A Coachable Parser of Natural Language Advice

Christodoulos Ioannou[1] and Loizos Michael[1,2]
[1]*Open University of Cyprus, Nicosia, Cyprus*
[2]*CYENS Center of Excellence, Nicosia, Cyprus*

Keywords: Knowledge Engineering, Knowledge Acquisition, Interactive Programming, Translation Policy, Machine Coaching.

Abstract: We present a system for parsing advice offered by a human to a machine. The advice is given in the form of conditional sentences in natural language, and the system generates a logic-based (machine-readable) representation of the advice, as appropriate for use by the machine in a downstream task. The system utilizes a "white-box" knowledge-based translation policy, which can be acquired iteratively in a developmental manner through a coaching process. We showcase this coaching process by demonstrating how linguistic annotations of sentences can be combined, through simple logic-based expressions, to carry out the translation task.

1 INTRODUCTION

In the not-so-distant future, Ethan relies on his Cognitive Car Assistant (CCA) to manage the driving behavior of his self-driving car. Every morning, as Ethan drives his car to work, he notices that when the road signs are blurry, the visibility is quickly diminished. He coaches his CCA that "When the road signs are blurry, activate the ESP system of the car, especially if visibility is reduced.", where ESP is understood to be the traction control system of the car in question. His CCA responds with "Noted. Whenever the road signs are blurry, I will activate the ESP system, especially if visibility is reduced.". Ethan also notes that "If ESP is inactive, decrease the car's maximum speed.", with his CCA responding with "Noted. I will decrease maximum speed when ESP is inactive.".

The interaction between Ethan and his CCA is an example application of machine coaching (Michael, 2019): a dialectical human-machine interaction protocol whereby a (not necessarily technically-savvy) human coach offers advice to a machine to improve the latter's decision-making policy in a certain domain of interest. As illustrated in the example above, this form of interaction relies heavily on the machine's ability to understand the natural language advice provided by the human. This understanding is not always straightforward and cannot necessarily be anticipated by the developer of Ethan's CCA, as further demonstrated below.

Prompted by an incident where Ethan's son drove the car at a high speed on the highway and the CCA failed to keep the vehicle at a sufficient distance from the leading cars, Ethan remarks to his CCA to "Increase distance from leading cars when Joe is driving.", with his CCA acknowledging the advice. Continuing their conversation, Ethan further advises his CCA to "Engage the flood lights when the road is wet.", with his CCA responding that "I cannot interpret the meaning of the action 'engage' in your latest advice.". Ethan proceeds to revise the advice translation policy by clarifying that in his personal lingo, 'engage' actually means 'activate', with his CCA confirming the translation policy revision, and replying with "Noted. When the road is wet I will activate the flood lights.".

As evidenced by the running example above, machine coaching can be utilized not only on the domain level — which, in our case, is that of self-driving vehicles — but also at a meta level on how advice for the domain level is to be parsed, ensuring that Ethan's personal linguistic idioms are faithfully translated and represented by his CCA, without requiring extensive retraining or sophisticated programming.

The architecture of the envisioned CCA could involve several modules, both symbolic and neural. Neural modules would be more suitable for low-level atomic processes that require less guidance or amendment by a human, need not be fully explainable, and at the same time might be complicated enough for a human to provide a policy to control their behavior.

Ioannou, C. and Michael, L.
A Coachable Parser of Natural Language Advice.
DOI: 10.5220/0012454500003636

In *Proceedings of the 16th International Conference on Agents and Artificial Intelligence (ICAART 2024) - Volume 2*, pages 500-510
ISBN: 978-989-758-680-4; ISSN: 2184-433X

Examples of such processes could be the identification of the road conditions and boundaries or the interpretation of the road signs. On the other hand, high-level processes that relate to the general plan of driving and which use sensor parameters and low-level processes outputs, would benefit from being implemented as an adjustable "white-box" symbolic module.

The same line of reasoning on choosing how a process is implemented applies not only to the domain-level problem, but also to the meta-level problem of parsing advice in natural language. Reliability, explainabilty, and auditability of translating advice into domain-level policy expressions become critical properties in a domain like self-driving vehicles, where mistakes could lead to injury or death.

Sidestepping the question of how the meta-level advice is, itself, communicated to the CCA in a natural manner — which could be done, for instance, through a graphical interface, or through a sufficiently structured form of natural language that would be amenable to off-the-shelf parsing methods — in this work we focus on demonstrating how meta-level advice (once itself parsed) can be utilized through the process of machine coaching *to iteratively improve the parsing of the domain-level advice*, while keeping both the domain-level (Michael, 2019) and the meta-level (Ioannou and Michael, 2021) decision-making processes efficient, expressive, and explainable.

At a high level (see Figure 1), the parsing system receives as input a piece of domain-level advice in natural language, and applies Natural Language Processing (NLP) tools to extract its linguistic annotations, including word tokens, word lemmas, parts of speech, named entities, and dependency relations between tokens. The system's translation policy is then applied on these annotations to generate a logic-based representation of the input sentence. In case the outcome is deemed to be incomplete or incorrect, an expert can coach the system towards updating its policy, including providing exceptions to parts of the policy that should be overridden in certain circumstances. By virtue of this interaction, the translation policy can be adapted to the idiosyncrasies and particularities of the language, the domain expert, or the application domain for which the logic expressions are generated.

