NL2UNIFOL: From Natural Language Sentences to Uniform First-Order Logic Formulae

Matteo Magnini[1], Davide Liga[1] and Luca Pasetto[1]

[1]University of Luxembourg, Esch-sur-Alzette, Luxembourg

Abstract

Automatic translation of Natural Language (NL) sentences into logic representation – such as First-Order Logic (FOL) formulae – is a task of great interest for many communities, including Knowledge Representation (KR), Natural Language Processing (NLP), normative Artificial Intelligence (AI), and AI in general. Recently, an increasing number of works have explored the use of Large Language Models (LLMs) to translate NL into FOL, with promising results. However, existing works mostly focus on the translation of a single NL sentence at a time, resulting in independent formulae that do not share a common predicate and constant name space.

This work presents a new framework – namely Natural Language to Uniform First Order Logic (NL2UNIFOL) – that leverages LLMs to translate NL sentences into a *uniform* FOL theory, where formulae share the same vocabulary. NL2UNIFOL is an end-to-end pipeline that: (i) translates NL sentences into preliminary FOL formulae; (ii) identifies and *safely* merges predicate and constant names to obtain a *uniform* vocabulary; (iii) finally generates the final FOL theory. We use NL-FOL pairs from the MALLS dataset to validate our framework.

Results show that in the majority of cases NL2UNIFOL is able to correctly create clusters of predicate and constant names to be merged together, which allows to obtain a *uniform* FOL theory. We also recognise that there are still a numerous amount of cases in which the represented name and the representative name semantically diverge, which motivates future work to further improve the symbol clustering process.

Keywords

Natural Language Processing, First-Order Logic, Large Language Models, Knowledge Representation

1. Introduction

The automatic translation of Natural Language (NL) sentences, or full corpora, into formal logic representations is a crucial task for many Artificial Intelligence (AI) communities. The task gains particular importance in legal and normative AI applications, where regulations and norms are often expressed in NL, but need to be formalized for automated reasoning. The manual translation of NL to formal logic is a time-consuming and error-prone process, that needs to be carried out by experts with knowledge of both domain and formal logic. The automatic translation into a formal language was traditionally approached through classical Natural Language Processing (NLP) techniques, such as rule-based systems and statistical models. Recently, there has been a growing interest in leveraging Large Language Models (LLMs) for this task, given their impressive capabilities in understanding and generating human-like text.

Existing works focus on the translation of individual NL sentences into independent First-Order Logic (FOL) formulae. In particular, there are contributions to three main dimensions: (i) the fine-tuning of an LLM to perform the translation task [1], (ii) the creation of a benchmark dataset [2, 1], and (iii) the definition of evaluation metrics to assess the quality of the generated formulae [3]. However, these works suffer from a common limitation: they consider the translation task only at the sentence level, resulting in independent formulae that do not share a common predicate and constant name space.

This is a crucial aspect for many applications, especially for those that require a posterior reasoning phase over the generated theories. For example, the project "a DJ for Machine Ethics: the Dialogue

Joint Workshop on Statistics and Knowledge Integration for Logic, Learning, Ethical Decisions, and LLMs (SKILLED-LLMs 2026), co-located with FLoC 2026, Lisbon, Portugal

matteo.magnini@uni.lu (M. Magnini); davide.liga@uni.lu (D. Liga); luca.pasetto@uni.lu (L. Pasetto)

https://matteomagnini.github.io (M. Magnini); https://www.uni.lu/fstm-en/people/davide-liga (D. Liga); https://lucapasetto.github.io (L. Pasetto)

0000-0001-9990-420X (M. Magnini); 0000-0003-1124-0299 (D. Liga); 0000-0003-1036-1718 (L. Pasetto)

Jiminy" (DJ4ME) [1] introduces an architecture for machine ethics, which uses persuasion dialogues between stakeholders' avatars to make moral decisions based on their norms and values. To support this, the project needs computational methods for *mining norms* from NL corpora and generating explanations of moral decisions in terms of these norms. Translating NL into a *uniform* logic theory is paramount for ensuring consistent reasoning and enabling transparent, explainable decision-making in such systems. If each norm is translated independently, with its own vocabulary, the reasoning engine would struggle to relate different norms and derive meaningful conclusions.

In this work, we present Natural Language to Uniform First Order Logic (NL2UNIFOL), a new framework that leverages LLMs to translate NL sentences into a *uniform* FOL theory. In the rest of this paper, the term "*uniform*" indicates that all formulae in the theory share the same vocabulary, i.e., shared predicate and constant name spaces. NL2UNIFOL is an end-to-end pipeline that proceeds in three main steps: (i) translate NL sentences into preliminary FOL formulae (*independent translation*), (ii) identify and *safely* merge predicate and constant names to obtain a *uniform* vocabulary (*symbol clustering*), (iii) finally generate the final FOL theory by replacing the predicate and constant names in the preliminary formulae with the merged ones (*reconciliation*).

We validate our framework using the MALLS dataset [2] and by exploiting different LLMs for the independent translation and symbol clustering step. Results are promising, showing that the clustering process correctly relate names that should be merged together in the majority of cases. However, we also recognise limitations that motivate future work.

2. Related Work

The translation of NL into formal logical representations has been a long-standing problem at the intersection of NLP, Knowledge Representation (KR), and automated reasoning. Early research established principled, compositional mappings from linguistic structure to logical formalisms such as higher-order logic, FOL, and discourse-level representations. Later work progressively shifted from purely symbolic pipelines to data-driven and neural models, trading theoretical guarantees for robustness and scalability. Recent approaches leverage LLMs to directly generate logical formulae from NL, substantially expanding linguistic coverage while raising new challenges related to consistency and theory-level coherence.

2.1. Classical NLP Approaches

Foundational work in formal semantics provided the theoretical basis for mapping NL into logic via compositional interpretation. Montague grammar established a precise correspondence between syntactic structure and higher-order logic, forming the conceptual backbone of many later semantic parsers [4]. Discourse Representation Theory (DRT) addressed limitations of sentence-level semantics by modeling quantification, anaphora, and discourse context, while maintaining a systematic translation into FOL [5]. Grammar-driven semantic parsing systems operationalised these theories by associating syntactic derivations with lambda-calculus expressions and logical forms. The Boxer system represents a mature instantiation of this paradigm, providing wide-coverage mappings from unrestricted text to DRT structures and FOL using symbolic rules and handcrafted lexical semantics [6]. An alternative line of work proposed controlled NLs such as attempto controlled English (ACE), which restrict linguistic expressivity to guarantee deterministic translation into FOL or description logics [7]. While classical approaches offer high precision, interpretability, and well-defined semantics, they typically operate at the sentence or discourse level and assume a fixed logical vocabulary. As a result, global unification of predicates and constants across large collections of extracted formulae is not explicitly addressed.

2.2. DNN-based Approaches

The introduction of Deep Neural Networks (DNNs) shifted semantic parsing toward data-driven models trained to map NL utterances directly to logical forms. Neural architectures enabled end-to-end learning

[1] https://icr.uni.lu/dj4me/index.html

of NL-to-FOL translation, alleviating the need for manually designed grammars and lexicons. More recent work explored the use of neural encoders to learn semantic structure implicitly through auxiliary tasks, including explicit translation into FOL. Chaturvedi and Asher showed that finetuning transformer-based models for NL-to-FOL translation improves the acquisition of predicate–argument structure compared to question-answering objectives [8]. Despite improved robustness to linguistic variation, these approaches remain supervised and operate on isolated sentence-level examples. Consequently, symbol reuse, predicate alignment, and theory-level consistency across multiple formulae are left implicit and unmanaged.

