Using the Meaning of Symbol Names to Guide First-Order Logic Reasoning

Claudia Schon

Hochschule Trier, Schneidershof, 54293 Trier, Germany

Abstract

When humans evaluate the validity of a logical conclusion, they typically consider the meaning of the conclusion, taking into account the context, leading to a naturally targeted approach that avoids irrelevant inferences. For example, if humans are asked to show that all *hammocks* are also *beds*, they will certainly not draw conclusions about *vehicles* or *weapons*. However, automated theorem provers do not usually take the context of the conclusion into account when selecting inference steps. Existing heuristics for selecting the clause for the next inference step usually do not take into account the meaning of symbol names, overlooking this valuable source of information about the context of a clause. Therefore, in the example above, clauses with symbol names such as *weapon* or *vehicle* could well be found in the processed clauses. However, since these clauses are not required for the actual proof, they are not helpful to the prover and tend to distract from the actual proof task. In this paper, we present an approach that uses natural language processing techniques to align the selection of the clause for the next inference step with the meaning of the current proof goal. By mapping symbol names to natural language words and representing them as vectors, we compute similarities between symbols and the proof goal, which inform the selection of the clause for the next inference. The approach has been implemented and evaluated. The experimental results show a significant reduction in the number of clauses processed during proof construction using the proposed method.

Keywords

Commonsense Reasoning, Automated Reasoning, heuristics for Automated Reasoning

1. Introduction

When humans evaluate the validity of a logical conclusion, they naturally consider the context and meaning of the conclusion and avoid irrelevant inferences. This intuitive approach is in line with Daniel Kahneman's model of human cognition based on two systems [1], where system 1 represents fast, instinctive and contextual thinking, while system 2 involves slower, more deliberate and logical reasoning. In the field of automated theorem proving (ATP), current systems [2, 3] predominantly use logic-based approaches similar to system 2, often not using the contextual insights corresponding to system 1.

For example, a human being asked to prove that all *hammocks* are also *beds* will intuitively avoid considering unrelated concepts such as *vehicles* or *weapons*. Conversely, automated theorem provers typically do not consider the contextual relevance of symbol names when selecting inference steps. Existing heuristics focus on syntactic properties without exploiting the semantic content embedded in symbol names, which can lead to the inclusion of irrelevant clauses and an inefficient proof process.

It is a key strength of human reasoning that we are able to combine both intuitive (system 1) and analytical (system 2) processes. It is therefore plausible that automated theorem proving could similarly benefit from integrating these two approaches, using the contextual understanding of system 1 alongside the logical strictness of system 2. Currently, however, the meaning of symbol names, and thus the contextual content of formulae, is not taken into account during proof construction in automated reasoning.

To address this gap, we propose an approach that integrates statistical methods from natural language processing into the reasoning process to improve the selection of the clause for the next inference step of a theorem prover. We do this by mapping symbol names to natural language words and representing them as vectors. This allows us to measure the semantic similarity between symbols and the goal of the proof. We use these similarities to derive symbol weights, which guide the selection of the clause for the next inference step in a more goal-directed way.

Our approach has been implemented and evaluated using the E theorem prover [2]. Experimental results show a significant reduction in the number of clauses processed during proof construction, indicating a more focused and efficient proof process. This work highlights the potential of combining logic-based and statistical methods to improve automated reasoning, in line with Kahneman's theory.

The main contributions of this paper are:

- The introduction of a heuristic for determining symbol weights based on similarity to the proof goal. These weights can be used to guide the selection of the clause for the next inference step in automated theorem proving.
- An evaluation of the approach, demonstrating a significant reduction in the number of clauses processed during proof construction for commonsense reasoning tasks.

The paper is structured as follows: after discussing related work (Sect. 2) and preliminaries (Sect. 3), we describe how to determine symbol weights based on the similarity of the symbol name to the proof goal in Sect. 4. In Sect. 5 we present experimental results of our approach. Finally, we summarize and discuss ideas for future work in Sect. 6.

2. Related Work

The selection of the clause for the next inference step has a very large influence on the performance of theorem provers. A perfect selection can make a search unnecessary, and a proof can be found directly. Most heuristics for selecting the clause for the next inference step are based on simple counting of symbols and the like [4]. In this way, weights are specified for symbols and then used to determine weights for clauses, which are used to select the clause for the next inference step.

10th Workshop on Formal and Cognitive Reasoning (FCR-2024) at the 47th German Conference on Artificial Intelligence (KI-2024, September 23 - 27), Würzburg, Germany

c.schon@hochschule-trier.de (C. Schon)

To make it possible to adjust symbol weights, the theorem prover E [2] offers the option of assigning a weight to each symbol. This can be done by specifying a weight function with `FunWeight`[1]. Although this option allows you to set symbol weights, it does not describe how to determine them.