The system has been implemented in Java, and uses Stanford CoreNLP (Manning et al., 2014) for natural language processing, and the Prudens framework (Markos and Michael, 2022) for knowledge representation and reasoning. The pipeline functions are available both as a Java library and as a web service. Web and desktop applications are also available for experimenting with the pipeline[1].

2 REPRESENTING A TRANSLATION POLICY

Following the syntax and semantics of the Prudens framework, the translation policy of our proposed system comprises expressions in the form of logical implications in a simple fraction of first-order logic, with associated priorities to resolve conflicts between them (cf. Section 2 of (Markos and Michael, 2022) for the syntax and semantics of the the Prudens framework).

Policy expressions are defined over a set of predicates, which constitute the atomic elements of the language in which the linguistic annotations of an input sentence are expressed. To represent these linguistic annotations, we define two predicate types.

The first predicate type represents a word token:

`token(Word, POS Tag, NER Flag,`

`NER Tag, Lemma, Position)`

where `Word` is a word in an input sentence, `POS Tag` is its part-of-speech (POS) label, `NER Flag` takes the values {`ner`,`nner`} to indicate whether the word is a named entity, `NER Tag` is the type of the named entity (if applicable, or otherwise is equal to the constant `o`), `Lemma` is the lemmatized form of the word, and `Position` is the word's position in the sentence.

For example, the system will generate the following `token` predicates for the words of the clause *Signs are blurry*:

```
token(signs, nns, nner, o, sign, 1);
token(are, vbp, nner, o, be, 2);
token(blurry, jj, nner, o, blurry, 3);
```

The second predicate type is actually a predicate schema, and it represents a dependency relation:

$<$ `dependency` $>$ (

`Parent Word Lemma, Parent Word Position,`

`Child Word Lemma, Child Word Position)`

where the predicate name $<$ `dependency` $>$ shows the dependency relation type. `Parent Word Lemma` and `Child Word Lemma` are the lemmatized forms of the two words that have that dependency. These two words appear in positions `Parent Word Position` and `Child Word Position` in the sentence, respectively.

Using the same clause as an example, the system will generate the following predicates, based on the

[1] https://nestor-system.github.io/webapp/

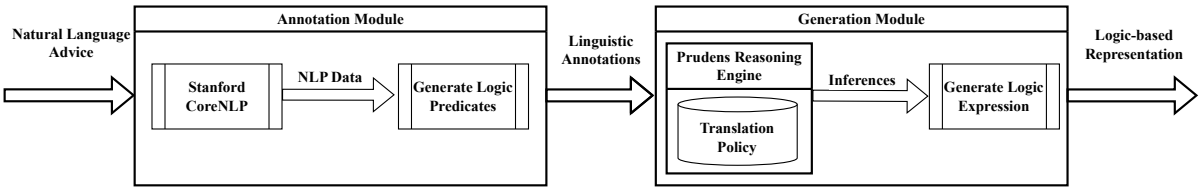

Figure 1: Parsing pipeline.

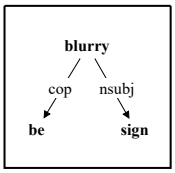

Figure 2: Dependency tree generated by NLP for the clause *Signs are blurry*.

dependencies identified by NLP as shown in Figure 2:

```
nsubj(blurry, 3, sign, 1);
cop(blurry, 3, be, 2);
```

To define how the logic expressions will be generated, a third type of predicate is required. This type of predicate is an action[2] and it is used as the head of an implication expression of the translation policy to specify how the outcome will be generated:

!generate(Type, Group,

$P_{1_1}[,P_{1_2},...]$ [, args, $A_{1_1}[,A_{1_2},...]$][, next,

$P_{2_1}[,P_{2_2},...]$ [, args, $A_{2_1}[,A_{2_2},...]$]][, next,...])

To describe the constituent parts of this predicate, we will use the terms *meta-predicate*, *meta-argument* and *meta-variable* for the description of predicates, arguments and variables of the translation policy expressions, and the terms *predicate*, *argument* and *variable* for the description of the expressions to be generated.

The meta-arguments Type and Group are used as driving switches for the *Generation Module* of the pipeline (see Figure 1) to guide it on how to process the inferred !generate actions. The meta-argument Type takes the values {head,body} to inform the *Generation Module* whether the inferred conclusion includes the head of the expression to be generated. The meta-argument Group is used to group translation policy expressions. Each group is processed separately by the *Generation Module*. Multiple !generate actions may be inferred for an input sentence. Group 0 is reserved for !generate actions with type head.

[2]Actions are predicates prefixed by the character "!" that represent a command and appear only in the head of an implication expression.

The meta-arguments $P_{i_1}, P_{i_2}, ...$ represent the constituents of a predicate name to be generated. If a predicate has arguments, then constant args follows to mark the end of the predicate name constituents and the start of the list of predicate's arguments, represented by the meta-arguments $A_{i_1}, A_{i_2}, ...$ If an argument of a generated predicate is a variable, then the variable placeholder constant vph_i is used, where i is a positive integer number. If the same placeholder constant is used in the same translation policy expression then it represents the same variable in the expression to be generated. The system automatically handles the naming of the variables as Xi. The predicates that a !generate meta-predicate represents are separated by the constant next. If a !generate action has type head, then the first predicate will be the head of the expression to be generated.