2.3. LLM-based Approaches

Large pretrained LLMs have recently demonstrated strong zero-shot and few-shot capabilities in translating NL into formal logical representations. Several works show that LLMs can generate executable FOL formulae via prompting or lightweight finetuning, often without task-specific architectures. Yang et al. demonstrate that instruction-tuned LLMs can effectively translate NL into FOL, achieving strong performance across diverse linguistic constructions [1]. Liu proposes a training-free, few-shot approach that reframes NL-to-FOL translation as a structured code generation task, improving generalization to complex formulae [9]. Vossel et al. show that fine-tuned LLMs can further improve logical accuracy and coverage in NL formalization tasks [3]. Beyond direct translation, several neurosymbolic systems integrate LLMs with external logical solvers to enhance faithfulness and reasoning reliability. Logic-LM combines language models with symbolic solvers to ensure logically consistent outputs during inference [10]. LINC adopts a neurosymbolic pipeline that couples LLMs with first-order logic provers, enabling explicit reasoning over generated formulae [11]. The FOLIO benchmark further highlights the role of FOL as an intermediate representation for evaluating logical reasoning in LLMs [2]. Within this landscape, LogicLLama represents a prominent approach explicitly targeting sentence-level NL-to-FOL translation using prompting and finetuning of LLMs. The method achieves high syntactic correctness and expressive logical forms, confirming the viability of LLMs as logic-aware language interfaces. However, LogicLLama and related approaches focus on translating individual sentences in isolation. Predicate naming, argument roles, and variable usage are resolved locally rather than at the level of an induced logical theory. This motivates approaches that augment LLM-based translation with mechanisms for global symbol unification and vocabulary induction across a domain-specific corpus.

2.4. Syntactic Normalisation of Symbols

The problem of identifying and normalising surface-level variants of symbols has been studied in classical NLP and information extraction. Early approaches rely on string-based similarity measures, such as edit distance, character n-gram overlap, and sequence alignment, to capture morphological variation and orthographic similarity between terms (e.g., singular/plural or spelling variants) [12]. Such methods have been adopted in schema and ontology matching, where lexical similarity between labels constitutes a primary signal for identifying candidate correspondences between entities and relations [13]. Lexical resources and paraphrase collections further support syntactic normalisation by providing curated equivalence classes of surface forms [14]. While these techniques are effective in detecting shallow variations, they remain insensitive to semantic relatedness and are typically applied in isolation, without considering downstream logical constraints. In the context of logic extraction, syntactic normalisation is insufficient to resolve semantically equivalent predicates whose names diverge lexically, nor does it provide safeguards against incorrect merges induced by superficial similarity alone.

2.5. Semantic Normalisation and Canonicalisation

Semantic normalisation addresses the limitation of surface-based methods by embedding symbols into continuous semantic spaces, where proximity reflects distributional or contextual similarity. Word and sentence embedding models have been widely employed to estimate semantic relatedness between terms, enabling clustering and alignment beyond lexical overlap [15]. Embedding-based canonicalisation has

been successfully applied in open knowledge base construction, where noun phrases and relation phrases are grouped into canonical forms to reduce redundancy and synonymy [16, 17]. These approaches demonstrate that semantic similarity provides a robust signal for unifying heterogeneous symbolic vocabularies extracted from text. However, existing canonicalisation methods are generally agnostic to logical structure and are not designed for predicate symbols with fixed arity or formal semantics. As a result, they may induce merges that are semantically plausible at the linguistic level but invalid when interpreted as logical predicates. The present work differs from prior semantic normalisation approaches by integrating embedding-based similarity with syntactic constraints and logical safety checks, enabling controlled unification of predicate and constant symbols within a first-order logical theory rather than an unstructured knowledge base.

3. Contribution

Inspired by the work of LogicLLama [1], we propose NL2UNIFOL, framework that leverages LLMs to translate NL sentences into a *uniform* FOL theory. We aim to overcome the limitation of existing works that focus on the translation of individual NL sentences into independent FOL formulae.

3.1. Motivation

In normative AI applications, such as legal reasoning and machine ethics, regulations and norms come from NL corpora. The source of these corpora could be official documents, such as European regulations like the General Data Protection Regulations (GDPRs) and the Markets in Financial Instruments Directive IIs (MiFID IIs), or textual descriptions coming from single individuals (or groups of individuals) about their moral values and principles. We will refer to these individuals as *stakeholders*. In the scenario of multiple stakeholders that want to provide their norms to an AI system – note that this is true also if we use official regulations – the terms used by each stakeholder to refer to the same concept may differ. There could be different names to refer to the same entity (e.g., "customer" vs "client"), or different predicates to refer to the same relation (e.g., "is employed by" vs "works for"). Alterations of a term could also happen due to typos or morphological variants such as singular/plural forms. Finally, synonyms, hypernyms, and paraphrases could be used by different stakeholders to refer to the same concept.

The misalignment of terms poses a challenge for the translation of NL norms into a formal logic theory. If each norm is translated independently, with its own vocabulary, a reasoning engine cannot relate different norms and derive meaningful conclusions. This motivates a framework that not only translates NL sentences into logic formulae, but also ensures that all formulae share the same vocabulary.

3.2. Framework Overview

Figure 1 illustrates the workflow of the NL2UNIFOL framework. Stakeholders, individuals or institutions, provide their norms in NL to the system. The norms are expected to concern the same domain, e.g., financial services or medical ethics, but may use different terminologies to refer to the same concepts as discussed in Section 3.1. An LLM is then used to independently translate each NL sentence into a preliminary FOL formula (1). The LLM is instructed to translate the sentence using a specific grammar and possibly using LLM-related techniques to improve the quality of the generated formulae, such as few-shot prompting or chain-of-thought prompting. The pseudo-algorithm of the independent translation step is reported in Section D.

Then, each preliminary formula is parsed to extract the predicate and constant names used (2). Differently from existing works on normalisation and canonicalisation of symbols, we must consider the constraints that come from the logical nature of the symbols. In particular, predicate comes with a name N and with a fixed arity A that must be taken into account during the merging process. Note that a predicate name could appear with different arities in different formulae, therefore we consider each $\langle N, A \rangle$ pair as a distinct symbol.

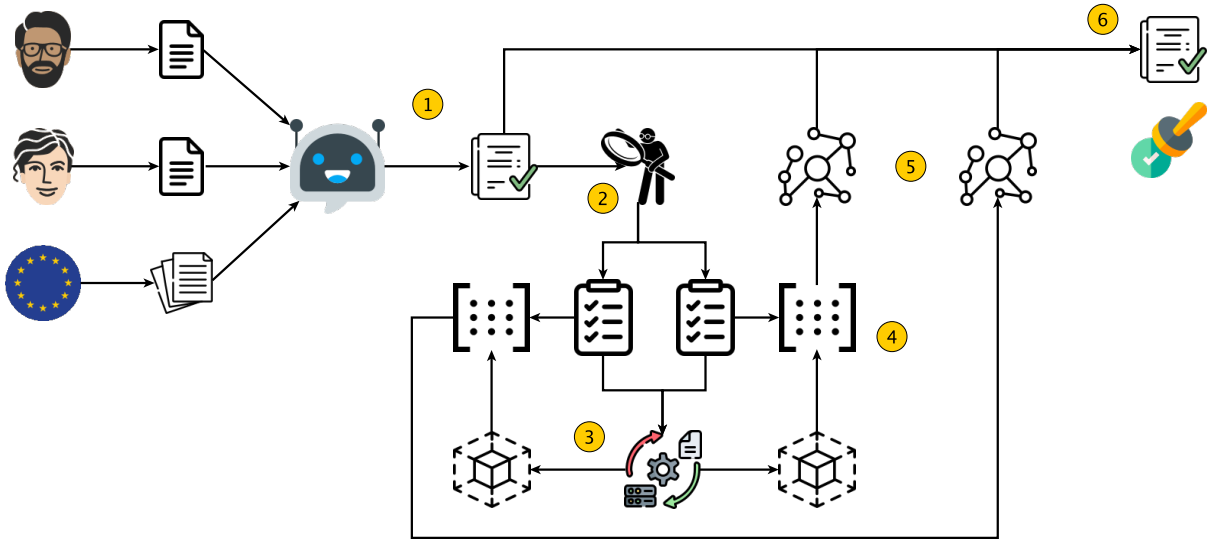

Figure 1: The workflow of the NL2UNIFOL framework. The pipeline consists of the following steps: (1) independent translation of NL sentences into preliminary FOL formulae, (2) parsing of formulae to extract predicate and constant names, (3) creation of the embedding-based representations of the extracted names, (4) computation of similarity score matrices for predicates and constants, (5) graph clustering to identify sets of names to be merged, and (6) generation of the final *uniform* FOL theory by replacing the names in the preliminary formulae with the merged ones.

