Knowledge-Based Translation of Natural Language into Symbolic Form

Christodoulos Ioannou[1] and **Loizos Michael**[1,2]
[1]Open University of Cyprus
[2]CYENS Center of Excellence
christodoulos.ioannou@st.ouc.ac.cy, loizos@ouc.ac.cy

Abstract

We consider the scenario of machines that receive human advice in natural language to revise their object-level knowledge for a domain of interest. Although techniques exist to translate such natural language advice into a symbolic form that is appropriate for machine reasoning, *the translation process itself* is typically pre-programmed and, thus, it is not amenable to dynamic and gradual improvement, nor can it be adjusted to the linguistic particularities of the advice giver. We seek to examine whether these limitations can be overcome through the use of a knowledge-based translation process.

In this position paper we take a first step in this investigation by demonstrating how such meta-level translation knowledge can be engineered to support the interpretation of object-level advice. While, admittedly, our engineering of the meta-level knowledge amounts to pre-programming, it nonetheless pushes towards the automated acquisition of this meta-level knowledge through advice-taking by demonstrating a key prerequisite: that it can be expressed, and reasoned with, under the same syntax and semantics as the object-level knowledge.

1 Introduction

An oft-quoted lesson from the early days of Artificial Intelligence is the importance of endowing machines with commonsense knowledge [McCarthy, 1989]. Equally oft-quoted is the realization that such commonsense knowledge cannot be directly pre-programmed into a machine, but can, more realistically, be acquired through some form of learning or, more specifically, through some form of advice-taking from a human [McCarthy, 1959]. But, most of the times, the requirement for pre-programming machines has not been completely side-stepped, but rather pushed up a level. Instead of pre-programming the object-level (domain) knowledge of machines, the majority of relevant AI research now effectively deals with the task of pre-programming machines at a meta-level (in the design of learning algorithms or architectures) that instructs machines how to acquire the object-level knowledge [Valiant, 2006; Michael and Valiant, 2008; Michael, 2016; Sap *et al.*, 2019]. Hence, one may ask: *(Q1) Can this meta-level knowledge also be acquired by means analogous to how the object-level knowledge is acquired?*

In the case of interest to our work, the meta-level knowledge represents a translation process (of object-level knowledge) from natural language into symbolic form. One could argue that question *(Q1)* is inconsequential in this case, since the meta-knowledge needed for translation purposes is predetermined and fixed, and comprises mostly generic rules applicable across advice givers. We agree only partly! Groups of people in different parts of the world or different individuals may use the same language differently. Therefore, being able to acquire meta-level knowledge by means analogous to how the object-level knowledge is acquired would allow the former knowledge to be treated as a policy of interpreting verbal commands. Individuals would be able to expand the policy to support interpreting additional sentence types, and customize it to incorporate omitted but implied meanings or different contexts when interpreting known sentence types.

Consequently, our sought goal is to design a process that will allow users to explain how to translate a sentence. For this reason, even though the translation process of natural language is user-agnostic in its most part, the amendment of this process should be supported. A translation process providing a dynamic and cognitively-light amendment procedure would also help decrease the needed amount of cognitively-heavy, batch-mode, offline pre-programming of meta-knowledge, making an affirmative answer to question *(Q1)* important.

The line of research that we pursue, and the initial results of which we present in this position paper, seek to provide some kind of response to *(Q1)*, in the particular setting of building a cognitive assistant able to help a human with a certain task [Leviathan and Matias, 2018]. Such a cognitive assistant cannot be pre-programmed, as it is expected to abide by its user's preferences when taking actions, which effectively necessitates the assistant to acquire its user-specific (object-level) knowledge through learning or advice-taking from the user [Bernard and Arnold, 2019]. For concreteness, in this paper we focus on knowledge acquisition through a process of advice-taking as originally proposed by McCarthy [1959] and further elaborated recently [Michael, 2017; MacLellan *et al.*, 2018] towards the development of the human-machine interaction protocol of Machine Coaching [Michael, 2019].

According to Machine Coaching, then, a cognitive assistant observes the state of the world and reasons using its cur-

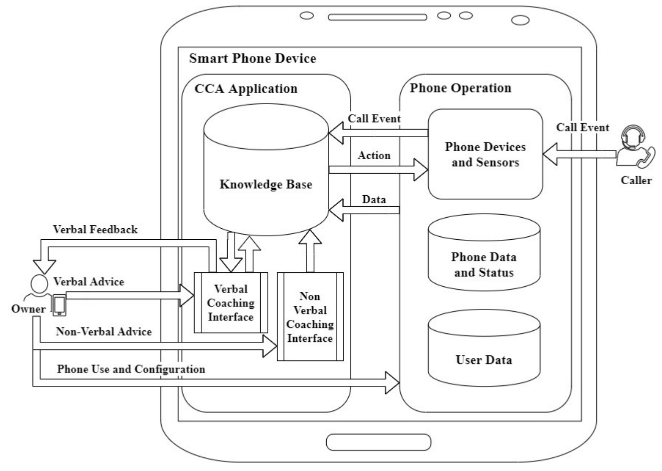

Figure 1: Architecture for a Cognitive Call Assistant.

rent object-level knowledge on how to act. A human coach observes the assistant and can inquire as to the reasons behind the assistant's choices. If the explanation is unconvincing or otherwise insufficient or unacceptable to the human, then the latter can offer advice back to the assistant on how to improve its knowledge towards better future decision-making. Importantly, we focus on the case where the interaction happens in natural language, and we are interested, more specifically, in understanding *how the natural language advice offered by the human to the assistant can be translated into some symbolic form that can be used for reasoning by the assistant*.

