Unifying Speech and Computation

Martin John Wheatman

Unifying Speech and Computation

Martin J. Wheatman

Yagadi Ltd., Preston, Lancashire PR3 2ND, UK
martin@wheatman.net

Abstract. A novel approach to programming computing machinery is demonstrated by the Enguage^{TM} language engine: programming by utterance. The running of a command is modeled as a deductive process; the mechanism by which meaning is ascribed to utterance—induction—is described. A full example of the factorial function is given. The paper then develops utterance not only as a form of issuing commands to hardware, but also of storing, retrieving, and manipulating spoken information—a programmable UI. Because such utterances can be generated by speech-to-text software, such interactive computation does not require a program as a written artifact.

Keywords: Programmable UI, Interactive Computation, Speech Understanding.

1 Introduction

The science of computing introduced by the Church Turing model (1, 2) is a layered approach: source code, as a written representation of algorithm, is compiled into a machine readable code. Programming languages use keywords and tokens to define syntax: an ostensibly unambiguous structure with which to specify that translation. A user interface layer is also required: we may code int n=1; but to be visible n must transcend the process address space. Typically, this means printing in some form. Text-to-speech and speech-to-text technology can fulfill this function. However, there is a need—and the ability—to turn speech directly into action, not merely to augment the control panel metaphor. The context-free approach is not conducive to understanding speech, however, because of its dependence on structure: speech simply does not use reserved words and tokens; often it is repeated, clichéd and unrefined (3). This paper shows how understanding can be created, not merely supplying values, but constructing behavior—functions—vocally.

The bias of language towards writing is not new; at the turn of the C20th, Saussure claimed the primacy of writing as a rigorous, academic endeavor, over the common, unrefined activity of speech (4). However, this primacy is misplaced: speech is a human cognitive activity—an innate ability; whereas, writing is a technology—a learnt skill. Further, writing is predated by speech, and can only ever be an approximation to it. Furthermore, there are many cognitive issues which prevents writing from being a universal medium. Any visual medium (think of keyboards on touch-screens!) disentitles those already excluded by physical or mental limitation—a digital society must be for all. Therefore, a truly universal machine must be implemented vocally, through speech understanding. The obvious caveat, perhaps, are the profoundly deaf or mute, who may prefer screens or, as with the deaf-blind, may have other methods of accessing utterance (5); however, access to computing should be for all. A way forward, here, is provided by Speech Act Theory (SAT) which defines a pragmatic explanation of language where understanding is judged by outcome (6). Rather than attributing meaning to words and word types, e.g. nouns and noun-phrases, SAT models understanding not on the words used, nor on what is meant meta-physically, but what the reaction is to the utterance in context.

Such understanding, not speech-to-text but utterance-to-action, is achieved by the lang*uage eng*ine, Enguage^{TM}. Examples can be found on (7) The specification of arbitrary text transformation as pattern and its associated list of intentions was originally devised for a software engineering project (8). These translations are a deductive (i.e. rule-following) process (9); their creation—the inductive process—has been found also to be deductive: a self-constructing, or *autopoietic*, process (10), much in the same way as a C compiler is written in C. Such interaction affords the ability to program by voice (11); which, as will be shown in this paper, removes the need for a program as a written artifact.

This paper describes the mechanisms supporting understanding in section 2, including written and vocal repertoires. In section 3, a simple function definition is described, and the test results of how the factorial function is constructed are presented. Section 4 develops the argument that, despite the limitations of the specialize solution, this research represents a novel computing system in itself. The significance is summarized section 5.

2 Understanding

Information Systems, such as SQL (12) or the HTTP protocol (13), place less emphasis on the translation to an underlying representation; and, more on the translation between an input request and an output response. Typically in textual form, this can be viewed abstractly as name="value", where the name is the URL and the value is the web page content. However, the value during mapping may be processed, such as in HTTP with PHP or JavaScript, or transmitted over TCP/IP, rather than it being an atomic mapping. This complexity-in-reference, or interpretant (14), can be seen in Enguage (9); however, both the request and response are arbitrary arrays of strings, allowing a bi-directional conversation to be constructed (15). Further, meaning is modeled within internal arbitrary arrays, or intentions, specifying action.