For example, the following !generate meta-predicate represents the logic implication is_bird(X1) implies fly(X1):

```
!generate(head, 0,
          fly, args, vph_1, next,
          is, bird, args, vph_1);
```

Two or more !generate meta-predicates can be combined to represent complex expressions, as shown in the following example, which represents the logic implication is_bird(X1), at(antartica) implies penguin(X1):

```
!generate(head, 0,
          penguin, args, vph_1, next,
          is, bird, args, vph_1);
!generate(body, 1, at, args, antartica);
```

3 DEMONSTRATION OF POLICY COACHING

The ultimate goal of our research program is to have Ethan to both advice the CCA on the domain-level policy (on how the self-driving car will behave), but also to offer meta-level advice on how to improve the

translation policy that parses the domain-level advice. In this work, we distinguish the two tasks, and assume that the latter will be done with the help of an expert.

In this Section, thus, we will follow the steps taken by Natalie, a *NAT*ural language *A*nd *Log*I*c* *E*xpert, as she coaches the system to identify clauses in conditional sentences and translate them into symbolic form. Natalie will interact with the system by invoking it on a corpus of conditional sentences in English that are used as pieces of domain-level advice to the CCA. Starting with an empty translation policy we demonstrate how the system can be used by an expert to iteratively coach a policy to translate user advice to domain-level expressions, thus showing that acquiring the translation knowledge without any pre-programming is feasible (Ioannou and Michael, 2021).

We consider a simplified version of the conversation of Ethan with his CCA. In this simplified version, an advice in natural language given by the user to the CCA is limited to a subset of zero conditional sentences, defined in the following paragraphs (see Section 3.1). Furthermore, the CCA is assumed to be able to perform only four actions on different car modules: to activate or deactivate a car module and to increase or decrease the value of a parameter of a car system. The CCA can take these actions according to the state of different sensors of the car. As the domain-level policy is not part of the scope of this work, it is not formally defined. For this scenario, we assume that the domain-level predicates to be generated are valid predicates of the domain-level policy and refer to sensors and/or car modules that the CCA can manage.

Below is the realisation of the simplified version of the conversation, according to the above assumptions and constraints (CCA replies are not considered):

Ethan: "If signs are blurry, activate ESP." **(S1)**

CCA: "When signs are blurry, I will activate ESP."

Ethan: "When ESP is inactive, decrease speed." **(S2)**

CCA: "If ESP is inactive, I will decrease speed."

Ethan: "Increase the distance when Joe is driving." **(S3)**

CCA: "I will increase the distance when Joe is driving."

Ethan: "Engage the lights when the road is wet." **(S4)**

CCA: "I cannot interpret action engage when the road is wet."

Ethan: Amends the translation policy and repeats the domain-level advice.

CCA: "When the road is wet I will activate the lights."

We will use sentences *S1, S2, S3, S4* of the conversation above in order to demonstrate the coaching process and the capability of the process to specialize or resolve conflicts.

In this section *Predicate* and *Subject* with capital first letter refer to relevant linguistic terms and the term *generated expression* will refer to a logic expression of the CCA's domain-level policy.

For every iteration step of the coaching process the following will be listed: the new expressions added to the *Translation Policy*, the *Translation Policy Inferences* on the sentence and the *Translated Symbolic Form* of the sentence as inferred by the translation policy emerged after each iteration step. The Linguistic Annotations of each sentence and the final *Translation Policy* emerged by the coaching process are listed in the Appendix.

3.1 Supported Domain-Level Advice Language

Conditional sentences describe what happens or what stands (consequence part) when a certain condition applies (antecedent part); e.g., *If the weather is bad tomorrow, John will take the car*. The order in which the two parts of the sentence are expressed or the word that is used to mark the condition (i.e., *if* or *when*) do not change the meaning of the sentence. Zero conditional sentences, specifically, express facts, rules or laws and both parts of the sentence are in the present simple tense; e.g., *If the road is bumpy, activate off-road mode* (Foley and Hall, 2003).

For simplicity of exposition, we consider a subset of zero conditional sentences that are suitable to be given as simple domain-level advice, in which the antecedent part consists of a *simple declarative to-be clause* and the consequent part consists of a *simple imperative clause*; e.g., *When the windshield is wet, start the wipers*.

A *simple declarative to-be clause* follows the structure: <*Subject*> <*To-Be Verb*> <*Predicate*>, where <*Predicate*> can be a *noun*, an *adjective* or a *verbal* and <*To-Be Verb*> can be either singular or plural; e.g., *Notifications are possible*, *Alarm is ringing* and *Mary is a colleague*.

A *simple imperative clause* follows the structure: <*Imperative Verb*> <*Object*>; e.g., *Notify user* or *Start the camera*.

For conciseness, pronouns, negation, word mod-

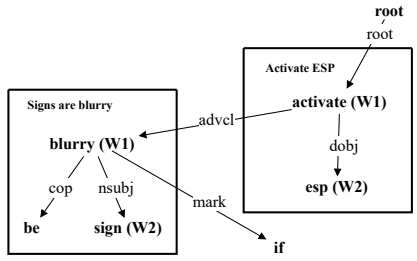

Figure 3: NLP-generated dependency tree for sentence (S1).

ifiers and multi-word nouns or verbs are not considered.

3.2 Identify Simple Declarative to-be and Imperative Clauses

Natalie chooses first to coach the system to identify the basic clauses of the supported language: *simple declarative to-be clauses* and *simple imperative clauses*. Based on her natural language expertise, she decides how to use linguistic annotations to identify these clauses. Natalie defines that a *simple declarative to-be clause* is identified when a word `W2` has a `nsubj` dependency to some word `W1`, and the <*To-Be Verb*> has a `cop` dependency to word `W1` (expression E001). Working in the same way, a *simple imperative clause* is identified when a word `W2` has a `dobj` dependency to some verb `W1` (expression E002). These patterns are illustrated in Figure 3.

