Natural Language Understanding with Knowledge

Jiquan Ngiam
May 2008

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Senior Thesis Advisor
Scott Fahlman (sef@cs.cmu.edu)

Abstract

This paper examines the problem of extracting structured knowledge from unstructured free text. The extraction process is modeled after construction grammars, essentially providing a means of putting together form and meaning. The knowledge base is not simply treated as a destination, but also an important partner in the extraction process. In particular, the ideas are implemented as a module closely tied to the Scone Knowledge Base system. I demonstrate that with a reasonable knowledge base and general construction rules, one can easily extract structured knowledge.

Keywords: natural language processing, construction grammar, knowledge representation

TABLE OF CONTENTS

1 INTRODUCTION

The ultimate goal of processing natural language in computer is really to be able to form an understanding of some piece of text and then reason over what has been understood. This paper aims to examine how we can move closer to this goal. The knowledge representation adopted is the Scone Knowledge Base (KB) system (Fahlman 2006). This is in contrast the to customary representations used for knowledge – first order logic or lambda calculus. The Scone KB system is a flexible efficient system that models knowledge in a semantic network.

I approach natural language understanding from the viewpoint that people are able to understand completely ungrammatical sentences (e.g. "john, mary lunch burritos"). Essentially, understanding natural language comes *mostly* from semantics. Hence, any approach to natural language understanding must first consider how the knowledge is going to be represented.

Construction grammar (Goldberg 2006) is a family of grammars that emphasizes the connection between syntax and semantics. They provide a linguistic basis for the approach adopted by this paper. In particular, the notion of constructions adopted most closely follows that of Radical Construction Grammar (Croft 2001).

This paper demonstrates an effective method of language processing by fusing knowledge with construction grammars. In particular, I will examine the extraction of knowledge structures from free text at a sentence level. These ideas have been implemented in LISP as a practical language-processing engine meant to be used in conjunction with the Scone KB system. The language engine is currently in use by other members of the Scone group.

2 SCONE KNOWLEDGE BASE SYSTEM

The Scone KB system represents knowledge as a semantic network. In this paper, KB elements are encased in curly brackets. For example, {physical object} refers to the Scone element that represents a typical physical object. An element can also have roles and relations specified over it. Furthermore, elements also have children or multiple parents and the roles and relations will be inherited accordingly. Scone efficiently supports multiple inheritance and exception.

Figure 1 shows how the sentence "Bush to fly over California fires" might be represented in a semantic network.