2.1 Unequivocal Response

Enguage models utterances, arbitrary lists of strings, and their context. From SAT, intentionality should provide perlocution, an unequivocal reply reassuring the user of the interpretation that has been made (16). A meaning is achieved by transforming an utterance into a reply, e.g. [“what”, “does”, “2”, “+”, “2”, “equal”] is replaced by [“2”, “+”, “2”, “equals”, “4”]. Each reply should be unequivocal like this: because it uses pattern matching; one utterance may match several patterns, so there may be many replies. The user is seeking the most appropriate reply and may be presented with each candidate in turn, in order of pattern complexity. The presentation of the next candidate is controlled by the user: no ..., is the pattern to re-present the last utterance (17). In practice, it has been found that this often involves simply selecting an alternative utterance. Thus, Enguage is a mediator of appropriate reply, rather than an interpreter, per se.

The structure for the meaning of an utterance, or *sign*, is composed of a pattern, and a *monad*, a list of internal utterances, serving as intentions with which to elicit action. The intentions are invoked until the action is a reply. If interpretation is positive, or felicitous, it is presented to the user. If there is anything infelicitous to interrupt this, intentions prefixed with *if not*, are followed. The interpretation of intentions continues until the end of the monad, or a *reply* ... (or *if not, reply* ...) is encountered.

2.2 From Written to Vocal Autopoietic Repertoire

A concept is supported by a repertoire of utterances. Written repertoires have been used to describe the action performed on matching an utterance (9). An example sign description, taken from the Enguage need repertoire, is:

```
On "SUBJECT needs PHRASE-OBJECTS":
  set output format to "QUANTITY,UNIT of,,LOCATOR LOCATION";
  OBJECTS exists in SUBJECT needs list;
  reply "I know";
  if not, add OBJECTS to SUBJECT needs list;
  if not, append OBJECTS onto SUBJECT needs list;
  then, reply "ok, SUBJECT needs ...".
```

A simple mapping, of the colon/semi-colon list, turns this pattern and monad into a list of standalone utterances in a self-creating, or autopoietic, repertoire (16), which constructs the machine representation of a sign. Uppercase words here represent contextual variables, set on matching the pattern and read from the environment by each intention. Lowercase pattern give a context sensitive framework for each pattern. A reply is a formatted answer. The answer is obtained by an underlying call to a traditional program, a database query or TCP/IP connection, and replaces the ellipsis in the reply. Intentions may voice change to the context within which an utterance is interpreted; so in the above example, Martin needs coffee may elicit ok, Martin needs coffee; but, its next utterance will elicit I know.

The written repertoire, outlined above, is now succeeded by a vocal autopoiesis (10). The main issue was the ability to represent variables and values without resorting to the unspoken distinction between case; plus, the ability to note when rules are being constructed rather than interpreted. The first was simply solved by noting variables with the word variable, and a descriptor if required, e.g. numeric variable quantity. The ellipsis is replaced by the (configurable) word whatever. Induction is noted by a variable which is set, then unset on the subsequent utterance of ok.

2.3 Structured Language

Some natural language does follow simple structure—*seven fifteen* is almost certainly a time—and this is incorporated into Enguage. One example of this is Number, which is implied in natural language, as in *I need a cup of coffee*, meaning *I need [quantity="1", unit="cup", object="coffee"];* which can be extracted from the pattern `i need NUMERIC-QUANTITY UNIT of PHRASE-OBJECT`. These valeus help contextualize subsequent utterances, *and another* meaning *[quantity="+1", unit="cup", object="coffee"]*. A numerical pattern variable is signified by the prefix NUMERIC-, which requires the evaluation of numeric expressions, such as *2 times 3 all squared*. Being in natural language, numeric expressions do not use parenthesis tokens, but even with this limit in complexity they remain very useful.

Floating qualifiers (18) can be extracted as temporal and spatial modifiers, e.g. *I am meeting my brother at the pub at 7pm*, meaning: *I am meeting my brother [time="7pm", location="the pub", locator="at"]*. Recent developments include late-binding floating qualifiers, allowing location, for example, to be identified by pattern, not *a priori* knowledge.

Further—and pertinent to this paper—expressions can be used to determine the meaning of utterances. In particular *the height of Martin is 194cm* implies that *height* is an attribute of whatever class from which *Martin* is instantiated. However, because *the square of n is n times n* contains *n times n*, an expression, it is implied that *square* is a function and that *n* is a parameter.

The next section details the example of these solutions in use in a spoken description of the mathematical function factorial.

3 A Working Description of Factorial

This section presents an example of how the complete description of a function—factorial—can be presented as a working *programme* of utterances. The British spelling is used here to distinguish a textual monad from the traditional representation of algorithm, of functions assignments and control structures. These examples form part of the Enguage text suite available at (19).