In [5], the selection of symbol weights is investigated. The proposed approach employs a graph neural network to predict symbol weights based on the structure of the clause normal form of the input problem. The system is trained using prior proof successes, and a novel scheme is introduced to balance positive clauses (derived from a proof) and negative clauses (not derived from a proof) during training.

Another approach is to use machine learning to derive new clause selection strategies from successful proofs [6].

The ENIGMA [7, 8, 9, 10] method represents a key advancement in internal guidance within ATPs by integrating machine learning models directly into the theorem proving process to influence decisions such as clause selection. Utilizing techniques like gradient-boosted trees and recursive neural networks, ENIGMA has significantly enhanced the performance of saturation-based provers like E, particularly in large-theory settings like the Mizar Mathematical Library [11].

To the best of our knowledge, none of the currently used approaches to determining symbol weights take into account the meaning of symbol names.

The problem of premise selection, where relevant formulae for a given proof task have to be selected from a knowledge base, is related to the problem considered in this paper. Especially in the area of large mathematical libraries, there are many approaches to premise selection that rely on machine learning methods [12]. The $\text{ATP}_{\text{BOOST}}$ method improves premise selection by framing it as a binary classification task and using the XGBoost algorithm. DeepMath [13] uses neural networks for premise selection. It is worth noting that DeepMath does not use manually created features. Another notable technique uses recurrent neural networks to capture dependencies between premises [14]. In addition, the work in [15] applies the transformer architecture to premise selection, achieving superior performance compared to earlier recurrent neural network-based methods. Similarly, the transformer-based Magnushammer [16] shows significant improvements in detection success rates through effective retrieval and ranking of relevant premises.

The Semantic Web [17] is one domain where the significance of symbol names has been assessed, revealing that the semantics embedded in the names of IRIs (Internationalized Resource Identifiers) reflect a type of social semantics that aligns with the formal meaning of the referenced resource.

Regarding premise selection in commonsense knowledge bases, there are a few approaches that consider the meaning of symbol names. One such approach is Similarity SInE [18], an extension of SInE selection that uses word embeddings to account for symbol similarity. Similarity SInE ensures that all the formulae selected by SInE are included, and additionally selects formulae for symbols that are similar to those appearing in the goal. Furthermore, [19] introduces a vector-based technique for premise selection which uses a word embedding to vectorize the formulae of a knowledge base and to use vector similarity for selection purposes. The SeVEn [20] selection takes the idea of vectorizing a knowledge base one step further by using a large language model for the vectorization step. These three methods take into account the meaning of the symbol names, but they are not used to select the clause for the next inference step.

We are not aware of any approaches that use the meaning of symbols to select the clause to be used in the next reasoning step.

[1]See the manual of E for details: https://wwwlehre.dhbw-stuttgart.de/~sschulz/WORK/E_DOWNLOAD/V_2.4/eprover.pdf

3. Preliminaries

We consider the following first-order logic reasoning task in this paper: given of a set of formulae called the knowledge base (KB) and a goal G. The task is to show that the KB implies G which can be reduced to showing that the conjunction of KB and $\neg G$ is unsatisfiable. To show the unsatisfiability, usually the conjunction of the KB and the negated goal are transformed into an equisatisfiable clause normal form and a contradiction is inferred from the clause set.

To achieve this contradiction, saturation-based provers organise the execution of their inferences using a given-clause algorithm as shown in Alg. 1. This algorithm maintains two sets of clauses: The set of processed clauses and the set of unprocessed clauses. Initially, all clauses are added to the set of unprocessed clauses, and the set of processed clauses is empty. In each step, the algorithm selects a so-called given clause from the set of unprocessed clauses, adds it to the set of processed clauses, and then performs all the inferences possible with the given clause and the set of processed clauses. The resulting new clauses are added to the set of unprocessed clauses. This process is repeated until either the empty clause is inferred, or the set of unprocessed clauses is empty (or a resource or time limit is reached).

Algorithm 1 Given Clause Algorithm

```
Require: Initial set of clauses initial_clauses
1:  processed_set ← {}
2:  unprocessed_set ← initial_clauses
3:  while True do
4:      if empty_clause ∈ processed_set then
5:          return "Proof Found"
6:      end if
7:      if unprocessed_set is empty then
8:          return "No Proof Found"
9:      end if
10:     given_clause ← select_clause(unprocessed_set)
11:     remove given_clause from unprocessed_set
12:     add given_clause to processed_set
13:     new_clauses ← generate_new_clauses(given_clause,
        processed_set)
14:     simplified_clauses ← simplify_clauses(new_clauses)
15:     for each clause in simplified_clauses do
16:         if clause ∉ unprocessed_set then
17:             add clause to unprocessed_set
18:         end if
19:     end for
20: end while
```

The choice of the given clause (line 10 in Alg. 1) strongly influences whether and how fast a proof is found. An unskilful choice of given clause can quickly lead to a large number of irrelevant clauses in the set of unprocessed clauses, making it difficult to find a proof. On the other hand, a skilful choice of the given clause can lead to only those inferences

being made that are relevant to the proof task, so that a proof can be found directly.

