An Empirical Modelling approach to Natural Language Commands

0218982

Abstract

The paper details an approach to Natural Language Processing (NLP) based on the principles of Empirical Modelling (EM). The suitability of EM as a tool for working with natural languages is discussed as well as how NLP can integrate EM ideals. A definitive notation for making use of natural language in EM IS presented, followed by an overview of an implementation, paying particular regard to how it makes use of the features of EM.

Keywords: Empirical modelling, natural language processing

1 Introduction

Natural language processing and Empirical Modelling are two seemingly unrelated topics, yet the latter provides an interesting base on which to build a method for exploring the use of natural language as a means of interacting with computers.

The words of computers and people are very different, and traditionally the onus has been on the human side to adapt in order to make interaction between the two possible. The use of natural language as a user interface should remove much of the burden from the user and has been a long-standing ambition of many computer scientists. During the past couple of decades much work has gone into constructing such interfaces but commercial success has been elusive.

Methods of natural language processing are traditionally based either on formal grammars or, as is now more common, statistical methods. Formal grammars generally lack the flexibility required for NLP and statistical methods, by their nature, work on very large amounts of data, making them unsuitable for use with smaller applications.

This paper provides an overview of an approach to natural language processing that has been combined with the ideals of Empirical Modelling to produce a more informal technique, one that provides flexibility and allows for exploration of language for use within smaller domains. It first provides an overview of the technique, followed by an interpretation based on EM. The relationship between the two is then discussed and a definitive notation introduced. Finally, an implementation is presented and possible future extensions considered.

2 An approach to NLP

The approach to natural language processing that is used in this paper is based on command interpretation within limited domains (Rawlinson, 2005) and is well-suited for use with Empirical Modelling. It relies on identifying semantic entities within the input and using the combination of these, along with any keywords present, to match sentences to commands. The recognition of entities is based strongly on dependency, carrying out actions on identification to convert text to a more usable form. The success of this approach relies on two assumptions:

1. That although natural languages do not follow a prescriptive grammar, sufficient formal structure should exist describing individual semantic entities to be able to identify them and deduce their meaning.

2. That within limited domains, all possible actions should be distinct enough to allow input sentences to be mapped to individual commands based on the semantic entities and keywords identified in the input.

These assumptions are far from unreasonable, formal structure, at least at some level, is a basic requirement of communication. Although natural language can appear blah informal, once individual semantic entities are identified the formal structure becomes much easier to perceive. For example, when presented with two unlabelled jugs[1] side-by-side, there are two primary ways of referring to an instance of

[1]This paper will make frequent references to the Jugs model (Beynon, 1988).

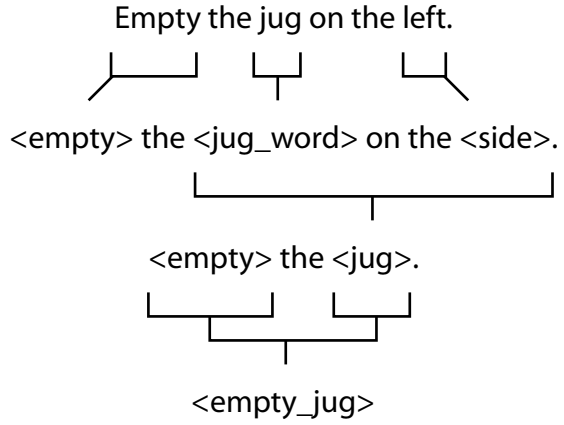

Figure 1: The parsing of an example sentence using the grammar in fig. 2.

one: either “the jug on the left”, or ”the left jug”, and although slightly contrived, these demonstrate the ease with which it is possible for semantic entities to be identified. Furthermore, being able to infer the meaning of a sentence (or rather in this case, map an input sentence to one of a set of known commands) based solely on these semantic entities identified with the text is also a safe assumption. Consider the command “empty the jug on the left”; reducing this to only those words that hold individual meaning, we are left with “empty jug left”, and although now half as long, the meaning is still intact, and conceivably easier to parse, at least for a computer.

An example of this approach can be seen in fig. 1.

3 An EM approach to NLP

Three of the key concepts of EM are observability, dependency, and agency, and these can be applied with the above approach to form an EM approach to natural language command recognition. First, consider some model and an input phrase that contains some action that is to be applied to that model.

Observables can be thought of as collections of symbols that hold some semantic meaning of importance to the model. There then exists a dependency between the existence and meaning of one observable, and the existence, meaning, and arrangement of the observables that comprise it. This forms a tree-like structure, moving from abstract concepts close to the root, to more concrete ones at the leaves. The premise is then that this can be taken to a point where a meaning of the phrase can be construed from the identified root observables and their locations, and if done ’correctly’, this meaning should match the intended meaning of the input phrase, allowing for the action to be carried out.

