Concept2Text: an explainable multilingual rewriting of concepts into natural language*

Flavio Bertini[1,4], Alessandro Dal Palù[1,4,**], Francesco Fabiano[2], Andrea Formisano[3,4,**] and Federica Zaglio[1]

[1]Department of Mathematical, Physical and Computer Sciences, University of Parma, Italy
[2]Department of Computer Science, New Mexico State University, Las Cruces, New Mexico, United States
[3]Department of Mathematics, Computer Science and Physics, University of Udine, Italy
[4]GNCS-INdAM, Gruppo Nazionale per il Calcolo Scientifico

Abstract

Automated and explainable data interpretation hinges on two critical steps: (i) identifying emerging properties from data and representing them into abstract concepts, and (ii) translating such concepts into natural language. While Large Language Models have recently demonstrated impressive capabilities in generating natural language, their trustworthiness remains difficult to ascertain. The deployment of an explainable pipeline enables its application in high-risk activities, such as decision making. Addressing this demanding requirement is facilitated by the fertile ground of knowledge representation and automated reasoning research. Building upon previous work that explored the first step, we focus on the second step, named *Concept2Text*. The design of an explainable translation naturally lends itself to a logic-based model, once again highlighting the contribution of declarative programming to achieving explainability in AI.

This paper explores a Prolog/CLP-based rewriting system designed to interpret concepts expressed in terms of classes and relations derived from a generic ontology, generating text in natural language. Its key features encompass hierarchical tree rewritings, modular multilingual generation, support for equivalent variants across semantic, grammar, and lexical levels, and a transparent rule-based system. We present the architecture and illustrate a simple working example that allows the generation of hundreds of different and equivalent rewritings relative to the input concept.

Keywords

Concept-to-text, Explainable AI, Natural Language, Prolog

1. Introduction

The concept of explainable Artificial Intelligence (explainable AI, or xAI) has emerged to encapsulate essential attributes of AI systems, including transparency and ethical behaviour while also ensuring accountability, security, privacy and fairness [1]. Across various domains, the adoption of AI systems hinges on their capacity to provide comprehensive insights into their internal operations, thus fostering interpretability and transparency of the decision-making process. The recent European Union AI Act aims to establish a harmonized legal framework to promote the development of artificial intelligence while safeguarding public interests such as health, safety, fundamental rights, democracy, and the environment [2]. In particular, it requires that AI systems be sufficiently transparent, explainable and well-documented. From a technical perspective, this simple requirement implies that the system must furnish supporting evidence for its outputs. While this objective remains distant for systems relying on deep neural networks, it presents an opportune challenge for the Logic Programming community, given the inherent explainability of its products. The AI Act adopts a risk-based approach, wherein

CILC 2024: 39th Italian Conference on Computational Logic, June 26-28, 2024, Rome, Italy
* Research partially supported by Interdepartmental Project on AI (Strategic Plan UniUD–22-25), by MaPSART-FAIR project, and by INdAM–GNCS project CUP E53C22001930001.
** Corresponding authors.
flavio.bertini@unipr.it (F. Bertini); alessandro.dalpalu@unipr.it (A. Dal Palù); ffabiano@nmsu.edu (F. Fabiano); andrea.formisano@uniud.it (A. Formisano); federica.zaglio@unipr.it (F. Zaglio)
0000-0001-6925-5712 (F. Bertini); 0000-0003-0353-158X (A. Dal Palù); 0000-0002-1161-0336 (F. Fabiano); 0000-0002-6755-9314 (A. Formisano)

the demand for system explainability is heightened in contexts deemed to carry unacceptable or high risks because they pose a threat to people or affect safety or fundamental rights, such as real-time and remote biometric identification systems and medical devices.

The demand for explainable AI (xAI) applications propels us beyond the "black-box" approach characteristic of traditional sub-symbolic AI. Unlike these systems, which often rely on machine learning or deep learning, the evolving landscape of AI necessitates elucidating the reasoning behind its decisions, particularly in high-risk scenarios, such as automatically describing an electrocardiogram in a medical report.

Our research centers on the process of transforming raw information (e.g., data series) into natural language descriptions. Designing an explainable framework entails an initial step for interpreting and abstracting features (Data2Concept), followed by translating concepts into natural language (Concept2Text). The first step, as outlined in [3, 4], involves identifying desired patterns in raw data and representing them as high-level descriptions or *concepts*. For instance, a time series might exhibit a consistent increase in values over time, which can serve as a logical fact for subsequent processing. These concepts can be enriched with additional knowledge, such as information about the time domain, measured quantities, and contextual factors.

In our previous work, we solely demonstrated a proof of concept for the second step. We presented a basic generator of natural language expressions, serving as a foundational precursor to the system elaborated upon in this paper. Subsequent extensions of that work have already led to the deployment of domain-specific applications. For instance, [5] delineates the utilization of Logic Programming in crafting an explainable decision-making support system tailored for learning analytics in academia.

This paper presents the design of a general and explainable Concept2Text pipeline. Its key features include:

- **Explainability:** The system is grounded in Logic Programming (LP) features, particularly focusing on rewriting rules and Constraint Satisfaction Problems, ensuring transparency for editing and scaling.
- **Modularity:** The system's architecture allows for seamless expansion to accommodate various domain-specific concepts and languages.
- **Tree Rewriting:** Concepts are represented as trees and undergo a series of stages, culminating in the translation of the concept into natural language. These stages progress from the conceptual to the grammatical, ultimately reaching the syntax level.
- **Variants:** To facilitate the generation of diverse semantic-equivalent sentences, each rewriting permits the creation of multiple versions that can be selected. Each stage of rewriting determines various levels of equivalence, ranging from conceptual and structural to grammatical and lexical.
- **Multi-Language Support:** The modular architecture of the system enables the creation of multiple language rule sets, allowing users to select different languages without affecting the overall structure. In this paper, we illustrate an example involving English and Italian.

The paper is structured as follows. Section 2 provides a concise overview of related work on the topic. Section 3 presents the design of the system, while Section 4 demonstrates some practical results. Finally, Section 5 offers concluding remarks.

2. Related Work

2.1. Ontologies

In information science, ontologies serve as organizational frameworks for structuring knowledge within specific domains. They define facts, properties, and their interrelationships using simple representational primitives, such as classes, attributes, and relations among class members [6]. Primarily, they elucidate the relationships between concepts relevant to a given discourse domain. Ontologies can encapsulate a broad spectrum of human knowledge, spanning from art and science to technology and medicine.

In practical terms, ontologies employ specific languages to specify concepts and relations, abstracting away implementation details. The Web Ontology Language (OWL) is a prominent example, designed to enable applications to process information and concepts autonomously, without direct human intervention [7]. Leveraging ontologies can greatly benefit computational reasoning for textual content generation by enabling computers to grasp word meanings and assemble them into complex sentences, akin to human language processing [8].