As explained earlier, the CCA supports only a specific set of actions. For this reason, Natalie decides that a *simple imperative clause* can be identified as an *action clause* only if the verb of such a clause is one of the following: *Activate*, *Deactivate*, *Increase* and *Decrease* (expression E003). The above are encoded in the initial translation policy listed below.

Expressions Added to Translation Policy (Iteration Step 1)[3]*:*

```
@Knowledge
E001 :: cop(W1, P1, be, PBe),
nsubj(W1, P1, W2, P2) implies
sdclause(_, W1, P1, W1, W2, P2, W2);

E002 :: token(Word1, POS_Tag,
              NER_Flag, NER_Tag, W1, P1),
?startsWith(POS_Tag, vb),
dobj(W1, P1, W2, P2) implies
siclause(_, W1, P1, W1, W2, P2, W2);

E003 :: siclause(Prefix,
                 W1, P1, WPredicate1,
                 W2, P2, WPredicate2),
?partOf(activate_deactivate_increase_decrease,
        W1) implies
aclause('!',
        W1, P1, WPredicate1,
        W2, P2, WPredicate2);

@Procedures
function startsWith(word, start) {
    return word.startsWith(start);}

function partOf(list, word) {
    return list.split("_").includes(word);}
```

Predicates `sdclause`, `siclause` and `aclause` are intermediate predicates defined by Natalie to represent a *simple declarative clause*, a *simple imperative clause* and an *action clause*, respectively. The arguments of these predicates are defined in such a way to hold information that is expected to be required in the following steps of the inference process. Placeholder constant "_" in a meta-predicate is ignored by the *Generation Module*.

Applying this policy on sentence (S1), Natalie gets the following results (see Linguistic Annotation of (S1) in the Appendix):

Translation Policy Inferences on (S1):

```
sdclause(_, blurry, 4, blurry,
         sign, 2, sign);
siclause(_, activate, 6, activate,
         esp, 7, esp);
aclause('!', activate, 6, activate,
        esp, 7, esp);
```

Natalie checks the translation policy output and verifies that the results are as anticipated. She also observes that the policy is generic enough to capture both plural and singular form clauses, as well as *Subjects* that are or are not tagged as named entities.

3.3 Translate Consequent and Antecedent Parts of a Sentence

Natalie specifies that when the *Verb* (`W1`) of an *action clause* in the consequent part of a sentence has `root` dependency to the `root` token, then `W1` should be the name of a domain-level *action*, in the generated expression head. The *Object* (`W2`) of the clause should become the name of a predicate in the generated expression body. Both generated predicates should have as argument the same variable (expression E004). Natalie also specifies that when the *Predicate* (`W1`) of a *simple declarative to-be clause* in the antecedent part of a sentence has an `advcl` dependency to some word in the sentence, then both the *Predicate* and the *Subject* (`W2`) of the *simple declar-*

[3]The Prudens language supports custom JavaScript functions which return a Boolean value and can be used as predicates. These functions are defined under the `@Procedures` directive.

ative to-be clause should be the names of two predicates in the generated expression body. Both generated predicates should have as argument the same variable (expression E005). These patterns are illustrated in Figure 3 and are encoded in the expressions listed below.

Expressions Added to Translation Policy (Iteration Step 2):

```
E004 :: root(root, 0, W1, P1),
aclause(Prefix, W1, P1, WPredicate1,
        W2, P2, WPredicate2) implies
!generate(head, 0, Prefix,
          WPredicate1, args, vph_1, next,
          WPredicate2, args, vph_1);

E005 :: advcl(WParent, PParent, W1, P1),
sdclause(_, W1, P1, WPredicate1,
         W2, P2, WPredicate2) implies
!generate(body, 1, _,
          WPredicate1, args, vph_1, next,
          WPredicate2, args, vph_1);
```

Natalie applies the new policy on sentence (S1) again, and she gets the following results:

Translation Policy Inferences on (S1):

```
sdclause(_, blurry, 4, blurry,
         sign, 2, sign);
siclause(_, activate, 6, activate,
         esp, 7, esp);
aclause('!', activate, 6, activate,
        esp, 7, esp);
!generate(body, 1,
          _, blurry, args, vph_1, next,
          sign, args, vph_1);
!generate(head, 0,
          '!', activate, args, vph_1, next,
          esp, args, vph_1);
```

Translated Symbolic Form of (S1):

```
blurry(X1), sign(X1), esp(X2) implies
!activate(X2);
```

The output of the system is as anticipated by Natalie, although she is not quite happy with how named entities are represented in the generated expression.

3.4 Revise Translation Policy for Handling NER Subject or Object

Natalie specifies that named entities should not appear as distinct parts of the generated expression, but should instead appear as constant arguments. For this reason, Natalie adds two new translation policy expressions (E006 and E007) to override the existing expressions (E004 and E005) that translate the antecedent and consequent parts of the sentence. The new expressions differentiate the output in case the *Subject* of a *simple declarative to-be clause* or the *Object* of a *simple imperative clause* are named entities. Since the new expressions are essentially a specialization of the respective existing translation policy expressions, the existing expressions are also inferred, when the new expressions are inferred. To handle this situation, Natalie adds an explicit conflict expression (E008) that forces only the specialized version of the expressions to be inferred (cf. Section 3.5 of (Markos and Michael, 2022) for the conflict semantics of the Prudens language). The above are encoded in the expressions listed below.