3.1 A Specialized Function Definition

An algorithm description as structured language (see 2.3 above), has been incorporated into an Expression mechanism, and is demonstrated in the Enguage test suite. At the time of writing, this only implements simple—non-recursive—functions. An example of a recursive function is given in 3.2 below.

Functions can either be created or evaluated. The creation sign description is:

```
On "the FUNCTION of AND-LIST-PARAMS is EXPR-BODY":
     #set the PARAMS of FUNCTION to BODY;
     perform "function create FUNCTION PARAMS / BODY";
```

```
      then, reply "ok, the FUNCTION of PARAMS is BODY".
```

The *and-list*, created for this sign, represents a list of parameters, such as *a and b and c*. The defines the induction of a function, such as *the sum of a and b is a plus b.* The EXPR- prefix matches an expression-as-value, confirming this as a function definition, as opposed to an entity-attribute operation, such as *the height of martin is 195*.

```
  On "what is the FUNCTION of PHRASE-PARAMS":
       perform "function evaluate FUNCTION PARAMS";
       if not, reply "I do not know";
       then, reply "the FUNCTION of PARAMS is ...".
```

The evaluation sign works by substituting the actual parameters for the formal parameters within the body. This, then, describes a number, much in the manner of substitution in Lambda Calculus (1). So, *what is the sum of 3 and 4*, is evaluated by substituting *3* and *4* into the function body *a plus b*. The resultant *3 plus 4* can be interpreted by the Number mechanism of Enguage giving *7*. This also supports *the factorial of 1 is 1*; however, whether this is interpreted as a value (by matching *the ATTR of ENTITY is VALUE*), or an expression, is immaterial—they both act in the same way.

Numerical expressions have been supported by Enguage for sometime, allowing verbal arithmetic to be performed. This is unlike written numerical expressions, as written expressions use parenthesis, whereas Enguage offers some control of the order of evaluation by the use of the word all: for example, *1 plus 2 times 3* would give *7*; whereas, *1 plus 2* ***all*** *times 3* would give *9*. At the moment the numerical abilities are hard-coded with English terms.

Recursive function descriptions such as *the factorial of n is n times the factorial of n minus one* have not yet been implemented due to time constraints, and the an alternative implementation which is described next.

3.2 A General Function Definition

An extensible method of programming vocally is described here. Vocal interpretation descriptions, inductive utterances, consist mainly of the patterns: *to X reply Y/this implies Z*; and, *interpret X thus/first Y/then Z/ok*. *Go on* is the unequivocal reply for induction until it is concluded by uttering *ok*. The *perform* intention introduces an internal command, so this might not reflect natural speech. Otherwise, this listing is self-documenting. This example forms part of the current Enguage unit test, from which these results below have been cut and pasted.

```
Factorial Description
=====================
user> what is the factorial of 4
enguage> I don't understand.

user> to the phrase what is the factorial of 1 reply 1
```

```
engauge> Go on.

user> ok
engauge> Ok.

user> what is the factorial of 1
engauge> 1.

user> interpret multiply numeric variable a by numeric variable b thus
engauge> Go on.

user> first perform numeric evaluate variable a times variable b
engauge> Go on.

user> ok
engauge> Ok.

user> multiply 2 by 3
engauge> 6.

user> interpret subtract numeric variable c from numeric variable d thus
engauge> Go on.

user> first perform numeric evaluate variable d - variable c
engauge> Go on.

user> ok
engauge> Ok.

user> subtract 2 from 3
engauge> 1.

user> interpret what is the factorial of numeric variable n thus
engauge> Go on.

user> first subtract 1 from variable n
engauge> Go on.

user> then what is the factorial of whatever
engauge> Go on.

user> then multiply whatever by variable n
engauge> Go on.

user> then reply whatever the factorial of variable n is whatever
engauge> Go on.

user> ok
engauge> Ok.

user> what is the factorial of 4
```

```
enguage> 24 the factorial of 4 is 24.
```

4 Discussion

This section argues Enguage's position as a novel approach to creating software. It starts by distancing itself from three existing approaches to language understanding. Then it explains how the results, given in section 3 above, equate to software engineering—creating programs.

4.1 Technology Review

This project is primarily an interface. It contains the ability to receive and present information, much in the same way as a windowing system, but in vocal rather than visual form.