In the medical realm, ontologies play a pivotal role in unlocking and facilitating computational reasoning, particularly in precision medicine and explainable AI [9]. Biomedical and health sciences extensively employ ontologies to represent knowledge across diverse areas, including diseases [10], gene products [11], phenotypic abnormalities [12], clinical trials [13], vaccine information [14], and the entire human anatomy [15].

2.2. Sub-symbolic text models

In recent times, large language models (LLMs) have surged in popularity, fueled by their remarkable ability to manipulate, summarize, generate, and predict textual content, after training on text corpora, in a manner reminiscent of human language [16]. LLMs, notably based on transformer-based neural network architectures, are characterized by hundreds of billions (or more) of parameters and trained on vast text datasets. Positioned within the realm of generative AI, they excel in generating content based on their training data.

Despite their impressive performance across various text processing tasks, LLMs harbor certain limitations, including susceptibility to misinterpreting instructions, producing potentially biased content, and generating factually incorrect information [17]. These limitations underscore a lack of control over the accuracy and consistency of the concepts synthesized within generated text, leading to undesirable outcomes such as the generation of fake news and plagiarism [18]. Such challenges align with the characterization of LLMs as typical "black box" systems, as described in the EU AI Act, wherein understanding and interpreting their inner workings are inherently challenging.

2.3. Symbolic text models

The utilization of Logic Programming techniques (particularly approaches based on Prolog, ASP, and CLP [19, 20]) in the extraction of concepts from raw data in compliance with xAI standards, and their subsequent translation into natural language expressions, remains a relatively unexplored area in the literature. For instance, Pereira proposed models of grammars and graph-based structures leveraging Prolog's unification [21].

The converse problem, which involves processing natural language text provided as input, has gained significant attention from the Logic Programming (LP) community [22]. Definite Clause Grammars (DCGs), a well-known tool introduced in the 1980s primarily for parsing natural and artificial languages using explicit grammar rules and Prolog syntactic sugar, have been instrumental in this endeavor [23]. DCGs are bidirectional in nature, offering a methodology not only for parsing but also for generating text from a controlled context-free grammar that adheres to Backus-Naur Form (BNF) rules. Additionally, they facilitate the modeling of multiple variants of the grammar tree. Furthermore, examples of formalization of language structure, although not rule-based or operational, can be found in the literature [24].

3. Model

We now focus into the design choices of the system. For the sake of simplicity, let us assume that a concept is articulated in terms of a set of general classes and the relations among them. While a graph could serve as a suitable structure to host such knowledge, with classes represented as nodes and relations as (hyper-)edges, we opt in this paper to operate with a spanning tree that captures the most pertinent details for description. Even if for now we manually generate the tree, in future work,

```
[class(student),
   [rel(attribute),attribute(plural)],
    [rel(attributive_spec),class(course)]
```

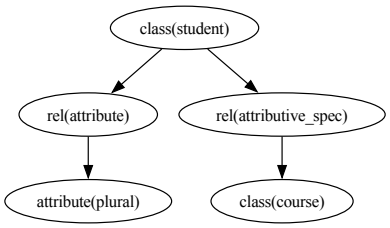


Figure 1: Example of an input concept (Prolog List on the left and corresponding tree on the right).

we intend to explore methods for automatic selection and regulation of the amount of information loss in this process. As an illustrative example, let us encode the concept of *students of a course*. Here, we encode the concepts of *student* and *course* using a predicate `class/1`, while utilizing the `rel/1` predicate to denote relevant relations attributed to them. For instance, we may wish to specify that *student* is plural and establish a relationship between students and a specific subset associated with those attending a *course*. Syntactically, we structure a Prolog nested list [Root,Child_1,…,Child_n], where children may contain further nested lists. Figure 1 illustrates the encoding (on the left) and the corresponding tree visualization (on the right).

The core idea revolves around processing a tree that describes a concept, as depicted above, to derive an equivalent grammar tree containing a well-formed sentence. We have devised a uniform rule-based rewriting system to drive this process. During rewriting, trees undergo structural and semantic modifications. Different sets of rules govern distinct stages in the transformation of the input tree. Although we initially considered using DCGs, we encountered two primary limitations: Firstly, DCG rules naturally model the structure of a specific tree rather than the rewriting relationships of a subtree into a new subtree. Although adaptable for tree rewriting, the head of DCG rules does not support a general tree shape. Secondly, in cases involving multiple alternative outputs of a rule (variants), explicit control over the selection process is necessary, as opposed to allowing the SLD resolution to explore all possibilities. Consequently, we opted to model a native tree-to-tree rewriting system instead of adapting DCGs for this purpose. Notably, our approach eliminates the need to generate rules for determining the well-formedness of a sentence in a specific language, as a DCG would. Instead, the rewriting rules are designed to ensure that the resulting trees ultimately conform to the grammar rules. Moreover, our approach ensures the lack of over-generation and under-generation of sentences w.r.t. the implicit grammar modeled as the set of rewriting rules.

Let us explore how tree rewriting is conceptualized (further details in Section 3.1.1). When making changes to a tree structure, we need to pinpoint specific conditions that trigger these modifications, typically based on the presence of certain subtrees and their relationships embedded in the larger structure. We expect a rule to encompass the lowest node that is an ancestor of all relevant subtrees, including locations that are affected by the rule rewriting (potentially elsewhere in the tree).

Each rule is responsible for constructing a new subtree that replaces the previous content hanging from that node. While some cases involve straightforward substitutions of one subtree with another, more complex scenarios can entail assembling intricate structures by combining existing components, rearranging their structure, and introducing new elements.

This level of generality enables the modeling of typical semantic equivalent rewritings as well as grammatical transformations (e.g., active vs. passive voice, word to pronoun substitution, etc.).

Concept rewriting into text is governed by tree rewriting rules, implemented through a fixed-point algorithm. We have identified different stages to structure the process. Although the fixed-point rewriting mechanism is common across all stages, we prefer to tailor rules and introduce barriers to fix-points. This approach offers several advantages, including the ability to accommodate language-independent stages alongside those requiring language-specific rules. This modular setup also contributes to a more organized system.