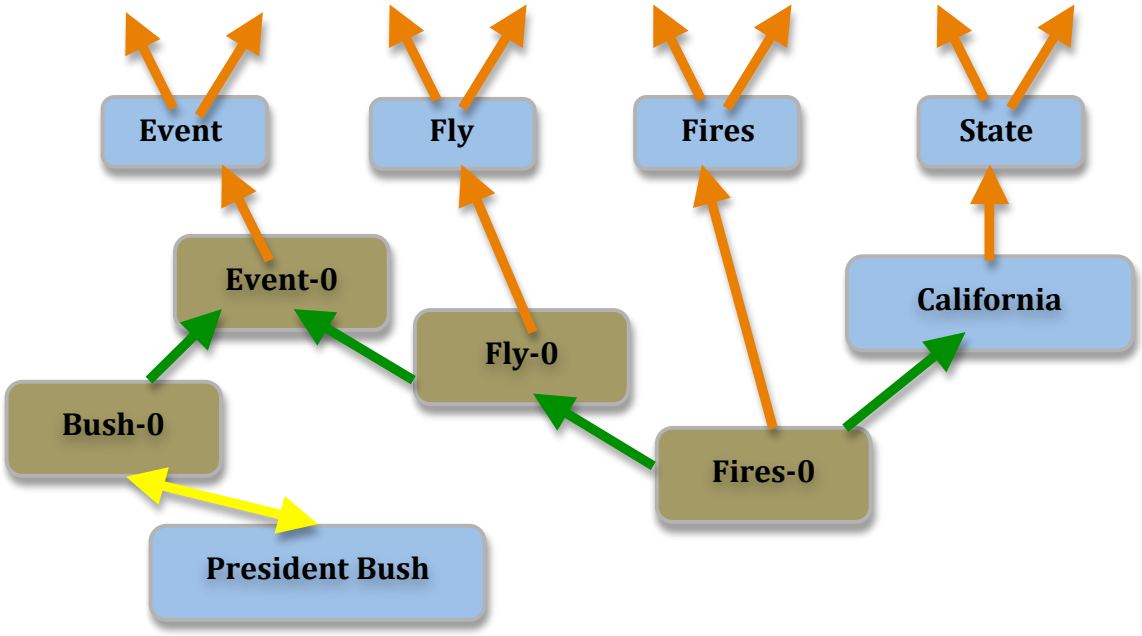

Figure 1: Example Semantic Network based on Scone

3 CONSTRUCTIONS

3.1 OVERVIEW

Constructions are basic pairings of form and meaning. Form refers to the surface information of an utterance. In some models of construction grammar, this includes phonological specifications, syntactical relations. On the other hand, meaning encompasses some form of structured knowledge such as schemas, frames, semantic networks etc. Constructions essentially represent expressions and their possible interpretations.

This approach to language processing distinguishes itself in that understanding becomes an integrated process from syntax to semantics. This is in contrast to classical approaches that have multiple black boxes doing specific tasks. However, at the same time, it requires a complex system to be built around these constructions so that all the information can be effectively and efficiently processed.

Unfortunately, there are few practical implementations of construction grammars. Bergen and Chang (2003) present Embodied Construction Grammar (ECG), a precise model for computational architectures. This model has been implemented in Simulation Based Learning and Language Learning models.

Forbus et. al (2007) demonstrate a system which learns by reading stories. The natural language portion of the system is based on Direct Memory Access Parsing (Martin and Riesbeck 1986). The phrasal patterns in DMAP are akin to constructions in a construction grammar framework.

This paper will be focusing more on a model of construction grammar that is able to extract structured knowledge from unstructured text. Hence, I adopt a simplified version of construction grammars, in which form is a pattern defined by a sequence of variable restrictions. On the other hand, semantics is defined by the related semantic network structure that the construction represents. The rest of this section will describe the model adopted.

3.2 VARIABLES AND ELEMENT RESTRICTIONS

In a construction, I use variables as bridges between form and meaning. Variables are scoped within each construction and appear both in the form and meaning. During language processing, variables will end up holding values based on the input text. They will then be used during the semantic evaluation.

LOCATION-ACTION CONSTRUCTION	
Variables	(*?u* {place}) (*?v* {action})
Pattern	"At" (*?u* {place}) (*?v* {action})
Semantics	*KB-set:* *?u* is-the {location} of *?v*

PERSON-ACTION-OBJECT CONSTRUCTION	
Variables	(*?x* {person}) (*?y* {action}) (*?z* {physical object})
Pattern	(*?x* {person}) (*?y* {action}) (*?z* {physical object})
Semantics	*KB-create:* *?w* = a new instance of *?y* (an action) *KB-set:* *?x* is-the {agent} of *?w* *KB-set:* *?z* is-the {object} of *?w*

Figure 2: Example Constructions

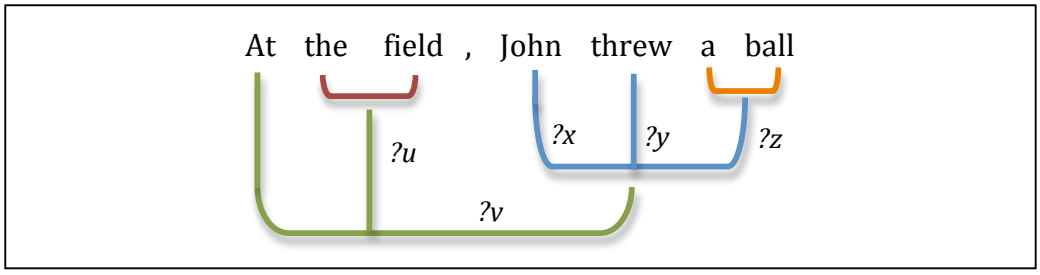

Figure 3: Example Matched Sentence

When they appear as part of the form of the construction, variables are associated with an element-restriction. This restriction bounds the possible values the variable can hold. In the Location-Action construction (Figure 2), the variable *?x* is restricted such that only elements which are {location} can be matched. Figure 3 demonstrates a match involving both example constructions.