Enguage is not a chatbot, the intention is not to maintain a conversation. Despite being concerned with an interface to natural language, this research has nothing to do with the Turing Test. The simple argument for this is that typical examples of such entrants (20, 21) use a keyword search to determine a theme and to reply with something convincingly intelligent. Where pattern matching is used, they resort to traditional methods of interpreting those patterns: that one of many preset values is used in reply to an utterance (21, 22). These responses are defined by some meta-language, typically represented in XML. In contrast, this research must consume the entire utterance, which is achieved solely by pattern matching, and must provide an unequivocal reply: no deception is intended.

Further, Enguage is not an evolution of existing voice based systems. While many personal assistants can be generalized as front ends to internet search, Amazon's Alexa (23) does present an interesting case. It allows devices to act upon commands such as, "Alexa, tell the garage door to open." However, because Alexa, and the development of Alexa skills, are rooted in traditional techniques—using web-based tools—it does not contain the ability to self-create which is central to Enguage. Thus, it falls short of a range of speech modes available (i.e. inductive as well as deductive) In short, we need to be able to say things like, "Tell Alexa how to open the garage door."

Furthermore, there is reluctance within the Computational Linguistics community to believe such an approach can work because of the reliance on syntax to define language. It takes a quantifiable coverage of natural language understanding, often quoting success at language understanding as a percentage rate. However, this is based on the analysis of language as it stood in the 1930's. Enguage achieves an appropriate interpretation, from which the user decides whether this is as intended; mediation of understanding, including the disambiguation of utterances, is part and parcel of the process of language, rather than being an exceptional, or failure, case. This more natural approach is that of speech act theory (16). Further, this disambiguation can also be automated to some extent, such as in lines 5 and 6 of the written repertoire in 2.1. If the first attempt to add an item to the needs list fails (such as in *martin needs to go to town*: the intention, *add to go to town to martin needs list* makes no sense), the second one, using a different phrase, *append to go to town onto martin need list* probably will. Enguage demonstrates interpretation being ordered by pattern complexity, presenting—but not guaranteeing—the most appropriate interpretation first.

4.2 Utterance as an Information Processing System

Enguage simply models an utterance as an arbitrary string, and maps one onto a reply: $u_{utterance} \rightarrow r_{reply}$. Rather than '→' being an atomic mapping, the rules for that mapping are constructed, also by voice, to capture the cultural reasoning behind the words. But how does this equate to being a computing system?

The instructions can be encoded in one language, say English, and the patterns can represent Chinese symbols; therefore, the mechanism behind Searle's Chinese Room thought experiment, can be implemented (24). This is not to confuse Enguage with Artificial Intelligence: Enguage, ostensibly, is nothing more than a user interface to underlying state-holding software (databases, internet services and the like). But it holds the utterance-to-action behavior, and because it can now construct these as functions, it can be seen as an information processing system in the traditional sense.

While the presented description of factorial resorts to calls to an internal command to perform arithmetic, this is conceptually no different to a CPU deferring processing to an arithmetic logic unit. The recursive nature of factorials is encapsulated entirely within what is effectively spoken word, thus the spoken word is computable. This demonstrates that loops can be implied even though looping is not generally seen as a reliable cognitive function.

5 Conclusion

Viewing computing as a process of mapping utterances to replies imposes few limitations: vocabulary is only limited by the abilities of the speech-to-text software. Enguage argues for utterances to not need a syntax—interpretation is all done by pattern matching. Perhaps patterns represent a custom, context sensitive, syntax; a flat structure constructed of constants and variables. Certainly, it seems, understanding works at two levels: the words which form a context sensitive framework which we share to engage in understanding, such as *i need ...*; secondly, there are the words which we use which are in fact personal to the speaker, such as *coffee*. What is my preference for coffee: its not found online (25).

The vocal abilities of Enguage will not make programming any easier. Indeed, it may result in programs being written down to be read. Arithmetic functions, such as factorials, will not be encoded within a natural language system, as they already are available in compiled libraries; however, this paper shows not only that the feat is possible, and that Enguage has the ability to support new functions orally, on-the-fly. It should also lead to other inductive language which is not available on deductive-only systems, such as—already implemented in Enguage—*X implies Y*. Further inductive examples are being developed, and an implementation of *why*.

Despite the progress that has been made, such an approach may take sometime to gain acceptance, since it breaks down the barrier between applications. Rather than running separate programs (word-processors, spreadsheets) to achieve different effects, one interpreter can handle many repertoires. This might not stop it being used for bespoke software, and this is where funding is being sought; but it may form a barrier to the adoption by, say, the mobile app community.