Expressions Added to Translation Policy (Iteration Step 3):

```
E006 :: root(root, 0, W1, P1),
token(Word2, POS_Tag, ner, NER_Tag, W2, P2),
aclause(Prefix, W1, P1, WPredicate1,
        W2, P2, WPredicate2) implies
!generate(head, 0, Prefix,
          WPredicate1, args, WPredicate2);

E007 :: advcl(WParent, PParent, W1, P1),
token(Word2, POS_Tag, ner, NER_Tag, W2, P2),
sdclause(_, W1, P1, WPredicate1,
         W2, P2, WPredicate2) implies
!generate(body, 1, _,
          WPredicate1, args, WPredicate2);

E008 ::
!generate(TYPE, GROUP, Prefix,
          WPredicate1, args, WPredicate2) #
!generate(TYPE, GROUP, Prefix,
          WPredicate1, args, vph_1, next,
          WPredicate2, args, vph_1);
```

Natalie applies the new policy on sentence (S1) once more, and she gets the following results:

Translation Policy Inferences on (S1):

```
sdclause(_, blurry, 4, blurry,
         sign, 2, sign);
siclause(_, activate, 6, activate,
         esp, 7, esp);
aclause('!', activate, 6, activate,
        esp, 7, esp);
!generate(body, 1,
          _, blurry, args, vph_1, next,
          sign, args, vph_1);
!generate(head, 0, '!', activate, args, esp);
```

Translated Symbolic Form of (S1):

```
blurry(X1), sign(X1) implies !activate(esp);
```

Natalie is happy with the outcome on sentence (S1). She proceeds to apply the same policy also on sentence (S2), which has a *simple declarative to-be clause* with a named entity *Subject* in the antecedent part of the sentence (see Figure 4 and Linguistic Annotation of (S2) in the Appendix). Natalie gets the following results, as anticipated:

Translation Policy Inferences on (S2):

```
sdclause(_, inactive, 4, inactive,
```

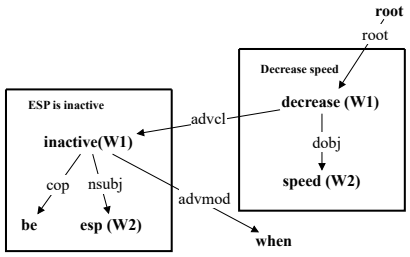

Figure 4: NLP-generated dependency tree for sentence (S2).

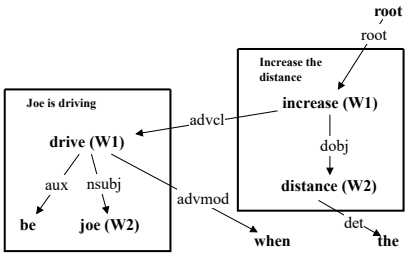

Figure 5: NLP-generated dependency tree for sentence (S3).

```
         esp, 2, esp);
siclause(_, decrease, 6, decrease,
         speed, 7, speed);
aclause('!', decrease, 6, decrease,
        speed, 7, speed);
!generate(body, 1,
          _, inactive, args, esp);
!generate(head, 0,
          '!', decrease, args, vph_1, next,
          speed, args, vph_1);
```

Translated Symbolic Form of (S2):

```
inactive(esp), speed(X1) implies !decrease(X1);
```

In either of the two sentences, one might note that two translation policy expressions with a `!generate` action are applicable: the generic one and the NER-specific one. Yet, the translation policy inference output includes only one of those two conclusions: the one coming from the expression with the higher priority. This is a direct consequence of the reasoning semantics of Prudens (Markos and Michael, 2022), which the system adopts in representing the translation policy, and which allows for conflicting expressions to co-exist, and resolves their conflicting conclusions gracefully, as needed.

3.5 Revise Translation Policy to Identify Declarative to-be Clauses with a Verbal Predicate

Natalie proceeds to apply the current policy on sentence (S3). The system fails to translate correctly the antecedent part of the advice (see Linguistic Annotation of (S3) in the Appendix).

Natalie observes that the antecedent clause *Joe is driving* of (S3) is not identified correctly by the system. This happens because *simple declarative to-be clauses* with a *verbal Predicate* are not identified yet by the current translation policy. So she proceeds to define that a *simple declarative to-be clause* with a *verbal Predicate* is identified when a word `W2` has a `nsubj` dependency to some word `W1` and the <*To-Be Verb*> has an `aux` dependency to word `W1` (see Figure 5). She encodes the above in a new expression (E009) that is added to the translation policy.