Element restrictions are normally specified by primitive elements (e.g. strings, numbers) or internal Scone elements. Furthermore, one can specify restrictions over the role and relations of the elements – for example, one could set up a variable that only matches {person} with {favorite color} being {red}.

3.3 FORMS AND PATTERNS

The form of a construction is essentially a pattern defined by an ordered list of variables and constants. For a construction to match, there must exist an input token to match each variable in the pattern. Furthermore, for a series of tokens to match a construction, the matches must appear in the same order in which the pattern is defined. For simplicity, I assume that variables do not repeat in the pattern.

An input stream of tokens is ideally a formed by elements already within the KB. However, tokens can also be words, which are then resolved by the language-processing engine. This resolution involves a disambiguation process, which is covered later.

One important notion is that when a construction matches a stream of input tokens, it can produce a new token to replace the tokens matched. This allows for a stream of tokens to be matched to some hierarchical composition of constructions. In the example sentence (Figure 3), since the *person-action-object* construction produces *?w*, an action, it can be used to match part of the pattern for the *location-action* construction.

3.4 INSTANTIATION

If a stream of tokens matches a construction, then one can choose to instantiate the construction to effectively produce the semantic network structure associated with the construction. Furthermore, the construction will also return a "head" element, which in essence refers to whole entity produced by the construction. For example, if "green elephant" were matched, then the construction would return an elephant instance with having color green.

3.5 HIERARCHICAL REPRESENTATION

Across constructions, one would often find overlapping senses and semantic information. For example, constructions for {pull}, {push} and {shove} refer to physical interactions between an {animate being} and another {physical object}.

Furthermore, even considering just a single word such as "open", we have multiple senses of the word under different uses – "open house", "open the box", "open source", etc. However, in all these cases, there is an overarching sense of the word in play. There is something that represents a container, items in the container and the act is to extract or provide some form of access to these items.

To efficiently represent these overarching senses, my model allows constructions to be hierarchically specified such that constructions inherit the semantics senses of their parents.

4 LANGUAGE PROCESSING ENGINE

4.1 MATCHING ENGINE

In effect, constructions resemble Context-Free Grammar rules, with the specifications involving KB elements. Since each construction is paired with a semantic representation that specifies the knowledge structure that the construction produces, constructions can be viewed as a specification of a CFG rule.

This paper presents a language-processing engine implemented in LISP based on the Earley parser (Earley 1970). Interpretation of a series of tokens is performed in three phases – a pre-processing phase, a core matching phase and finally instantiation.

In the pre-processing phase, each variable of the pattern in each construction is linked to all other constructions that can produce some Scone structure that can satisfy the variable's restriction. This sets up constructions in a way that represents a context-free grammar.

The core of the matching engine is largely based on the Earley parser. In the matching phase, the interpreter runs left to right accumulating only constructions that are valid up till that token. In order to keep the space/time requirements manageable, the partially matched constructions are scored, ranked and pruned.

The matching phase will conclude with all possible matches. From these, the best construction is selected based on its rank and instantiated. This happens in a recursive fashion depending on the composition of the final match.

4.2 SCORING AND SELECTION

In practice, one would find that a single sentence would match to multiple constructions. With scoring mechanisms, one can better select the construction to use. Furthermore, scoring permits flexibility in the system. For example, partial matching described later is implemented based in conjunction with the scoring mechanisms.

Constructions are scored by a weighted average of various sub-scores. The sub-scores for a construction include

1. *Length of the construction*
   Scores are assigned in proportion to the length of the construction. Longer constructions are thus preferred. The hypothesis is that longer constructions are more specific matches since they have more variables to be filled.

2. *Completion (matched) status of the construction*
   At any given time during the matching process, many constructions will only partially matched. Scores are assigned to constructions in proportion to the percentage of its pattern that has already been matched. This score is meant to influence selection effects if a limit is placed on the total number of possible constructions.

