# A Pragmatic Approach to Disambiguation in Text Understanding

Martin Wheatman(✉)

Yagadi Ltd., Withinreap Barn, Moss Side Lane, Thornley,
Preston PR3 2ND, UK
martin@wheatman.net

**Abstract.** This paper describes a novel disambiguation mechanism for a text understanding system, which utilizes a general correction function, also described. This mechanism is comprised of: contextual ordering; transactional persistent memory; and, interactivity aligning machine action with user intent. The latter clearly demonstrates the use of language at a Pragmatic level – directedness toward user intent is afforded, purely through a speech interface. This work is supported by a Java implementation *enguage* and the *iNeed* app.

**Keywords:** Machine understanding · Disambiguation · Pragmatism · Semiotics

## 1 Introduction

Being able to speak naturally to computers would be of great benefit for all, including many hitherto disenfranchised groups, other than, perhaps, the functionally mute or profoundly deaf. Speech-to-text services provided by mobile operating systems can disambiguate with surprising accuracy [1]. However, understanding the resultant text remains problematic, which is exacerbated by the many forms of ambiguity in natural language [2]. Simple command based systems, such as Amazon Fire OS's Alexa [3], can successfully apply an operation to an entity, by support fixed phrases such as "ask…to…". However, such slot-filling does not uncover intended meaning in text.

Structural analysis – syntax and semantics – is not sufficient to provide insight into the intention of the speaker [4]. Two sentences with valid, identical structures may have entirely different meanings: *the Jumbo has landed* is an everyday occurrence; whereas, *the Eagle has landed* is one of the most iconic phrases of the C20th. Linguistic studies, such as modelling concordance with corpora analysis software [5], defer any issues with intended meaning to the researcher. Indeed, text understanding software must be able to differentiate between identical sentences. Further, deducing intentional thrust should also be addressed: while *I have a coffee* is the negation of *I need a coffee*, *I have to go to town* has the same meaning as *I need to go to town*.

Human speech, however, is not unduly plagued by misinterpretation: intentionality can be identified and clarified effectively by interaction aside from the main discourse. It is where written speech is unclear, without recourse to an author that problems occur: the spoken word seems to lose its life on the written page.


M.C.C. Baranauskas et al. (Eds.): ICISO 2016, IFIP AICT 477, pp. 143–148, 2016.
DOI: 10.1007/978-3-319-42102-5_16

## 2 Enguage

The disambiguation mechanism described has been developed for *enguage*, an experimental text understanding interface for mobile devices [6–9]. It uses the speech-to-text and text-to-speech services of mobile operating systems, allowing utterances to affect a response. To do this it constructs, on-the-fly, a set of potential interpreters for each utterance. Each micro-interpreter, or sign, consist of: a pattern, with which to match an utterance, and a corresponding train of thought, consisting of further utterances. Together, these signs along with persistent contextual data are combined into what is generally referred to as *interpretant* that is constructed during the process of interpretation [10]. Thus, meaning is based upon pattern matching rather than adherence to a syntax, so that *the Eagle has landed* is distinct from *the X has landed*.

A pattern consists of constant boilerplate tokens, represented here in lowercase, and variable hotspots, e.g. *the X has landed*. A hotspot can be flagged as representing a phrase if prefixed with *PHRASE-*, where the terminator is the token following the pattern or the end of utterance, e.g. *may name is PHRASE-NAME*, where NAME would match *Martin* or *Martin Wheatman*. Matched values become part of the overall context, e.g. X = "Jumbo", along with the particular train of thought for this sign.

Signs are organized into *repertoires* each of which supports a computable concept. For example, the *need* concept in the iNeed app [11], in essence, is a list manipulated with the repertoire: *i do not need anything*; *i need X*; *i do not need X*; and, *what do i need*. A sign construction repertoire [12] is built into enguage so that a repertoire itself is also a set of natural language utterances: text understanding is self-constructed, or autopoietic, in the same manner that a C compiler is written in C. Thus, natural language is interpreted in natural language: it is decomposed into more specific terms until an unequivocal conceptualization reached.

## 3 Disambiguation Mechanisms

Three algorithms are used to disambiguate utterances. Firstly, one gives an objective order to signs, and thus a defined order to meaning. Secondly, two in-built repertoires afford the user the ability to implicitly affect this ordering. The first of these is the ability to re-search the interpretant; second, based on the first, gives the user the ability to choose a meaning by revealing and hiding interpretations, determined by the feedback, or perlocution [4], within the ordered signs.