Translating natural language into symbolic form is, admittedly, a well-studied problem, and numerous relevant parsing tools are available [Kamath and Das, 2019]. To our knowledge, systems built upon such parsing tools are effectively pre-programming the translation (meta-level) knowledge of the cognitive assistant on how to interpret the (object-level) advice from a human. Our aim, then, is to examine whether an alternative to pre-programming is feasible and effective. Can this meta-level knowledge be learned? Or more specifically: *(Q2) Can meta-level knowledge (of how to interpret object-level advice) be acquired through Machine Coaching?*

Question *(Q2)* is our main long-term research question. We take a first step towards answering this question by demonstrating how the meta-level knowledge can be engineered to achieve its goal of interpreting object-level advice. Although engineering the meta-level knowledge does, in fact, amount to pre-programming, it nonetheless pushes towards the automated acquisition of this meta-level knowledge through Machine Coaching by demonstrating the key prerequisite *that it can be expressed, and reasoned with, under the same syntax and semantics as the object-level knowledge*, which itself is amenable to acquisition through Machine Coaching.

2 A Machine Coaching Example

As our running example, we consider a Cognitive Call Assistant (CCA) installed on a smart phone that takes actions upon receiving incoming calls, with an architecture as in Figure 1.

Upon receiving an incoming call, the CCA uses its current (object-level) knowledge to decide on how to act. The user can interact with the CCA to help it improve its knowledge, so that future incoming call events can be handled more appropriately (according to the user's personal preferences). The interaction may have the form of verbal advice, handled by the Verbal Coaching Interface, or non-verbal commands handled by the Non-Verbal Coaching Interface (which includes applications that any standard smart phone has, like calendars, maps, etc.). The dialogue below demonstrates this:

1. **Event:** Incoming call from some caller.
2. **User:** Declines. "Decline calls when busy."
3. **CCA:** "When do you consider yourself to be busy?"
4. **User:** Uses calendar and map applications and perhaps verbal commands to instruct that she considers herself busy when being in a meeting at work.
5. **Event:** (Later in the week…) Incoming call from John while user is in a work meeting.
6. **CCA:** Declines the call.
7. **User:** "Why did you decline this call?"
8. **CCA:** "Because you are at work and at a meeting, so I conclude that you are busy."
9. **User:** "Send SMS saying 'Busy! Will call back later.' to the caller when the call is important. If John is the caller then the call is important."

The example above is an instance of a Machine Coaching interaction used to improve the object-level (domain) knowledge of the CCA. The Verbal Coaching Interface is responsible for the translation of natural language into symbolic form. We consider this module's policy to be implemented through meta-level knowledge expressed and reasoned with under the same syntax and semantics as the object-level knowledge, and, thus, amenable to the same means of acquisition.

We assume that the Verbal Coaching Interface ships with an initial version of the meta-level (translation) knowledge. Compared to the CCA's object-level (domain) knowledge, the training of which is primarily the responsibility of the user, the initial version of the meta-level knowledge is engineered by experts to support some minimal level of translation. If the end user is unsatisfied with the translation process (i.e., if the translated symbolic form of a natural language sentence is wrong or unexpected, or a natural language feature is unsupported), the end user should be able to advise and revise the meta-level knowledge as done for the object-level knowledge.

Apparently, the initial version of the meta-level knowledge might not be rich enough to allow the user to explain and correct all errors or handle all unknown advice sentences. Since we do not expect the user to be fluent in the symbolic form or to be a natural language expert, she must be able to ask an expert for help, who will advise, in her stead, the translator module using logic-based communication and the Machine Coaching protocol to improve the translation process and/or adjust it to her needs. We illustrate this type of interaction by extending the conversation of our running example.

Suppose that the Verbal Coaching Interface fails to translate the last advice sentence from the user to the CCA during their last interaction from above. The dialogue continues:

10. **CCA:** "I am unable to interpret the advice 'If John is the caller then the call is important'? Do you want to consult an expert to help improve the translation policy?"
11. **User:** "Yes, please!"
12. **CCA:** "Please provide additional information that you wish to pass on to the expert."
13. **User:** "By this sentence I mean that I wish all calls coming from John to be labeled as important. This, in turn, will trigger a different treatment by the CCA."
14. **Event:** The advice that could not be interpreted, along with the additional information provided by the user are sent to an expert. The expert machine coaches the CCA to improve its meta-level knowledge.
15. **CCA:** "Expert advice has been received and the translation policy has been amended. Please continue!"

Thus, using Machine Coaching (albeit communicating directly in the symbolic form of the meta-knowledge), the expert can dynamically and gradually incorporate additional features of natural language (e.g., word synonyms, sentence types, etc.) and user-specific adjustments (e.g., personalized phrase disambiguation, local dialect, etc.) that might have not been part of the initial version of the meta-level knowledge.

3 Overview of Related Work

Our primary focus is on translating natural language advice or questions into symbolic form. A lot of work has been done in information and rule extraction from natural text, using ontologies or other scope-related information [Delannoy *et al.*, 1993; Dragoni *et al.*, 2016; Hassanpour *et al.*, 2011]. Unlike that work, our methodology does not seek to find important words / phrases based on the scope of the application, and no ontologies are used. Instead, we investigate the structure and relations within natural language sentences without taking the context into account, which has the advantage that our methodology is application-oblivious. Although not using ontologies or scope-related annotations might lead to vital information being unavailable when rules are extracted, this can be gradually remedied through Machine Coaching.

Other work has also been done in symbolic analysis of natural language text with the use of Controlled Natural Languages (CNL) [Kuhn, 2014]. In our approach, although similar in some ways with CNL approaches [Kain and Tompits, 2019], we do not strictly define a controlled natural language, but instead we start by considering a basic subset of natural language that we can then gradually and iteratively expand.

