Syntax-Semantic Mapping for General Intelligence: Language Comprehension as Hypergraph Homomorphism, Language Generation as Constraint Satisfaction

Ruiting Lian[2,4] *and Ben Goertzel*[2,3,4] *and Shujing Ke*[1,2,4] *and Jade O'Neill*[1] *and Keyvan Sadeghi*[2] *and Simon Shiu*[1] *and Dingjie Wang*[2,4] *and Oliver Watkins*[2] *and Gino Yu*[2]

[1] Hong Kong Poly U, Dept. of Computer Science
[2] Hong Kong Poly U, School of Design
[3] Novamente LLC
[4] Dept. of Cognitive Science, Xiamen University

Abstract. A new approach to translating between natural language expressions and hypergraph-based semantic knowledge representations is proposed. Language comprehension is formulated in terms of homomorphisms mapping syntactic parse trees into semantic hypergraphs, and language generation as constraint satisfaction based on constraints derived via applying the inverse relations of these homomorphisms. This provides an elegant approach to implementing semantically savvy NLP systems, and also to thinking about the feedbacks between syntactic and semantic processing that are the crux of generally intelligent NLP. A prototype of the approach created using the link parser and the OpenCog Atom semantic representation is described, and initial results presented. Routes to extending this prototype into something useful for aiding generally intelligent dialogue systems are discussed.

1 Introduction

Human language interaction is a large part of human-level AGI. The Turing Test, the most widely accepted evaluation metric for human-level AGI, is entirely focused on natural language dialogue; but even if one sets the Turing Test aside as many researchers advocate [1], there is no disputing the key value of human-like conversation as an indicator of human-like intelligence.

However, the fields of AGI and NLP are currently almost entirely disjoint. Few of the existing proto-AGI systems deal with language; and nearly all NLP systems are centered on extremely specialized rule-based or statistical methods that require careful customization for effective processing in new domains. If one's goal is AGI, two main options present themselves:

1. Create an AGI system capable of fairly general learning via experience, but little or no built-in linguistic mechanism, and have it learn human language.

Variations abound, including robotic and virtual embodiment versus pure chat-based learning.
2. Create an AGI system including relatively sophisticated computational linguistics mechanisms, and have it use these to communicate, while modifying and improving them based on its experience.

A conceptual framework for thinking about the second option was presented in [2], in the context of enabling an AGI system to use virtually and/or robotically embodied experience to revise linguistic knowledge initially supplied to it via traditional computational linguistics means. Subsequent to publication of [2], some practical attempts were made to work toward implementing the ideas described there, in the OpenCog framework [3]. However, various technical difficulties were encountered, due to limitations of the specific computational linguistics tools integrated with OpenCog (the RelEx language comprehension system [4], and associated RelEx2Frame [5] and NLGen systems [6]). The present paper describes a new approach to integrating computational linguistics tools with OpenCog, differing from the RelEx approach in significant respects, created with the purpose of making it easier for OpenCog's general-purpose learning algorithms to interact synergetically with its dedicated computational linguistics components.

The key ideas of the approach presented here are:

1. Language comprehension, in its intermediate stages, is carried out via application of graph rewrite rules to syntactic parse trees. These rules, together, effect a hypergraph homomorphism that maps a parse tree into a subhypergraph of an OpenCog Atomspace (the "Atomspace" being OpenCog's weighted, labeled hypergraph based knowledge store). That is: **syntax to semantics mapping via hypergraph homomorphism**.
2. The more advanced stages of language comprehension are carried out via generic inference mechanisms, acting on the hypergraphs produced by the above rewrite rules. This is relied upon systematically, in the sense that the above rewrite rules don't try to perform subtle disambiguation, generally trying to disambiguate only enough to figure out the argument structures of the semantic relations being depicted in a sentence, and the basic semantic nature of each relation.
3. If a parser is capable of accepting semantic guidance midway through the parsing process, this guidance may be obtained via applying the above rewrite rules to partial parses and obtaining information regarding the semantic meaningfulness of the results
4. Language generation is done by applying the inverse relations of the above graph rewrite rules to a semantic hypergraph intended for expression, and thus obtaining a set of constraints corresponding to said hypergraph. A sentence expressing the hypergraph is then generated by solving the constraint satisfaction problem. That is: **semantics to syntax mapping via finding solutions to the constraints posed by the inverse relations of hypergraph homomorphisms**.