The concept-to-text rewriting process can be broken down into several stages, each addressing specific objectives. The overall construction of the final tree benefits from the iterative tree rewriting performed at each stage's fix-points. Depending on the rules activated at each stage, we achieve different objectives as outlined below:

1. **Equivalent Concepts:** Concepts represented as relations among classes can be rewritten into equivalent forms. The output remains a concept-based tree containing more detailed descriptions or equivalent concepts captured by specific class-dependent properties and ontological relations. This stage ensures semantic variants of the concept, leading to the furthest but still equivalent final text.
2. **Concept2Structure:** This stage transforms the tree of concepts and their relations into a prototype of grammar structure. It constructs the components of a sentence (noun and verbal subtrees, as well as complements) and introduces certain structural relationships (e.g., potential information forwarding across subtrees). Classes and relations retain their ontology descriptions.
3. **Structure2Grammar:** While maintaining the overall structure, this stage translates each class and relation into grammar lexemes and/or other simple grammatical forms.
4. **Coordination:** This stage ensures that subtrees are coordinated, as necessary, to match gender and number for nouns, verbs, etc.
5. **Inflection and Sorting:** Responsible for producing the correct inflections for nouns and adjectives, as well as conjugations for verbs. Additionally, it computes the correct word order for words within the same phrase through the resolution of language-dependent Constraint Satisfaction Problem (CSP).
6. **Syntax:** Applies local rules to consecutive words to ensure syntactic properties are met (e.g., contractions, ellipsis, etc.).

In the following we provide some details about key aspects of implementing this model.

3.1. Implementation

3.1.1. Tree rewriting

Our objective is to devise rules flexible enough to handle common language properties, such as subtree swapping and restructuring. The sequence of transformations outlined in the preceding section has been realized using Prolog. Here, we provide a brief overview of the main components of this implementation.

Each transformation in the process takes a tree as input and produces a list of trees as output (representing possible variants according to a rule). A tree is represented as a Prolog list `[RootInfo|Children]`, where `Children` is the list of child trees.

The phases of the process operate as a pipeline (as discussed earlier), with each phase rewriting the output of the previous one by applying a set of rewriting rules. These rule sets are unique and tailored to the specific requirements of each phase. However, all rules are uniformly described using Prolog clauses that define the predicate `rule(Lang,Type,Name,Tree,RewTree)`, where:

- `Lang`: Specifies the target language of the translation, which remains consistent throughout the process. Currently, the accepted values for `Lang` are limited to English and Italian.
- `Type`: Indicates a specific phase of the rewriting process, as described in Section 3. Acceptable values for `Type` include the atoms: `equiv_concept`, `concept2structure`, `structure2grammar`, `coordination`, `inflection` and `syntax`.
- `Name`: Identifies the applied rule, distinguishing a specific rewriting among those possible in the phase `Type`.
- `Tree`: Represents the tree to be rewritten by the clause.
- `RewTree`: The rewritten version of `Tree`.

Each phase repeatedly applies its rules to the input `Tree`, until a fix-point is reached (i.e., no more rules of `Type` are applicable). This is implemented by the following Prolog clause:

```
1 fixpoint_rewrite(Lang, Type, InputTree, OutputTree, [Rewriting|Rewritings]) :-
2   rewrite(Lang, Type, InputTree, TempTree, Rewriting, Changed),
3   ( (Changed == 'changed')  % if fix-point reached
4       -> fixpoint_rewrite(Lang, Type, TempTree, OutputTree, Rewritings)
5       ; (OutputTree=TempTree, Rewritings=[]) ).
```

where, the predicate `rewrite/6`, occurring in line 2, implements a Breadth-First Search (BFS)-based rewriting of `InputTree`:

```
6 rewrite(Lang, Type, InputTree, OutputTree, RuleTree, Changed) :-
7   mk_empty_queue(EmptyQ), % EmptyQ is an empty queue
8   enqueue(InputTree-OutputTree-RuleTree, EmptyQ, QofTrees),
9   bfs_rewrite(Lang,Type, QofTrees, 'unchanged', Changed).

11 bfs_rewrite(_Lang, _Type, QofTrees, Changed, Changed) :-
12   is_empty_queue(QofTrees). % no more sub-trees
13 bfs_rewrite(Lang, Type, QofTrees, ChangetTilNow, Changed) :-
14   dequeue(Tree-RewTree-RuleTree, QofTrees, QofTrees1),
15   multiple_rewrite_tree(Lang, Type, Tree, NewTree, Applied),
16   ((Applied == 'nop') -> (TempChanged=ChangetTilNow) ; (TempChanged='changed')),
17   (isLeafTree(NewTree) % if it is leaf build a leaf for RuleTree
18     -> (NewTree=RewTree, singletonTree(Applied, RuleTree), QofTrees2=QofTrees1)
19     ; (getChildrenTrees(NewTree, Root, Children), % Gets the sub-trees and
20        length(Children, Num),                     % their number
21        length(Ls, Num), length(Rs, Num), % Creates two lists of Num fresh variables
22        getChildrenTrees(RewTree, Root, Ls),     % Constraints the structure of the
23        getChildrenTrees(RuleTree, Applied, Rs), % terms RewTree and RuleTree
24        enqueueTriples(Children, Ls, Rs, QofTrees1, QofTrees2)) ),
25   bfs_rewrite(Lang,Type, QofTrees2, TempChanged, Changed).
```

BFS traversal is implemented in the standard manner, utilizing a queue that stores, for each unvisited element, a Prolog term of the form `Tree-RewTree-RuleTree`. Here, `Tree` represents the subtree to be visited (and possibly rewritten), while `RewTree` and `RuleTree` are initially variables that will be instantiated during the traversal.

Specifically, `RewTree` will be instantiated to the rewritten version of `Tree`, while `RuleTree` will be a term isomorphic in shape as `RewTree`, describing the applied rule(s) for each node of `RewTree`.

It is important to note that the information gathered in `RewTree` plays a vital role in ensuring the explainability of the approach. `RewTree` serves as a description of the justification for each rewrite performed. This is achieved by recording the `Name` argument found in the definition of the clause(s) of `rule/5` (as seen earlier) used in the rewriting process. Currently, this information comprises a rule ID, but richer knowledge can also be easily managed if needed.

Each dequeued subtree (line 14) undergoes rewriting by `multiple_rewrite_tree/5` (line 15), which iteratively applies rules of `Type` to `Tree` until a fix-point is reached. The resulting tree `NewTree` and `Applied`, which gathers the names/justifications of all applied rules, reflect this step. The Prolog atom `'nop'` is used if no rules are applied.

In line 24, the predicate `enqueueTriples/5` enqueues a term `Child-L-R` for each child tree of `NewTree`. Here, `L` and `R` are fresh variables (created in line 21 as members of lists with as many variables as the number of subtrees to be enqueued), to be instantiated in the subsequent part of the BFS visit.

At the conclusion of the visit, `OutputTree` will be instantiated to the result of the rewriting phase. Both `OutputTree` and the last argument of `fixpoint_rewrite/5` (i.e., `[Rewriting|Rewritings]`) will have an isomorphic structure (as mentioned, lists representing trees). Therefore, the explanation for each node in `OutputTree` can be found in the corresponding node of `[Rewriting|Rewritings]`.