Our proposed architecture starts by extracting dependency relations between words, along with Part of Speech (POS) and Named Entity Recognition (NER) annotations [Grishman, 1997]. This information is subsequently used to identify patterns in the semantic relations that hold in a sentence, which support the construction of a rule capturing what a user wishes to communicate through that sentence. The key departure of our work from past works [Akbik and Broß, 2009; Gerber and Ngomo, 2012; Bos, 2015] is that the translation process itself is encoded as a collection of meta-rules, constituting the meta-level knowledge of a cognitive assistant.

As we have argued, this is a first step towards being able to acquire this knowledge gradually and dynamically, without the need of a pre-specified ontology, and in a manner that allows the meta-level knowledge (and hence the translation process of the object-level advice) to be amenable to adjustment to the linguistic particularities of each human.

4 Architecture and Methodology

The pipeline design of the CCA's Verbal Coaching Interface responsible for the interpretation of a piece of advice given by the user is shown in Figure 2. According to the pipeline, a piece of advice given verbally by the user is parsed (in Module 1) into a collection of predicates that describe the dependencies in the sentence of the advice. These predicates act as input to the meta-level knowledge of the CCA, with which the CCA reasons (in Module 2) to turn the piece of advice into an object-level rule, thus interpreting the advice. This object-level rule is then used to revise, through Machine Coaching, the CCA's object-level knowledge, and it is this object-level knowledge that is utilized by the CCA when deciding how to handle future incoming calls. The details of the Machine Coaching revision mechanism are inconsequential for this work, and can be found elsewhere [Michael, 2019].

In the remainder of this section we discuss in more detail the key components of the architecture described above.

4.1 User Language for Advice-Giving

For the purposes of this work, we restrict the types of sentences in which object-level advice can be expressed to the ones below [Huddleston and Pullum, 2002]. The numbers in brackets after each sentence type are references to the respective sentence examples that are provided in Appendix A:

1. Simple Declarative Sentences (**SDS**)
   - 1.1. Simple Declarative To-Be Sentences
     - 1.1.1. with Adjective or Noun Predicate *[1–3]*
     - 1.1.2. with Verb Predicate *[4–5]*
   - 1.2. Simple Declarative Verb Sentences
     - 1.2.1. with Object *[6]*
     - 1.2.2. without Object *[7–9]*
2. Simple Imperative Sentences (**SIS**) *[10]*
3. Conditional Sentences (zero conditional)
   *Below **SDC** stands for Simple Declarative Clause and **SIC** stands for Simple Imperative Clause.*
   - 3.1. Imperative Conditional Sentences (**ICS**)
     - 3.1.1. If/When SDC, SIC *[11–12]*
     - 3.1.2. If SDC, then SIC *[13]*
     - 3.1.3. SIC if/when SDC *[14]*
     - 3.1.4. all above with Implied Subject in SDC *[15–16]*
   - 3.2. Declarative Conditional Sentences (**DCS**)
     - 3.2.1. If/When SDC, SDC *[17–19]*
     - 3.2.2. If SDC, then SDC *[20]*
     - 3.2.3. SDC if/when SDC *[21–22]*
     - 3.2.4. all above with Implied Subject in SDC *[23]*

For the purposes of simplicity, sentences with negation are not considered in this initial set of supported sentence types.

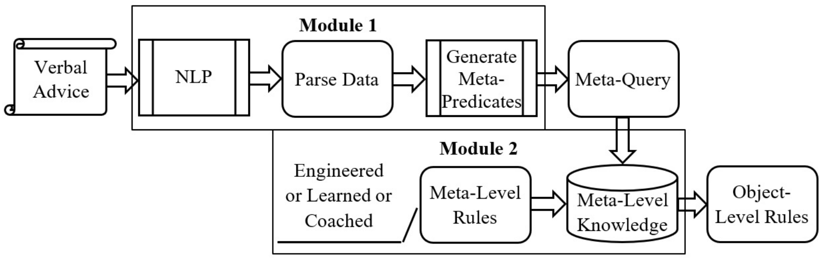

Figure 2: Implementation pipeline for a Verbal Coaching Interface.

4.2 Syntax of the Meta-Level Knowledge

Given an object-level advice sentence, we apply Natural Language Processing (NLP) tools to get dependency, part of speech, and named entity recognition data (in the form of a set of meta-predicates) and apply dependency analysis to manually identify patterns across these sets, which we later use to engineer meta-rules. We have defined and consider three types of meta-predicates for use in the body of meta-rules:

$<$ **Dependency** $>$ (
`Parent_Word`, `Parent_Word_Position`,
`Child_Word`, `Child_Word_Position`)

pos($<$ `POS_Tag` $>$, `Word`, `Word_Position`)

ner($<$ `NER_Tag` $>$, `Word`, `Word_Position`)

where $<$ `Dependency` $>$, $<$ `POS_Tag` $>$, and $<$ `NER_Tag` $>$ are placeholders for the dependency relation between two words, for the Part of Speech of a word, and for the Named Entity Recognition tag of a word, respectively, as identified by NLP. In addition to the above meta-predicates, a special meta-predicate is used in the head of meta-rules to represent predicates and variables of the generated object-level rule:

ruleterms(
`Head_Terms`, `Head_Vars`,
`Body_Terms`, `Body_Vars`)

where `Head_Terms` and `Body_Terms` are lists of lists that represent object-level predicates for the head and the body of the generated rule, respectively. Each sub-list represents a single predicate, and each sub-list member represents a part of the predicate (in case a predicate is a multi-word concatenation). `Head_Vars` and `Body_Vars` are also lists of lists and represent the corresponding variables of the object-level predicates.

