asked "Which ball is Bob's." The current agent will say "I don't know," unless someone has told it which ball is Bob's before, or it has overheard someone referring to a particular ball as Bob's. But with PLN integrated, then the agent will be able look around, find a red ball and say (for instance) "The ball near the fountain" (if the ball near the fountain is in fact red).

Next, there are known shortcomings in the NLP infrastructure we have used in the PetBrain, some of which have been mentioned above, e.g. the use of hardcoded rules in places where there should be experientially adaptable rules. Remedying these shortcomings is relatively straightforward within the OpenCog architecture, the main step being to move all of these hardcoded rules into the Atomspace, replacing their interpreters with the PLN chainer, and then allowing PLN inference to modify the rules based on experience.

Apart from NLP improvements and PLN integration, what else is missing in the PetBrain, restricting its level of general intelligence? It is missing a number of important cognition components identified in the OpenCogPrime AGI design. Adaptive attention allocation is prime among these: the ECAN framework (GPI^+) provides a flexible capability for assignment of credit and resource allocation, but needs to be integrated with and adapted to the virtual agent control context. Concept creation is another: inference about objects and linguistic terms is important, but inference becomes more powerful when used in synchrony with methods for creating new concepts to be inferred about. Finally the PetBrain's motivational architecture is overly specialized for the "virtual dog" context and we intend to replace it with a new architecture based on Joscha Bach's MicroPsi (Bac09). These are not trivial aspects but they are well-understood and well-documented.

There is also the question of whether virtual worlds like Multiverse are sufficiently rich to enable a young artificial mind to learn to be a powerful AGI. We consider this non-obvious at present, and in parallel with our virtual-worlds work we have been involved with using the PetBrain to control a Nao humanoid robot (GdG98). The OpenCog framework is flexible enough that intricate feedback between robotic sensorimotor modules and cognitive/linguistic modules (such as those described here) can be introduced without changing the operation of the latter.

References

Joscha Bach. *Principles of Synthetic Intelligence*. Oxford University Press, 2009.

Collin F. Baker, Charles J. Fillmore, and John B. Lowe. The berkeley framenet project, 1998.

et al Ben Goertzel. An inferential dynamics approach to personality and. emotion driven behavior determination for virtual animals, 2008.

et al Ben Goertzel. An integrative methodology for teaching embodied non-linguistic agents, applied to virtual animals in second life, 2008.

Ben Goertzel and Hugo de Garis. Xia-man: An extensible, integrative architecture for intelligent humanoid robotics, 1998.

Ben Goertzel, Izabela Freire Goertzel, Hugo Pinto, Mike Ross, Ari Heljakka, and Cassio Pennachin. Using dependency parsing and probabilistic inference to extract relationships between genes, proteins and malignancies implicit among multiple biomedical research abstracts, 2006.

Ben Goertzel and David Hart. Opencog: An open-source platform for agi, 2008.

Ben Goertzel. Lida and a theory of mind, 2008.

Ben Goertzel. A pragmatic path toward endowing virtually-embodied ais with human-level linguistic capability, 2008.

Ben Goertzel. Opencogprime: A cognitive synergy based architecture for embodied general intelligence, 2009.

Ben Goertzel and Cassio Pennachin. The novamente cognition engine, 2006.

Ben Goertzel, Joel Pitt, Matthew Ikle, Cassio Pennachin, and Rui Liu. Glocal memory: a design principle for artificial brains and minds. *Neurocomputing, Special Issue of Artificial Brain*, to appear.

Jerry R. Hobbs. Resolving pronominal references. *Lingua*, (2):232–239, 1978.

Ruiting Lian, Ben Goertzel, Rui Liu, Michael Ross, Murilo Queiroz, and Linas Vepstas. Sentence geration for artificial brains: a glocal similarity matching approach. *Neurocomputing, Special Issue of Artificial Brain*, to appear.

Daniel D. K. Sleator and Davy Temperley. Parsing english with a link grammar, 1991.

Yuanyuan Tian, Richard C. McEachin, Carlos Santos, David J. States, and Jignesh M. Patel. Saga: a subgraph matching tool for biological graphs. *Bioinformatics (Oxford, England)*, 23(2):232–239, Jan. 2007.

Zhiping Zheng. Bibliography on automated question answering, May 2009. `http://www.answerbus.com/bibliography/index.shtml`.