The final stage (syntax rewriting) is executed by a simplified version of the aforementioned program. The tree is flattened, and leaves are extracted in order to form a straightforward list of words comprising the sentence. This list of words is then rewritten until a fixpoint is reached, using a set of rules that only inspect pairs of consecutive words.

Here are some additional details about `rule/5`. Each rule generates a list of trees as alternative variants. When applying a rule, we must select one variant from this list. We've chosen a random selection strategy that takes into account previous choices. This approach has proven particularly effective in preventing the repetition of the same structure in different parts of the final sentence. We keep track of the choice history for each rule using simple assertions. In cases where the same rule is fired multiple times, we ensure that the last choice is avoided if possible.

3.1.2. Language independent rules

The first stage, responsible for handling equivalent concepts, is language-independent. While classes and relations must be named according to a specific language (english in the paper), this naming convention does not affect the generation of a specific language, as they will be converted later according to language-dependent rules.

Now, let us introduce another working example to illustrate some key features contained in the stages described above. We'll model the concept of an *interval* that specifies the use of the *year* class as the unit of measure (uom):

```
> [class(interval),[rel(attribute),attribute(uom(class(year)))],...]
```

If a common knowledge ontology is accessible, we could utilize the information that `is_a(year,time)`, implying that the class *interval* can be further specified as an interval of time. Moreover, additional semantic knowledge about equivalences could inform the rewriting rules that an interval of time is equivalent to the class *period*.

Implementing rules that trigger whenever common knowledge adds some information is straightforward. In this case, a sequence of rewritings could be:

```
> [class(interval),[rel(attributive_spec),attribute(class(time))],...]
> [class(period),...]
```

where `attributive_spec` represents the specification attribute provided by the class *time*.

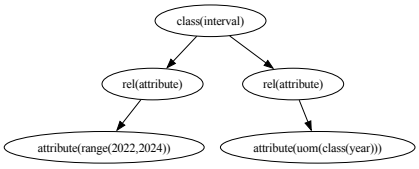
Let also discuss some potential equivalences that can be drawn for an interval that deals with a range of numbers V_1 and V_2, e.g.:

```
> [class(interval),[rel(attribute),attribute(range(V1,V2))],...]
```

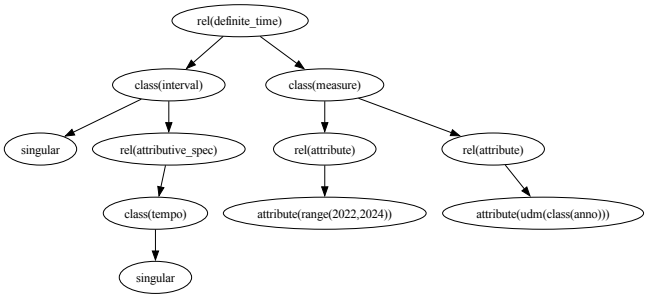
Focusing on the treatment of the class *interval*, we can propose various alternative versions. These include: (i) presenting the interval as a simple measure of a range, (ii) using it as a temporal complement to introduce the interval (e.g., during the interval ...), and (iii) employing a more refined version with the addition of a relative subordinate (e.g., during the interval that spans ...). Furthermore, the actual measure itself (the range between two numbers) can be expressed using different prepositions (from ... to, between ..., starting from ... until). It is evident that the combinatorial explosion of such rewritings gives rise to a rich set of alternative options even at the concept stage.

Figure 2 illustrates a simplified example demonstrating the application of two rewriting rules described above. It is noteworthy how the classes are matched and rewritten: from (a) to (b), the class *interval* is replaced with the left subtree, introducing the concept of measure; from (b) to (c), the class *measure*, denoting a range, is rewritten into a nesting of two complements (source and goal) with measures of simple numeric quantities. Let us show a simplified snippet of the first rule:

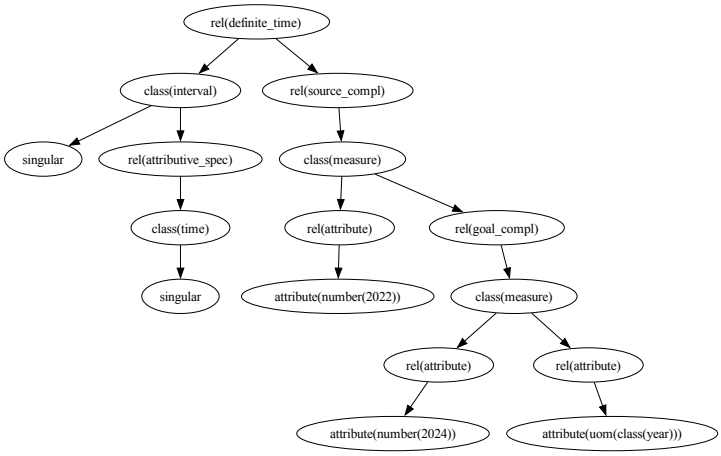
```
1  rule(_Lang,equiv_concept,equiv_interval,[Root|C],
2               [[Root|C2], ... ] ]):- %%% list of equivalent trees
3       %%% firing condition
4       member([class(interval)|C1],C),
5       member([rel(attribute),attribute(range(V1,V2))],C1)
6       member_non_var([rel(attribute),attribute(uom(class(Uom)))],C1),
7       %%% prepare new tree structure
8       %%% (depending whether isa relation in known)
9       ( common_knowledge_isa(Uom,Isa),!,
10        Int=[class(interval),[rel(attribute),attribute(singular)],
11        [rel(attributive_spec),[class(Isa),singular]]];
12        Int=[class(interval),[rel(attribute),attribute(singular)]]),
```



(a) Input concept



(b) Rewriting of concept *interval*



(c) Rewriting of concept *measure* with range

Figure 2: Example of rewriting with concept equivalence.

```
13        replace(C,[class(interval)|C1],[[rel(definite_time),Int,[class(measure)|C1]]],C2),
14        ...
```

Let us assume a predicate `replace/4` that takes the input list, the element to be replaced, the replacement list (enclosed by an additional list, in case multiple elements need to be inserted), and returns the output list.