Each advice sentence is assumed, in this work, to contain a single conditional, corresponding, thus, to a single object-level rule. This single rule, however, might involve the application of multiple meta-rules on the given advice sentence, with each meta-rule contributing part of the generated object-level rule; see, e.g., Example Sentence 2 in Section 4.3.

4.3 Engineering the Meta-Level Rules

For each of the sentence types considered in Section 4.1, we have engineered one or more associated meta-rules, which are presented in Appendix B. Three meta-rules are discussed below as part of two examples that demonstrate the key ideas.

Each example comprises an input sentence and its type, the intuitively expected formal rule corresponding to that sentence, the dependency parse tree for that sentence, the applicable meta-rules that are triggered by that sentence type, the inferences drawn from the meta-rules, and ultimately the generated object-level rule. Within each dependency parse tree, the bounding boxes of the respective figures highlight the patterns that trigger the meta-rules. Each variable W and P in the meta-rules corresponds, respectively, to an arbitrary word in the sentence and the word's position in that sentence.

Example Sentence 1: *A call is important.*

Type: 1.1.1. Simple Declarative To-Be Sentence with Adjective Predicate.

Expected Rule: `call(X) implies important(X);`

Dependency Parse Tree: see Figure 3 (left).

Applicable Meta-Rules:

$\texttt{root(root,0,W}_1\texttt{,P}_1\texttt{), cop(W}_1\texttt{,P}_1\texttt{,be,\_), nsubj(W}_1\texttt{,P}_1\texttt{,W}_2\texttt{,\_)}$
$\texttt{implies ruleterms([[W}_1\texttt{]], [[X1]], [[W}_2\texttt{]], [[X1]]);}$

Meta-Inferences:

`ruleterms([[important]], [[X1]], [[call]], [[X1]])`

Generated Rule:

`call(X1) implies important(X1);`

Example Sentence 2: *Send SMS when call is important.*

Type: 3.1.3. Imperative Conditional Sentence (SIC when SDC).

Expected Rule:
`call(X), important(X), SMS(Y) implies send(Y);`

Dependency Parse Tree: see Figure 3 (right).

Applicable Meta-Rules:

$\texttt{advcl(\_,\_,W}_1\texttt{,P}_1\texttt{), cop(W}_1\texttt{,P}_1\texttt{,be,\_), nsubj(W}_1\texttt{,P}_1\texttt{,W}_2\texttt{,\_)}$
$\texttt{implies ruleterms([[]], [[]], [[W}_2\texttt{],[W}_1\texttt{]], [[X1],[X1]]);}$

$\texttt{root(root,0,W}_1\texttt{,P}_1\texttt{), pos(verb,W}_1\texttt{,P}_1\texttt{), dobj(W}_1\texttt{,P}_1\texttt{,W}_2\texttt{,\_)}$
$\texttt{implies ruleterms([[W}_1\texttt{]], [[X2]], [[W}_2\texttt{]], [[X2]]);}$

Meta-Inferences:

`ruleterms([[]], [[]], [[call],[important]], [[X1],[X1]])`

`ruleterms([[send]], [[X2]], [[SMS]], [[X2]])`

Generated Rule:

`call(X1), important(X1), SMS(X2)`
`implies send(X2);`

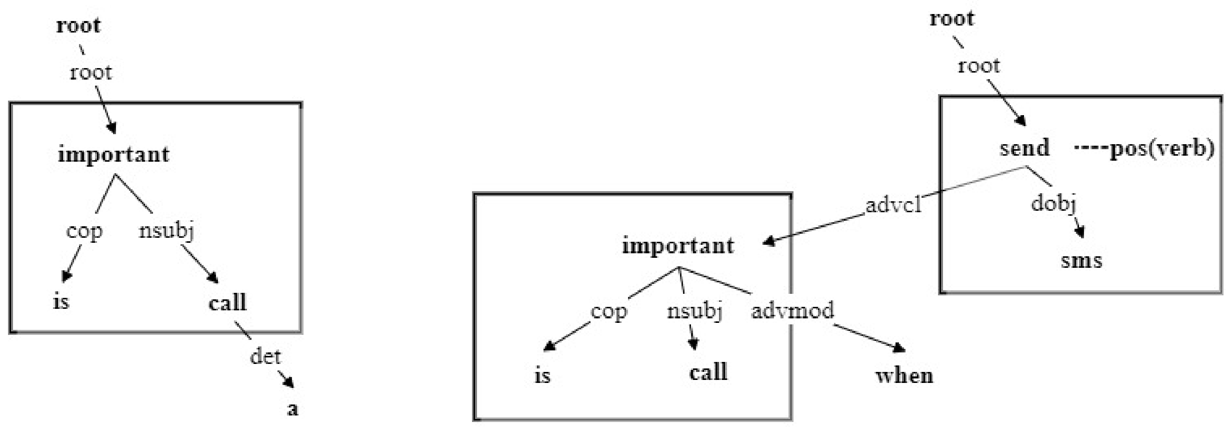

Figure 3: Dependency parse trees for Example Sentence 1 (left) and Example Sentence 2 (right).

In addition to the example meta-rules that are presented above, there are versions of those meta-rules that cope with the case of sentences with a named entity subject, since the resulting object-level rules need to be different in such cases.

In terms of the methodology followed in engineering meta-rules, we note that this involved the manual analysis of multiple sentences of each of the considered types, and the identification and clustering of dependency patterns. In a sense, then, we have manually-induced these meta-rules from a training set of sentences, having in mind that the engineered meta-rules are expected to apply and work well also with sentences (of the same types) that were not part of the training set.

