A Table Based Inference Engine for Natural Language Processing

Dave Davies, 2005

Abstract

This paper, the third in a series [Davies 2005a, 2005b], addresses the problem of Natural Language Inference with a strong emphasis on Software Engineering and, specifically, table based techniques and application of the Link Matrix data structure. We look primarily at the resolution or verification process and issues related to flexibility, efficiency and performance in large Knowledgebases. The application domain is Natural Language Agent systems.

Introduction

There has long been a recognition that computer languages should be developed to describe problems rather than just their solutions [eg. Winograd 1979]. Natural Language systems are an obvious candidate for this role. Algorithms for NP parsing and inference have been extensively discussed in the literature and many NL systems have been constructed and reported [Cunningham 1997]. Despite this extensive body of work NL systems have still not reached the point where they form a pervasive component of Information and Knowledge Systems. A primary motivating force behind this work is to enhance the usability, flexibility and connectivity of NLP in a way that can enable it to perform such a role.

Consequently, Software Engineering for the Natural Language inference problem is viewed here as having four principal objectives or requirements:

1. Language scope: The language definition, a subset of English in our case, should be broad enough to adequately describe the problem domain in forms that are intuitively accessible to the end user.
2. Flexibility: The parser grammar and production system policy settings should be directed by rule sets that are human readable and editable. They should be capable of runtime update or modification. Code modules should conform to a standard plugin format allowing alternative inference policies or algorithms to be selected to suit a particular inference task or problem domain and provide connectivity with the underlying operating system and networks.
3. Performance: A general requirement for efficient inference is 'minimal searching'. Only those statements in the Knowledgebase (KB) that are directly relevant to the current inference task should be accessed. A form of associative memory should be used to optimise access to transient information in the inference process.
4. Stability: Both the Parser and Inference Engine should be robust, coping in an orderly manner with ill-formed statements or queries.

We look at the inference resolution or, more specifically, the unification process and issues related to efficiency and performance in large Knowledgebases. Data and architectural structures have been developed to specifically address key issues of NLP [Davies 2005a]. The algorithms used have been designed, or modified, to work closely with the underlying structures. This process has evolved through several iterative stages with each iteration tightening the partnership between algorithm and structure.

System Architecture

The current version of this system has been constructed using Java technology. This choice was based on Java's cross-platform capability but also on considerations such as system and network connectivity, security, Java's large library infrastructure and the availability of the JavaSpeech API that defines interface conventions for rule-based systems and speech interfaces.

In overview, the system uses the Model-View-Controller pattern to provide a layered architecture. Here we consider only the Model layer. Within the Model layer a sublayer structure is used along established lines [eg. Mizoguchi 1982] with text conversion, parsing, inference and dialogue control layers. A review of NL architecture is provided in [Cummingham 1997]

To provide a flexible runtime 'plugability' a general Action class defines common inputs, output and activation for inbuilt Actions such as Parse, Respond (to query), Resolve (inference) and Transform (statement form). This approach allows for the future inclusion of a runtime choice of Parser and other Actions. Java's Reflection capability allows new Action subclasses to be defined and instantiated at runtime from library classes providing a flexible interface with the GUI, operating system or network functionality provided by the libraries.

The elements of the system of most direct relevance to the Inference Engine are the Link Matrix and HashMap based data structures described previously [Davies 2005a]. The Link Matrix is a structure that links all instances of a given word as a linked list and, in doing so, links all statements in the Knowledgebase that contain that word. All words are linked. The link structure allows the performance of KB access in the inference process to be optimised. Only statements that are directly related to the inference process are accessed.

A Whiteboard structure is provided in the system as an Object Oriented and adorned variant on the conventional Blackboard structures that have been used in many rule-based systems. Some of the adornments provided are: structured HashKey objects that facilitate tracing and debugging; hash key grammar conventions that allow complex keys to be constructed in a systematic, and thus predictable, manner; infrastructure for hashtable based arrays, lists and trees; automatic creation of Arrays; and nested Iterators that are particularly useful in dealing with ambiguity.

The Parser

The parser has been described in detail in a previous paper in this series [Davies 2005b]. Here we need only look at the parser output. The parser generates two data structures on the completion of a parse. These structures are stored by the parser in a hashtable using a key generated from the parsed statement and are thus available to any module, such as the Inference Engine, that has the sentence and can thus access the necessary key.

The first structure is a parse tree illustrated in Figure 1 with the *sentence* label as the root node and the Part of Speech representation of the original sentence as the leaves. In between, the parser stores and information on sentence structure that may be of value in the inference process. Efficient guided and unguided tree search methods are provided. The Inference Engine is able to request, for example, the subject phrase of the condition fact of a conditional statement.