The idea behind the merging process of predicate and constant names is to leverage both syntactic and semantic similarity. In order to compute the semantic similarity between names, we create their embedding-based representations using a transformer (3). We believe that the best way to represent the meaning of a predicate or constant name is to consider its context of use, i.e., all the FOL formulae in which it appears, following the distributional view that semantic similarity emerges from shared usage patterns [18, 19, 15]. In addition to the semantic similarity, we also compute the syntactic similarity between names using string-based metrics, such as Levenshtein distance [12, 20]. Then, we combine the semantic and syntactic similarity scores to obtain two final similarity score matrices: one for predicates and one for constants (4). The similarity score matrices are then used to create two graphs, one for predicates and one for constants, where nodes represent names and edges connect names with similarity score above a given threshold (5). We apply the connected components algorithm to identify clusters of names to be merged together. Finally, we generate the final *uniform* FOL theory by replacing the predicate and constant names in the preliminary formulae with the merged ones (6).

3.3. Framework Details

The first and last steps of the framework are not the main focus of this work, as for the independent translation we can directly rely on existing works such as LogicLLama [1] while the generation of the final FOL theory is a trivial replacement task. Instead, steps 2 to 5 are the core of our contribution, as they implement the symbol clustering process that allows to obtain a *uniform* vocabulary.

Symbols Collection In the second step of the framework, we parse each preliminary FOL formula to extract the predicate and constant names used. During the parsing we also keep track of the arity – for predicates only – and of the occurrences of each name in the formulae. Predicates with different arities cannot be merged together, while occurrences will be used to identify the most frequent name in each cluster during the generation of the final FOL theory. We also want to keep track of the argument positions in which each name appears, as this information will be useful to create the embedding-based representations in the next step.

Names Embeddings Embedding-based representations of predicate and constant names are constructed by explicitly modelling their context of use in the third step. For each name N, its context is defined as the collection of all preliminary FOL formulae in which N occurs. Rather than relying solely on the surface form of a name, the context captures how the symbol is employed within the logical theory. For a predicate N, the context is encoded as a textual description of its argument structure and usage patterns. Specifically, we construct a string of the form *"Predicate 'N' relates something like '$T_{1,1}$', ..., 'T_{1,n_1}' to something like '$T_{2,1}$', ..., 'T_{2,n_2}', ..., to something like '$T_{A,1}$', ..., 'T_{A,n_A}'."*, where A denotes the arity of the predicate and $T_{i,j}$ are the terms observed in the i-th argument position across the preliminary formulae. This representation summarises the typical semantic roles played by each argument position of the predicate. Analogously, the context of a constant N is encoded as a textual description of the predicates and argument positions in which it appears. In this case, we construct a string of the form *"Constant 'N' is used as argument 'i' of predicates 'P_1', ..., 'P_m', ..., as argument 'j' of predicates 'P_1', ..., 'P_n'."*, where P_k denotes predicates in which the constant occurs and $i, \ldots, j$ indicate the corresponding argument positions. These context strings are then used as inputs to a transformer-based encoder to obtain dense semantic representations of predicate and constant names.

Similarity Matrices Computation In the fourth step, for each pair of predicate names $\langle N_i, A_i \rangle$ and $\langle N_j, A_j \rangle$, we compute their semantic similarity as the cosine similarity between their embedding-based representations obtained in the previous step. This results into a matrix $M_{p,sem}$ where each entry $M_{p,sem}[i, j]$ contains the semantic similarity score between N_i and N_j. Additionally, we compute their syntactic similarity, for instance by using the normalized Levenshtein distance between the surface forms of N_i and N_j. This results into a matrix $M_{p,syn}$ where each entry $M_{p,syn}[i, j]$ contains the syntactic similarity score between N_i and N_j. Finally, we take into account the arity constraint by creating a matrix M_a where entries corresponding to pairs with different arities are set to zero. The final predicate similarity matrix M_p is then computed as a weighted combination of the three matrices:

$$M_p = ((1 - \alpha_p) \cdot M_{p,sem} + \alpha_p \cdot M_{p,syn}) \odot M_a$$

where $\alpha_p \in [0, 1]$ is a hyperparameter that controls the weight of syntactic similarity for predicates and $\odot$ denotes the element-wise multiplication. Similar computations are performed for constant names, with the only difference that there is no arity constraint to consider. This results into the final similarity matrix M_c for constants:

$$M_c = (1 - \alpha_c) \cdot M_{c,sem} + \alpha_c \cdot M_{c,syn}$$

where $\alpha_c \in [0, 1]$ is a hyperparameter that controls the weight of syntactic similarity for constants.

Graph Clustering In the fifth step, we create two graphs from M_p and M_c respectively. In each graph, nodes represent names and edges connect names with similarity score above a given threshold τ_p (resp., τ_c). We then apply the connected components algorithm to identify clusters of names to be merged together. The hyperparameters τ_p and τ_c control the granularity of the merging process: higher values result into more conservative merges, while lower values lead to more "aggressive" merges. The choice of these hyperparameters is crucial to identify the best trade-off that maximises the number of correct merges while minimising the number of incorrect merges.

4. Methodology

We designed a simple but realistic experiment to evaluate the effectiveness of NL2UNIFOL.

4.1. Dataset

We chose the MALLS[2] dataset presented in the work of Han et al. [2] for our experiments. The dataset consists of $\langle NL, FOL \rangle$ pairs, where each NL sentence is paired with its corresponding FOL formula.

[2] https://huggingface.co/datasets/yuan-yang/MALLS-v0

It comes already divided into a training set of $27,284$ pairs, and a test set of $1,000$ pairs, with a term vocabulary size of $49,394$. MALLS is particularly suited for evaluating our framework, as it contains a considerable amount of records and the NL sentences come from multiple domains, including animals, legal, medical, and many others. Moreover, it represents our scenario of multiple stakeholders because the NL sentences are generated independently, and they do not share a common predicate/constant name space.

We cannot use MALLS as it is in its entirety, because the sentences and the formulae concern many different domains. To replicate our use case, we should consider only sentences and formulae that concern the same domain. Therefore, we select a subset of the pairs in MALLS that concern the *nutrition* domain. The resulting datasets contain 748 and 27 pairs for evaluation and few-shot prompting respectively. The datasets[3] have been generated by filtering the original entries by asking GPT-5 nano[4] if the content is related to food recommendation.

4.2. Embeddings

To capture semantic regularities among symbols in FOL rules, we construct embedding spaces for both predicates and constants by leveraging NL descriptions derived from their usage patterns. Specifically, we start from a set of predicate-term co-occurrence matrices, where each matrix corresponds to a specific argument position of predicates. Rows correspond to predicates (identified by name and arity), columns correspond to terms, and binary values indicate whether a term appears in a given argument position. From these matrices, we generate canonical sentences describing predicates.

For each predicate, we iterate over its argument positions and extract the set of terms observed in that position. If terms are available, we produce a phrase of the form *"something like 'term$_1$', ... "*, otherwise we fall back to a generic placeholder ("a logic variable"). These argument-wise descriptions are then combined into a full sentence of the form: *"Predicate 'P' relates ..."*, where the arguments are connected using positional structure.

