Natural Language Programming

H. Chase Stevens
s1107496

School of Informatics
University of Edinburgh

Informatics Research Review

January 2015

Abstract

The aim of natural language programming is to allow written natural language specifications to be transformed into corresponding computer-executable formal language representations. Research in this field has largely tackled this task as performed in restricted domains; focusing on restricted subsets of both natural language specifications and programs. Three domains in particular have received special attention in the literature, these being the use of formalized, restricted variants of natural language specifications for the creation of programs, the translation of unrestricted natural language descriptions or directives into imperative formal languages, and the translation of natural language specifications or questions into declarative formal languages. There are also numerous methodologies and techniques used in natural language programming, the most prominent being those used in the field of machine translation, including semantic parsing, tree transduction, and, most recently, translation via artificial neural network.

1 Introduction

- Expansion of abstract.
- More formal exploration of various goals and interpretations of natural language programming.
- Possible brief mention of historical focus and successes/failures.
- Motivation:
  1. Allowing non-programmers to write programs or, more generally, get computer to perform desired tasks.
  2. Reduction of workload for programmers.

2 Background

Broadly, approaches to natural language programming can be split into two categories: those that attempt to make formal languages more closely resemble natural language, and those that attempt to handle unrestricted natural language expressions. While the former approach has been largely abandoned in favor of the latter within the more recent literature, an overview is given here of both types, in order not only to establish a historical context for the problem of natural language programming but also to provide sufficient juxtaposition to enable a comparison of the approaches' relative strengths and weaknesses. The latter approach, in turn, can be subdivided into those approaches targeting domain-specific declarative languages (often query languages) and those which target more general imperative languages.

2.1 Formalized natural language subsets

Under the "formalized natural language subset" paradigm, rather than a natural language text being translated into a corresponding formal language representation, a formal language itself is designed to, as much as possible, resemble and capture some subset of natural language. As such, expressions written in systems using this approach are directly parsed and executed, without having to undergo any other processing. In general, this approach shares many traits with other pre-statistical natural language processing methodologies; systems employing this approach are tightly bound to specific domains, such as database querying and processing, and can be quite fragile.

A prominent example of such a system is NLC, which is designed for processing data stored in tables [1] [2]. The system allows for the description of procedures which query and manipulate these data through English-language commands, for example, "Choose a row in the matrix" or "Add five [to] the 2nd positive entry in col 2" [1]. NLC is emblematic of a "good old-fashioned AI" approach in that its lexicon consists solely of terms explicitly pre-defined. This means that the system is only able to recognize a very small vocabulary of 450 words and misspellings thereof [1]. A hand-written grammar is used to parse natural language statements, and while this

approach in conjunction with the limited vocabulary produces good translations of the recognized subset of English, it also severely limits what can be legally expressed (for example, both declarative statements and questions are disallowed) and presents a clear issue in terms of the system's scalability. Nevertheless, experimental results [2] testify that NLC is perceived by users as providing a means of performing data manipulation that is both easier and faster than its contemporary alternatives. On the other hand, results also demonstrate that a significant proportion of users failed to achieve desired results using NLC, and also highlight users' frustrations in having to re-formulate English-language expressions in order to target NLC's subset of English.

The "ROBOT" system employs a similar methodology to NLC, but with some notable distinctions, namely, that ROBOT allows both imperative commands ("PRINT THE NAME, MODEL OF CAR, AND INSURANCE COMPANY OF ALL EMPLOYEES") and questions ("WHAT DID WE SELL CUSTOMER 24618?") to be used, and that ROBOT leverages data labels (e.g. column names) from the database to be queried, incorporating these labels into its lexicon [3]. It should be noted, however, that the latter is not an automatic process, but requires mappings between lexemes and data labels or values to be entered manually by database administrators. While users' success rate with ROBOT differs little from NLC, the system has the advantage of not requiring users to know the structure of tables in order to query them.

Unlike both NLC and ROBOT, the system presented in Heidorn [4] allows for declarative English-language specifications of scenarios for simulation. While the overall approach used is similar to that of NLC, including a limited hand-written lexicon, it notably differs in offering an interactive interface. In this way, the user is allowed to deliver a specification, then is asked by the system to clarify, elaborate on, and correct the system's interpretation of this specification. Despite the use of interactive corrective components in this and other systems having been noted as being an important feature for users [5], very little more recent work has incorporated this, perhaps because more modern statistical or translation-based approaches do not lend themselves as easily to interactive prompts.

There are several issues that pervade early natural language subset approaches to database querying, as highlighted in [5]. The permitted vocabulary is often tightly bound to a specific database, which both serves to limit the usefulness and applicability of the natural language interface and requires users to have an intimate knowledge of the targeted database. Without being given a formal specification of the grammar and lexicon, users also often have to re-formulate queries repeatedly in order to achieve the results they want.