To select the given clause, heuristics are usually used that assign a weight to each clause. The clause with the lowest weight is then selected as the given clause. Common heuristics use the age of the clause (older clauses are favoured and receive a lower weight) or are based on a weighted symbol count. These heuristics are often used in combination. The ratio of clauses selected by age to clauses selected by symbol count is called the pick-given ratio. A well-known ratio is the pick-given ratio of 5, which means that the heuristic selects the clause with the lowest weighted symbol count five times and the oldest clause once. There are many other heuristics for selecting clauses. See Sect. 2 for some pointers in this regard.

In the following, the set of all predicate and function symbols in a formula F is represented as $sym(F)$. In a slight abuse of notation, we use $sym(KB)$ to denote the set containing all predicate and function symbols occurring in a knowledge base KB.

3.1. Distributional Semantics

Our approach uses the meaning of the names of the function and predicate symbols. To do this, we rely on the distributional semantics of natural language. This is best explained using Firth's famous statement:

> You shall know a word by the company it keeps. [21]

In other words: If we look at very large texts, we can observe that words that occur in a similar context are similar [22].

One approach in this area are word embeddings [23], [24]. These are vector representations of words, usually learned using neural networks on very large amounts of text. For a given set of words V, also called a vocabulary, a word embedding is a function, $f : V \to \mathbb{R}^n$, that assigns an n-dimensional vector to each word in the vocabulary. Since our approach uses existing word embeddings and does not train new ones, we refrain from going into details on the training process of word embeddings.

It is very interesting that words with similar meanings are mapped to similar vectors. Usually, the similarity of two word vectors is calculated with the help of the cosine similarity.

Definition 1 (Cosine similarity of two vectors)**.** *Let $u, v \in \mathbb{R}^n$, both non-zero. The* cosine similarity *of u and v is defined as:*

$$cos\_sim(u, v) = \frac{u \cdot v}{||u||\ ||v||}$$

The cosine similarity of two vectors u and v takes values between -1 and 1. For exactly opposite vectors the value is -1, for orthogonal vectors the value is 0 and for equal vectors the value 1. The more similar two vectors are, the greater is their cosine similarity.

In our approach, we use a trained word embedding to determine similarities between the names of symbols in a knowledge base and the proof task. The next section describes how we do this.

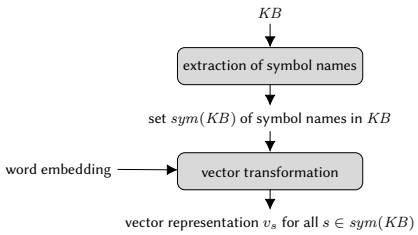

Figure 1: Overview of the preprocessing step which determines a vector representation for all function and predicate names occurring in a knowledge base KB.

4. Guiding the Reasoning Process by the Meaning of Symbol Names

Automated theorem provers usually calculate the weight of a clause from the weights of the symbols occurring in the clause. For example, the theorem prover E can be given values for weights for function and predicate symbols. These weights then determine which weights the clauses receive and therefore which clauses are preferred in the reasoning process.

We now present an approach to determine symbol weights, and thus clause weights, that uses the meaning of symbol names. As described above, we assume that there are multiple proof tasks to solve for a given knowledge base. In the following, when we talk about a knowledge base or a goal, we refer to the knowledge base or the goal in this scenario.

Our approach uses a word embedding. It is important that the word embedding fits to the domain of the knowledge base. For example if the knowlege base is Yago [25] it would be suitable to use a word embedding which was learnt on wikipedia articles. In the following, we refer to the word embedding used, as introduced above, as a function $f : V \to \mathbb{R}^n$, where V is the vocabulary of the word embedding.

Fig. 1 shows the first step of our approach, which determines a vector representation for all function and predicate symbols occurring in the knowledge base under consideration. To achieve this, in a first step, all function and predicate symbols are extracted from the knowledge base resulting in the set $sym(KB)$. Next, during the vector transformation, all extracted symbols $s \in sym(KB)$ are mapped to the vocabulary V of a word embedding. This mapping step is typically not trivial and highly depends on the knowledge base in use. It builds upon an existing WordNet mapping for the knowledge base, if available, which is then semi-automatically extended. We do not go into detail here and refer the reader to [26] for more information. In the following we assume that this step is possible for all symbols extracted from the knowledge base and that there is a mapping function $map : sym(KB) \to V$ which performs this step. Once a function or predicate symbol $s \in sym(KB)$ has been assigned a word $map(s)$ from the vocabulary of the word embedding used, the vector $v_s = f(map(s))$ corresponding to the word is assigned to the symbol. This vector v_s is used as the vector representation of the symbol s. This step only needs to be performed once for a knowledge base. Therefore, it can be considered as a pre-processing step.