5 Preliminary Empirical Evaluation

Towards empirically evaluating whether the engineered meta-level knowledge adequately addresses the task of interpreting user advice, we have implemented in Java the pipeline outlined in Section 4. Natural language is parsed using the Stanford CoreNLP Library [Manning *et al.*, 2014], the meta-level knowledge is implemented, for quick prototyping purposes, in SWI-Prolog, and the Java Prolog Library (JPL) is used to interface with the meta-level knowledge. The meta-level knowledge can be invoked to translate advice from natural language into symbolic form through a desktop application or a web-service call (http://cognition.ouc.ac.cy/nestor/).

This position paper does not provide a full-fleshed quantitative empirical evaluation, which is part of our ongoing work. However, as part of a preliminary evaluation, we created a custom data-set that includes at least one sentence of each type considered, along with its variants, and applied our engineered meta-level knowledge on that data-set. The list of the generated object-level rules can be found in Appendix A. That outcome was evaluated by the authors qualitatively, and below we present some early findings from this evaluation:

- The generated object-level rules reasonably capture the meaning of their associated natural language sentences.
- In some cases, slightly altering the syntax of a sentence (e.g., removing the comma from the sentence *"If John is busy, send SMS"*), alters the produced dependency tree.
- Simple sentences with omitted words and implied meanings cannot be handled appropriately.
- NLP sometimes labels incorrectly words with multiple meanings, resulting in incorrect dependency trees.

6 Conclusion and Future Work

We have presented our progress in developing a translation process from natural language into symbolic form that is, itself, expressible in the same symbolic form. Even though the meta-level knowledge developed in the context of this paper was engineered, our initial results provide some confidence that the meta-level knowledge itself could be acquired through a process of Machine Coaching, as is the case with the object-level knowledge. This, in turn, suggests an affirmative answer to the two questions posed in this paper, providing for a translation process that is amenable to dynamic change in an elaboration tolerant manner [McCarthy, 1998].

Towards solidifying our initial work and findings, we are currently in the process of porting the meta-level knowledge from SWI-Prolog to Prudens, a first-order logic language and framework (http://cognition.ouc.ac.cy/prudens/) with semantics that supports directly the paradigm of Machine Coaching. This will be followed by the undertaking of a quantitative empirical study to evaluate the efficiency, effectiveness, and explainability of a machine-coachable translation process.

More specific goals to be pursued in future work include:

- demonstrating the feasibility of machine coaching for the object-level task by utilizing engineered meta-rules;
- evaluating more thoroughly the ability of our engineered meta-level knowledge to cope with a varied collection of advice sentences of the types already supported;
- examining how the sentence types that are supported can be extended (e.g., by supporting negation, logical connectors, implied meanings, pronoun disambiguation, textual entailment [Levesque *et al.*, 2012; Korman *et al.*, 2018]);
- supporting other natural languages, and quantifying the extent to which meta-level knowledge developed for one natural language can be transferred (even partially) to another;

- experimenting with the use of machine learning techniques for the acquisition of meta-level knowledge;
- acquiring meta-level knowledge through a Verbal Coaching Interface, analogously to the acquisition method of object-level knowledge that is described in this paper.

The last direction from above raises, in fact, certain interesting theoretical questions: Is there a meta-language rich enough to support its own verbal-coachability without the need for the involvement of experts? Is there a way to decide or to prove this claim? Or is there a need for an infinite hierarchy of meta-languages needed to support the verbal-coaching of knowledge at lower levels of the meta-language hierarchy?

Even without concrete answers to these theoretical questions, central in our future work is to include a knowledge-based translation process in a cognitive assistant that can be verbally-coached, and which will also support the machine-coaching (even if not the verbal-coaching) of the translation process itself, so that the latter can improve over time and adapt to the linguistic particularities of its current user.

Acknowledgements

This work was supported by funding from the EU's Horizon 2020 Research and Innovation Programme under grant agreements no. 739578 and no. 823783, and from the Government of the Republic of Cyprus through the Deputy Ministry of Research, Innovation, and Digital Policy.

References

[Akbik and Broß, 2009] Alan Akbik and Jügen Broß. Wanderlust: Extracting Semantic Relations from Natural Language Text using Dependency Grammar Patterns. In *Proceedings of the WWW Workshop on Semantic Search*, volume 48, 2009.

[Bernard and Arnold, 2019] Denys Bernard and Alexandre Arnold. Cognitive Interaction with Virtual Assistants: From Philosophical Foundations to Illustrative Examples in Aeronautics. *Computers in Industry*, 107:33–49, 2019.

[Bos, 2015] Johan Bos. Open-Domain Semantic Parsing with Boxer. In *Proceedings of the 20th Nordic Conference of Computational Linguistics*, pages 301–304, 2015.

[Delannoy *et al.*, 1993] Jean François Delannoy, C. Feng, Stan Matwin, and Stan Szpakowicz. Knowledge Extraction from Text: Machine Learning for Text-to-Rule Translation. In *Proceedings of the ECML Workshop on Machine Learning and Text Analysis*, pages 7–13, 1993.

[Dragoni *et al.*, 2016] Mauro Dragoni, Serena Villata, Williams Rizzi, and Guido Governatori. Combining NLP Approaches for Rule Extraction from Legal Documents. In *Proceedings of the 1st Workshop on Mining and Reasoning with Legal Texts*, 2016.

[Gerber and Ngomo, 2012] Daniel Gerber and Axel-Cyrille Ngonga Ngomo. Extracting Multilingual Natural-Language Patterns for RDF Predicates. In *Proceedings of the 18th International Conference on Knowledge Engineering and Knowledge Management*, pages 87–96. Springer, 2012.

[Grishman, 1997] Ralph Grishman. Information Extraction: Techniques and Challenges. In *Proceedings of the International Summer School on Information Extraction*, pages 10–27. Springer, 1997.