3. *Base score for the construction*
   Each construction can be pre-assigned a base-score. By default, all constructions have a base-score of 1. This base-score is intended to reflect the frequency of use of the construction. The idea is to have this score change as the system encounters more sentences and learns from feedback.

4. *How well the underlying tokens and constructions match the pattern*
   A cumulative score is assigned by computing how well the underlying tokens and constructions match the pattern. If a variable is satisfied by another construction, the score from that construction is then used to compute this value. In the case that an actual token is used to fill the variable, the specificity of the match comes into consideration.

4.3 PRACTICALITIES

One of the major objectives was to provide an engine that was practical and applicable to future projects that wish to process unstructured text together with Scone. Many options have been implemented to this extent to make the system extremely flexible and easy to use.

For example, one would find that the same construction could often apply when matching to both individual variables and a set formed by individuals of that type. This is handled as a special case in the construction engine and a variable option can be set to enable it.

4.4 PERFORMANCE

The construction engine performs extremely efficiently with all demonstrations with the code taking no longer than a second for each sentence.

In general, the runtime of the engine can be bounded by the scoring mechanisms. The mechanisms allow one to easily limit the number of constructions allowed. The entire engine has a complexity of $O(c \cdot n^3)$ where n is the number of input tokens and c is the number of constructions.

Since n is normally 15–20 in general English sentences, one can expect very reasonable performance with the engine.

5 CONTEXT-DEPENDENT PROCESSING

5.1 CFG PROBLEMS

Unfortunately, the context-free grammar approach for this model of construction grammar is insufficient on its own. Firstly, I note that there is no real distinction between terminals and non-terminals in this model. All terminals and non-terminals are simply KB elements. Next, the symbols in the language follow a hierarchy defined by the KB. For example, the {action} element, when used as a variable restriction, should also match all sub-types of actions, such as {kick} and {push}. As a result, a direct application of a CFG parser will be insufficient. These issues become evident in two situations highlighted in this section.

To be more precise, the output of construction is often used to fill a variable in another construction. However, whether a construction's output can be used to fill a variable is often not certain until the actual input tokens are available. Hence, a degree of context-dependent processing is required.

For example, consider a construction that has a variable (*?x* {action} with {object}-role set to an {edible}). Then, the *person-action-object* construction (Figure 2) cannot be used as the variable filler unless that it sets the {object}-role of the action to an edible. This in turn depends on the variable *?z*, which is based on the input.

5.1 CHARACTERIZING CONSTRUCTION OUTPUTS AND HANDLING SETS

We often need to form sets while processing the text. For example, a recipe text might indicate mixing the tuna, mayonnaise, celery and salt. Ideally, we will have very general constructions that form sets for all kinds of elements.

On the other hand, one can imagine having kitchen action constructions that only work over sets that contain only edible items. However, the output of the set-creation construction can only be used in the kitchen action construction only when the created set has edible members.

As a result, after set construction, we need to be able to characterize the typical member of the set so as to be able to determine whether it can be used as part to fill another construction. Hence, whether a construction can be applied depends on the context or actual input tokens.

In a more general setting, we need to characterize the roles and relations of elements based on context as well. For sets, this simply means characterizing the {member} role of the set.

5.2 General and Specific Constructions

Another issue arises with general and specific constructions. General constructions are constructions meant to be applicable in a wide variety of ways. For example, a color construction that matches ({color} {physical object}) can be one that creates an instance of the physical object with the color role appropriately set. Specific constructions are more domain-oriented. For example, one can talk about kitchen actions which use only {kitchen appliance}.

Certainly, a "green oven" is still a kitchen appliance, but it also matches the color construction. Conversely, a "green leaf" is not an appliance but matches the color construction. Hence, to determine whether the color construction can be used to produce an object for {kitchen appliance}, we need to determine what the color construction produces. This is context dependent.

To resolve this issue, I categorize constructions in two ways – those that can be determined prior to processing and those that need to be checked for context. In order to keep the number of constructions processed to a manageable size, the constructions for the latter category are kept to those that produce elements which are super-sets of the expected type. This does not cover all the cases but it is hypothesized that they are sufficient for most purposes.

During processing, when constructions are matched, they are also linked to some representative element pre-existing in the KB. This representative element is then used for checking whether a later match is possible. This simplified approach seems to be able to handle most of the cases, especially when the constructions specified are not too complex.