Ultimately, computing is not simply the running of programs, it includes the creation of programs, a significant component of which are functions. Turing argued for the use of finite-sized values which can be read, processed and written to/from positions on an endless tape (2). Modern information systems combine these alphabetic characters to compose the large, unwieldy, values which he shunned. With an arbitrary textually mapping between utterance and reply (8) any function, including function constructing functions, can be represented. Turing highlighted one of many issues with writing; while it is sparse enough for words to be guessed as a glance, multiple-repeated digits are incomprehensible. But, speech-to-text/text-to-speech software can adequately generate and consume Turing's otherwise un-memorable numbers. Thus, these large values can be spoken, and what they represent—programs—need no longer be a written artifact.

References

1. Church, A.: An Unsolvable Problem of Elementary Number Theory, *American Journal of Mathematics*. 58 (2): 345–363 (1936).
2. Turing, A. M.: On Computable Numbers with application to the Entscheidungsproblem, *Proceedings of the London Mathematical Society* 2(42) pp. 230–265 (1936).
3. Andersen, P. B.: *A Theory of Computer Semiotics (Human Computer Interaction Vol. 3)* Cambridge University Press, UK (1997).
4. Saussure, F. de: *Course in General Linguistics*, 3rd Ed. Eds. Bally, C., Sechehaye, A., Riedlinger, A. Transl. 1983 Harris, R. Duckworth London (1915).
5. Sense https://www.sense.org.uk/content/computing-aids, last accessed 2017/12/01
6. Austin, J. L.: *How to do Things With Words*, 2nd Edn. Oxford University Press, Oxford and New York (1962).
7. Wheatman, M. J.: https://www.youtube.com/channel/UCjTXFBHVNhC4dd_ZBxgx0ww
8. Liu., K., Wheatman, M. J.: Automating Software Design Pattern Transformation. 7th IEEE International Conference on Industrial Informatics CF-000825 (2009)
9. Wheatman, M. J.: A Semiotic Analysis of: If we are holding hands, whose hand am I holding. *Computing and Information Technology 22* (special issue: LISS 2013, 2014) (2014)
10. Wheatman, M. J.: An Autopoietic Repertoire. In: 34th SGAI International Conference Proceedings Research and Development in Intelligent Systems, Springer-Verlag (2014).
11. Wheatman, M.J.: Programming Without Program or How to Program in Natural Language Utterances. In: 37th SGAI International Conference on Artificial Intelligence Proceedings, pp 61–71, Springer Nature (2017).
12. Chamberlin, D., Boyce, R. F.: SEQUEL: A Structured English Query Language. In: ACM SIGFIDET Workshop on Data Description, Access and Control Proceedings, pp. 249–64, Association for Computing Machinery, (1974).
13. HTTP RFC, https://tools.ietf.org/html/rfc7230, last accessed 2018/04/09

14. Peirce, C. S.: Logic as Semiotic, In: *The Philosophical Writings of Peirce, Selected Writings*, Ed. Buchler, J., Dover Publications, pp 98-119 (1955)
15. Wheatman, M. J.: A Semiotic Model of Information System. In: 13th International Conference on Informatics and Semiotics in Organizations Proceedings, Eds. R. Jorna, K. Liu and N. R. Faber, (2011).
16. Searle, J. R.: *Speech Acts: An Essay in the Philosophy of Language*, Cambridge University Press, Cambridge (1969).
17. Wheatman, M. J.: A Pragmatic Approach to Disambiguation in Text Understanding. Proceedings of the 17th ICISO, pp143–148 (2016)
18. Wheatman, M.J.: Context-dependent Pattern Simplification by Extracting Context-free Floating Qualifiers. 36th SGAI International Conference Proceedings pp 209–217, Springer Nature (2016).
19. Wheatman, M. J.: Enguage source code, https://github.com/martinwheatman/enguage.and
20. Weizenbaum, J. ELIZA—A Computer Program for the Study of Natural Language Communication between Man and Machine. In: *Communications of the ACM* Volume 9 pp36-45 (1966).
21. ALICE AI Foundation (2017) www.alicebot.org/about.html las accessed 2017/12/01
22. Wilcox, B: ChatScript: https://en.wikipedia.org/wiki/ChatScript, last accessed 2017/12/01
23. Amazon Alexa, https://developer.amazon.com/alexa, last accessed 2017/12/01
24. Searle, J.: Minds, Brains, and Programs, In: *Behavioral and Brain Sciences* 3, pp 417-424 (1980).
25. Wheatman, M. J.: What Google Doesn't Know. In: *IT Now Spring 2017*, British Computer Society, pp 48-49 (2017)