Indeed, the specificity with which users have to formulate their queries or directives, coupled with the disadvantage that even a carefully worded sentence may fail to succeed, raises the question of whether these systems as a whole are more effective than well designed formal query languages (such as SQL), alternative database management UIs, or other more traditional approaches. This is the case with the both NLC and the system presented in

Heidorn [4], in which a full specification of the problem resembling a more verbose version of what might alternatively be expressed in source code is required from the user: while the user is spared having to learn a formal language's syntax, they instead are forced to use English to unambiguously explicate a scenario in a high level of detail.

2.2 Approaches targeting declarative formal languages

- Maybe split into 3 subsections - query languages (SQL, XPath), functional (e.g. ROBOT, string manipulation), regular expressions (are these just a special case of query?) - problem: some papers (ge, mooney) span multiple?

[6] describes a system which generates simple programs (in C) from descriptions such as "find the maximum and minimum numbers". This is achieved through the use of a recurrent neural network, which reads input and generates programs on a character-by-character (not token-by-token) basis. This is a preliminary study with few mentioned result. This approach has the disadvantage that the source code generated is not guaranteed to be syntactically valid, but it appears that the programs generated using this technique require only a few corrections to perform correctly.

[7] demonstrates a means of taking a restricted natural language, for which there already exists a suitable syntactic parser, and leveraging this to create both formal language procedures and queries. The technique assumes one or more lambda-calculus representations exist for each terminal and non-terminal node in the syntactic tree, which itself is assumed to be binarized. These can then be composed into a number of possible resultant expressions; supervised learning is used to identify features that are most likely to indicate correct semantic representations.

[8] describes a technique for taking natural-language query specifications and converting them into a derivative of XPath, allowing for the querying of X/HTML documents. The problem is tackled from two sides: the XPath derivative used includes high-level predicates (e.g. "`definition(a)`", "`about(a)`") so as to be more clearly derivable from the natural language descriptions than standard XPath; likewise, the system only allows the use of a restricted set of lexical items which directly correspond to these high-level predicates, making parsing and subsequent semantic conversion an easier task. Improved results are achieved by using techniques from information retrieval to evaluate the relevance of various generated formal-language queries' results to the original natural-language query. The leveraging of domain-specific operations in this paper both serves to produce impressive results but also limit the technique's applicability to cases where high-level general procedures have already been defined in the target language.

[9] describes a means of learning rules for translating natural language instructions and queries into formal language queries and procedures, assuming a syntactic parser is available for the natural language specifications. To do so, mappings are learned from subtrees of the syntactic parse of the natural language description to subtrees of the equivalent formal-language

representation. The method shows an improvement not only in quality of translation but also in speed versus previous methods.

The approach taken in [10] uses natural language descriptions of desired functionality in combination with user-provided examples to create string query and manipulation programs as formalized using a declarative DSL. The authors improve on previous work by requiring the user to specify both positive and negative examples not only for the overall task, but also individual constituents of the natural language description provided. For example, given an input like "Replace any combination of 6 whole numbers with 'X'," the user would be able to provide examples of "6 whole numbers". By using these examples for particular phrases within the natural language specification, [10] is able to constrain the search space of possible programs adhering to the description and, in doing so, improve the accuracy both with which sub-components of the program are mapped to description constituents and with which these sub-components are composed together to reflect the overall program description.

Lastly, [11] illustrates a bi-directional technique for transforming both natural language descriptions into source code and vice-versa. When the system is trained, both natural language descriptions and their corresponding source code are parsed into trees; correspondences between sub-trees are then learned and used for tree transduction. This process as applied to the Geoquery dataset results in source code translations that perform better than baseline approaches, but fall short of results from state-of-the-art systems.

- Need to determine if "Semantic Parsing on Freebase from Question-Answer Pairs" [12] should be included here or not.

2.3 Approaches targeting imperative formal languages

- What happened to C++ one? is that raza 2015?
- First thing here needs to be moved to previous section (regex is definitely DSLish/declarative)

[13] describes a method for generating regular expressions from natural language descriptions, e.g. "three letter word starting with X". To achieve this, the authors employ a semantic categorical grammar which targets the lambda calculus. Individual features of the description, once parsed, can then be unified into a single expression, which is then translatable into a regular expression. The authors claim that their algorithm shows significant improvement over previous work, and crucially exceeds 50% accuracy in translation.

[14] attempts to convert user-provided natural language descriptions of procedures into non-executable program "skeletons", which can then be edited into working programs by the user. To do so, the system described attempts to find three types of features within the descriptions: "steps", "loops", and "comments". Shallow syntactic parses are used to identify these features and specify in what way they should be transformed into the skeleton. This technique appears to be relatively brittle and has the disadvantage of requiring

human intervention to produce executable code, however, given the successes in machine-assisted translation, further work in this area has the potential to achieve favorable results.

3 Discussion and Conclusion

- Comparison of relative strengths/weaknesses of various approaches

There are several notable areas in which more research is clearly warranted. The literature reveals relatively little exploration of the use of artificial neural networks for natural language programming, despite repeated successes in traditional machine translation tasks. The investigation of whether recent advances in neural network training techniques and architecture design - such as the incorporation of an external stack, which has proven to be of benefit in other natural language processing tasks [15] - might successfully be applied to the problem of natural language programming seems very likely to bear fruit. In particular, the application of recurrent neural networks, which are well suited for use with tree-structured data, to natural language programming tasks appears to be a natural next step.