For the second rule applied in the example, a simplified code snippet could be:

```
1  rule(_Lang,equiv_class,measure_range, [Root|C],
2               [[Root|C4], ... ]):-  %%% list of equivalent trees
3               member([class(measure)|C1],C),
4               member([rel(attribute),attribute(range(N1,N2))],C1),
5               (El=[rel(attribute),attribute(uom(class(U)))],
6                member(El,C1),!,Uom=[El];  %% there is a UoM specified
7               Uom=[]),
8               replace(C,[rel(attribute),attribute(range(N1,N2))],[[]],C2),
                    replace(C2,El,[[]],C3),
9               %%% replace subtree at measure class with single measures
10              replace(C3,[class(measure)|C1],[[rel(source_compl),
11              [class(measure),[rel(attribute),attribute(number(N1))],
12              [rel(goal_compl),[class(misura),[rel(attribute),attribute(number(N2))]|Uom]
                    ]|Uom]]],C4),
13        ...
```

The second stage (concept to structure) is essentially language independent. It involves creating

internal nodes to house information about their subtrees. Specifically, classes and relations denote specific grammar subtrees referred to as phrases. These phrases can include nouns, verbs, propositions, relative phrases, etc., identified through the analysis of each subtree and the presence of various relations.

During the rewriting process, concepts are structurally adapted into phrases within the tree. For instance, a subject may have a verb relation as one child, and an object may be associated with the verb as its child. This three node branch at the concept level is flattened, resulting in three ordered siblings (subject, verb, object).

Internal nodes are enriched with explicit descriptions of their subtrees. This information is encoded using a predicate `info/4`, which specifies the type of phrase (using standard linguistic terminology such as np, vp, pp, rp for noun, verbal, propositional, relative phrases), the subtype (e.g., subject, object), and the gender and number attributes that apply to the subtree.

The fourth stage is language independent as it is responsible for linking (via unification) the gender and number variables contained in the `info/4` nodes. Some links are predetermined by default (e.g., between subject and verb), but in other cases, a previous rewriting may have required an explicit coordination (e.g., a relative subordinate phrase must match the gender and number of the associated noun, as soon as that subtree is create in a next stage). At this stage, all coordinations are enforced essentially by ensuring unification where necessary.

The remaining stages are language dependent and will be discussed in the next section.

3.1.3. Language dependent rules

The overall structure allows us to focus on language-dependent rules for specific stages: structure2grammar, inflection, and syntax. One advantage of this model is that we can easily plug in sets of rules without modifying the system.

The structure2grammar stage is responsible for translating classes and relations into their counterparts associated with the target language. Typically, an explicit mapping of classes to grammar lexemes is required. While we are exploring automated tools to streamline this phase, for small domain-specific applications, manual crafting is an option. At this stage, synonyms can be suggested as variants. Classes and relations can generate a list of lexemes, and the tree structure can be modified accordingly. In general, each object is encapsulated by a parent node that provides its type (e.g., noun, adjective, number, verb, preposition, etc.). This tagging allows a simple reasoning when determining the correct order of elements within a phrase.

Let us now provide some details of the inflection stage. Here, the rewriting process governs the selection of appropriate inflections for nouns, adjectives, and verbs, considering their gender and number. This is typically accomplished by consulting a dictionary that explicitly associates lexemes with words. When dealing with verbs, additional considerations come into play to determine the correct tense. Auxiliary verbs are generated according to the rules of the target language.

After the inflection process, another crucial step is required: arranging the words in the correct order within each phrase. Languages have various rules governing the order of words in a phrase, and attempting to cover all possible cases would be impractical due to the combinatorial explosion of possibilities.

To address this challenge, we have devised a set of rules that describe local and partial orderings among subsets of words within the phrase. By combining these partial orders, we can derive the correct total order of words. These orderings may depend on word types and specific words themselves. For example, the fact that in the English language an adjective appears before a noun, is rendered by imposing a constraint on their relative positions in the noun phrase.

We represent this network of sorting constraints as a Constraint Satisfaction Problem (CSP). While solving the CSP itself is straightforward, developing accurate order constraints requires careful tuning. Thus, the flexibility of a CSP allows to support the updates of the constraints considered.

The final stage addresses the enforcement of writing rules for adjacent words. Every language possesses distinct rules governing word combinations, typically controlled through local pattern matching

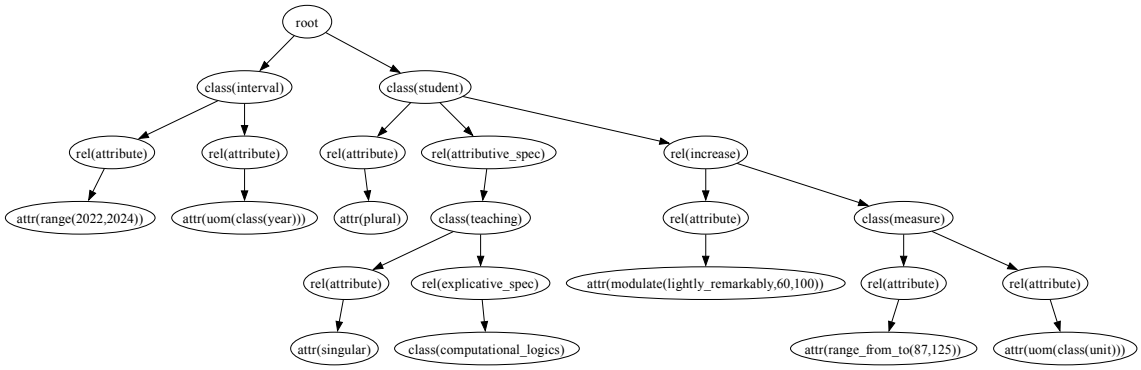


Figure 3: Input concept for a full sentence

at the word or character level. For instance, this stage handles scenarios where two words are merged into contractions or a single letter is removed to indicate ellipsis.

Operating on the leaves of the tree, which at this stage contain the individual words of the sentence, this stage ensures adherence to language-specific rules. By conducting a Depth-First Search (DFS) traversal of the tree, the correct order for assembling the text is recovered.

4. Results

Let us consider a complete sentence that embodies the typical elements of a concept description generated by a Data2Concept framework. This sentence includes temporal characterization over a range in time (class *years*), describing a phenomenon that has increased over time (by a range of values). It also specifies the unit of measurement (integer units), employs past tense, and allows for the potential use of a relative subordinate clause to specify the interval of time. This showcases how several descriptions can be generated by simply changing classes and relations within this prototype concept.

In the following listing, we provide the concept tree represented in the form of a list. Lines 1–3 describe the interval with attributes for the numerical range and the unit of measure (years). Line 4 defines the subject (class student, plural), and lines 5–6 specify that the students are associated with a course. Line 7 adds information that the course pertains to computational logics. Another attribute of the course, described in line 8, specifies that an increase occurred. Its attributes include modulation through an adverb (intensity of 60% on a linear scale where adverbs are selected according to the intensity) and the actual measure, providing a numerical range and the unit of measure (class unit).

```
1  [class(interval),
2   [rel(attribute),attr(range(2022,2024))],
3   [rel(attribute),attr(uom(class(year)))]],
4  [class(student),plural,
5    [rel(attributive_spec),
6      [class(course),singular,
7        [rel(explicative_spec),class(computational_logics)]]],
8    [rel(increase),
9      [rel(attribute),attr(modulate(lightly_remarkably,60,100))],
10     [class(measure),
11       [rel(attribute),attr(range_from_to(87,125))],
12       [rel(attribute),attr(uom(class(unit)))]]]]
```

Figure 3 depicts the corresponding tree representation of the list.