A true solution to this problem would be to actually instantiate constructions during processing and use those instantiations as representative elements. These instantiations are later removed when processing completes. However, this is not implemented.

6 GENERALIZATION

6.1 OVERVIEW

Generalization for constructions was motivated by how learning of language develops. Children often learn that certain patterns of language translate into the same meaning and then generalize over these patterns. An example of this generalization happens with past-tense inflexions. Goldberg (2006) also suggests pre-emption and openness as a means of non-linguistic generalization. These ideas are closely related to the generalization approach taken in this paper.

In the context of this paper, the aim of generalization is to empower a system to perform one-shot learning – to be able to apply existing constructions to novel sentences and words. For example, consider constructions that match specifically only "blue pencil", "orange highlighter" and "black pen". In this case, a possible target construction to learn is one which matches (*?x* {color}) (*?y* {writing tool}) and proceeds to assert that *?x* is the color that *?y* produces.

6.2 GENERALIZING OVER A SET

In this paper, I also explore an algorithm for generalization over a set that uses spreading activation. The aim of generalization over a set is to provide a description of a typical element of the set. This description can be simply be a single KB element, thus indicating that all sub-types of the element are regarded as being in the set. It could also be a set of KB elements, which implies that the original set had different clusters in it.

The generalization algorithm places an amount of activation on each element in the set and proceeds to pass this activation to its parent nodes via a transfer function. The algorithm is analogous to a voting system.

Let us define

$$\alpha(x) = activation\ \ at\ \ node\ \ x$$

$$f(x,y) = amount\ \ of\ \ activation\ \ from\ \ x\ \ to\ \ y$$

$$N(y) = \#\ \ of\ \ direct\ \ children\ (activation\ \ sources)\ \ of\ \ y$$

$$p = generalization\ \ eagerness\ \ parameter,\ 0 < p \leq 1$$

$$\alpha(x) = \begin{cases} 1 & if\ \ x\ \ is\ \ in\ \ the\ \ set \\ \sum f(y,x) & otherwise \end{cases} \qquad f(x,y) = \frac{\alpha(x)}{\left[N(x)\right]^p}$$

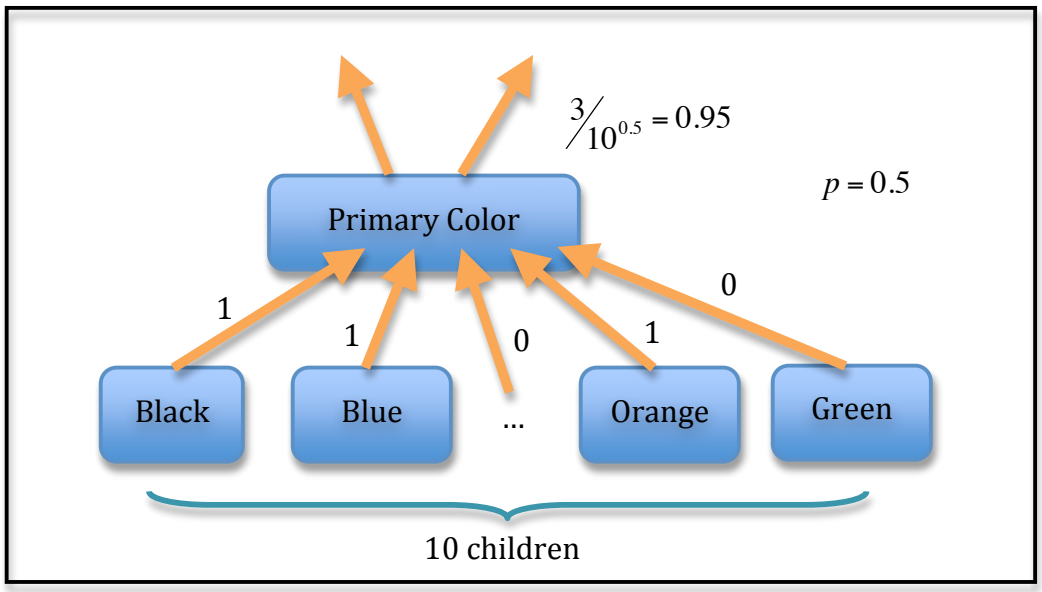

Figure 4: Activation Example

One can interpret this in terms of a voting system, where by $\alpha(x)$ represents the voting power of x and $f(x,y)$ represents the effective vote of x. Hence, the more children (source activations) that x have, the more it needs to increase its effective vote. Figure 4 demonstrates this idea of activation pictorially.