Find example to support of ANN successes in MT

Further, translation efforts for generalized natural language descriptions have largely targeted more conventional imperative languages (such as C++ [6]) despite greater success in more declarative languages, such as CLang [9] and GeoQuery [16]. It's unclear what properties if any have driven the successes in the latter: two possible explanations are that either the inherently stateful nature of the imperative paradigm poses a challenge, or that declarative languages may tend toward simpler grammars. Regardless, this suggests an opportunity to attempt targeting a concise but non-domain specific functional language.

A particular difficulty for natural language programming is the lack of large sets of training data, one notable exception being the recently released IFTTT corpus [17]. Without these, modern statistical approaches are severely limited in effectiveness. Further, what corpora are available do not offer parallel translations of natural language descriptions to multiple formal languages, but instead target a single formal language (e.g. the GeoQuery corpus). Likewise, corpora appear to be available largely for only a single natural language - English - which impedes work on natural language programming in other languages. A potential solution for this would be to create a corpus by generating natural language descriptions of code - a conceivably easier task, and one that has been previously tackled in the literature [18]. Alternatively, function or method comments written for publicly accessible codebases online could be leveraged as natural language descriptions of the code they accompany, provided these comments can be trusted to describe the code to a sufficient level of detail and accuracy.

References

[1] B. W. Ballard and A. W. Biermann, "Programming in natural language: "NLC" as a prototype," in *Proceedings of the 1979 annual conference*. ACM, 1979, pp. 228–237.

[2] A. W. Biermann, B. W. Ballard, and A. H. Sigmon, "An experimental study of natural language programming," *International journal of man-machine studies*, vol. 18, no. 1, pp. 71–87, 1983.

[3] L. R. Harris, "User oriented data base query with the ROBOT natural language query system," *International Journal of Man-Machine Studies*, vol. 9, no. 6, pp. 697–713, 1977.

[4] G. E. Heidorn, "English as a very high level language for simulation programming," *ACM SIGPLAN Notices*, vol. 9, no. 4, pp. 91–100, 1974.

[5] R. A. Capindale and R. G. Crawford, "Using a natural language interface with casual users," *International Journal of Man-Machine Studies*, vol. 32, no. 3, pp. 341–361, 1990.

[6] L. Mou, R. Men, G. Li, L. Zhang, and Z. Jin, "On end-to-end program generation from user intention by deep neural networks," *arXiv preprint arXiv:1510.07211*, 2015.

[7] R. Ge and R. J. Mooney, "Learning a compositional semantic parser using an existing syntactic parser," in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*. Association for Computational Linguistics, 2009, pp. 611–619.

[8] X. Tannier and S. Geva, "XML retrieval with a natural language interface," in *String Processing and Information Retrieval*. Springer, 2005, pp. 29–40.

[9] R. J. Kate, Y. W. Wong, and R. J. Mooney, "Learning to transform natural to formal languages," in *Proceedings of the National Conference on Artificial Intelligence*, vol. 20, no. 3. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005, p. 1062.

[10] M. Raza, S. Gulwani, and N. Milic-Frayling, "Compositional program synthesis from natural language and examples," 2015.

[11] R. Karampatsis, "Translating natural language into source code via tree transduction," Master's thesis, University of Edinburgh, 2015.

[12] J. Berant, A. Chou, R. Frostig, and P. Liang, "Semantic parsing on freebase from question-answer pairs." in *EMNLP*, 2013, pp. 1533–1544.

[13] N. Kushman and R. Barzilay, "Using semantic unification to generate regular expressions from natural language." North American Chapter of the Association for Computational Linguistics (NAACL), 2013.

[14] R. Mihalcea, H. Liu, and H. Lieberman, "NLP (natural language processing) for NLP (natural language programming)," in *Computational Linguistics and Intelligent Text Processing*. Springer, 2006, pp. 319–330.

[15] S. Das, C. L. Giles, and G.-Z. Sun, "Learning context-free grammars: Capabilities and limitations of a recurrent neural network with an external stack memory," in *Proceedings of The Fourteenth Annual Conference of Cognitive Science Society*, 1992.

[16] Y. W. Wong and R. J. Mooney, "Learning for semantic parsing with statistical machine translation," in *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*. Association for Computational Linguistics, 2006, pp. 439–446.

[17] C. Quirk, R. Mooney, and M. Galley, "Language to code: Learning semantic parsers for if-this-then-that recipes," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL-15)*, 2015, pp. 878–888.

[18] G. Sridhara, E. Hill, D. Muppaneni, L. Pollock, and K. Vijay-Shanker, "Towards automatically generating summary comments for java methods," in *Proceedings of the IEEE/ACM international conference on Automated software engineering*. ACM, 2010, pp. 43–52.