The set of rules we developed for generating basic variants resulted in a satisfactory combinatorial expansion of rewritings. Specifically, when testing the concept described above, we obtained more than

500 distinct sentences for both English and Italian languages. In Table 1, we present examples of output text for English rules (1–3) and for Italian (4–6). It is noteworthy how the rewriting of equivalence and grammar structures is capable of providing remarkable differences, while preserving semantics perfectly, thanks to the strict transitivity of the applied equivalences. The resulting text appears natural, although some additional synonyms could be included to enrich and diversify certain words (e.g., student, significantly, class).

1. During the interval of time that has spanned from 2022 until the year 2024 students of the class of computational logics have significantly increased from 87 to 125 units.
2. There has been a pronounced growth of students of the class of computational logics starting from 87 until 125 units in the interval of time between the years 2022 and 2024.
3. From the year 2022 and during the next 2 years students of the class of computational logics have significantly incremented starting from 87 until 125 units.
4. C'è stato un incremento deciso di studenti del corso di logica computazionale da 87 fino a 125 unità tra gli anni 2022 e 2024.
5. Durante l'intervallo di tempo che è intercorso dal 2022 all'anno 2024 gli studenti del corso di logica computazionale sono decisamente incrementati a partire da 87 fino a 125 unità.
6. Nel periodo tra gli anni 2022 e 2024 gli studenti del corso di logica computazionale sono decisamente cresciuti a partire da 87 fino a 125 unità.

Table 1
Output examples. 1–3 for English and 4–6 for Italian.

In Figures 4, 5, and 6, we present a sequence of tree rewritings for each stage, beginning with the input illustrated in Figure 3. The rewritten trees are depicted after each stage has reached its fixed point.

5. Conclusions

We introduced a Prolog-based rewriting system that converts concepts, represented as trees, into natural language. The system's modularity allows for easy adaptation to various domain-specific applications and output languages. Moreover, the system adheres to explainable AI standards by offering transparency and verifiability. Initial findings demonstrate its effectiveness when combined with an xAI concept extractor, thereby forming a comprehensive data-to-text explainable pipeline.

This work opens different lines of research for further exploration. Developing a comprehensive rule set for accurately translating classes into suitable grammar synonyms is a complex task, as the most appropriate choices depend heavily on context. We aim to devise automatic methods to retrieve such preferences and stylistic usages.

While we currently focus on a tree-like set of relations associated to a concept, other approaches may yield a general graph. Converting this graph into a spanning tree, or alternatively, synthesizing the information in a controlled manner, could facilitate the creation of guided concept summaries.

The entire pipeline is versatile and applicable across various domains, particularly in scenarios where reports are generated based on data analytics. We intend to extend this methodology to automated analysis of ECGs and other medical data, financial data, and more broadly, to produce trustworthy Business Intelligence (explainable automated reporting). Furthermore, we aim to leverage our findings to enable real-time and interactive dialogue, facilitating robot-guided sessions for cognitive impairment rehabilitation.

References

[1] A. B. Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. García, S. Gil-López, D. Molina, R. Benjamins, et al., Explainable artificial intelligence (xAI): Concepts, taxonomies, opportunities and challenges toward responsible ai, Information fusion 58 (2020) 82–115.

[2] European Commission, Regulation of the European Parliament and of the Council. Laying down harmonised rules on Artificial Intelligence (Artificial Intelligence Act), 2021. https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:52021PC0206.
[3] A. Dal Palù, A. Dovier, A. Formisano, Towards explainable data-to-text generation, in: A. Dovier, A. Formisano (Eds.), Proc. of the 38th Italian Conference on Computational Logic, CILC 2023, Udine, Italy, June 21-23, 2023, volume 3428 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023.
[4] A. Dal Palù, A. Dovier, A. Formisano, An xAI approach for data-to-text processing with ASP, in: E. Pontelli, et al. (Eds.), Proc. of the 39th International Conference on Logic Programming, ICLP 2023, Imperial College London, UK, 9th July 2023 - 15th July 2023, volume 385 of *Electronic Proceedings in Theoretical Computer Science, EPTCS*, 2023, pp. 353–366. doi:10.4204/EPTCS.385.38.
[5] F. Bertini, A. Dal Palù, A. Formisano, A. Pintus, S. Rainieri, L. Salvarani, Students' careers and AI: a decision-making support system for academia, in: Proceedings of the 2023 Italia Intelligenza Artificiale - Thematic Workshops, (Ital-IA 2023), Pisa, Italy, May 29-30, 2023, volume 3486 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023, pp. 272–277.
[6] S. Staab, R. Studer, Handbook on ontologies, Springer Science & Business Media, 2013.
[7] D. L. McGuinness, F. Van Harmelen, et al., OWL web ontology language overview, W3C recommendation 10 (2004) 2004.
[8] R. Speer, J. Chin, C. Havasi, Conceptnet 5.5: An open multilingual graph of general knowledge, in: Proceedings of the AAAI conference on artificial intelligence, volume 31, 2017.
[9] M. A. Haendel, C. G. Chute, P. N. Robinson, Classification, ontology, and precision medicine, New England Journal of Medicine 379 (2018) 1452–1462.
[10] L. M. Schriml, et al., Human disease ontology 2018 update: classification, content and workflow expansion, Nucleic acids research 47 (2019) D955–D962.
[11] M. Ashburner, et al., Gene ontology: tool for the unification of biology, Nature genetics 25 (2000) 25–29.
[12] S. Köhler, et al., The human phenotype ontology in 2021, Nucleic acids research 49 (2021) D1207–D1217.
[13] B. Smith, M. Ashburner, C. Rosse, J. Bard, W. Bug, W. Ceusters, L. J. Goldberg, K. Eilbeck, A. Ireland, C. J. Mungall, et al., The OBO foundry: coordinated evolution of ontologies to support biomedical data integration, Nature biotechnology 25 (2007) 1251–1255.
[14] Y. Lin, Y. He, Ontology representation and analysis of vaccine formulation and administration and their effects on vaccine immune responses, Journal of biomedical semantics 3 (2012) 1–15.
[15] N. F. Noy, M. A. Musen, J. L. Mejino Jr, C. Rosse, Pushing the envelope: challenges in a frame-based representation of human anatomy, Data & Knowledge Engineering 48 (2004) 335–359.
[16] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, et al., A survey of large language models, arXiv preprint arXiv:2303.18223 (2023).
[17] Y. Wang, W. Zhong, L. Li, F. Mi, X. Zeng, W. Huang, L. Shang, X. Jiang, Q. Liu, Aligning large language models with human: A survey, arXiv preprint arXiv:2307.12966 (2023).
[18] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, P. Fung, Survey of hallucination in natural language generation, ACM Computing Surveys 55 (2023) 1–38.
[19] A. Dovier, A. Formisano, E. Pontelli, Parallel answer set programming, in: Y. Hamadi, L. Sais (Eds.), Handbook of Parallel Constraint Reasoning, Springer, 2018, pp. 237–282. doi:10.1007/978-3-319-63516-3_7.
[20] A. Dovier, A. Formisano, G. Gupta, M. V. Hermenegildo, E. Pontelli, R. Rocha, Parallel logic programming: A sequel, Theory Pract. Log. Program. 22 (2022) 905–973. doi:10.1017/S1471068422000059.
[21] F. C. N. Pereira, Grammars and logics of partial information, in: Proc. 4th ICLP, 1987, pp. 989–1013.
[22] F. C. N. Pereira, S. M. Shieber, Prolog and natural-language analysis, Microtome Publishing, 2002.
[23] Y. Matsumoto, H. Tanaka, H. Hirakawa, H. Miyoshi, H. Yasukawa, BUP: a bottom-up parser embedded in prolog, New Generation Computing 1 (1983) 145–158.
[24] M. den Dikken, The Cambridge handbook of generative syntax, Cambridge University Press, 2013.