Consider an effective vote of 1 to be a base-line (since all nodes in the set have such an effective vote already). For a node to have an effective vote of at least 1, $f(x,y) \geq 1 \Rightarrow \alpha(x) \geq [N(x)]^p$

Since $0 < p \leq 1$, $0 < [N(x)]^p \leq N(x)$. Hence, p controls the amount of votes required before the effective vote of the node becomes 1.

The basic algorithm basically activates all the nodes based on the input set and removes the node with the highest activation and the nodes in the input set that have been covered (subtypes) by this node. This repeats until all the nodes have been covered.

This approach works well in practice. Figure 5 demonstrates how a set of elements can be effectively generalized or described using this algorithm. The example had $p = 0.25$ for all nodes and used the Standard Upper Merged Ontology (Niles & Pease 2001), which was imported into the Scone KB.

Input *(in no particular order)*

```
{Spain} {France} {Italy} {Belgium}
{Lieutenant} {Colonel} {Military General}
{Male} {Female}          {Red} {Blue} {Yellow}
{January} {February} {March} {April}
{Right Angle} {Acute Angle} {Polygon} {Cone} {Cylinder}
```

Output

```
{Sex Attribute} {European Nation} {Military Officer}
{Primary Color} {Month} {Geometric Figure}
```

Figure 5: Generalization Example

6.3 GENERALIZING CONSTRUCTIONS

From generalization of a set, one can also experiment with generalizing constructions. The intuition is that if two constructions map variables in a way such that the resulting knowledge structure is the same, then, one could generalize over the variables to form a resulting construction that can be applied to other cases.

For example, one might have very specific constructions that only match phrases such as "chocolate box", "candy box", "metal box", "post box" etc. One can then easily see a few constructions going on here:

{food} box	⇒	container for storing {food}
{material} box	⇒	container made of {material}
post box	⇒	container for sending letters

Given enough examples (in the form of constructions) of {food} box, one can use the generalization algorithm described in 6.2 to then create a construction that exactly matches {food} box. Furthermore, the eagerness of this generalization can be parameterized as described earlier. This form of generalization serves as a means of learning new constructions and even one-shot learning of very general constructions.

These ideas have been implemented as a proof-of-concept. The algorithm implemented takes in a set of constructions and proposes new constructions based on that set. The key idea is to cluster constructions together by running their variable restrictions using the algorithm in 6.2. This is repeated recursively for each cluster until we have a bucket of constructions that essentially have generalizations specified over each element. This provides a new construction.

This form of learning is unfortunately very limited since it requires constructions to be of the very similar forms. One can imagine a specific construction that only matches "opened the chocolate box" (i.e. the entire phrase is the form itself, "chocolate box" is not matched by a separate construction). This algorithm will not be able to generalize over such examples. For these examples, one would first need to discover the inner structure of the sentence.

7 PARTIAL MATCHING

7.1 OVERVIEW

While processing real world text, one would not normally expect perfect matches from the constructions. First, there would be chunks in the text that do not match any construction. Second, sentences are often fragmented so that constructions can only be partially matched.

7.2 SKIPPING TOKENS

A natural approach to partial matching is to skip input tokens when they do not match. This could be due to the token being unrecognized (a new word) or the constructions being considered do not expect the token. However, even when tokens are skipped, the construction that matched overall must still have valid tokens matched to it, hence, we are still able to generate a corresponding knowledge structure.

This idea of partial matching has been implemented in the engine by influencing the scoring function. When tokens are skipped, a penalty (negative score) is added to the matching score. Furthermore, a series of contiguous skips suffers a greater penalty that increases exponentially.