Since we assume that for a large number of goals it has to be proven that they are entailed from the same knowledge

base, the effort of this preprocessing step is relativized.

Fig. 2 provides an overview of the process of proving a goal using the symbol-name heuristic. First, a vector representation of the goal is computed for a given goal G. To do this, all function and predicate symbols are extracted from the goal leading to the set $sym(G)$ and mapped to words in the vocabulary of the word embedding used. For this, the above mentioned function map is used. It is important to use the same word embedding that was chosen for the knowledge base in the pre-processing step. The next step is to compute the vector representation for the goal: For each symbol $s \in sym(G)$ in the goal, the vector v_s of the word $map(s)$ is looked up in the word embedding. The vectors of all symbols in $sym(G)$ are summed and divided by the number of symbols in the goal resulting in vector v_G:

$$v_G = \frac{\sum_{s \in sym(G)} v_s}{|sym(G)|} \quad (1)$$

In other words, the vector representation of a goal corresponds to the average of the vector representations of all the symbols in the goal.

In the next step, weights for symbols in the knowledge base are computed based on the vector representations of the function and predicate symbols in the knowledge base and the vector representation of the goal. To compute the weight of a symbol $s \in sym(KB)$, we first compute the similarity of its vector representation v_s and the vector representation of the goal v_G. Cosine similarity is used for this. The values of cosine similarity are between -1 and 1. The higher the similarity of the two vectors, the higher the value. On the other hand, symbol with a low value are preferred by provers. Therefore, we convert the cosine similarities into weights as follows:

We assume a default weight of 1000. From this we subtract 1000 times the cosine similarity. This allows us to account for three decimal places in the cosine similarity. This results in the weight of a symbol as

$$weight(s, G) = 1000 - cos\_sim(v_s, v_G) \quad (2)$$

The weights of the symbols calculated in this way are now passed to the theorem prover together with the proof task. The prover calculates the weights for the clauses from the weights for the symbols and uses them to control the proof process.

A first approach would be to pass all calculated weights for symbols to the theorem prover. Experiments have shown that this does not necessarily lead to the best results. Our experiments have shown that it is most advantageous to pass only the weights of the 10 highest scoring symbols and leave all other symbol weights at the default value of 1000.

5. Experimental Results

To evaluate the approach, we need a knowledge base with function and predicate symbols that are, or can be, mapped to natural language words. We also need a large number of proof tasks for this knowledge base. Adimen SUMO [27] is a first-order logic ontology based on the Suggested Upper Merged Ontology (SUMO). Since a mapping to WordNet synsets is available for most of the symbols of Adimen SUMO, this ontology is interesting for our evaluation. WordNet is an extensive lexical database of the English language,

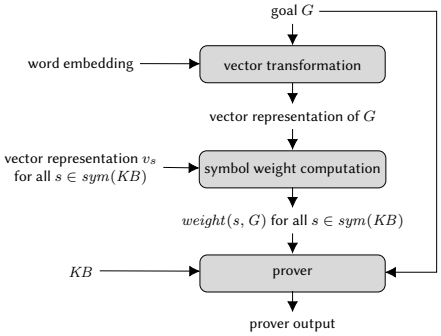

Figure 2: Overview of the process of proving goal G from knowlege base KB using the symbol name heuristic. Note that the vector representation of KB used here was computed in the preprocessing step depicted in Fig. 1 which determines a vector representation for all function and predicate names occurring in a knowledge base KB.

which groups words according to their meaning and forms synonymous groups, called synsets. For the vectorization we use the pretrained word embedding ConceptNet Numberbatch [28] which is well suited for the commonsense area.

The proof tasks considered in the experiments were created to test the application of the closed world assumption (CWA) to first-order logic ontologies [29]. Each proof task consists of a single formula, the goal, for which it must be shown that it can be inferred from the Adimen SUMO ontology. We call the set of all of these proof tasks the *Adimen SUMO CWA problems* in the following.

These Adimen SUMO problems are challenging even for highly optimised theorem provers. Usually theorem provers use selection techniques for this kind of problems, which try to select the formulas relevant for a given proof task from the knowledge base.

In all our experiments, we use the theorem prover E with a timeout of 10 seconds. We call it with different parameters combinations[2]:

Table 1
Prover configurations considered in the experiments.