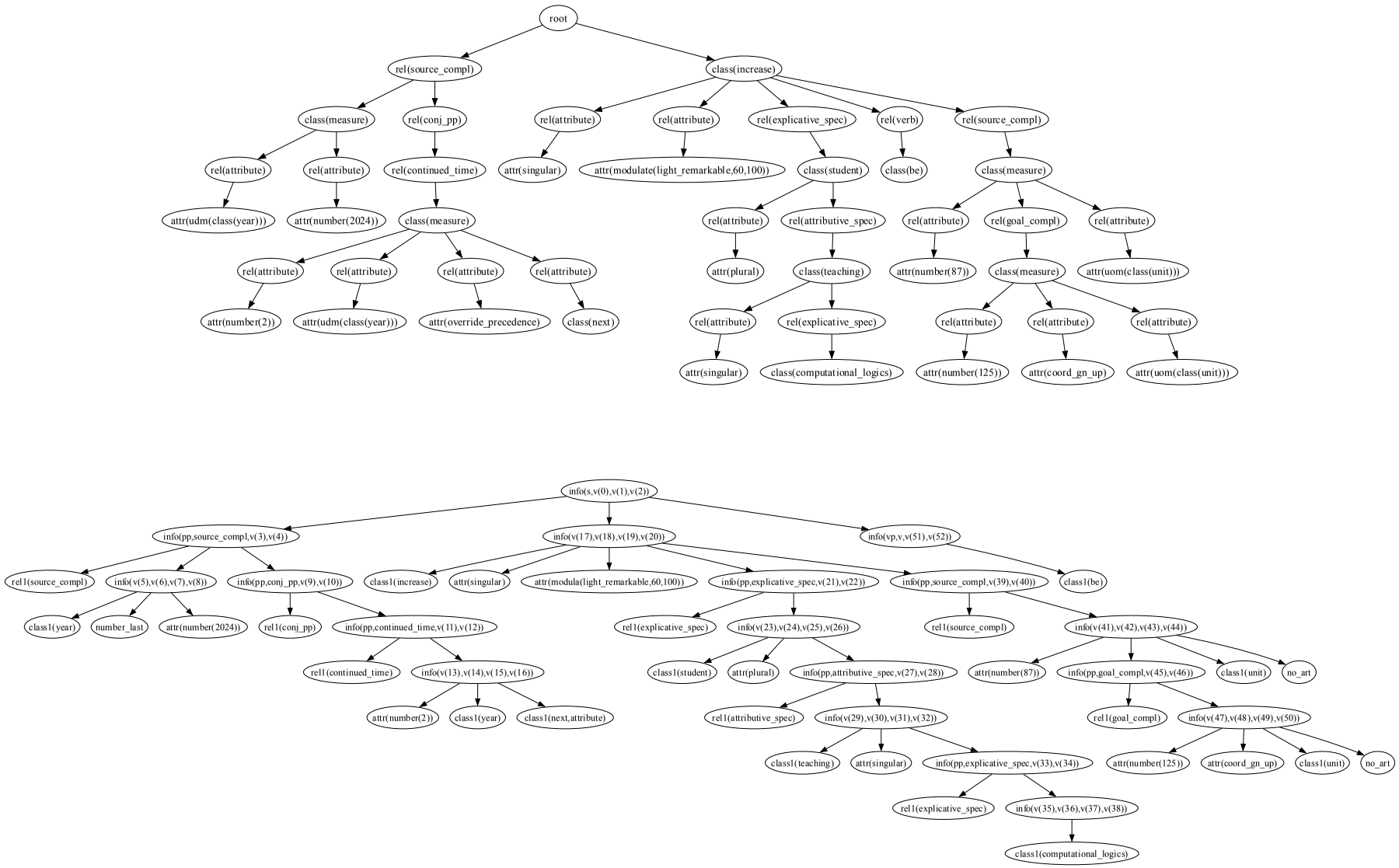


Figure 4: After concept equivalence (left) and after concept2structure (right)