Expressions Added to Translation Policy (Iteration Step 4):

```
E009 :: aux(W1, P1, be, PBe),
nsubj(W1, P1, W2, P2) implies
sdclause(_, W1, P1, W1, W2, P2, W2);
```

Natalie applies the new policy on sentence (S3) again, and she gets the following results, as anticipated:

Translation Policy Inferences on (S3):

```
siclause(_, increase, 1, increase,
         distance, 3, distance);
sdclause(_, drive, 7, drive,
         joe, 5, joe);
aclause('!', increase, 1, increase,
        distance, 3, distance);
!generate(body, 1, _, drive, args, joe);
!generate(head, 0,
          '!', increase, args, vph_1, next,
          distance, args, vph_1);
```

Translated Symbolic Form of (S3):

```
drive(joe), distance(X1) implies
!increase(X1);
```

3.6 Revise Translation Policy to Adjust to Ethan's Personal Lingo

Natalie applies the current policy on sentence (S4) which the parsing system, as expected, is unable to translate because verb *engage* is a particularity of Ethan's language (see Linguistic Annotation of (S4) in the Appendix). Natalie "translates" Ethan's meta-level advice *When I say engage I mean activate* into a logic expression (E010) that identifies a *simple imperative clause* as an *Activate action clause*, if the verb of the clause is *engage* (see Figure 6), and adds it to the translation policy.

Expressions Added to Translation Policy (Iteration Step 5):

```
E010 :: siclause(Prefix,
                 W1, P1, WPredicate1,
                 W2, P2, WPredicate2),
?startsWith(W1, engage) implies
```

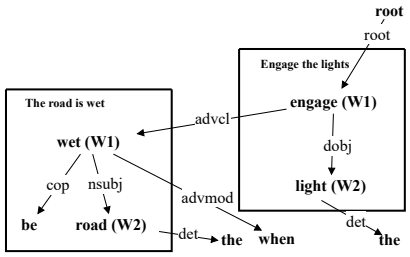

Figure 6: NLP-generated dependency tree for sentence (S4).

```
aclause('!', W1, P1, activate,
        W2, P2, WPredicate2);
```

The system returns the following results, as anticipated:

Translation Policy Inferences on (S4):

```
sdclause(_, wet, 8, wet,
         road, 6, road);
siclause(_, engage, 1, engage,
         light, 3, light);
!generate(body, 1,
          _, wet, args, vph_1, next,
          road, args, vph_1);
aclause('!', engage, 1, activate,
        light, 3, light);
!generate(head, 0,
        '!', activate, args, vph_1, next,
        light, args, vph_1);
```

Translated Symbolic Form of (S4):

```
wet(X1), road(X1), light(X2) implies
!activate(X2);
```

4 RELATED WORK AND CONCLUSIONS

We have demonstrated an implemented system with a novel take on knowledge acquisition through the use of machine coaching, as applied on the problem of parsing natural language advice. In doing so, the system allows for the development of explainable translation policies that can be easily debugged, iteratively enhanced, and customized. This work does not aspire to demonstrate a comprehensive and fully expressive parser, but rather to demonstrate the potential of using a coaching methodology towards that end.

For the purposes of demonstrating the idea of a coachable parser and for the sake of simplicity, we have used in this paper simple zero conditional sentences for acquiring gradually the translation policy. Obviously, the system can be used for more complex sentences and other types of conditional sentences and natural language structures, or even for languages other than English as long as reliable NLP tools and dependency parsers are available.

Alternative methodologies have been proposed for specific domains requiring extraction of structured logic from text (Delannoy et al., 1993; Dragoni et al., 2016; Hassanpour et al., 2011). Such methodologies can take advantage of pre-defined information related to the application domain. Although machine coaching may not be suitable for the extraction of all required knowledge for a system from day one, this is compensated by the advantages of explainability and adaptability. Also, the gradual acquisition of knowledge does not require restriction of the language used and allows customization to user specific needs. Approaches using Controlled Natural Languages (CNL) (Kuhn, 2014; Kain and Tompits, 2019) can be restrictive to the language form and the application domain.

To a certain extent, the problem of parsing natural language advice relates to the argument mining problem of extracting logical argument structures from unstructured text sentences. Both neural and symbolic methodologies have been used for argument mining, and extensive progress has been made over the past years, and especially after the social media rise, which provided new information-rich domains of application (Lippi and Torroni, 2016). Neural systems, in particular, have shown a great promise over the last few years (Devlin et al., 2019; Zhao and Eskenazi, 2016; Wen et al., 2017; Rajendran et al., 2018), even though they still suffer from high training cost and inefficient results in specific domains. Inherently, these systems use "black-box" approaches, which do not provide accurate, deep reasoning for decision explanations (Valmeekam et al., 2022; Wei et al., 2022; Zhou et al., 2023), a feature that along with acquired knowledge and concept accuracy is very important in certain domains (Nye et al., 2021; Jansen et al., 2017). Our work demonstrates that a coachable policy for argument mining might offer a cheap and lightweight "white-box" complementary approach to the existing ones, which could potentially provide an explainable but also flexible and adaptable solution; and one that could be part of a neural-symbolic architecture (Tsamoura et al., 2021) that utilizes both symbolic coaching and neural learning (Michael, 2023; Yang et al., 2023; Defresne et al., 2023).

Our system was developed to act as a tool whose generated expressions could be used to coach an application-specific policy (Ioannou and Michael, 2021). Our fictional character Natalie acts, then, as a facilitator in enhancing the usability of the machine coaching paradigm for such application-specific users, since the latter need not be experts in logic, and can, instead use natural language for communicating

the revisions they wish to make to the application-specific policy. Interestingly, this courtesy is not (and it is unclear whether it is even possible to be) extended to Natalie herself, since she needs to communicate her revisions for the translation policy directly in logic (Ioannou and Michael, 2021). Nonetheless, future work could seek to mitigate Natalie's burden by providing her or the user, for example, with visual aids in constructing expressions for the translation policy, or asking her to complete the less cognitively demanding, but perhaps more time consuming, task of evaluating numerous potential translations, and then letting another process that combines machine learning and machine coaching to actually generate the translation policy (Markos et al., 2022).