Name in the Diagrams	Description
noAuto	E without any parameters
auto	E with `auto` parameter
weighted_topk	E with symbol weights for the k symbols most similar to the goal
weighted_top10_auto	E with symbol weights for the 10 symbols most similar to the goal and `auto` parameter

In the first experiments we omitted the selection and ran E (without any parameters) on the Adimen SUMO CWA problems and collected problems where E was able to find a proof within a timeout of 10 seconds. This resulted in 129 problems that we used for the first experiments. This test set is rather small, as there are not many problems that E can solve without parameters and without preselecting the

[2]Detailed information on the different parameters used for the theorem prover E can be found in the manual which is available at: https://wwwlehre.dhbw-stuttgart.de/~sschulz/WORK/E_DOWNLOAD/V_2.4/eprover.pdf

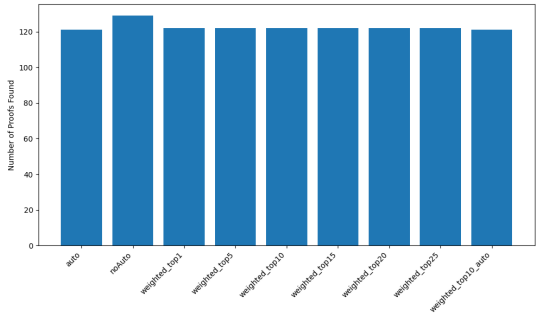

Figure 3: Number of proofs found on the 129 problems E was able to find a proof without any parameters within 10 seconds. The different E configurations listed in Tab. 1 were considered.

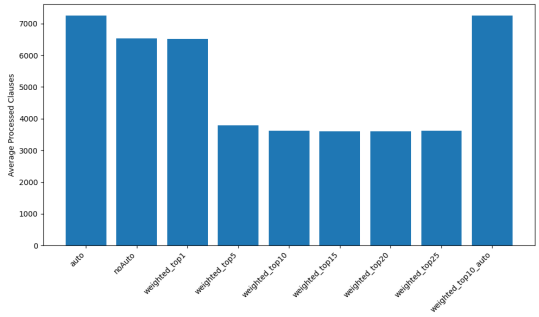

Figure 4: Average number of clauses processed during proof construction. Different parameters for E were considered. Problems considered were 129 Adimen SUMO problems E (without any parameters) was able to find a proof within a timeout of 10 seconds. The different E configurations listed in Tab. 1 were considered.

formulas.

Fig. 3 shows the number of proofs found for the different E configurations we considered. We see that the configuration with no parameters finds the highest number of proofs with 129 proofs, which is not surprising because of the way the problems were chosen. Furthermore, we see that the number of proofs found using the configuration with the `auto` parameter (121 proofs), as well as the configurations using the similarity-based weights, results in slightly fewer proofs being found (122 for all configurations of E using similarity-based symbol weights without `auto` parameter).

Since our approach aims at influencing the selection of the clause for the next inference step in such a way that the selected clause fits the thematic context of the goal, the number of processed clauses is of particular interest for the evaluation. This number is usually much larger than the number of clauses in the actual proof. The smaller the number of processed clauses, the more targeted the proof. Ideally, the number of processed clauses is equal to the number of clauses in the proof, and no unnecessary conclusions have been drawn.

Fig. 4 shows the average number of clauses processed by the different E configurations we considered. We can see that E with the `auto` parameter processes the most clauses. The number of clauses processed for E without any parameter and E using the similarity-based weight for only the symbol most similar to the goal are comparable. Again, this is not surprising as one would not expect a single symbol weight to have much influence. The configurations of E

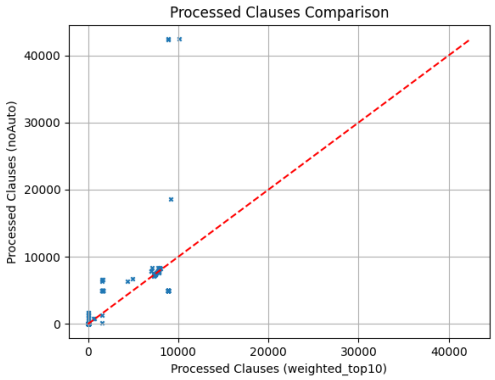

Figure 5: Number of clauses processed by E with the calculated symbol weights of the 10 symbols most similar to the goal (x-axis) and number of clauses processed by E without any parameters (y-axis). Problems considered were 129 Adimen SUMO problems E was able to find a proof within a timeout of 10 seconds. Each point corresponds to one proof. Points on the first bisector correspond to proofs where both configurations have processed the same number of clauses. Points above (below) the first bisector correspond to proofs where E without any parameters has processed more (fewer) clauses than E with the set symbol weights.

using similarity-based symbol weights for the 5 to 25 symbols most similar to the goal are comparable in terms of the number of clauses processed. The best configuration seems to be the one with 10 symbol weights. Compared to the configuration using E without parameters, the configurations using symbol weights for 5 to 25 symbols process only about half the number of clauses. This shows the usefulness of the presented approach. We have also investigated whether the combination of the `auto` parameter and the symbol weights for the 10 symbols most similar to the goal has advantages. However, we observe that the number of clauses processed by this configuration is comparable to the use of the `auto` parameter alone.