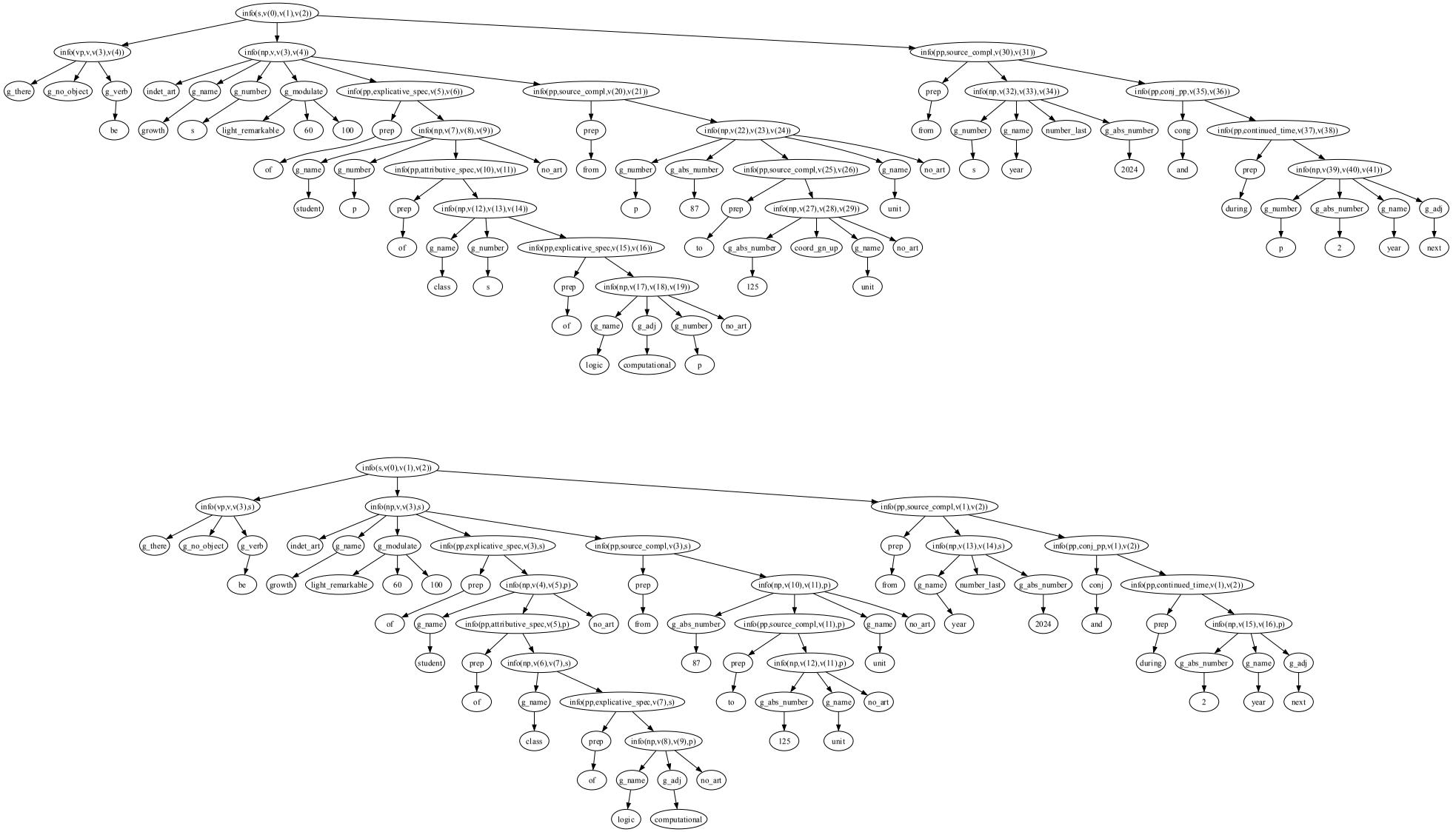


Figure 5: After structure2grammar (left) and after coordination (right)

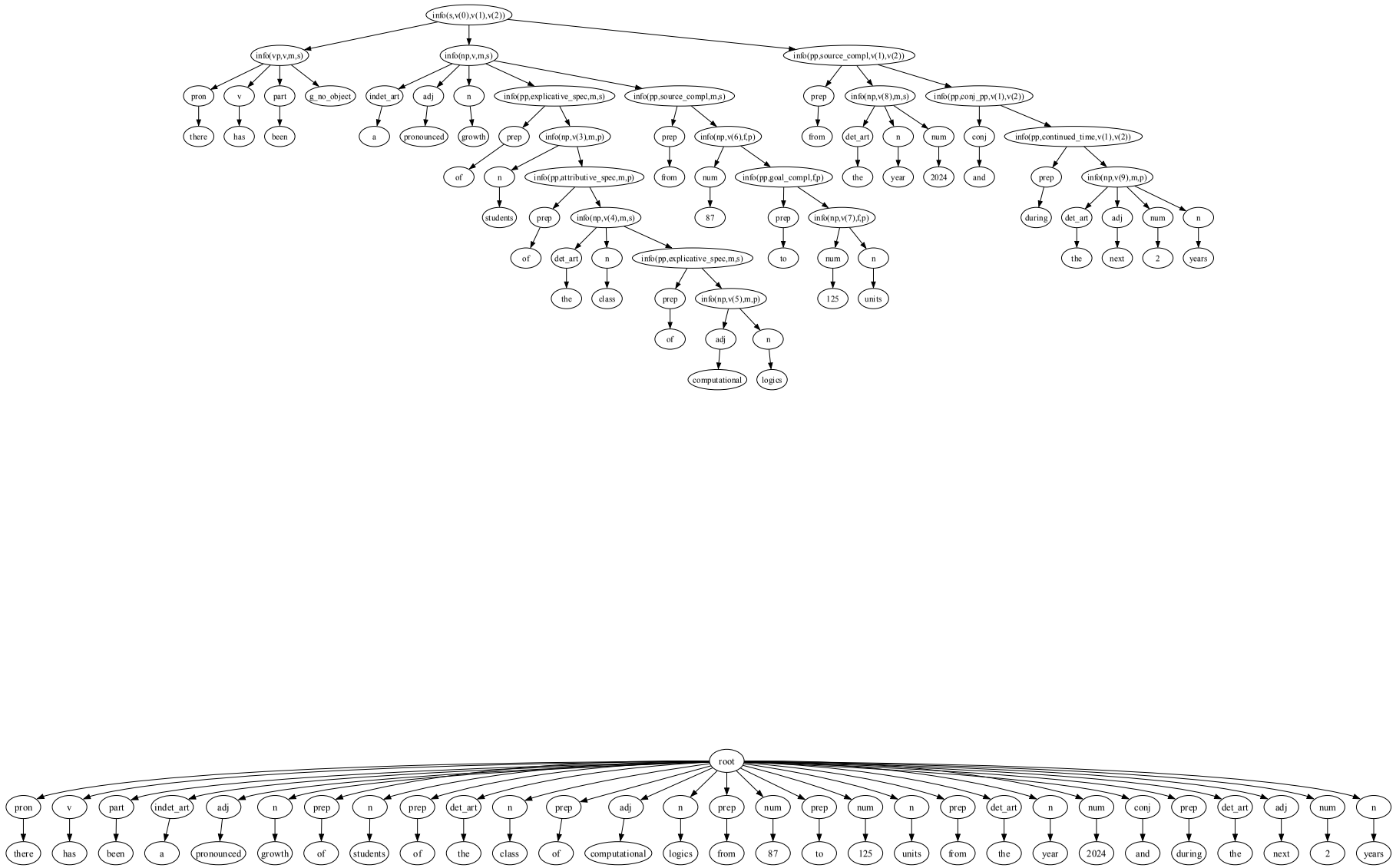


Figure 6: After inflection (left) and after syntax (right)