Beyond our initial empirical evaluation of the system, a more systematic user study will help us assess the accuracy, the performance and the usability of the demonstrated system and methodology. Future work could focus on this task by identifying suitable datasets and benchmarks that would quantify the claimed advantages of the proposed ideas (Ruder, 2021).

ACKNOWLEDGMENTS

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

REFERENCES

Defresne, M., Barbe, S., and Schiex, T. (2023). Scalable coupling of deep learning with logical reasoning. In *International Joint Conference on Artificial Intelligence (IJCAI 2023)*.

Delannoy, J. F., Feng, C., Matwin, S., and Szpakowicz, S. (1993). 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.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Dragoni, M., Villata, S., Rizzi, W., and Governatori, G. (2016). Combining NLP approaches for rule extraction from legal documents. In *Proceedings of the 1st Workshop on Mining and Reasoning with Legal Texts*.

Foley, M. and Hall, D. (2003). *Longman advanced learners' grammar: a self-study reference & practice book with answers*. Longman London and New York.

Hassanpour, S., O'Connor, M. J., and Das, A. K. (2011). 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.

Ioannou, C. and Michael, L. (2021). Knowledge-based translation of natural language into symbolic form. In *Proceedings of the 7th Linguistic and Cognitive Approaches To Dialog Agents Workshop - LaCATODA 2021 (IJCAI 2021)*, volume 2935, pages 24–32. CEUR-WS.

Jansen, P., Sharp, R., Surdeanu, M., and Clark, P. (2017). Framing QA as building and ranking intersentence answer justifications. *Computational Linguistics*, 43(2):407–449.

Kain, T. and Tompits, H. (2019). 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.

Kuhn, T. (2014). AsSurvey and classification of controlled natural languages. *Computational Linguistics*, 40(1):121–170.

Lippi, M. and Torroni, P. (2016). Argumentation mining: state of the art and emerging trends. *ACM Trans. Internet Technol.*, 16(2).

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J. R., Bethard, S., and McClosky, D. (2014). The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60.

Markos, V., Thoma, M., and Michael, L. (2022). Machine coaching with proxy coaches. In *Proceedings of the 1st International Workshop on Argumentation and Machine Learning (ArgML'22)*, volume 3208, pages 45–64. CEUR-WS.

Markos, V. T. and Michael, L. (2022). Prudens: An argumentation-based language for cognitive assistants. In *Proceedings of the 6th International Joint Conference on Rules and Reasoning (RuleML+RR'22)*, pages 296–304, Berlin, Germany. Springer.

Michael, L. (2019). Machine coaching. In *Proceedings of the IJCAI Workshop on Explainable Artificial Intelligence (XAI'19)*, pages 80–86, S.A.R. Macau, P.R. China.

Michael, L. (2023). Autodidactic and coachable neural architectures. In Hitzler, P., Sarker, M. K., and Eberhart, A., editors, *Compendium of Neurosymbolic Artificial Intelligence*, volume 369 of *Frontiers in Artificial Intelligence and Applications*, pages 235–248. IOS Press.

Nye, M., Tessler, M. H., Tenenbaum, J. B., and Lake, B. M. (2021). Improving coherence and consistency

in neural sequence models with dual-system, neuro-symbolic reasoning. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*.

Rajendran, J., Ganhotra, J., Singh, S., and Polymenakos, L. (2018). Learning end-to-end goal-oriented dialog with multiple answers. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3834–3843, Brussels, Belgium. Association for Computational Linguistics.

Ruder, S. (2021). Challenges and opportunities in NLP benchmarking. Technical report, ruder.io A blog about natural language processing and machine learning.

Tsamoura, E., Hospedales, T., and Michael, L. (2021). Neural-symbolic integration: a compositional perspective. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI'21)*, pages 5051–5060.

Valmeekam, K., Olmo, A., Sreedharan, S., and Kambhampati, S. (2022). Large language models still can't plan (a benchmark for llms on planning and reasoning about change). In *NeurIPS 2022 Foundation Models for Decision Making Workshop*.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., brian ichter, Xia, F., Chi, E. H., Le, Q. V., and Zhou, D. (2022). Chain of thought prompting elicits reasoning in large language models. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K., editors, *Advances in Neural Information Processing Systems*.

Wen, T.-H., Vandyke, D., Mrkšić, N., Gašić, M., Rojas-Barahona, L. M., Su, P.-H., Ultes, S., and Young, S. (2017). A network-based end-to-end trainable task-oriented dialogue system. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 438–449, Valencia, Spain. Association for Computational Linguistics.

Yang, Z., Ishay, A., and Lee, J. (2023). Coupling large language models with logic programming for robust and general reasoning from text. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 5186–5219, Toronto, Canada. Association for Computational Linguistics.

Zhao, T. and Eskenazi, M. (2016). Towards end-to-end learning for dialog state tracking and management using deep reinforcement learning. In *Proceedings of the 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 1–10, Los Angeles. Association for Computational Linguistics.

Zhou, D., Schärli, N., Hou, L., Wei, J., Scales, N., Wang, X., Schuurmans, D., Cui, C., Bousquet, O., Le, Q. V., and Chi, E. H. (2023). Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations*.