### 3.1 Ordering

Signs are ordered in increasing pattern complexity, so the least-complex matching pattern is matched first: *the eagle has landed* is matched before *the X has landed*. Further, *the X has landed* is matched before *the PHRASE-X has landed*. This is to ensure that specialized–singular–variables take precedence.

To achieve this patterns are hashed on their complexity: primarily on the number and complexity of hotspots (i.e. a phrased hotspot is given a higher value) and

secondarily on the length of (number of tokens in) its boilerplate. Length of boilerplate is significant where there is a phrased hotspot: the shorter the boilerplate, the higher the complexity values. This is because a phrased hotspot potentially matches an infinite number of tokens, so is given an arbitrarily high value (e.g. MAXINT), then any boilerplate and non-phrased hotspots are subtracted from this, meaning that *i have to PHRASE-X* matches before *i have PHRASE-X*, i.e. (MAXINT – 3) < (MAXINT – 2).

### 3.2 Correction

While speech-to-text systems are remarkable, they can create unintended translations: speech can be misheard. Because the speech-to-text in iNeed is presented directly to enguage, the ability to correct after the fact is required. Further, and pertinent to this paper, natural language is ambiguous: enguage cannot guarantee the above ordering achieves the intended match.

#### 3.2.1 General Correction

Enguage provides an inbuilt sign for general correction which uses a simple transactional persistent memory system. Changes to persistent memory are written to a provisional overlay. If one utterance is simply followed by another, the current provisional overlay is transferred to the database before it is repopulated. To obtain a correction, the provisional overlay is deleted before a new one is created.

The correctional sign has the pattern *no PHRASE-X*. On matching this pattern, the provisional overlay is removed and the utterance *PHRASE-X* is uttered. This allows the user to retract an utterance. It requires that operations that physically cannot be undone are initiated by several interlocking concepts. Further, there is a commitment to the design of repertoires such that *no PHRASE-X* is not matched to non-correctional (*no son of mine X* springs to mind). Being almost entirely composed of hotspot (one phrased hotspot with very little boilerplate), this will be almost the last pattern to be matched. So, all utterances beginning with *no* must explicitly be matched. However, this will not include the single answer *no* as there is no associated hotspot. Further, many utterances such as *no don't take him to the gallows* will work fine, as this is a natural use of this form.

This ability to affect changes to the operation of the software shows that this system is pragmatic, that the correction mechanism is implicit in the conversation between user and app – based on assessment of perlocution: replies from utterances.

#### 3.2.2 Special Correction

There is also a modification to this correctional mechanism which disambiguates utterances. It is triggered if the new utterance, without the initial *no*, is the same as the last. The first match of an interpretant pattern is then ignored, finding the next one. These signs are then swapped in the interpretant order to maintain this new context on subsequent utterances. If another *no PHRASE-X* is uttered, and the X matches the previous utterance, both the first and second matched patterns are ignored to find the third, and so on. This is dependent on the replies in interpretant – perlocution – to reflect the intentions of the user: it is for the speaker to decide if they have been

misunderstood. This is seen as being the natural way that misunderstanding is corrected in human discourse:

user: *The Eagle has landed.*
app: *Ok, we've a bunch of guys here about to turn blue!*
user: *No the Eagle has landed.*
app: *Ok, the eagle has returned to its nest.*

In practice, is has been found that a corrected utterance is automatically chosen by a speaker, rather than repeating the phrase to force disambiguation. For example, in the above interaction, the user reply may very well be *No the Eagle has returned to the nest*. This will not result in any signs being swapped.

### 3.2.3 Pragmatism

The reflecting of user intent shows that this process is pragmatic, that the actions taken (or proposed as being taken) depend on the interaction of the user and app, not in some hypothetical absolute meaning. It requires that the repertoire unequivocally reflects the app's interpretation: the repertoire should plainly confirm the specific action taken in response to each utterance. So, *my name is X* should have a reply of *ok, your name is X*. There is a minor issue here in the enguage interpreter; there is a obtuse option which returns only up to the comma. So *ok, your name is X* will reply *ok*. This mode can be used if the user is happy with the operation of the app, or if they are sure there is no ambiguity in the repertoire. Otherwise, the default (verbose) mode should be used, which replies the full phrase.