5. A database of previously comprehended sentences may be used to direct the system toward more natural-sounding solutions to the constraint satisfaction problem, as may inference on the hypergraphs inferred by applying known rewrite rules to partially generated sentenced.

Our main focus here will be on points 1 and 4, though the other points will be mentioned briefly as appropriate. The critical difference between this approach and previous approaches attempted with OpenCog, is that the graph rewrite rules serving as the core of the syntax/semantics mapping process are simple enough to be treated as "cognitive content" by OpenCog's learning and reasoning processes.

Due to length limitations, we have placed a number of tables and examples associated with the paper in Supplementary Information available online at `http://wp.goertzel.org/?page_id=406`.

2 The OpenCog Integrative AGI Framework

The work described here is part of the larger project of developing OpenCog, an open-source AGI software framework. OpenCog has been used for commercial applications in the area of natural language processing and data mining; e.g. see [4]. It has also been used to control virtual agents in virtual worlds, at first using an OpenCog variant called the OpenPetBrain [7], and more recently in a more general way using a Minecraft-like virtual environment [8]. It is the platform for the in-progress implementation of the OpenCogPrime design aimed ultimately toward AGI at the human level and beyond.

Conceptually founded on the "patternist" systems theory of intelligence outlined in [9], OpenCogPrime combines multiple AI paradigms such as uncertain logic, computational linguistics, evolutionary program learning and connectionist attention allocation in a unified architecture. Cognitive processes embodying these different paradigms interoperate together on a common neural-symbolic hypergraph knowledge store called the Atomspace ("Atom" being a term inclusive of Nodes and Links, where the latter includes hyperlinks). The interaction of these processes is designed to encourage the self-organizing emergence of high-level network structures in the Atomspace. Further review of OpenCog will be omitted here for space reasons; the reader is referred to [3] and various references linked from `http://opencog.org`.

The Atomspace Representation OpenCog's "Atomspace" knowledge representation is a generalized hypergraph formalism which comprises a specific vocabulary of Node and Link types, used to represent declarative knowledge and also, indirectly, other types of knowledge as well. There is a specific vocabulary of a couple dozen node and link types with semantics carefully chosen to reflect the needs of OpenCog's cognitive processes. Simple examples of OpenCog links, in the notation commonly used with OpenCog, are:

```
InheritanceLink Ben_Goertzel animal <.99>
```

```
EvaluationLink <.7>
        chase
        ListLink
                cat
                mouse
```

Examples using nodes with English-word labels provide convenient examples, but in fact most nodes in a practical OpenCog system will generally be automatically learned and not correspond directly to any human-language concept.

What's important about the AtomSpace knowledge representation is mainly that it provides a flexible means for compactly representing multiple relevant forms of knowledge, in a way that allows them to interoperate – where by "interoperate" we mean that e.g. a fragment of a chunk of declarative knowledge can link to a fragment of a chunk of attentional or procedural knowledge; or a chunk of knowledge in one category can overlap with a chunk of knowledge in another category (as when the same link has both a (declarative) truth value and an (attentional) importance value). In short, any representational infrastructure sufficiently flexible to support

- compact representation of all the key categories of knowledge playing dominant roles in human memory
- the flexible creation of specialized sub-representations for various particular subtypes of knowledge in all these categories, enabling compact and rapidly manipulable expression of knowledge of these subtypes
- the overlap and interlinkage of knowledge of various types, including that represented using specialized sub-representations

would probably be acceptable for OpenCog's purposes. The Atom formalism satisfies the relevant general requirements and has proved workable from a practical software perspective.

3 Link Parsing and RelEx

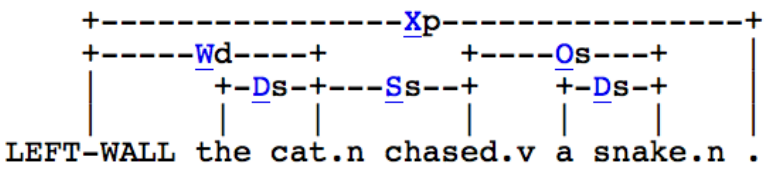
The novel NLP approach described here utilizes a syntax parsing framework called link parsing [10], and (to a lesser extent, and in a more temporary way), an add-on to the link parser called RelEx (for Relationship Extractor). The conceptual essence of the approach is not tied to these particular tools, but its current practical implementation is.