Fig. 5 shows a direct comparison of the number of processed clauses for the E configuration without using any parameters and the E configuration using the similarity based weights for the 10 symbols most similar to the proof goal. We observe that the majority of the points in Fig. 5 are above the first bisector. Thus, we see that for most problems the number of clauses processed could be reduced by using the similarity-based symbol weights.

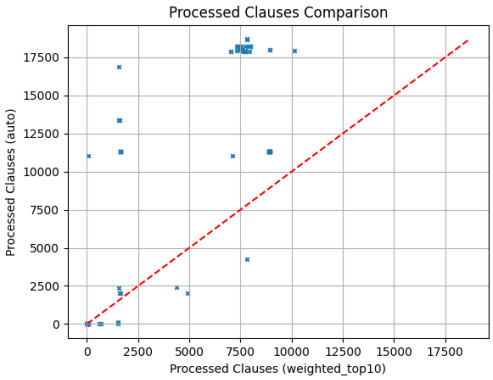

Figure 6: Number of clauses processed by E with the calculated symbol weights of the 10 symbols most similar to the goal (x-axis) and number of clauses processed by E using the auto parameter (y-axis). Problems considered were 129 Adimen SUMO problems E (without any parameters) was able to find a proof within a timeout of 10 seconds. Each point corresponds to one proof. Points on the first bisector correspond to proofs where both configurations have processed the same number of clauses. Points above (below) the first bisector correspond to proofs where E using the auto parameter has processed more (fewer) clauses than E with the 10 similarity-based symbol weights.

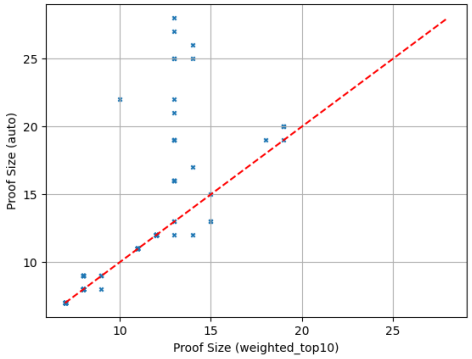

Figure 7: Sizes of proofs found with E using the calculated symbol weights of the 10 symbols most similar to the goal (x-axis) and number of clauses processed by E using the `auto` parameter (y-axis). Problems considered were 129 Adimen SUMO problems E (without parameters) was able to find a proof within a timeout of 10 seconds. Each point corresponds to one proof. Points on the first bisector correspond to proofs where the proofs of both configurations have the same size. Points above (below) the first bisector correspond to larger (smaller) proof sizes for E with the `auto` parameter compared to E with 10 similarity-based symbol weights.

Fig. 6 shows a direct comparison of the number of clauses processed for the E configuration using the `auto` parameter and the E configuration using the similarity-based weights for the 10 symbols most similar to the proof goal. The vast majority of the points in Fig. 6 are above the first bisector. Many are also at a large distance from it. For example, we see a cluster of problems for which E with auto-parameter processed around 17500 clauses, but for which the configuration with symbol weights processed only about 7500 clauses. This illustrates that the use of similarity-based symbol weights was able to drastically reduce the number of clauses processed in many cases.

Since we observe large differences in the number of clauses processed, the question arises whether the sizes of the proofs found also differ. In Fig. 7 we see the direct comparison of the sizes of proofs constructed with E using the `auto` parameter with the sizes of proofs constructed with E and the similarity-based symbol weights of the 10 symbols most similar to the goal. We observe that the proofs are often smaller when E was using similarity-based symbol weights. In future work, we plan to analyze the different proofs in more detail to determine how they differ and whether the shorter proofs are easier to understand.

Fig 8 shows the corresponding comparison, where we compared the proof size for proofs constructed with E without any parameters with the proof sizes of proofs constructed with E and the similarity-based symbol weights of the 10 symbols most similar to the goal. Here we observe that the size of some proofs is smaller using the configuration of E with symbol weights. However, the differences are not as large as in Fig 7, and in most cases the sizes do not differ.