7.3 PARTIAL CONSTRUCTIONS

The alternative to skipping tokens is to skip parts of the form in a construction. However, this is much more tricky since constructions represent an expression of some abstract meaning. It is hard to determine which part of the pattern for the construction plays a key role in determining the meaning. I note that these ideas have not been implemented.

On the other hand, partially matching constructions allow for interesting analysis of phrases. In a partial match, one could create placeholder elements for the unfilled variables. These elements can then be filled in later when more information is available. For example, in processing a construction that expects a {country} and it is skipped over, one might be able to backtrack and find an unmatched string to suggest that that string might represent a {country} that is not in the KB.

8 WORD SENSE DISAMBIGUATION

In processing natural language, the same written word can often have multiple ambiguous meanings. From the perspective of construction grammars, one could create a construction for each word that converts the written form of the word into its semantic counterpart. However, this would result in an explosion in terms of the number of constructions.

In classical natural language approaches, word sense disambiguation is often performed prior to parsing. However, this paper approaches disambiguation differently – from the perspective of constructions.

Instead of having to disambiguate words prior to processing, I approach disambiguation by doing it only when it is required. In the case of the language engine, this happens when we wish to check whether a word string can be used to fill some variable in a construction. At this point of time, we have information on the variable we are trying to fill and this gives the disambiguation process a hint on which sense of the word to take. For example, the word "place" can both refer to an {action} or {location}. But, these two concepts are semantically separated; therefore the variables in the constructions will reflect the sense of the word we want.

However, I note that even with this approach, ambiguities can occur and there might times where the system can select the wrong sense in order to fulfill a construction. On the other hand, such an approach coupled with collocation considerations would be able to give good guesses. In order to handle bad guesses, an ideal system should perhaps be able to backtrack and change its original decision on the word sense.

9 DISCUSSION

One might be motivated to ask how this approach differs from the CFG parsers that have been widely used for syntactical parsing. Syntactical parsing with CFG parsers have operated by first tagging words in sentences according to some basic types and then building a parse tree over several rules. The approach here does not perform any tagging, but instead uses elements from a KB in place of them. This allows for a more general form of representation (the elements in the KB can be tagged) and further, allows for establishing a link to semantics.

10 FUTURE WORK

A more precise formalism of the system described in this paper might be useful for theoretical reasons. Rather than having scoring mechanisms, one could use notions from stochastic context-free grammars. Stolcke (1995) demonstrates a probabilistic Earley parser with the same complexity bounds as the original parser. With more precise notions of the relations between the constructions, it might be easier to characterize and perform some form of automated learning over these parameters.

Constructions form the key of this approach to language understanding. However, coming up with constructions for a whole language is a tedious task. Hence, automated learning of constructions is certainly an interesting area to explore. They key problem in learning constructions is that one needs to figure out the underlying grammatical structure of the language. Chang and Gurevich (2004) present a framework for learning constructions in Embodied Construction Grammar.

The work here has been concerned with sentence level processing of language. Ultimately, this construction grammar based system should fit into a dialogue understanding system. However, a new set of problems arises at a dialogue level. The hope is that the knowledge structures created at the sentence level together with the KB can help resolve some of the problems.

11 ACKNOWLEDGEMENTS

I would like to thank Professor Scott Fahlman for his valuable guidance and advice. I would also like to thank Cinar Sahin, Ben Lambert and members of the Scone Group for their feedback and help in understanding Scone.

REFERENCES

Chang, N. and Gurevich, O. (2004). Context-Driven Construction Learning. *Proceedings of the 26th Annual Meeting of the Cognitive Science Society*. Chicago, IL.

Croft, W. A. (2001). *Radical Construction Grammar: Syntactic Theory in Typological Perspective*. Oxford: Oxford University Press.

Bergen, B. K. and Chang, N. (2005). Embodied Construction Grammar in simulation-based language understanding. In Jan-Ola Östman and Mirjam Fried, eds. *Construction Grammars: Cognitive Groundings and Theoretical Extensions*. Philadelphia, PA: John Benjamins. 147-190.

Earley, J. (1970). An efficient context-free parsing algorithm, *Communications of the Association for Computing Machinery*, *13*:2:94-102.

Fahlman, S. E. (2006). Marker-Passing Inference in the Scone Knowledge-Base System. *First International Conference on Knowledge Science, Engineering and Management (KSEM'06)*, Guilin, China.