3.1 Link Grammar

The essential idea of link grammar is that each word comes with a feature structure consisting of a set of typed connectors . Parsing consists of matching up connectors from one word with connectors from another. Consider the sentence:

The cat chased a snake

The link grammar parse structure for this sentence is:



There is a database called the "link grammar dictionary" which contains connectors associated with all common English words. The notation used to describe feature structures in this dictionary is quite simple. Different kinds of connectors are denoted by letters or pairs of letters like S or SX. Then if a word W1 has the connector S+, this means that the word can have an S link coming out to the right side. If a word W2 has the connector S-, this means that the word can have an S link coming out to the left side. In this case, if W1 occurs to the left of W2 in a sentence, then the two words can be joined together with an S link.

The rules of link grammar impose additional constraints beyond the matching of connectors – e.g. the planarity and connectivity metarules.. Planarity means that links don't cross. Connectivity means that the links and words of a sentence must form a connected graph – all the words must be linked into the other words in the sentence via some path.

The graph rewrite rules at the center of the NLP approach described here map link parses into semantic Atom structures.

3.2 RelEx

RelEx is an English-language semantic relationship extractor, designed to post-process the output of the link parser. It can identify subject, object, indirect object and many other dependency relationships between words in a sentence; it generates dependency trees, resembling those of dependency grammars. The output of the current version of RelEx on the example sentence given above is:

```
singular(cat)
singular(snake)
_subj(chase, cat)
_obj(chase, snake)
past(chase)
```

A list of the important RelEx relationship types is included in this paper's online Supplementary Info.

RelEx currently works via creating a tree with a FeatureNode corresponding to each word in the sentence, and then applying a series of rules to update the entries in this FeatureNode. The rules transform combinations of link parser links into RelEx dependency relations, sometimes acting indirectly via dynamics

wherein one rule changes a feature in a word's FeatureNode, and another rule then takes an action based on the changes the former rule made.

OpenCog also contains a system called RelEx2Frame, that translates RelEx output into relationships involving the frames and arguments defined in the FrameNet ontology, and code for translating link parser links, RelEx and RelEx2Frame relationships, into Atoms. The new NLP approach presented here replaces RelEx2Frame and RelEx both, but utilizes RelEx in a temporary role to help generate data to enable the learning of graph rewrite rules mapping link parses into Atom structures.

3.3 NLGen

Language generation is a complex, multi-phase process. There is an abstract cognitive aspect, concerned with figuring out what is appropriate to say in the current context. And then there is the "surface realization" aspect, concerned with translating conceptual content into a grammatical, comprehensible statement.

OpenCog's current language generation software, called NLGen, is focused on surface realization, and is based on an approach called SegSim, which takes an Atom set in need of linguistic expression and matches its subsets against a data-store of (sentence, link parse, RelEx relationship set, Atom set) tuples, produced via applying OpenCog's NL comprehension tools to a corpus of sentences. Via this matching, it determines which syntactic structures have been previously used to produce relevant Atom subsets. It then pieces together the syntactic structures found to correspond to its subsets, to form overall syntactic structures corresponding to one or more sentences. This process works unproblematically for relatively simple sentences, but sometimes becomes tricky for sentences involving conjunctions or other complex syntactic forms.

4 Mapping Syntax to Semantics via Hypergraph Homomorphisms

The core idea of the proposed new approach to natural language comprehension is to map syntactic parses (e.g. link parse graphs) into semantic interpretations (e.g. Atom sets) via applying rewrite rules. Each rewrite rule takes as input a subgraph of a syntactic parse graph satisfying certain constraints, and outputs an Atom hypergraph. In practice the rules required seem to take the form of pairs (G, A), where

- G is a graph whose nodes are either words or variables, and whose links are link-parser link types
- A is a hypergraph whose nodes are either words, variables or special linguistic nodes (drawn from a small vocabulary of such), and whose hyper-edges are OpenCog Atom types (e.g. InheritanceLink, EvaluationLink).
- The lists of variables in G and A must be the same

We shall call rules matching this description "simple mapping rules."

For the simple mapping rules actually needed for handling human language, the constraint that **each edge in G maps into a single hyper-edge in A** appears to hold true. Mathematically, this latter constraint implies that each of the rewrite rules is individually a **graph homomorphism** [11], which then implies that a collection of rewrite rules applied together is also a graph homomorphism.

These observations about the practical nature of the rules required, are drawn from inspection of the actual rules used in the RelEx system currently, which appear sufficient to cover a wide variety of English syntax and semantics. According to our best understanding, extending RelEx to increase its coverage and accuracy would be unlikely to break any of the observations made here regarding the basic mathematical nature of the rules involved.