[Hassanpour *et al.*, 2011] Saeed Hassanpour, Martin J. O'Connor, and Amar K. Das. A Framework for the Automatic Extraction of Rules from Online Text. In *Proceedings of the International Workshop on Rules and Rule Markup Languages for the Semantic Web*, pages 266–280. Springer, 2011.

[Huddleston and Pullum, 2002] Rodnry Huddleston and Geoffrey K. Pullum. *The Cambridge Grammar of the English Language*. Language. Cambridge: Cambridge University Press, 2002.

[Kain and Tompits, 2019] Tobias Kain and Hans Tompits. Uhura: An Authoring Tool for Specifying Answer-Set Programs Using Controlled Natural Language. In *Proceedings of the 16th European Conference on Logics in Artificial Intelligence*, pages 559–575. Springer, 2019.

[Kamath and Das, 2019] Aishwarya Kamath and Rajarshi Das. A Survey on Semantic Parsing. In *Proceedings of the 1st Conference on Automated Knowledge Base Construction*, 2019.

[Korman *et al.*, 2018] Daniel Z. Korman, Eric Mack, Jacob Jett, and Allen H. Renear. Defining Textual Entailment. *Journal of the Association for Information Science and Technology*, 69(6):763–772, 2018.

[Kuhn, 2014] Tobias Kuhn. A Survey and Classification of Controlled Natural Languages. *Computational Linguistics*, 40(1):121–170, 2014.

[Levesque *et al.*, 2012] Hector Levesque, Ernest Davis, and Leora Morgenstern. The Winograd Schema Challenge. In *Proceedings of the 13th International Conference on the Principles of Knowledge Representation and Reasoning*, 2012.

[Leviathan and Matias, 2018] Yaniv Leviathan and Yossi Matias. Google Duplex: An AI System for Accomplishing Real-World Tasks Over the Phone. Technical report, Google AI Blog, 2018.

[MacLellan *et al.*, 2018] Christopher J. MacLellan, Erik Harpstead, Robert P. Marinier III, and Kenneth R. Koedinger. A Framework for Natural Cognitive System Training Interactions. *Advances in Cognitive Systems*, 6:1–16, 2018.

[Manning *et al.*, 2014] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. The Stanford CoreNLP Natural Language Processing Toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, 2014.

[McCarthy, 1959] John McCarthy. Programs with Common Sense. In *Proceedings of the Teddington Conference on the Mechanization of Thought Processes*, pages 75–91, 1959.

[McCarthy, 1989] John McCarthy. Artificial Intelligence, Logic and Formalizing Common Sense. In *Philosophical*

Logic and Artificial Intelligence, pages 161–190. Springer, 1989.

[McCarthy, 1998] John McCarthy. Elaboration Tolerance. In *Proceedings of the 4th International Symposium on Logical Formalizations of Commonsense Reasoning*, pages 198–216, 1998.

[Michael and Valiant, 2008] Loizos Michael and Leslie G. Valiant. A First Experimental Demonstration of Massive Knowledge Infusion. In *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning*, pages 378–388. AAAI Press, 2008.

[Michael, 2016] Loizos Michael. Cognitive Reasoning and Learning Mechanisms. In *Proceedings of the 4th International Workshop on Artificial Intelligence and Cognition*, volume 1895, pages 2–23. CEUR-WS.org, 2016.

[Michael, 2017] Loizos Michael. The Advice Taker 2.0. In *Proceedings of the 13th International Symposium on Commonsense Reasoning*, volume 2052. CEUR-WS.org, 2017.

[Michael, 2019] Loizos Michael. Machine Coaching. In *Proceedings of the IJCAI Workshop on Explainable Artificial Intelligence*, pages 80–86, 2019.

[Sap *et al.*, 2019] Maarten Sap, Ronan Le Bras, Emily Allaway, Chandra Bhagavatula, Nicholas Lourie, Hannah Rashkin, Brendan Roof, Noah A Smith, and Yejin Choi. Atomic: An Atlas of Machine Commonsense for If-Then Reasoning. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*, volume 33, pages 3027–3035, 2019.

[Valiant, 2006] Leslie G. Valiant. Knowledge Infusion. In *Proceedings of the 21st AAAI Conference on Artificial Intelligence*, volume 6, pages 1546–1551. AAAI Press, 2006.

A Evaluation Data and Results

Sentence 1: *Students are smart.*

Type: 1.1.1. Simple Declarative To-Be Sentences with Adjective or Noun Predicate.

Generated Rule:

```
student(X1) implies smart(X1);
```

Sentence 2: *Eagle is a bird.*

Type: 1.1.1. Simple Declarative To-Be Sentences with Adjective or Noun Predicate.

Generated Rule:

```
bird(eagle);
```

Sentence 3: *Gifts from my wife are precious.*

Type: 1.1.1. Simple Declarative To-Be Sentences with Adjective or Noun Predicate (as part of a complex sentence).

Generated Rule:

```
gift(X1) implies precious(X1);
```

Sentence 4: *John is working.*

Type: 1.1.2. Simple Declarative To-Be Sentences with Verb Predicate.

Generated Rule:

```
work(john);
```

Sentence 5: *Students are studying.*

Type: 1.1.2. Simple Declarative To-Be Sentences with Verb Predicate.

Generated Rule:

```
student(X1) implies study(X1);
```

Sentence 6: *Birds have wings.*

Type: 1.2.1. Simple Declarative Verb Sentences with Object.

Generated Rule:

```
birds(X2), wing(X1) implies have(X2, X1);
```

Sentence 7: *A man cries.*

Type: 1.2.2. Simple Declarative Verb Sentences without Object.