Analogously, we construct sentences for constants by analyzing in which predicates and argument positions they occur. For each constant, we collect all predicates in which it appears, grouped by argument position. This information is encoded into sentences such as *"Constant 'c' is used as argument i of predicate 'P'..."*, possibly aggregating multiple predicates.

Given the set of generated sentences, we build the embedding space by using an encoder to obtain dense vector representations. In particular, we use the `all-MiniLM-L6-v2` model, a lightweight transformer-based encoder that provides a good trade-off between efficiency and semantic representation quality. To ensure stable similarity computations, we normalise each embedding vector to unit length using ℓ_2 normalisation. This allows us to compute similarities via dot product, which corresponds to cosine similarity in the normalised space.

4.3. Hyperparameters

The choice of the hyperparameters in steps 4 and 5 of the framework is crucial to obtain a successful symbol clustering process. In particular, we have four hyperparameters to tune: α_p and τ_p for predicates, and α_c and τ_c for constants. To keep the discussion focused, we analyse the effect of the predicate-related hyperparameters, noting that the same considerations apply to constants.

Figure 2 illustrates how the number of merged predicates (blue curves) varies as a function of α_p and τ_p. The figure also reports the number of non-trivial clusters (orange curves), i.e., clusters containing more than one predicate. The plot is generated using the training portion of the MALLS dataset restricted to the nutrition domain, as described in Section 4.1.

As expected, when $\tau_p = 1$ no merges occur, independently of the value of α_p. As τ_p decreases, the number of non-trivial clusters initially increases, since new connections appear in the similarity graph and isolated predicates start forming clusters. However, after a critical threshold, the number

[3] https://github.com/DJ4ME/BlueFairy/tree/main/evaluation/data
[4] https://developers.openai.com/api/docs/models/gpt-5-nano

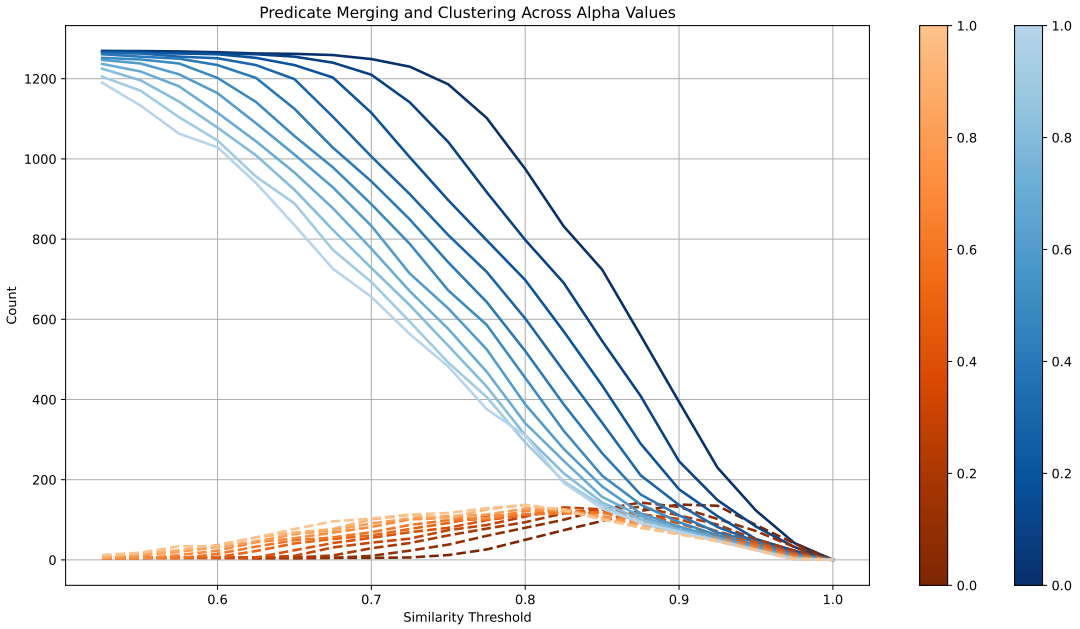

Figure 2: Predicate clustering behaviour varying α_p and τ_p hyperparameters. Blue lines represent the overall number of merged predicates. Orange lines represent the number of predicate clusters with more than one predicate (i.e., actual merges). The darker the lines, the lowest the value of α_p. Values of α_p range from 0 (purely semantic similarity) to 1 (purely syntactic similarity) with step 0.1. Values of τ_p range from 0.525 to 1 with step 0.025.

of non-trivial clusters decreases again. This happens because existing clusters start merging with each other faster than new clusters are created, leading to a collapse of the cluster structure. This rise-and-collapse behaviour is characteristic of percolation phenomena and has been extensively studied in the literature [21].

The parameter α_p controls the trade-off between semantic and syntactic similarity. Increasing α_p consistently shifts all curves towards lower values of τ_p. While the maximum number of non-trivial clusters attained by the curves is approximately the same across different values of α_p, higher values of α_p are associated with more elongated curve shapes. This indicates that giving more weight to syntactic similarity makes the merging process less sensitive to the precise choice of the threshold τ_p, whereas lower values of α_p lead to sharper transitions.

How to select suitable values for τ_p and α_p? While maximising the number of non-trivial clusters is desirable, this criterion alone is insufficient. Indeed, a large number of clusters can be obtained at the cost of an excessive number of merges, ultimately leading to overly coarse and less interpretable groupings. To explicitly capture this trade-off, we introduce a merge-aware scoring function.

Let $C(\alpha_p, \tau_p)$ denote the number of clusters with cardinality greater than one, and let $M(\alpha_p, \tau_p)$ be the total number of predicate merges. We define a normalised merge ratio:

$$R(\alpha_p, \tau_p) = \frac{M(\alpha_p, \tau_p)}{\max_{\alpha_p, \tau_p} M(\alpha_p, \tau_p)}.$$

The selection criterion is then based on the following score:

$$S(\alpha_p, \tau_p) = C(\alpha_p, \tau_p)\,(1 - R(\alpha_p, \tau_p)).$$

This formulation favours configurations that simultaneously yield a high number of non-trivial clusters while penalising excessive merging.

For a fixed value of α_p, we analyse the behaviour of $S(\alpha_p, \tau_p)$ as a function of τ_p and identify its maximum:

$$\tau_p^\star(\alpha_p) = \arg\max_{\tau_p} S(\alpha_p, \tau_p).$$

As in the previous formulation, we are interested in stable optima rather than sharp peaks. Therefore, we analyse the local discrete variations of S around $\tau_p^\star$ using finite differences:

$$\Delta^- S(\alpha_p, \tau_p) = S(\alpha_p, \tau_p) - S(\alpha_p, \tau_p^-),$$
$$\Delta^+ S(\alpha_p, \tau_p) = S(\alpha_p, \tau_p^+) - S(\alpha_p, \tau_p).$$

where τ_p^- and τ_p^+ denote the neighbouring threshold values. The admissible fluctuation scale is estimated via the InterQuartile Range (IQR) of all absolute variations of S across τ_p. A maximum is considered *stable* if both $|\Delta^- S(\alpha_p, \tau_p^\star)|$ and $|\Delta^+ S(\alpha_p, \tau_p^\star)|$ lie within this range.

Let $\mathrm{A}_s = \{\alpha_p \mid \tau_p^\star(\alpha_p) \text{ is stable}\}$ be the set of values yielding stable maxima. The optimal hyperparameters are then selected as:

$$(\alpha_p^\star, \tau_p^\star) = \arg\max_{\alpha_p \in \mathrm{A}_s} S(\alpha_p, \tau_p^\star(\alpha_p)).$$

In practice, when multiple threshold values achieve the same maximum score for a given α_p, we select $\tau_p^\star(\alpha_p)$ as the median of the corresponding values, further increasing robustness.