## 4 The Test

A test repertoire, which doesn't rely on any special cases or context, has been devised to fully demonstrate this disambiguation mechanism. This repertoire includes three signs with identical patterns, each with a separate significant train of thought.

```
On "this is a test":
    perform "list add _user needs meaning one";
    then, reply "this reaches meaning one".
On "this is a test":
    perform "list add _user needs meaning two";
    then, reply "this reaches meaning two".
On "this is a test":
    perform "list add _user needs meaning three";
    then, reply "this reaches meaning three".
```

The test pattern, *this is a test*, is ambiguous because it is all boilerplate and used on more than one sign. By reuttering this phrase, the progression through the distinct trains-of-thought is achieved. When the matched patterns are exhausted, enguage will first reply with the default *I do not understand*, before repeating the search. A full demonstration of this can be found at [13], a transcript of which is:

user: *This is a test.*
app: *This reaches meaning one.*
user: *No this is a test.*
app: *This reaches meaning two.*
user: *No this is a test.*
app: *This reaches meaning three.*
user: *No this is a test.*
app: *I don't understand.*
user: *No this is a test.*
app: *This reaches meaning one.*

## 5 Contribution and Further Work

Enguage is a text understanding interface allowing developers to create context dependent natural language apps. It provides many generic features for the conversation engineer, of which disambiguation is possibly the most important as it provides the key to context dependent processing across all repertoires. To be certain, it is not an AI chatbot designed for the Turing Test [14].

The mechanism overcomes structural issues with syntax and semantics by exploiting, at a pragmatic level, the human ability to detect and direct intent. It builds upon pattern matching so that meaning is determined by the boilerplate and in the interpretation of hotspot values, rather than individual word types and structures comprising a syntax tree. It utilizes a general correction mechanism, aiding its simplicity; it avoids the many individual forms of ambiguity.

This is not a mechanism for avoiding ambiguity; ambiguity will always occur, it simplifies the interaction needed when it is deemed to have occurred. In that it affords the user to direct intent, it is pragmatic. However, it is restricted to a discourse system – where there is interaction between the app and user—it cannot aid applications which are restricted to the structural domain: it will be of no use to the semantic analyses of corpora [5] which does not work within a conversation.

This work also serves to highlight differences between the spoken and written word. Current spoken word speech systems do not, in general, register pauses and intonation which could aid disambiguation. However, working with text, an arbitrary list of strings could also include emojis and other non-vocal ephemera to help identify intent. Written repertoires—including the autopoietic repertoire – use punctuation to express clearly. Further work includes producing a simplified, completely vocal autopoietic repertoire: *X implies Y* works already, but *X is conceptually Y* needs developing. Furthermore, context-free data types, such as *number* and *when*, have been instigated which needs to be enhanced by the concepts of *where* and *class*.

**Dedication.** This paper is dedicated to my father, James Malcolm Wheatman 11.2.1931 – 7.4.2016.

## References


1. Huang, X., Baker, J., Reddy, R.: A historical perspective of speech recognition. Commun. ACM **57**(1), 94–103 (2014)
2. Wikipedia. https://en.wikipedia.org/wiki/Ambiguity. Downloaded 26 Jan 2016
3. Amazon. https://developer.amazon.com/public/solutions/alexa/alexa-skills-kit/getting-started-guide. Downloaded 26 Jan 2016
4. Austin, J.L.: How to Do Things with Words. Oxford University Press, Oxford (1976). Ed. by Urmson, J.O., Sbisa, M.
5. Anthony, L.: AntConc Software (2015). http://www.laurenceanthony.net/software.html. Downloaded 26 Jan 2016
6. Wheatman, M.J.: A semiotic analysis of, "if we are holding hands, whose hand am I holding". J. Inf. Technol. **22**(Special Issue on Logistics), 41–52 (2014)
7. Wheatman, M.J.: https://www.youtube.com/watch?v=HsJyrdtk0GM. Uploaded 8 Jul 2015
8. Wheataman, M.J.: https://www.youtube.com/watch?v=ngdHVV3hwpE. Uploaded 18 Sept 2015
9. Wheatman, M.J.: https://play.google.com/store/apps/details?id=com.yagadi.iNeed. Accessed 22 Feb 2016
10. Peirce, C.S.: Collected Papers, vol. 1–8 (1935–58). Ed. by Hartshorne, C., Weiss, P.
11. Wheatman, M.J.: (2015). https://github.com/martinwheatman/enguage/iNeed/
12. Wheatman, M.J.: An autopoietic repertoire. In: Proceedings of AI-2014 34th SGAI International Conference on Artificial Intelligence, Cambridge, UK (2014)
13. Wheatman, M.J.: https://www.youtube.com/watch?v=Ig2Xd1QDu6E. Accessed 20 Jan 2016
14. Loebner, H.G.: In response. Commun. ACM **37**(6), 79–82 (1994)