In a second set of experiments, we selected formulas from 500 of the Adimen SUMO CWA problems with the SInE selection [30] (recursion depth 3, generosity 5.0) and used the resulting formula sets as test data for our evaluation. It is important to note that the SInE selection is incomplete. It is therefore possible that not all the necessary formulae

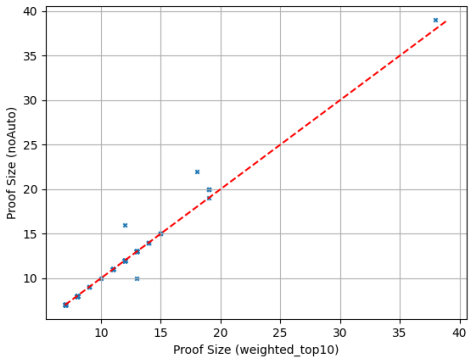

Figure 8: Sizes of proofs found with E using the calculated symbol weights of the 10 symbols most similar to the goal (x-axis) and number of clauses processed by E without any parameter (y-axis). Problems considered were 129 Adimen SUMO problems E was able to find a proof within a timeout of 10 seconds. Each point corresponds to one proof. Points on the first bisector correspond to proofs where the proofs of both configurations have the same size. Points above (below) the first bisector correspond to larger (smaller) proof sizes for E without any parameters compared to E with 10 similarity-based symbol weights.

for a proof were selected. Consequently, it is unlikely that a proof can be found for all of these 500 problems.

Since the E configuration, which combines symbol weights with the auto parameter, performed poorly in our first experiments, we do not include this configuration in the following experiments.

In Fig. 9 we can see that nearly the same number of proofs can be found by the different configurations of E. The low number of around 200 found proofs can be explained by the incompleteness of the SInE selection.

Fig. 10 shows the average number of clauses which were processed during proof construction. We observe that the

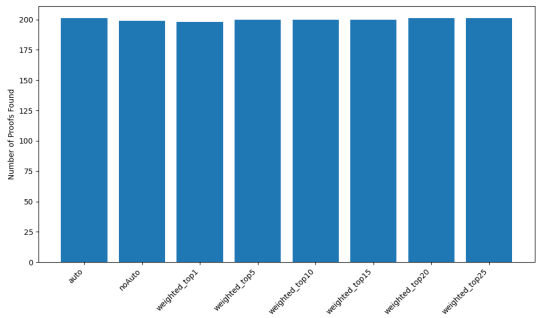

Figure 9: Number of proofs found on the result of selecting with SInE (recursion depth 3, benevolence 5.0) for 500 Adimen SUMO problems. For a description of the used E configurations see Tab. 1.

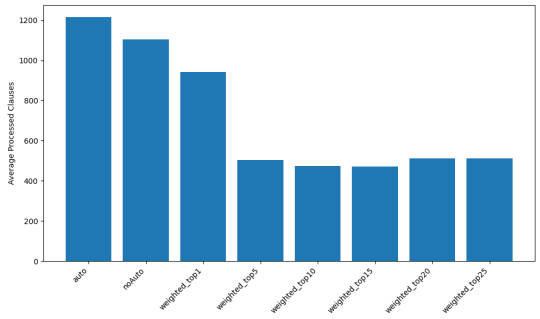

Figure 10: Average number of clauses processed during proof construction. Different parameters for E were considered. Problems considered were 500 Adimen SUMO problems for which formulae were preselected using SInE (recursion depth 3, benevolence 5.0). For a description of the used E configurations see Tab. 1.

average number of clauses processed by E with the `auto` parameter is the highest value with 1213.85. The average number of clauses processed by E without parameters is slightly lower with 1103.68. In comparison, E with the 10 best similarity-based symbol weights processes on average only 473.10 clauses, which is much less.

Fig. 10 shows that the similarity-based weights lead to a reduction in the average number of clauses processed. However, it is not clear how this reduction is distributed across the individual problems. Therefore, in Fig. 11 we directly compare the number of processed clauses of E without any parameters (y-axis) with the number of processed clauses of E with the symbol weights of the 10 symbols most similar to the goal (x-axis). For each proof we see a point in the graph. We observe that, with a few exceptions, the number of processed clauses is significantly lower for the E configuration with goal similarity-based symbol weights than for E without parameters.

As in the previous experiments, we also compare the sizes of the found proofs. Fig. 12 and Fig. 13 show the corresponding results. We observe that for the problems considered in the experiments with preselected formulae, the differences in proof size are not that large. There are many proofs where the proof sizes are the same. Furthermore, for both E configurations, there are proofs where the proof size is smaller and proofs where the proof size is larger. Comparing the proof sizes of E without any parameters with the configuration using 10 similarity-based symbol weights, we observe slightly more cases where the proof size is lower using symbol weights. But the differences are really small.

Compared to the experiments with the non preselected formulas (see Fig. 7 and Fig. 8), the differences in the proof sizes are much smaller here. A possible explanation for this could be the pre-selection of formulae with SInE. This preselection means that there are significantly fewer formulae available, which considerably limits the choices when constructing the proofs. We intend to investigate this in more detail in future work.