This merge-aware criterion naturally avoids degenerate regimes. Configurations with very low thresholds, which induce large connected components through excessive merging, are penalised via the factor $(1 - R)$. Conversely, configurations with very high thresholds, which produce few or no merges, result in low values of C and are therefore also disfavoured. As a result, the selected hyperparameters correspond to a balanced regime where meaningful clusters emerge without collapsing into overly large structures. Importantly, this formulation implicitly regularises the trade-off between semantic and syntactic similarity controlled by α_p, without requiring additional heuristics.

4.4. Validation

To validate NL2UNIFOL, we first perform single NL sentence translations into FOL formulae using a set of LLMs. The code is public available on GitHub[5]. The models that we consider are: Qwen 2.5 Instruct (7B), LLama 3 Instruct (8B), Mistral 7B Instruct (7B), and Phi 3.1 Instruct (7B). We choose instruct models because they are fine-tuned to follow instructions, which is crucial for our task. For the scope of this paper, we are not interested in achieving state-of-the-art performance in NL-to-FOL translation, but in the symbol clustering process that allows to obtain a *uniform* vocabulary. For this reason, we consider relatively small models rather than larger ones, which would be more expensive to run. Nonetheless, we still expect good performance.

We did prompt engineering and performed both no-shot and few-shots strategies. We used the same prompt for all the models, and we used the same few-shot examples for all the models (the datasets with 27 examples described in Section 4.1). The full prompt reported in Section B of the Appendix, and it includes instructions, rules, and possibly examples.

Then, we select the results from the best-performing model – according to syntactic correctness (see Section A of Appendix) – and we apply the symbol clustering process described in Section 3.3. Finally, we evaluate the quality of the resulting clusters by eliciting the type of errors that occur in the merging process by manually inspecting the merged predicates and constants.

5. Results

Figure 3 reports the syntactic correctness of the generated FOL formulae across different LLMs and prompting strategies. We can observe that all the models achieve better performance with few-shot prompting, with Qwen 2.5 Instruct being the best-performing model in both settings.

We then apply the symbol clustering process to the formulae generated by Qwen 2.5 Instruct with few-shot prompting, using the procedure described in Section 4.3 to select the optimal hyperparameters

[5] https://github.com/DJ4ME/BlueFairy/

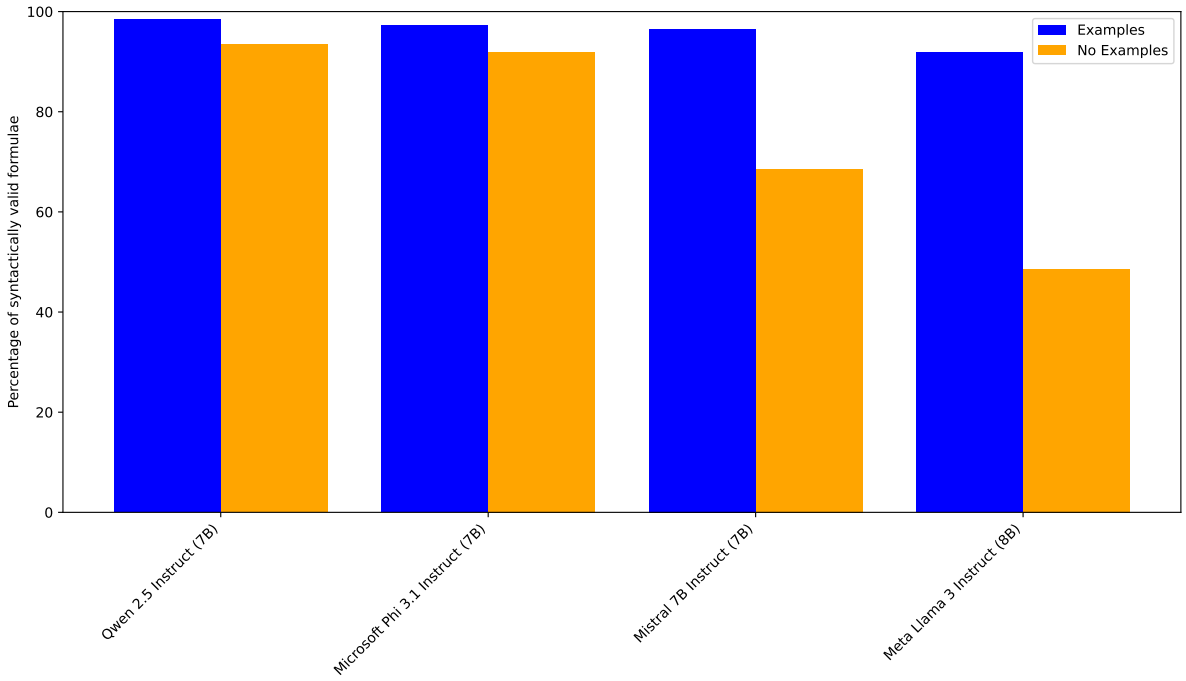

Figure 3: Syntactic correctness of the generated FOL formulae across different LLM families and prompting strategies.

for merging. We generated all the possible clustering for predicates using the following grid of hyperparameters: $\alpha_p \in \{0.0, 0.1, \ldots, 1.0\}$ (steps of 0.1) and $\tau_p \in \{0.525, 0.55, \ldots, 1.0\}$ (steps of 0.025). In total, we generated $11 \times 20 = 220$ different clusterings for predicates. The best hyperparameters selected by the procedure are $\alpha_p^\star = 0.1$ and $\tau_p^\star = 0.9$, which yield a total of 307 merged predicates grouped into 140 non-trivial clusters. Figure 4 illustrates the resulting predicate clusters. In total, from 1170 unique predicates extracted form the generated formulae, we reduce them to 863 unique predicates after merging, which corresponds to a reduction of $(1170 - 863)/1170 \approx 26.2\%$.

5.1. Merging Errors

To evaluate the quality of the obtained clusters, we manually inspect the merged predicates, and we categorise the errors that occur in the merging process. We introduce a relation-level taxonomy aimed at characterising the quality of individual predicate-to-representative mappings. Rather than assessing whether a cluster is globally coherent, this perspective focuses on the semantic fidelity of each unification decision, i.e., whether a given predicate is appropriately subsumed by its assigned representative.

Valid Mapping (OK) captures cases in which the representative is a faithful semantic generalisation of the predicate. The mapping preserves the essential meaning of the predicate while possibly abstracting away surface-level lexical or morphological variation. Such relations constitute the desired behaviour of the system.

Semantic Mismatch (Esem) captures cases in which the representative is not semantically compatible with the predicate. The mapping introduces a conceptual shift that cannot be justified as a generalisation, resulting in a violation of semantic consistency.

Over-Generalisation (Egen) captures cases in which the representative is semantically valid but excessively abstract with respect to the predicate. While still broadly related, it fails to preserve informative distinctions, leading to a loss of specificity in the mapping.

Over-Specification (Espe) captures cases in which the representative is too specific with respect to the predicate. In these cases, the mapping introduces unnecessary constraints or details that are not supported by the predicate semantics, resulting in an overly narrow representation.

