**Algorithm 2: natural language-driven templates** Algorithm 2 does not just fill in the question templates, but from a selected type of questions, fetches all axioms following the related axiom prerequisites, processes the contents of the ontology by fetching the vocabulary elements of a selected axiom, picks an appropriate variant of a template that the vocabulary can be used in, and makes some linguistic adaptation before generating the whole question.

The improvements incorporated were partially informed by the analysis of the sentences generated by Algorithm 1 that were evaluated as 'bad' questions. There are three major changes:

- using class expressions to generate questions, rather than only the declared domain and range of OPs. This generates more sensible questions because it only uses actually asserted and inferred knowledge.
- improving common grammar issues, availing of SimpleNLG and WordNet, for subject and verb agreement, gerund form generation, and article checking. Also, basic part of speech (POS) tagging (by using WordNet) for the classes and OPs was added, to get the appropriate form and to assist with selecting the appropriate template. For instance, with a template *Define a [Thing].*: if *[Thing]* is a noun, one can use this template, but if *[Thing]* is a verb, one has to take the template variant *Define [Thing].* and change the verb (infinitive form) to its gerund form. This improves the sentence from, e.g., "Define a eat" to the grammatically better "Define eating".
- choosing the appropriate template for a given axiom by considering the POS of classes and OPs, and classifying the OP. We designed an algorithm based on an FSM that classifies the name given to an OP to find the appropriate template for an axiom, and provides the appropriate equivalent text. The FSM strategy is a sequence detector to determine the category of an OP and chunks it; e.g., is-eatenBy is transformed into a list of words (is, eaten, by). Then it detects each component, and from that, the POS of each token is obtained, and, finally, it generates the appropriate group of words, which will constitute the question: "is eaten by". The algorithm considers 6 categories, listed and illustrated in Table 1. For instance, an OP that belongs to the category OP_IS_PAST_PART_BY cannot use the template *Does a [Thing] [ObjectProperty] a [Thing]?*. For example, for the axiom $\mathsf{Leaf} \sqsubseteq \exists \mathsf{eaten\text{-}by}.\mathsf{Giraffe}$, the appropriate template is *Is a [Thing] [ObjectProperty] a [Thing]?* and a correct generated question would be "Is a leaf eaten by a giraffe?" rather than "Does a leaf eaten by a giraffe?".

Regarding implementation, OWL API, Hermit Reasoner, SimpleNLG, and WordNet are used by the algorithm to generate questions.

**Algorithm 2**

**Input: type of questions**, ontology, set of templates
Fetch all asserted and inferred **axioms** that satisfies the **axiom prerequisites** for the selected **type of questions** using Reasoner
Select **one axiom** randomly
Fetch the **vocabulary elements** *vocabularyelements* of the selected axiom
Check the **category of the Object Property (OP)** of the selected axiom (Table 1 presents the categories and some examples)
Get the **appropriate template** and **formatted vocabulary element** ($vocOP$) with respect to the type of questions, the selected axiom and the category of the OP from the OP classification {The OP Classifier algorithm (included in the supplementary material) uses **SimpleNLG** to obtain the third-person singular in the present simple tense and the past participle form of a verb, and **WordNet** to check the POS (Part-Of-Speech) of words such as verb, noun, adjective and preposition)}
**for each** $voc$ **in** *vocabularyelements* **do**
    $voc$ = **checkFormat**($voc$) {camel case, hyphen, underscore}
    **if isClass**($voc$) == $true$ **then**
        **if POS**($voc$) == *noun* **then**
            **if isCountableNoun**($voc$) == $true$ **then**
                $formattedVoc$ = **addArticle**($voc$, "a"/"an"/"") {Choose the right article a/an/(empty) for $voc$ by using **SimpleNLG**, if the axiom contains an existential quantifier, do not insert any article}
            **else**
                $formattedVoc$ = $voc$ {$formattedVoc$ is the formatted vocabulary element to fill in the appropriate template}
            **end if**
        **else if POS**($voc$) == $infinitive\_verb$ **then**
            $formattedVoc$ = **gerundForm**($voc$) {By using **SimpleNLG**}
        **else**
            $formattedVoc$ = $voc$ {No transformation if $voc$ is neither a noun nor an infinitive verb}
        **end if**
    **else if isQuantifier**($voc$) == $true$ **then**
        $formattedVoc$ = **addQuantifier**($voc$, "some"/"only") {for $\exists$, use "some"; and for $\forall$, use "only"}
    **else if isObjectProperty**($voc$) == $true$ **then**
        $formattedVoc$ = $voc$ {No further transformation, here, $voc$ = $vocOP$; it is already formatted by the Object Property Classifier)}
    **end if**
    Fill in the selected **appropriate template** with $formattedVoc$ {the formatted vocabulary element}
**end for**
**Output: Question**

Table 1: Object Property classification with illustrative examples

| **Object property** | **Category** | **Equivalent text** |
|---|---|---|
| eats | OP_VERB | eats |
| participate-in | OP_VERB_PREP | participates in |
| hasState | OP_HAS_NOUNS | has a state |
| isContiguousPortionOf | OP_IS_NOUNS_PREP | is a contiguous portion of |
| containedIn | OP_IS_PAST_PART_PREP | is contained in |
| eaten-by | OP_IS_PAST_PART_BY | is eaten by |