A simple example of such a rule is (G,A) where

$$G = \{S_*(v_1, v_2),\ O_*(v_1, v_3)\}$$

$$A = (EvaluationLink\ v_1\ v_2\ v_3)$$

This maps a verb v_1 with subject v_2 and object v_3 into an OpenCog EvaluationLink with v_1 as the predicate and (v_2, v_3) as the argument list. E.g. S_* refers to any of the link parser subtypes of S. Of course, most rules are more complex than this.

While the transformations the RelEx system carries out are capable of being formulated as simple mapping rules of the above form, they are not actually implemented that way, but instead are implemented via an iterative process of updating the FeatureNodes in a feature tree representing the words in a sentence. This is disadvantageous for two reasons:

1. Rewrite rules in the "simple mapping rule" format are easy to represent as Atoms themselves (using the format (ImplicationLink P_G P_A), where e.g. P_G is an AndLink joining the OpenCog Links corresponding to the link-parser links in the graph G). This makes it straightforward to OpenCog cognitive algorithms like PLN (Probabilistic Logic Networks) to re-weight, generalize and modify the rewrite rules, enabling the system's linguistic understanding to evolve via experience.
2. If one explicitly knows the rewrite rules used for comprehension, one can then turn these same rules around and use them for generation, as will be described below. This is a quite general and elegant approach to generating surface realizations for sentences.

With this in mind, we have recently implemented a novel approach to automatically learning simple mapping rules roughly equivalent to the current RelEx rules. We have used the link parser and RelEx to create a corpus of (sentence, link parse, RelEx relationship set, Atom set) tuples, and then used OpenCog's Fishgram pattern-mining system to automatically learn simple mapping rules from this corpus. While there is inevitably some noise in the results from the

this process, in essence what one finds is a collection of simple mapping rules that gives mainly the same results as the traditional OpenCog pipeline "link parser → RelEx → Atomspace". Systematic evaluation of the quality of these learned rules will be presented in a later paper.

The application of our current simple mapping rules to the example sentence given above yields the output

```
(ExistsLink
        z
        (ExistsLink
         y
                (ExistsLink
                        x
                        (AndLink
                                (InheritanceLink x Cat)
                                (InheritanceLink y snake)
                                (InheritanceLink z eat)
                                (InheritanceLink z past)
                                (EvaluationLink z x y)
                        )
                )
        )
)
```

Further example mappings are given in the paper's online Supplementary Info, along with explanation of the Atom types involved.

5 Mapping Semantics to Syntax via Constraint Satisfaction

To carry out "surface realization" and generate natural language expressing the concepts in Atom sets, it suffices to reverse the graph rewrite rules described in the previous section. However, this is not entirely simple, because the rules create homomorphisms rather than isomorphisms. Any one Atom structure may be produced by many different link-grammar structures, because there are many grammatical ways to produce any given idea. But not all the grammatical structures corresponding to different subsets of a given Atom set needing articulation, will necessarily be grammatically compatible with each other. So one has a constraint satisfaction problem, which in general will have multiple solutions, with varying levels of syntactic ambiguity and subjective human naturalness.

More precisely, suppose we have an Atom set $A = \{A_i\}$; and let $R = \{R_j\}$ denote the set of all graph rewrite rules R_j with the property that R_j maps at least one link parse subtree into some nonempty subset of $\{A_i\}$. Let $R^i \in R$ denote the set of rewrite rules that produce an Atom set including the particular Atom A_i; we may write $R^i = \{R^i_k\}$, with $R = \bigcup_i R^i$. Let $m_g(r)$ denote the proposition that the rewrite rule r matches some subgraph of the graph g.

Given this set-up, the problem of generating a sentence expressing the Atom set A boils down to finding some link parse g that

– parses correctly according to link grammar
– satisfies the expressiveness condition

$$\bigwedge_i \bigvee_k m_g(R_k^i),$$

– satisfies an assumed "aesthetic condition", initially: that it would either not parse or not satisfy the expressiveness condition if any of its words were removed

Given a link parse g, producing the relevant sentence is trivial. The task of generating a sentence-set expressing A reduces to choosing a way to partition A into subsets, so that each can be acceptably expressed via a single sentence.