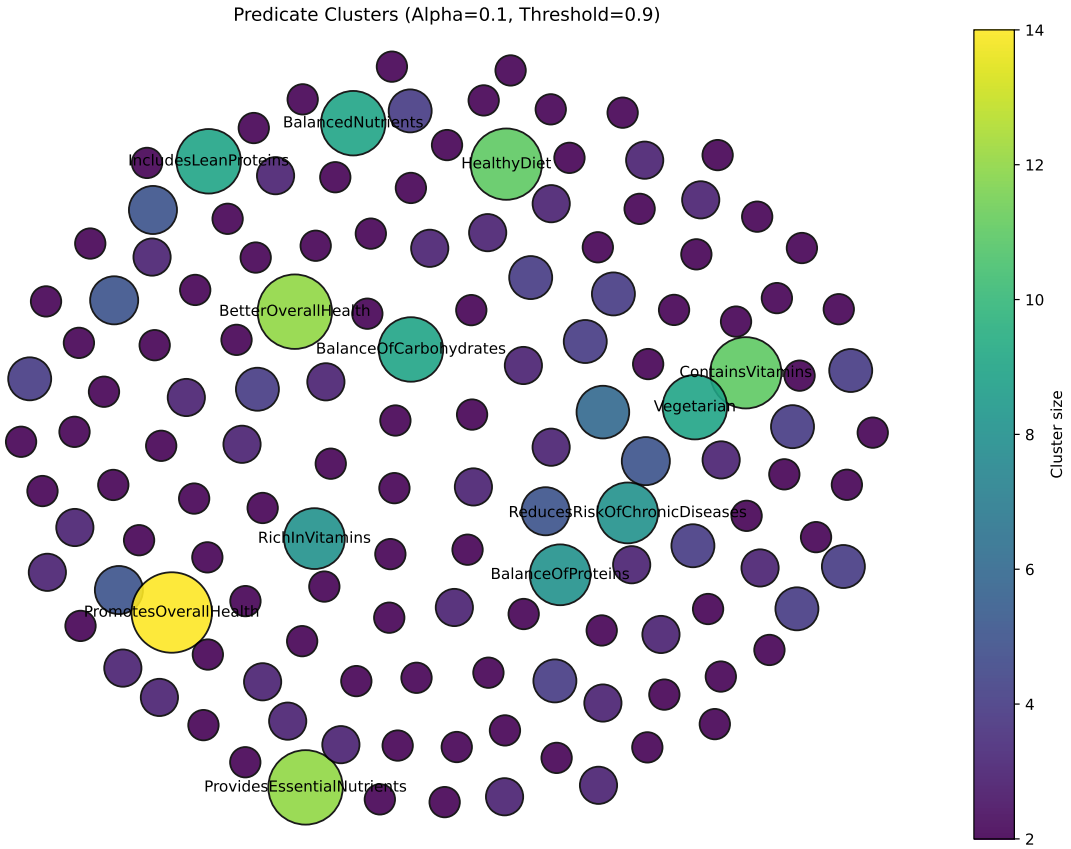

Figure 4: Non-trivial predicate clusters obtained for $\alpha_p = 0.1$ and $\tau_p = 0.9$. Bubbles represent the clusters, with size proportional to the number of predicates in the cluster. The representative names of big clusters are reported as labels inside the bubbles.

Compositional Reduction (Ecomp) captures cases in which the representative preserves only a subset of the predicate's semantic components. Multi-attribute or compositional predicates are partially mapped, resulting in incomplete semantic coverage.

Polarity Inversion (Epol) captures cases in which the representative expresses a meaning that is semantically opposite or contradictory to that of the predicate. This includes inversions in evaluative, causal, or directional properties, thereby breaking the intended semantic alignment.

Ontological Shift (Ecat) captures cases in which the predicate and representative belong to different ontological types, such as processes being mapped to states or entities to attributes. Even when a loose semantic relation exists, the structural mismatch invalidates the mapping.

Unsupported Generalisation (Ehall) captures cases in which the representative cannot be justified as a semantic generalisation of the predicate, often reflecting spurious or externally introduced abstractions. Such mappings indicate a failure of grounding in the underlying semantic space.

The proposed relation-level taxonomy provides a fine-grained diagnostic tool for analysing unification behaviour, enabling the identification of systematic biases that may not be visible at the cluster level. Examples of each error category are reported in Section C.

5.2. Analysis

We apply the proposed taxonomy to categorise the errors observed in the predicate merging process. The three authors of this paper independently annotated the merged predicates according to the defined categories, and we then applied majority voting to obtain a final categorisation for each predicate. In case of ties, we applied a severity-based tie-breaking strategy, where the most severe error category among the tied ones is selected as the final category. The severity-based value ordering is defined as

Category	Evaluator 1	Evaluator 2	Evaluator 3	Majority	ChatGPT
OK	144	141	133	141	177
Egen	46	48	55	46	33
Espe	20	24	21	22	5
Ecomp	20	21	24	21	29
Ecat	38	31	22	32	25
Esem	33	37	18	39	31
Epol	6	5	3	6	5
Ehall	0	0	31	0	2

Table 1
Distribution of error categories across annotators, majority voting (with severity-based tie-breaking), and GPT predictions.

follows: *Epol* > *Esem* > *Ecat* > *Ecomp* > *Ehall* > *Espe* > *Egen* > *OK*.

Table 1 reports the evaluation results of the error analysis. In addition to the individual and aggregate human annotations, the table also includes an automatic annotation obtained by ChatGPT, which was prompted to classify the merged predicates according to the same taxonomy. Results show that, according to all evaluators, in almost half of the cases the merging process produces valid mappings – 45.9% according to Majority and 57.7% according to ChatGPT –, while in the remaining cases it introduces various types of errors, with over-generalisation being the most common one, followed by semantic mismatch.

Reasoning about the severity of the observed errors, we can see that a significant portion of them are relatively mild. *Egen* is not necessarily a severe error – or an error at all in some contexts – it depends on the specific use case and the desired level of abstraction (similarly for *Espe*). *Ecomp* if correctly identified, can be easily mitigated by splitting the representative predicate into multiple predicates that capture the different semantic components. Also, *Ehall* is not necessarily a severe error, as it can be caused by the presence of spurious or externally introduced abstractions that are not grounded in the underlying semantic space, but it does not necessarily imply a violation of semantic consistency. Overall, these less severe errors account for ∼29% and 22.5% of the cases according toMajority and ChatGPT annotations respectively.

The other error categories instead are more severe. In particular, the *Epol* category is the most severe one, but it is also one of the least common having only a 5 and 6 cases according to Majority and ChatGPT annotations respectively over the 307 relations ($1 - 2\%$). Considering also the *Esem* and *Ecat* categories, which are the second and third most severe ones, all of them account for 25.1% and 19.9% of the cases according to Majority and ChatGPT annotations respectively.

6. Conclusion

In this paper, we introduced NL2UNIFOL, a framework for translating NL normative statements into a uniform FOL representation. The core of the framework is a symbol clustering process that merges predicate and constant names based on a combination of semantic and syntactic similarity. We validated the framework on a subset of the MALLS dataset. Results show that the symbol clustering process is effective in merging a significant portion of predicates. However, we also acknowledge that the merging process is far from perfection, and it introduces various types of errors, with over-generalisation and semantic mismatch being the most common ones.

This work sets the stage for future research in the area of NL translation into logic formulae. Future work will explore the integration of LLMs in support of the clustering process. Also, experiments on different datasets and domains will be needed. Finally, we aim to explore other kind of target logics, such as modal logics, which are more suitable to represent normative statements.

Acknowledgments

Matteo Magnini was supported by the Luxembourg National Research Fund (FNR) through the project "A DJ for Machine Ethics: the Dialogue Jiminy" (O24/18989918/DJ4ME).
Luca Pasetto was supported by FNR through the project "Logical methods for Deontic Explanations" (INTER/DFG/23/17415164/LODEX).

Declaration on Generative AI

During the preparation of this work, the authors used Copilot to speed up the LaTeX writing process.

References

[1] Y. Yang, S. Xiong, A. Payani, E. Shareghi, F. Fekri, Harnessing the power of large language models for natural language to first-order logic translation, in: L. Ku, A. Martins, V. Srikumar (Eds.), Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024, Association for Computational Linguistics, 2024, pp. 6942–6959. doi:10.18653/V1/2024.ACL-LONG.375.

[2] S. Han, H. Schoelkopf, Y. Zhao, Z. Qi, M. Riddell, W. Zhou, J. Coady, D. Peng, Y. Qiao, L. Benson, L. Sun, A. Wardle-Solano, H. Szabó, E. Zubova, M. Burtell, J. Fan, Y. Liu, B. Wong, M. Sailor, A. Ni, L. Nan, J. Kasai, T. Yu, R. Zhang, A. R. Fabbri, W. Kryscinski, S. Yavuz, Y. Liu, X. V. Lin, S. Joty, Y. Zhou, C. Xiong, R. Ying, A. Cohan, D. Radev, FOLIO: natural language reasoning with first-order logic, in: Y. Al-Onaizan, M. Bansal, Y. Chen (Eds.), Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024, Association for Computational Linguistics, 2024, pp. 22017–22031. doi:10.18653/V1/2024.EMNLP-MAIN.1229.