Generated Rule:

```
man(X1) implies cry(X1);
```

Sentence 8: *John speaks.*

Type: 1.2.2. Simple Declarative Verb Sentences without Object.

Generated Rule:

```
speak(john);
```

Sentence 9: *Eagles fly high.*

Type: 1.2.2. Simple Declarative Verb Sentences without Object (as part of a complex sentence).

Generated Rule:

```
eagles(X1) implies fly(X1);
```

Sentence 10: *Answer the phone.*

Type: 2. Simple Imperative Sentences.

Generated Rule:

```
phone(X1) implies answer(X1);
```

Sentence 11: *If John is busy, send SMS.*

Type: 3.1.1. Imperative Conditional Sentences (If/When SDC 1.1.1, SIC).

Generated Rule:

```
busy(john), sms(X1) implies send(X1);
```

Sentence 12: *If John is working, send SMS.*

Type: 3.1.1. Imperative Conditional Sentences (If/When SDC 1.1.2, SIC).

Generated Rule:

`work(john), sms(X1) implies send(X1);`

Sentence 13: *If John has a phone, then send SMS.*

Type: 3.1.2. Imperative Conditional Sentences (If SDC 1.2.1, then SIC).

Generated Rule:

`phone(X3), have(john, X3), sms(X1) implies send(X1);`

Sentence 14: *Send SMS when John is busy.*

Type: 3.1.3. Imperative Conditional Sentences (SIC if/when SDC 1.1.1).

Generated Rule:

`busy(john), sms(X1) implies send(X1);`

Sentence 15: *Send SMS if busy.*

Type: 3.1.4. Imperative Conditional Sentences (SIC if/when SDC with implied Subject).

Generated Rule:

`busy, sms(X1) implies send(X1);`

Sentence 16: *If tired, play the guitar.*

Type: 3.1.4. Imperative Conditional Sentences (If/When SDC with implied Subject, SIC).

Generated Rule:

`tired, guitar(X1) implies play(X1);`

Sentence 17: *If John is clever, Anna is happy.*

Type: 3.2.1. Declarative Conditional Sentences (If/When SDC 1.1.1, SDC 1.1.1).

Generated Rule:

`clever(john) implies happy(anna);`

Sentence 18: *When John is bad, Anna is crying.*

Type: 3.2.1. Declarative Conditional Sentences (If/When SDC 1.1.1, SDC 1.1.2).

Generated Rule:

`bad(john) implies cry(anna);`

Sentence 19: *When boss pays well, employees are happy.*

Type: 3.2.1. Declarative Conditional Sentences (If/When SDC 1.2.2 as part of a complex sentence, SDC 1.1.1).

Generated Rule:

`pay(boss), employee(X1) implies happy(X1);`

Sentence 20: *If John is rich, then Anna has money.*

Type: 3.2.2. Declarative Conditional Sentences (If SDC 1.1.1, then SDC 1.2.1).

Generated Rule:

`rich(john), money(X1) implies have(anna, X1);`

Sentence 21: *Dad is happy when paint shines.*

Type: 3.2.3. Declarative Conditional Sentences (SDC 1.1.1 if/when SDC 1.2.2).

Generated Rule:

`paint(X3), shine(X3), dad(X1) implies happy(X1);`

Sentence 22: *Dad washes the car when the sun shines.*

Type: 3.2.3. Declarative Conditional Sentences (SDC 1.2.1 if/when SDC 1.2.2).

Generated Rule:

```
sun(X3), shine(X3), car(X1), dad(X2)
implies wash(X2, X1);
```

Sentence 23: *When tired, head is aching.*

Type: 3.2.4. Declarative Conditional Sentences (If/When SDC with implied Subject, SDC 1.1.2).

Generated Rule:

`tired, head(X1) implies ache(X1);`

B Engineered Meta-Level Knowledge

```
:- dynamic pos/3. :- dynamic ne/3.
:- dynamic root/4. :- dynamic det/4.
:- dynamic dobj/4. :- dynamic advmod/4.
:- dynamic advcl/4. :- dynamic punct/4.
:- dynamic nsubj/4. :- dynamic cop/4.
:- dynamic mark/4. :- dynamic case/4.
:- dynamic nmod/4. :- dynamic amod/4.
:- dynamic nmod_poss/4. :- dynamic aux/4.
:- dynamic ccomp/4. :- dynamic ner/3.

%Utilities %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% addvarlist - Add indexed var list
addvarlist(0, _, []).

addvarlist(N, OFFSET, [[VARNAME]|BASEVLL]) :-
N > 0,
VARINDEX is OFFSET + N,
string_concat('X', VARINDEX, VARNAME),
PREVN is N - 1,
addvarlist(PREVN, OFFSET, BASEVLL).

% addvar - Add indexed var
addvar(0, _, []).

addvar(N, OFFSET, [VARNAME|BASEVL]) :-
N > 0,
VARINDEX is OFFSET + N,
string_concat('X', VARINDEX, VARNAME),
PREVN is N - 1,
addvar(PREVN, OFFSET, BASEVL).

% Meta-Level Knowledge Definition %%%%%%%%%%
% Simple TO BE Clauses w. adj. or noun pred.
sdc(_, W1, [[W1]], [[W2]], [[]], [[]]) :-
cop(W1, P1, be, _),
nsubj(W1, P1, W2, P2), ner(_, W2, P2), !.
```

```
sdc(OFFSET, W1, [[W1]], HVL, [[W2]], BVL) :-
cop(W1, P1, be, _), nsubj(W1, P1, W2, _),
addvarlist(1, OFFSET, HVL),
addvarlist(1, OFFSET, BVL).

% Simple TO BE Clauses with verb Predicate
sdc(_, W1, [[W1]], [[W2]], [[]], [[]]) :-
aux(W1, P1, be, _),
nsubj(W1, P1, W2, P2), ner(_, W2, P2), !.

sdc(OFFSET, W1, [[W1]], HVL, [[W2]], BVL) :-
aux(W1, P1, be, _), nsubj(W1, P1, W2, _),
addvarlist(1, OFFSET, HVL),
addvarlist(1, OFFSET, BVL), !.

% Simple Transitive Verb Clauses
sdvc( _, W1,
    [[W1]], [[W2, W3]], [[]], [[]]) :-
pos(TAG, W1, P1),
sub_string(TAG, 0, _, _, 'vb' ),
nsubj(W1, P1, W2, P2), ner(_, W2, P2),
dobj(W1, P1, W3, P3), ner(_, W3, P3), !.

sdvc( OFFSET, W1,
    [[W1]], [[W2|HVL]], [[W3]], BVL) :-
pos(TAG, W1, P1),
sub_string(TAG, 0, _, _, 'vb'),
nsubj(W1, P1, W2, P2), ner(_, W2, P2),
dobj(W1, P1, W3, _), addvar(1, OFFSET, HVL),
addvarlist(1, OFFSET, BVL), !.

sdvc( OFFSET, W1,
    [[W1]], [HVL], [[W2]], BVL) :-
pos(TAG, W1, P1),
sub_string(TAG, 0, _, _, 'vb'),
nsubj(W1, P1, W2, _),
dobj(W1, P1, W3, P3), ner(_, W3, P3),
addvar(1, OFFSET, PHVL),
append(PHVL, [W3], HVL),
addvarlist(1, OFFSET, BVL), !.

sdvc( OFFSET, W1,
    [[W1]], [HVL], [[W2], [W3]], BVL) :-
pos(TAG, W1, P1),
sub_string(TAG, 0, _, _, 'vb' ),
nsubj(W1, P1, W2, _), dobj(W1, P1, W3, _),
addvar(2, OFFSET, HVL),
addvarlist(2, OFFSET, BVL), !.

% Simple Verb Clauses
sdvc(_, W1, [[W1]], [[W2]], [[]], [[]]) :-
pos(TAG, W1, P1),
sub_string(TAG, 0, _, _, 'vb' ),
nsubj(W1, P1, W2, P2), ner(_, W2, P2), !.

sdvc(OFFSET, W1, [[W1]], HVL, [[W2]], BVL) :-
pos(TAG, W1, P1),
sub_string(TAG, 0, _, _, 'vb' ),
nsubj(W1, P1, W2, _),
addvarlist(1, OFFSET, HVL),
addvarlist(1, OFFSET, BVL).

% Simple Imperative Clauses with subject
sic([[W1]], [['X1']], [[W2]], [['X1']]) :-
pos(TAG, W1, P1),
sub_string(TAG, 0, _, _, 'vb' ),
dobj(W1, P1, W2, _), !.

% Simple Imperative Clauses without subject
sic([[W1]], [['X1']], [[]], [[]]) :-
pos(TAG, W1, _),
sub_string(TAG, 0, _, _, 'vb' ).

% Simple Imperative Sentences
sis(HTL, HVL, BTL, BVL) :-
root(root, 0, W1, _),
sic([[W1]], HVL, BTL, BVL),
append([], [[W1]], HTL).

% Simple Declarative Sentences
sds(HTL, HVL, BTL, BVL) :-
root(root, 0, W1, _),
sdc(0, W1, HTL, HVL, BTL, BVL), !.

sds(HTL, HVL, BTL, BVL) :-
root(root, 0, W1, _),
sdvc(0, W1, HTL, HVL, BTL, BVL).

% Simple IF conditional clause A
scc([[]], [[]], BTL, BVL) :-
mark(W1, P1, 'if', _), advcl(_, _, W1, P1),
sdvc(2, W1, [[W1]], CHVL, CBTL, CBVL),
append(CBTL, [[W1]], BTL),
append(CBVL, CHVL, BVL), !.

% Simple WHEN conditional clause A
scc([[]], [[]], BTL, BVL) :-
advmod(W1, P1, when, _), advcl(_, _, W1, P1),
sdvc(2, W1, [[W1]], CHVL, CBTL, CBVL),
append(CBTL, [[W1]], BTL),
append(CBVL, CHVL, BVL), !.

% Simple IF conditional clause B
scc([[]], [[]], BTL, BVL) :-
mark(W1, P1, 'if', _), advcl(_, _, W1, P1),
sdc(1, W1, [[W1]], CHVL, CBTL, CBVL),
append(CBTL, [[W1]], BTL),
append(CBVL, CHVL, BVL), !.

% Simple WHEN conditional clause B
scc([[]], [[]], BTL, BVL) :-
advmod(W1, P1, when, _), advcl(_, _, W1, P1),
sdc(1, W1, [[W1]], CHVL, CBTL, CBVL),
append(CBTL, [[W1]], BTL),
append(CBVL, CHVL, BVL), !.

% Simple IF cond. clause w. implied subject
scc([[]], [[]], [[W1]], [[]]) :-
mark(W1, P1, 'if', _), advcl(_, _, W1, P1), !.

% Simple WHEN cond. clause w. implied subject
scc([[]], [[]], [[W1]], [[]]) :-
advmod(W1, P1, when, _), advcl(_, _, W1, P1).

ruleterms(HTL, HVL, BTL, BVL) :-
scc(HTL, HVL, BTL, BVL).
ruleterms(HTL, HVL, BTL, BVL) :-
sds(HTL, HVL, BTL, BVL), !.
ruleterms(HTL, HVL, BTL, BVL) :-
sis(HTL, HVL, BTL, BVL).
```