A strength and weakness of this approach is that, in most practical cases, this constraint satisfaction problem will have many solutions. Selection among the various solutions could be approached in many ways, e.g. via evaluating various solutions and choosing the one with the highest word tuple probabilities relative to a large reference corpus; or via proceeding as in the current NLGen system, and choosing solutions whose fragments are known via past NL comprehension experience to have been used in real human-generated sentences.

6 Conclusions and Future Work

We have presented a novel approach to bidirectional syntax/semantics transformation. The ideas described do not purport to solve the whole problems of generally intelligent natural language comprehension or generation, but merely to provide an elegant mechanism for connecting syntactic and semantic aspects of linguistic intelligence. Currently the suggested approach to comprehension has been implemented but not yet thoroughly validated; and the suggested approach to generation is in the midst of implementation. Once implementation is complete the software will be used to help an OpenCog system to carry out natural language dialogue in the context of controlling a virtual agent in a video game world. Of course, this dialogue application will involve a host of other components as well, most critically a dialogue control mechanism based on OpenCog's "OpenPsi" framework for motivated action [12].

As we have focused on the syntax/semantics transformation aspect here, we have not said too much about what happens at either end of the transformation process (e.g. link parsing, and PLN inference). However, it bears emphasis that a major goal of the ideas presented here is to enable the processes at the different ends of the transformations to work more closely together. For generally intelligent language processing, parsing and generation should be guided by semantic inference. This sort of linguistic cognitive synergy [13] should in principle be relatively straightforward given a solid implementation of the ideas presented here. A partially-parsed or generated sentence can be mapped into Atoms using rewrite rules, and the interpretation of the resultant Atom structure can be used to estimate the semantic viability of the sentence fragment. The original implementation of the link parser would not allow this sort of semantic guidance of

parsing, but a variant of the link parser using SAT solving to do the parsing has been implemented, which is much more flexible in this regard. Experimenting with this sort of dynamic should be fascinating, and should move us closer to generally intelligent language processing.

Another possible direction for development is to allow the link parser dictionary itself (which contains most of the link grammar framework, since link grammar is highly lexicalized) to be adapted via the system's experience. Changes to the link parser dictionary would then lead to automatic modification of the rewrite rules, and could be validated or refuted based on the consequences of these changes as determined by inference. This would eliminate any "hard-coded linguistic content" aspect from the OpenCog NLP process, rendering all such content free for cognitive modification, as one desires in an AGI system.

References

1. Adams, S., Goertzel, B., et al: Toward a roadmap toward human-level artificial general intelligence. Subm. for publication (2010)
2. Goertzel, B.: A pragmatic path toward endowing virtually-embodied ais with human-level linguistic capability. IEEE World Congress on Computational Intelligence (WCCI) (2008)
3. Goertzel, B.e.a.: Opencogbot: An integrative architecture for embodied agi. Proc. of ICAI-10, Beijing (2010)
4. Goertzel, B., Pinto, H., Pennachin, C., Goertzel, I.F.: Using dependency parsing and probabilistic inference to extract relationships between genes, proteins and malignancies implicit among multiple biomedical research abstracts. In: Proc. of Bio-NLP 2006. (2006)
5. Goertzel, B.e.a.: A general intelligence oriented architecture for embodied natural language processing. In: Proc. of the Third Conf. on Artificial General Intelligence (AGI-10), Atlantis Press (2010)
6. Lian, R., Goertzel, B., Et, A.: Language generation via glocal similarity matching. Neurocomputing (2010)
7. Goertzel, B., Et Al, C.P.: An integrative methodology for teaching embodied non-linguistic agents, applied to virtual animals in second life. In: Proc.of the First Conf. on AGI, IOS Press (2008)
8. Goertzel, B., Pitt, J., Cai, Z., Wigmore, J., Huang, D., Geisweiller, N., Lian, R., Yu, G.: Integrative general intelligence for controlling game ai in a minecraft-like environment. In: Proc. of BICA 2011. (2011)
9. Goertzel, B.: The Hidden Pattern. Brown Walker (2006)
10. Sleator, D., Temperley, D.: Parsing english with a link grammar. Third International Workshop on Parsing Technologies. (1993)
11. Voloshin, V.: Introduction to Graph and Hypergraph Theory. Nova Science (2009)
12. Cai, Z., Goertzel, B., Zhou, C., Zhang, Y., JIang, M., Yu, G.: Dynamics of a computational affective model inspired by drners psi theory. Cognitive Systems Research (2011)
13. Goertzel, B.: Cognitive synergy: A universal principle of feasible general intelligence? In: ICCI 2009, Hong Kong. (2009)