APPENDIX

Linguistic Annotation of Sentences

Linguistic Annotation of (S1):

```
token(root, root, nner, o, root, 0);
token(if, in, nner, o, if, 1);
token(signs, nns, nner, o, sign, 2);
token(are, vbp, nner, o, be, 3);
token(blurry, jj, nner, o, blurry, 4);
token(activate, vbp, nner, o, activate, 6);
token(esp, nnp, ner, system, esp, 7);
root(root, 0, activate, 6);
mark(blurry, 4, if, 1);
nsubj(blurry, 4, sign, 2);
cop(blurry, 4, be, 3);
advcl(activate, 6, blurry, 4);
dobj(activate, 6, esp, 7);
```

Linguistic Annotation of (S2):

```
token(root, root, nner, o, root, 0);
token(when, wrb, nner, o, when, 1);
token(esp, nnp, ner, system, esp, 2);
token(is, vbz, nner, o, be, 3);
token(inactive, jj, nner, o, inactive, 4);
token(decrease, vb, nner, o, decrease, 6);
token(speed, nn, nner, o, speed, 7);
root(root, 0, decrease, 6);
advmod(inactive, 4, when, 1);
nsubj(inactive, 4, esp, 2);
cop(inactive, 4, be, 3);
advcl(decrease, 6, inactive, 4);
dobj(decrease, 6, speed, 7);
```

Linguistic Annotation of (S3):

```
token(root, root, nner, o, root, 0);
token(increase, vb, nner, o, increase, 1);
token(the, dt, nner, o, the, 2);
token(distance, nn, nner, o, distance, 3);
token(when, wrb, nner, o, when, 4);
token(joe, nnp, ner, person, joe, 5);
token(is, vbz, nner, o, be, 6);
token(driving, vbg, nner, o, drive, 7);
root(root, 0, increase, 1);
det(distance, 3, the, 2);
dobj(increase, 1, distance, 3);
advmod(drive, 7, when, 4);
nsubj(drive, 7, joe, 5);
aux(drive, 7, be, 6);
advcl(increase, 1, drive, 7);
```

Linguistic Annotation of (S4):

```
token(root, root, nner, o, root, 0);
token(engage, vb, nner, o, engage, 1);
token(the, dt, nner, o, the, 2);
token(lights, nns, nner, o, light, 3);
token(when, wrb, nner, o, when, 4);
token(the, dt, nner, o, the, 5);
token(road, nn, nner, o, road, 6);
token(is, vbz, nner, o, be, 7);
```

```
token(wet, jj, nner, o, wet, 8);
root(root, 0, engage, 1);
det(light, 3, the, 2);
dobj(engage, 1, light, 3);
advmod(wet, 8, when, 4);
det(road, 6, the, 5);
nsubj(wet, 8, road, 6);
cop(wet, 8, be, 7);
advcl(engage, 1, wet, 8);
```

Translation Policy Emerged from Coaching Process

```
@Knowledge
E001 :: cop(W1, P1, be, PBe),
nsubj(W1, P1, W2, P2) implies
sdclause(_, W1, P1, W1, W2, P2, W2);

E002 :: token(Word1, POS_Tag,
              NER_Flag, NER_Tag, W1, P1),
?startsWith(POS_Tag, vb),
dobj(W1, P1, W2, P2) implies
siclause(_, W1, P1, W1, W2, P2, W2);

E003 :: siclause(Prefix,
                 W1, P1, WPredicate1,
                 W2, P2, WPredicate2),
?partOf(activate_deactivate_increase_decrease,
        W1)
implies
aclause('!',
        W1, P1, WPredicate1,
        W2, P2, WPredicate2);

E004 :: root(root, 0, W1, P1),
aclause(Prefix, W1, P1, WPredicate1,
        W2, P2, WPredicate2) implies
!generate(head, 0, Prefix,
          WPredicate1, args, vph_1, next,
          WPredicate2, args, vph_1);

E005 :: advcl(WParent, PParent, W1, P1),
sdclause(_, W1, P1, WPredicate1,
         W2, P2, WPredicate2) implies
!generate(body, 1, _,
          WPredicate1, args, vph_1, next,
          WPredicate2, args, vph_1);

E006 :: root(root, 0, W1, P1),
token(Word2, POS_Tag, ner, NER_Tag, W2, P2),
aclause(Prefix, W1, P1, WPredicate1,
        W2, P2, WPredicate2) implies
!generate(head, 0, Prefix,
          WPredicate1, args, WPredicate2);

E007 :: advcl(WParent, PParent, W1, P1),
token(Word2, POS_Tag, ner, NER_Tag, W2, P2),
sdclause(_, W1, P1, WPredicate1,
         W2, P2, WPredicate2) implies
!generate(body, 1, _,
          WPredicate1, args, WPredicate2);

E008 ::
!generate(TYPE, GROUP, Prefix,
          WPredicate1, args, WPredicate2) #
!generate(TYPE, GROUP, Prefix,
          WPredicate1, args, vph_1, next,
          WPredicate2, args, vph_1);

E009 :: aux(W1, P1, be, PBe),
nsubj(W1, P1, W2, P2) implies
sdclause(_, W1, P1, W1, W2, P2, W2);

E010 :: siclause(Prefix,
                 W1, P1, WPredicate1,
                 W2, P2, WPredicate2),
?startsWith(W1, engage) implies
aclause('!', W1, P1, activate,
        W2, P2, WPredicate2);

@Procedures
function startsWith(word, start) {
    return word.startsWith(start);}

function partOf(list, word) {
    return list.split("_").includes(word);}
```