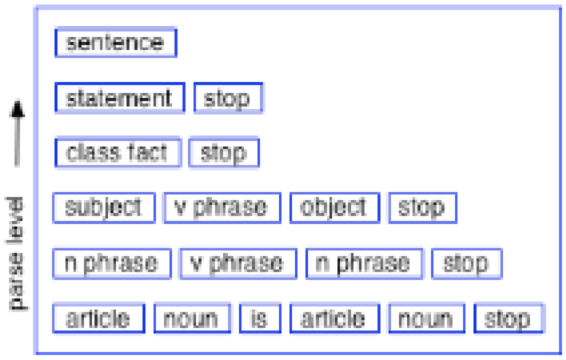

Figure 1: Parse tree for the sentence: ‘A dog is a mammal.’

In the second data structure provided by the parser the parse tree is represented in a compact integer form with one integer for each word in the sentence. Each integer encodes a reduced trace of the path from the root to that leaf. This compact form discards all but four of the parse levels keeping information at what may be described as the phrase, clause, fact and sentence levels.

When the KB is being accessed in the inference process this coded form provides an efficient check for the syntactic context of a word through a simple integer comparison, bit-masked if necessary to broaden the context.

The Dialogue Manager

In the dialog manager the inference problem is posed as the verification of a *query_statement* that has been derived through a transform, or sequence of transforms, from an initial *query*. For example the *query* ‘Does Tom eat meat?’ is transformed to the *query_statement* ‘Tom does eat meat.’.

The Inference Engine

The Knowledgebase (KB) used in the inference process consists of lexical, logical and semantic rules such as: ‘Devours means eats.’ or ‘A cat is an carnivore’ and specific data facts such as ‘Tom is a black cat’ or ‘Tom eats meat’. The inference process we are addressing is the use of known statements about the world to unify, through either logical or semantic equivalences, two specific statements: a statement to be tested and a set of known facts.

The unification process consists of two main phases: Expansion and Test. In the Expansion phase alternative interpretations of each phrase of the ‘*subject verb_phrase object’* form of the query statement are derived from either equivalence statements (e.g. ‘A means B’) or class definitions (e.g. ‘A is a B’) from the KB. The test phase combines alternate phrases to form new forms of the original statement and tests to see if they are present in the KB providing either logical or semantic support for the inference

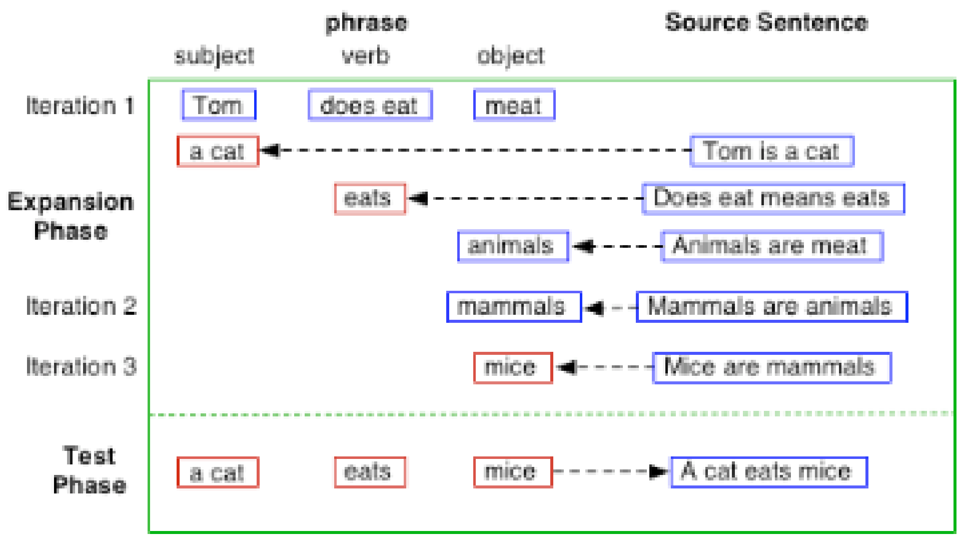

Figure 2: The expansion and test phases for the statement: '*tom does eat meat*'

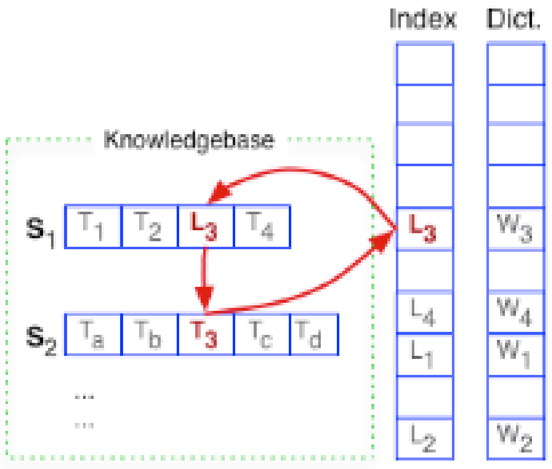

Figure 3: The Link Matrix

The Expansion Phase

The search for alternate phrases in the KB is performed using the Link Matrix illustrated in Figure 3 [Davies 2005a]. It is a performance and space efficient representation of the KB. It consists of htree parts, a dictionary, an index and the Knowledgebase. Word information (ASCII, POS etc.) is stored in the dictionary table. For each word in the dictionary a second structure, the index, stores a link to an instance in the KB.