These relationships can be captured through agency. By considering agents as some meaningful sequence of observables with one replacement observable, applying them to the phrase replaces any instances of the sequence with the replacements observable. If actions are also included, that are carried out upon successful matching, the observables can be given a value, that is then dependent on its action and ’child’ observables. Adding these actions allows for the textual representation of the entities to be converted to something more useful, such as from ”thirty two” to 32.

Matching a phrase that contains identified semantic entities can then be carried out by a similar agent that has a set of observables to match rather than a sequence. If all the observables are present in the string, the agents action is executed.

4 Relationship with EM

> “In the EM environment, there is not only a chance to utilise one’s experience, but also a chance to explore new experiences which is often a source of enhancing one’s knowledge.”
>
> - Rasmequan and Russ (2002)

A key feature of many models in EM is their flexibility and openness to interpretation. In general they are never really considered ’finished’ and are often re-used or expanded into areas beyond their original intended interpretation. The development of models of language using the approach detailed in this paper shares similar features. As they are not built on any formal analysis of language, but on the user’s own experience of, and interpretation of, language, different users are likely to have different views of what is required. This means the language associated with a model can be continually developed as different people make use of it and include their own understanding.

One difference is that it is very difficult to re-interpret the meaning of language. Even though normal models are open to interpretation, meaning is often implied by such things as variable names. The ability to re-interpret a model becomes more difficult the more concrete you try to make it. Notations such as `%scout` and `%sasami` are a step in this direction; a picture of a jug can be difficult to interpret as something else. Language is an extreme case of

this because the very purpose of language is to convey meaning. So although parts of a model of language may be extremely reusable, such as ones recognising times and dates, the model as a whole is extremely unlikely to fit in with other interpretation. The language used to interact with a model, is based on the current interpretation of the model, rather than the model itself.

The language used with a model can also be considered as a means of knowledge storage. Tradition methods of NLP usually use some form of knowledge-base in order to disambiguate meaning. However, in this case the meaning is built into the model; the dependencies set up between semantic entities and the actions carried out upon recognition both implicitly and explicitly define various relationships and so can be considered a form of knowledge.

5 A definitive notation

Definitive notations are a major tool used in EM and so a notation use of several features of the AOP and has been designed to be as intuitive as possible. The basic template for an agent is:

```
type name = pattern
{
    action
};
```

Where `type` can be either “agent” or “command” as appropriate, `name` is the name of the agent, appearing between angled-brackets, `pattern` is a space-separated list of terms, which are detailed below, and `action`, which is optional, is the code that is to be executed if the pattern is matched.

These are 4 types of term, with 2 simply being used to ease implementation.

```
"literal" <agent> (regex) ‘eden‘
```

Literals and agents are part of the core of this approach, regular expressions were added in order to reduce the number of agents required for simple matches, making possible agents such as:

```
agent <jug> = (jug|glass|container);
```

Regular expressions are also allowed when referencing other agents, which is used as the mechanism for combining several similar agents in to one. Access to `%eden` observables improves integration with other notations and allows for dependencies on other parts of the model.

```
command <empty_jug> = <empty> <jug>
{
    empty_jug($p2);
};

agent <empty> =
            (empty|pour (out|away));

agent <jug> = <jug_[12]>;

agent <jug_1> =
            <jug_word> "on the" <side>
{
    $v = $p3;
};

agent <jug_2> = <side> <jug_word>
{
    $v = $p1;
};

agent <side> = (left|right)
{
    switch ( $p1 )
    {
        case "left" :  $v = 1;
        case "right" : $v = 2;
    }
};

agent <jug_word> =
            (jug|glass|container);
```

Figure 2: An example grammar.

Access to the identified terms is provided to the action code by allowing variables of the form `$p`*x* to be used, where *x* is the number of an identified term.

An example combining most features of the notation can be seen fig. 2. It carries out commands to empty one of two jugs.

This notation is non-deterministic, and deliberately so; to achieve otherwise would require enforcing some form of arbitrary order or priority on semantic entities, which is highly undesirable. In most cases this should not present a problem as most sematic entities, especially in an environment where their recognition is controlled, should be reasonably distinct and rarely overlap.

6 Implementation considerations

One of the key considerations was to decide on the involvement of the Agent-Oriented Parser (Brown, 2000; Harfield, 2003) in the implementation. A cursory investigation suggests that the AOP could be an ideal starting pointing, however, its use in a major way was rejected for several reasons.

- Firstly, it supports a relatively small number of simple operations, which although ideal for its intended use, can make the construction of rules that do not match them slightly awkward. A sequence to be recognised can potentially require one agent per term, making what could be a simple recognition into a more complex set of agents.
- Secondly, the AOP matches the entire input string, whereas the approach in this paper routinely requires only partial matches. The could be overcome by add dummy terms either side of an agent, but yet again, this is adding unnecessary complexity.
- Finally, matching a command to an input sentence requires consideration of individual parts of the string in a potentially arbitrary order. The AOP provides no mechanism for this beyond including agents of every possible combination This is possible for simpler commands, but the number of agents grows quickly, potentially requiring, for example, 24 agents to recognise a command with just 4 terms.