[3] F. Vossel, T. Mossakowski, B. Gehrke, Advancing natural language formalization to first order logic with fine-tuned llms, CoRR abs/2509.22338 (2025). doi:10.48550/ARXIV.2509.22338. arXiv:2509.22338.

[4] B. H. Partee, H. L. W. Hendriks, Montague grammar, in: J. van Benthem, A. ter Meulen (Eds.), Handbook of Logic and Language, North Holland / Elsevier, 1997, pp. 5–91. doi:10.1016/B978-044481714-3/50004-7.

[5] H. Kamp, U. Reyle, From Discourse to Logic - Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory, volume 42 of *Studies in linguistics and philosophy*, Springer, 1993. doi:10.1007/978-94-017-1616-1.

[6] J. Bos, Wide-coverage semantic analysis with boxer, in: J. Bos, R. Delmonte (Eds.), Semantics in Text Processing. STEP 2008 Conference Proceedings, Venice, Italy, September 22-24, 2008, Association for Computational Linguistics, 2008. URL: https://aclanthology.org/W08-2222/.

[7] N. E. Fuchs, K. Kaljurand, T. Kuhn, Attempto controlled english for knowledge representation, in: C. Baroglio, P. A. Bonatti, J. Maluszynski, M. Marchiori, A. Polleres, S. Schaffert (Eds.), Reasoning Web, 4th International Summer School 2008, Venice, Italy, September 7-11, 2008, Tutorial Lectures, volume 5224 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 104–124. doi:10.1007/978-3-540-85658-0_3.

[8] A. Chaturvedi, N. Asher, Learning semantic structure through first-order-logic translation, in: Y. Al-Onaizan, M. Bansal, Y.-N. Chen (Eds.), Findings of the Association for Computational Linguistics: EMNLP 2024, Association for Computational Linguistics, Miami, Florida, USA, 2024, pp. 6669–6680. doi:10.18653/v1/2024.findings-emnlp.390.

[9] J. Liu, Few-shot natural language to first-order logic translation via code generation, in: L. Chiruzzo, A. Ritter, L. Wang (Eds.), Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume

1: Long Papers), Association for Computational Linguistics, Albuquerque, New Mexico, 2025, pp. 10939–10960. doi:10.18653/v1/2025.naacl-long.547.

[10] L. Pan, A. Albalak, X. Wang, W. Y. Wang, Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning, in: H. Bouamor, J. Pino, K. Bali (Eds.), Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023, Association for Computational Linguistics, 2023, pp. 3806–3824. doi:10.18653/V1/2023.FINDINGS-EMNLP.248.

[11] T. Olausson, A. Gu, B. Lipkin, C. E. Zhang, A. Solar-Lezama, J. B. Tenenbaum, R. Levy, LINC: A neurosymbolic approach for logical reasoning by combining language models with first-order logic provers, in: H. Bouamor, J. Pino, K. Bali (Eds.), Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023, Association for Computational Linguistics, 2023, pp. 5153–5176. doi:10.18653/V1/2023.EMNLP-MAIN.313.

[12] V. I. Levenshtein, Binary codes capable of correcting deletions, insertions, and reversals, Soviet physics. Doklady 10 (1965) 707–710. URL: https://api.semanticscholar.org/CorpusID:60827152.

[13] J. Euzenat, P. Shvaiko, Ontology matching, Springer, 2007. doi:10.1007/978-3-540-49612-0.

[14] J. Ganitkevitch, B. V. Durme, C. Callison-Burch, PPDB: the paraphrase database, in: L. Vanderwende, H. D. III, K. Kirchhoff (Eds.), Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 9-14, 2013, Westin Peachtree Plaza Hotel, Atlanta, Georgia, USA, The Association for Computational Linguistics, 2013, pp. 758–764. URL: https://aclanthology.org/N13-1092/.

[15] N. Reimers, I. Gurevych, Sentence-bert: Sentence embeddings using siamese bert-networks, in: K. Inui, J. Jiang, V. Ng, X. Wan (Eds.), Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019, Association for Computational Linguistics, 2019, pp. 3980–3990. doi:10.18653/V1/D19-1410.

[16] S. Vashishth, P. Jain, P. P. Talukdar, CESI: canonicalizing open knowledge bases using embeddings and side information, in: P. Champin, F. Gandon, M. Lalmas, P. G. Ipeirotis (Eds.), Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018, ACM, 2018, pp. 1317–1327. doi:10.1145/3178876.3186030.

[17] C. Jiang, Y. Jiang, W. Wu, Y. Zheng, P. Xie, K. Tu, COMBO: A complete benchmark for open KG canonicalization, in: A. Vlachos, I. Augenstein (Eds.), Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2023, Dubrovnik, Croatia, May 2-6, 2023, Association for Computational Linguistics, 2023, pp. 340–357. doi:10.18653/V1/2023.EACL-MAIN.26.

[18] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, J. Dean, Distributed representations of words and phrases and their compositionality, in: C. J. C. Burges, L. Bottou, Z. Ghahramani, K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States, 2013, pp. 3111–3119. URL: https://proceedings.neurips.cc/paper/2013/hash/9aa42b31882ec039965f3c4923ce901b-Abstract.html.

[19] M. Nickel, K. Murphy, V. Tresp, E. Gabrilovich, A review of relational machine learning for knowledge graphs, Proc. IEEE 104 (2016) 11–33. doi:10.1109/JPROC.2015.2483592.

[20] W. W. Cohen, P. Ravikumar, S. E. Fienberg, A comparison of string distance metrics for name-matching tasks, in: S. Kambhampati, C. A. Knoblock (Eds.), Proceedings of IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03), August 9-10, 2003, Acapulco, Mexico, 2003, pp. 73–78. URL: http://www.isi.edu/info-agents/workshops/ijcai03/papers/Cohen-p.pdf.

[21] D. Stauffer, A. Aharony, Introduction to Percolation Theory, Taylor & Francis, 2018. doi:10.1201/9781315274386.

A. FOL Grammar

To ensure the syntactic correctness of the generated FOL rules, we adopt a Context-Free Grammar (CFG) closely aligned with prior work on NL translation into FOL. Differently from previous approaches based on NLTK CFG parsers, we implement the grammar using the `Lark` parsing library, which enables a more compact and declarative specification. This choice reduces implementation complexity and improves maintainability, while providing access to efficient parsing algorithms. The grammar used in our experiments is reported below in a normalised form for readability:

$$
\begin{aligned}
\textit{sentence} &::= \textit{formula} \mid \textit{sentence bin\_op sentence} \mid (\ \textit{sentence}\ ) \\
\textit{formula} &::= \textit{literal} \mid \neg\ \textit{formula} \mid \textit{quantifier formula} \\
&\quad \mid (\ \textit{formula}\ ) \mid \textit{formula bin\_op formula} \\
\textit{quantifier} &::= \forall\ \textit{var} \mid \exists\ \textit{var} \\
\textit{literal} &::= \textit{pred}(\textit{terms}) \mid \neg\ \textit{pred}(\textit{terms}) \\
\textit{terms} &::= \textit{term} \mid \textit{term},\ \textit{terms} \\
\textit{term} &::= \textit{var} \mid \textit{const} \\
\textit{bin\_op} &::= \oplus \mid \vee \mid \wedge \mid \rightarrow \mid \leftrightarrow
\end{aligned}
$$

In this grammar, predicates (*pred*) are uppercase identifiers, variables (*var*) are lowercase symbols, and constants (*const*) include alphanumeric strings and comparison operators. Given a candidate formula generated by the LLM, we attempt to parse it using the Lark parser.