Forbus, K. D., Riesbeck, C. K., Birnbaum, L., Livingston, K., Sharma, A., Ureel, L. (2007). Integrating Natural Language, Knowledge Representation and Reasoning, and Analogical Processing to Learn by Reading, *Proceedings of the 22nd Conference on Artificial Intelligence (AAAI-07)*, Vancouver, BC.

Goldberg, A. (2006). *Constructions at Work: The Nature of Generalization in Language*. Oxford: Oxford University Press.

Martin, C. E. and Riesbeck, C. K. (1986). Uniform Parsing and Inferencing for Learning. *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, PA. 257-261.

Niles, I., and Pease, A. (2001). Towards a Standard Upper Ontology. In Chris Welty and Barry Smith, eds. *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001)*, Ogunquit, ME.

Stolcke, A. (1995). An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics, 21*:165-201.

APPENDIX – RECIPE EXAMPLE

The following examples on recipes require the core and cooking knowledge bases to be loaded.

Example Construction:

```
(define-construction

  ;; Name
  "a and b - forms a set"

  ;; Parent
  nil

  ;; Variable List
  ((v1 {edible} instance) (v2 {edible} instance)
   (v3 "and" "or" string)
   (v4 {set} (inst-code (new-indv nil {set}))))

  ;; Pattern List
  (v1 v3 v2)

  ;; Trigger List
  ((*c-head* v4))

  ;; Scone Operations
  (new-is-a (get-the-x-role-of-y {member} v4) {edible})
  (x-is-a-y-of-z v1 {member} v4)
  (x-is-a-y-of-z v2 {member} v4))
```

In this construction **"a and b – forms a set"** is the English name of the construction. There are 4 variables in the construction – **v1** thru **v4**. **v1** and **v2** are used to match to two edible elements, while **v3** is used to match to either the string **"and"** or **"or"**. **v4** is not used in the pattern but is produced by the code, the **inst-code** option of **v4** has been set to produce it. The **head** of the construction is **v4**, and there are 3 Scone operations which follow, essentially saying that the set produced represents a set of edible objects and that **v1** and **v2** are part of this set.

Note that the representation adopted for the actions are simplified so as to focus on the use of the construction engine.

The following is the original text of a recipe:

Chocolate Chip Butter Cookies

Melt butter in a microwave or double boiler; stir in vanilla. Cool completely. In a large bowl, combine flour and sugar; stir in butter mixture and chocolate chips (mixture will be crumbly). Shape into 1-in. balls. Place 2 in. apart on ungreased baking sheets; flatten slightly. Bake at 375 degrees F for 12 minutes or until edges begin to brown. Cool on wire racks.

The constructions defined (see appendix) allows for the following sentences to be matched as expected.

```
(cp-interpret "Melt butter in a microwave or double-boiler")
(cp-interpret "stir in vanilla")
(cp-interpret "Cool completely")
(cp-interpret "In a large bowl , combine flour and sugar")
(cp-interpret "Stir in butter mixture and chocolate chips")
(cp-interpret "Shape into 1 in. balls")
(cp-interpret "flatten slightly")
(cp-interpret "Cool on wire racks")
(cp-interpret "Place on ungreased baking sheets")
(cp-interpret "Place 2 in. apart on ungreased baking sheets")
(cp-interpret "Bake at 375 degrees F")
```

The constructions also generalized to various other sentences in other recipes.

```
(cp-interpret "In a bowl , mix together tuna , celery , mayonnaise
               and salt")
(cp-interpret "Mix into cheese mixture")
(cp-interpret "Combine the chocolate chips and cream in a medium
               metal bowl")
(cp-interpret "Stir pecan halves into the chocolate")
```

However, there were many other sentences that could not successfully processed as well. Some examples of unsuccessful sentence from other recipes follow.

```
When cycle is completed, turn dough onto a lightly floured surface.
Cover and let rise in a warm place for 30 minutes.
Bring a large pot of lightly salted water to a boil.
```

These sentences could not be processed mainly due to expressions that were not yet captured by the constructions or representation used in this demonstration. On the other hand, it is not difficult to conceive the constructions required for them.