Despite these incompatibilities, the AOP will not be cast aside completely, it is an invaluable tool when used as intended, in this case to convert from the definitive notation above in to something more usable in `%eden`.

7 An implementation

Initially the AOP was used to produce a notation that recognised agents in the definitive notation as detailed in section 5 and converted them into a form more easily usable in `%eden`. Specifically it produces a list containing the agent type, the agent name, a list of the terms to match, and the code to run when the pattern is recognised. The agents are then passed to a function that first creates an `%eden` observable and then a procedure that passes the agent to the general parsing function. The procedure is then made dependent on either the other agents in the agent's pattern, or if none are present, it is made dependent on the sentence currently be parsed.

The main function of the implementation is the one that carries out general parsing. It essentially attempts to match an agent to the sentence being parsed, and on doing so modifies the sentence to reflect the action of the agent and updates the agent. In doing so, any other agents that are dependent on either the sentence or the agent just matched are passed to the function to be matched as well.

The sentence being parsed is a list of tokens, with each token being a list that contains the token's type: either a literal or an agent, the agents name if required, and the current value of the token. Initially the sentence is a single literal token.

In order to receive the natural language input, a dummy notation was made using the AOP that simply passes the text to a function that creates the sentence to be parsed.

The implementation is available at /csucfg/em/nlp/ when in the Department of Computer Science at the University of Warwirck (January 2006).

8 Further work

There are three primary way in which the ideas in this paper can be expanded, one being more specific than the other.

The first is to expand the proposed definitive notation to increase its similar to the notation found in Rawlinson (2005), that is, to allows more complex patterns to be recognised. Instead of simply using a sequence of terms, it could be extended to allow combination similar to regular expressions. This would not increase the expressive power of the notation, but would perhaps help reduce the number of agents required and therefore the overall complexity of the grammar.

The second is to have some mechanism for dealing with the dependency of actions and state. So for example, breaking from the Jugs models and moving on to the Lift (Beynon, 2003a), a command such as "go to floor seven" could assert the state of begin on floor seven, which would depend on being in the lift at floor 7 and leaving it, which would require being in the lift and pressing the button for floor 7, which would require entering the lift etc. This would hopefully make specifying commands a simpler task and open-up the possibility of exploring state and action dependency.

The final possibly for continuing the work of this paper that has been considered is to add some form of reasoning behind the language, which could be a

version of the previous suggestion but applied in a more intelligent way. Hopefully reaching the levels of projects such as SHRDLU (Winograd, 1968-70), but in a more generic and flexible way.

9 Conclusion

This paper details an attempt to bring together Empirical Modelling with an approach to natural language processing and in doing so it explores the relationship between the two, proposes a definitive notation, and provides an overview of an implementation. The two approaches are shown to complement each other remarkably well, allowing for natural language interaction with a model to be explored and in doing so increase the understanding of not only the language associated with a single model, but also how language can be used for interaction with computers in general.

Acknowledgments

Thanks go to Steve Russ, Meurig Beynon, and the other members of the Empirical Modelling group at the University of Warwick, for introducing the author to the possibilities and applications of EM.

References

The empirical modelling website. URL http://www.dcs.warwick.ac.uk/modelling, January 2006.

Meurig Beynon, Jugs model. URL http://empublic.dcs.warwick.ac.uk/projects/jugsBeynon1988/, 1988.

Meurig Beynon, Lift model of netherlands scenario. URL http://empublic.dcs.warwick.ac.uk/projects/liftBeynon2003/, 2003a.

W.M. Beynon. Radical empiricism, empirical modelling and the nature of knowing. In *In Proceedings of the WM 2003 Workshop on Knowledge Management and Philosophy*, April 2003b.

W.M. Beynon. Empirical modelling and the foundations of artificial intelligence. In *Computation for Metaphors, Analogy and Agents*, Lecture Notes in Artificial Intelligence 1562, pages 322–364. Springer, 1999.

W.M. Beynon and S.B. Russ. The interpretation of states: a new foundation for computation? In *Proc. PPIG'92*, January 1992.

Rodney A. Brooks. Intelligence without representation. *Artif. Intell.*, 47(1-3):139–159, 1991a.

Rodney A. Brooks. Intelligence without reason. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 569–595, 1991b.

C. Brown. An agent-based parsing system in eden, 3rd year project, University of Warwick, 2000.

Antony Harfield. Agent-oriented parsing with empirical modelling, 3rd year project, University of Warwick, 2003.

Suwanna Rasmequan and S.B. Russ. Knowledge modelling. In *Proceedings of the IASTED Conference on Applied Modelling and Simulation*. MIT, November 2002.

T.P. Rawlinson. Natural language processing in limited domains, 3rd year project, University of Warwick, 2005.

Stuart Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach*. 2003.

Terry Winograd, Shrdlu. URL http://hci.stanford.edu/ winograd/shrdlu/, 1968-70.