B. Prompt for NL-to-FOL Translation

We report the content of the system prompt used for NL-to-FOL translation in our experiments. The placeholder `{FEW_SHOT_EXAMPLES}` is replaced with the few-shot examples from the dataset in the few-shot prompting setting, while it is left empty in the no-shot prompting setting. After the end of the prompt, the NL sentence to be translated is appended as input to the LLM.

Content of the system prompt used for NL-to-FOL translation

```
You are a formal logic translation engine.
Your task is to translate a single natural-language normative statement into a
First-Order Logic (FOL) formula.

You MUST follow these rules strictly:

1. OUTPUT FORMAT
- Output ONLY a single FOL formula.
- Do NOT add explanations, comments, or extra text.
- Do NOT repeat the input sentence.
- Do NOT introduce natural language in the output.

2. LOGIC FRAGMENT
- Use classical First-Order Logic.
- Allowed operators: ∀, ∃, ∧, ∨, ⊕, ¬, →, ↔.
- Use parentheses to avoid ambiguity.
- Use only predicates (no functions, no constants unless explicitly required).
- Use only variables x, y, z, w (reuse them if needed).

3. PREDICATES
- Predicates must be in English.
- Predicate names must be in CamelCase.
- Predicates represent properties or relations explicitly stated or clearly
implied.
- Do NOT invent new predicates unless logically required by the sentence.
```

```
- Do NOT encode modality (no “ought”, “should”, “allowed”) – only factual
structure.

4. QUANTIFIERS
- Use ∀ for general rules, obligations, or universal claims.
- Use ∃ only when the sentence explicitly asserts existence.
- Do NOT introduce existential quantifiers unless strictly required.

5. IMPLICATIONS
- Normative rules MUST be represented as implications (→).
- The antecedent contains conditions.
- The consequent contains outcomes or conclusions.
- Avoid vacuous implications whenever possible.

6. CONSISTENCY
- Assume an abstract, non-empty domain of entities.
- Do NOT assume temporal, probabilistic, or causal semantics.
- Do NOT use deontic operators.

7. STYLE
- Prefer simpler formulas over complex ones.
- Do not nest implications unless necessary.
- Do not flatten logical structure incorrectly.

–

EXAMPLES
{FEW_SHOT_EXAMPLES}

–

Now translate the following normative statement into FOL:
```

C. Examples for Error Categories

To support a clearer understanding of the annotation scheme, we report below one representative example for each category introduced in Section 5.1. The examples are selected from the evaluation set and are intended to illustrate the semantic motivation behind each label.

Valid Mapping (OK). A representative example is: `BalancedMixOfProteins` → `BalanceOfProteins`. In this case, the mapping preserves the essential meaning of the predicate while abstracting away purely lexical differences, resulting in a correct semantic generalisation.

Over-Generalisation (E_{gen}). A representative example is: `ContainsVitaminC` → `ContainsVitamins`. Here, the representative is semantically compatible with the original predicate but introduces an excessive level of abstraction, removing the distinction between specific and general vitamin content.

Over-Specification (E_{spe}). A representative example is: `ContainsProtein` → `IncludesLeanProteins`. In this case, the representative introduces additional constraints not implied by the original predicate, resulting in an overly specific interpretation.

Semantic Mismatch (E_{sem}). A representative example is: `BalanceOfFruits` → `BalanceOfFats`. Despite superficial lexical similarity, the two predicates refer to different nutritional concepts, making the mapping semantically inconsistent.

Compositional Reduction ($\mathbf{E}_{\text{comp}}$). A representative example is: `ProvidesEssentialVitaminsAndMinerals` → `ContainsVitamins`. The representative preserves only part of the original compositional meaning, omitting the reference to minerals and therefore reducing semantic coverage.

Polarity Inversion ($\mathbf{E}_{\text{pol}}$). A representative example is: `HealthyFats` → `Unhealthy`. In this case, the representative reverses the evaluative meaning of the predicate, leading to a polarity contradiction.

Ontological Shift ($\mathbf{E}_{\text{cat}}$). A representative example is: `Healthy` → `HealthyDiet`. The mapping changes the ontological type from a general attribute to a specific entity class, introducing a structural mismatch beyond lexical variation.

Unsupported Generalisation ($\mathbf{E}_{\text{hall}}$). A representative example is: `MadeFromLeaves` → `MadeFromButter`. Here, the representative cannot be justified as a semantic generalisation of the original predicate and instead introduces an unrelated concept, reflecting a spurious abstraction.

The reported examples are meant to be illustrative rather than exhaustive. They highlight recurring patterns observed during the annotation process and clarify the semantic boundaries between categories.

D. NL2FOL Pseudoalgorithm

Algorithm 1 reports the pseudoalgorithm of the NL2UNIFOL framework described in Section 3.2.

Algorithm 1: The NL2UNIFOL pipeline. Step 1 (independent translation) can rely on an existing NL-to-FOL model and step 6 (reconciliation) is a direct name replacement; steps 2–5 realise the symbol clustering. A *symbol* σ is either a predicate $\langle N, A\rangle$ or a constant. Here $\mathrm{ctx}(\sigma, \Phi_0)$ is the context string of σ (Sec. 3.3); lev is the normalised Levenshtein distance; $\mathrm{occ}(N)$ is the number of occurrences of N in Φ_0; $r(\kappa)$ is the representative of cluster κ; $\kappa(\sigma)$ is the cluster containing σ; and $\mathbb{1}[\cdot]$ is the indicator function.

Input : NL sentences $S = \{s_1, \ldots, s_n\}$; translation LLM $\mathcal{T}$; encoder $\mathcal{E}$; syntactic weights $\alpha_p, \alpha_c \in [0, 1]$; thresholds $\tau_p, \tau_c \in [0, 1]$

Output : Uniform FOL theory Φ

$\Phi_0 \leftarrow \{\, \mathcal{T}(s) \mid s \in S \,\}$

$(\mathcal{P}, \mathcal{C}) \leftarrow \mathrm{parse}(\Phi_0)$ `// predicates` $\langle N, A\rangle$ `and constants, with arities and occurrences`

foreach $\sigma \in \mathcal{P} \cup \mathcal{C}$ **do**

- $\mathbf{v}_\sigma \leftarrow \mathcal{E}\big(\mathrm{ctx}(\sigma, \Phi_0)\big)$
- $\mathbf{v}_\sigma \leftarrow \mathbf{v}_\sigma / \|\mathbf{v}_\sigma\|_2$

foreach $\langle N_i, A_i\rangle, \langle N_j, A_j\rangle \in \mathcal{P}$ **do**

- $M_{p,sem}[i, j] \leftarrow \cos(\mathbf{v}_i, \mathbf{v}_j)$
- $M_{p,syn}[i, j] \leftarrow 1 - \mathrm{lev}(N_i, N_j)$
- $M_a[i, j] \leftarrow \mathbb{1}\,[A_i = A_j]$

$M_p \leftarrow \big((1 - \alpha_p)\, M_{p,sem} + \alpha_p\, M_{p,syn}\big) \odot M_a$

$M_c \leftarrow (1 - \alpha_c)\, M_{c,sem} + \alpha_c\, M_{c,syn}$ `// constants, without the arity mask`

foreach $(M, \tau) \in \{(M_p, \tau_p), (M_c, \tau_c)\}$ **do**

- $G \leftarrow$ graph with node set the corresponding symbols and edge (i, j) iff $M[i, j] > \tau$
- $\mathcal{K} \leftarrow \mathrm{connectedComponents}(G)$
- **foreach** $\kappa \in \mathcal{K}$ **do** $r(\kappa) \leftarrow \arg\max_{N \in \kappa} \mathrm{occ}(N)$

$\Phi \leftarrow$ rewrite each formula of Φ_0, replacing every symbol σ with $r(\kappa(\sigma))$

return Φ