Each sentence in the KB is represented by an array of links - one for each word in the sentence. Each link points to a previous instance of that word which, in turn, links to another instance in a circular list. The dictionary index entry for the word is included in this list. This provides the search with linked lists of the instances of each word in the target pattern. The word lists link all sentences in the Matrix that contain the target words and the set of sentences in each word set is the target set.

To optimise phrase matching performance, the target word with the lowest instance count in the KB is used as the start word in the match. The sentences containing that word are tested in turn for the target phrase. The integer parse array generated by the parser is used to test the suitability of each target word instance to ensure that it is in the correct syntactic position in the correct sentence type. If the context is correct then pattern matching is attempted using a conventional algorithm. The complexity of the search is thus proportional to the product of the smallest instance count in the target and the number of words in the target.

The Test Phase.
In the Test phase the sets of alternate phrases generated in the Expansion phase are combined exhaustively to form candidate solution statements, logically or semantically equivalent representations of the target statement. The KB is then accessed to see if any of these atatements are presented as fact. For example, using the phrases shown in Figure 2, a statement such as '**A cat + eats + mice**' can be formed and used to generate a key. This key can then be used to access the statement in the KB if it is present.

When a candidate is found in the KB the inference has been confirmed and the process is either terminated or allowed to continue to find further solutions. The two phases may be repeated iteratively until all solutions have been found. If conditional statements are encountered the *condition fact* is verified recursively. Under the current inference policy the verification is delayed until the *consequent statement* is confirmed as needed in a resolution.

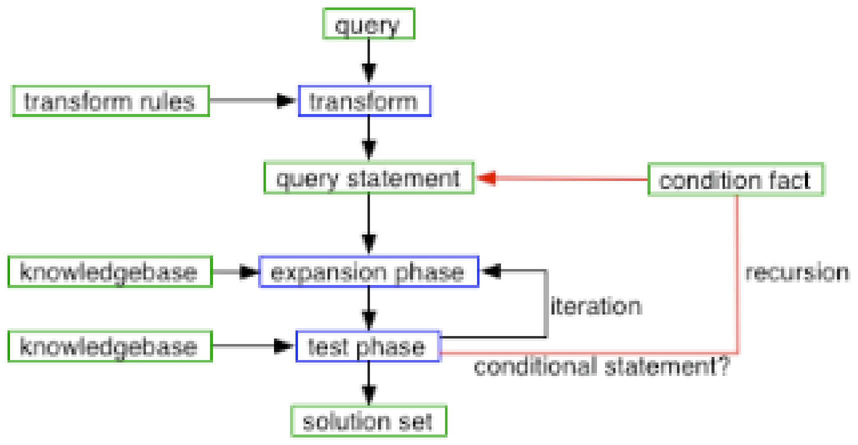

Figure 4: Overview of the inference process

In both phases the whiteboard is used to store intermediate results. In the case of recursion the whiteboard is shared by each recursive step so that the partial results of one recursion can be accessed by any other to avoid duplication or circular referencing.

Output
The full resolution process is outlined in Figure 4. The Test phase returns a solution set to the caller or, alternatively, any object can register as an Observer to receive messages on the

progress of the inference process and control it. This provides the potential for terminating runaway inferences or, more generally, runtime control of inference policy.

The solution set contains one item for each solution found. A solution is presented in the form of a linked list of the antecedents for each phrase as shown in Figure 2. This enables tracing and display of the inference process to be performed retrospectively.

Conclusions

Acknowledgements

I would like to take this opportunity to express my appreciation to the many people who have helped bring this project to its current state. It's inception was made possible by the entrepreneurial vision of Bill Fisher and the financial support of the Australian Industrial Development Incentives Board. Its development has been greatly facilitated by the skill and dedication of Arthur Zawadski, Xiaoming Zhang and Darren Hitchman though many others, particularly the graduate students in the University of Canberra IT Project course, have provided both energy and valuable insights and feedback.

References

Winograd, Terry **1979**, Beyond Programming Languages, Communications of the ACM, 22(7).
Mizoguchi, Fumio and Kondo, Shozo **1982,** A Software Environment for Developing a Natural Language Understanding System, COLING'82, North-Holland.
Cunningham, Hamish; Humphries, Kevin; Gaizauskas, Robert; Wilkes, Yorick **1997**, Software Infrastructure for Natural Language Processing, Proceedings of the fifth conference on Applied Natural Language Processing, Morgan-Kaufmann
Davies, David R. L. **2005a**, Data Structures for Natural Language Processing, Paper 1 in this series
Davies, David R. L. **2005b**, A Table Based Parser for Natural Language Processing, Paper 2 in this series