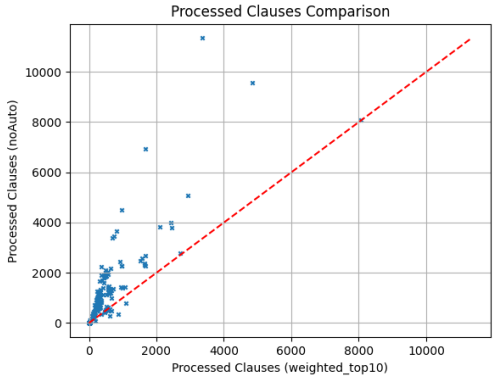

Figure 11: Number of clauses processed by E with the calculated symbol weights of the 10 symbols most similar to the goal (x-axis) and number of clauses processed by E without any parameters (y-axis). Problems considered were 500 Adimen SUMO problems for which formulae were preselected using SInE (recursion depth 3, benevolence 5.0). Each point corresponds to one proof. Points on the first bisector correspond to proofs where both configurations have processed the same number of clauses. Points above (below) the first bisector correspond to proofs where E without any parameters has processed more (fewer) clauses than E with the set symbol weights.

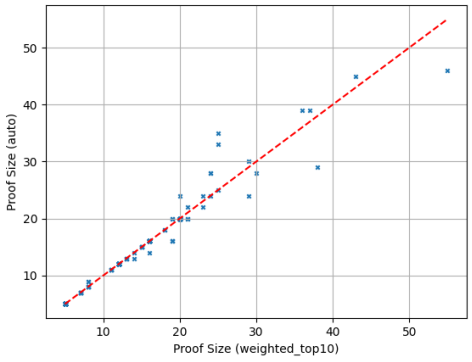

Figure 12: Sizes of proofs found with E using the calculated symbol weights of the 10 symbols most similar to the goal (x-axis) and number of clauses processed by E using the `auto` parameter (y-axis). Problems considered were 500 Adimen SUMO problems for which formulae were preselected using SInE (recursion depth 3, benevolence 5.0). Each point corresponds to one proof. Points on the first bisector correspond to proofs where the proofs of both configurations have the same size. Points above (below) the first bisector correspond to larger (smaller) proof sizes for E with the `auto` parameter compared to E with 10 similarity-based symbol weights.

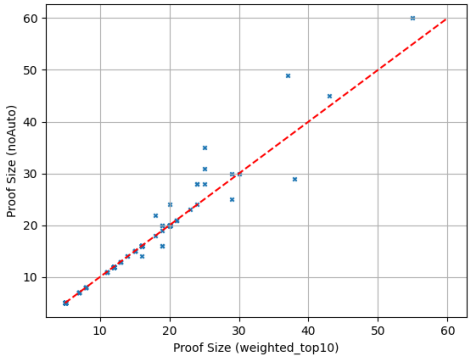

Figure 13: Sizes of proofs found with E using the calculated symbol weights of the 10 symbols most similar to the goal (x-axis) and number of clauses processed by E without any parameter (y-axis). Problems considered were 500 Adimen SUMO problems for which formulae were preselected using SInE (recursion depth 3, benevolence 5.0). Each point corresponds to one proof. Points on the first bisector correspond to proofs where the proofs of both configurations have the same size. Points above (below) the first bisector correspond to larger (smaller) proof sizes for E without any parameters compared to E with 10 similarity-based symbol weights.

6. Summary and Future Work

In this paper we have developed an approach to clause selection in automated theorem provers that aims to make the selection of the clause for the next inference step more goal-directed. The approach is based on the assumption that symbol names in knowledge bases have meaningful names that can be mapped to natural language words. If this assumption is correct, techniques from natural language processing can be used to determine the similarity between the proof goal and the symbols in the knowledge base. These similarities are then used to derive symbol weights that guide the selection of clauses for the next inference step. The approach has been implemented and evaluated using the theorem prover E. Our experimental results show a significant reduction in the number of clauses processed during proof construction, indicating a more goal-directed and efficient proof process.

In our experiments, we further observed that the use of similarity-based symbol weights has an impact on proof size (especially compared to the configuration using the `auto` parameter). In future work, we would like to analyze the differences in the proofs we found in more detail and investigate whether there are also differences in the comprehensibility of the proofs.

References

[1] D. Kahneman, Thinking, Fast and Slow, Macmillan, 2011.

[2] S. Schulz, S. Cruanes, P. Vukmirovic, Faster, higher, stronger: E 2.3, in: P. Fontaine (Ed.), Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Natal, Brazil, August 27-30, 2019, Proceedings, volume 11716 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 495–507. URL: https://doi.org/10.1007/978-3-030-29436-6_29. doi:10.1007/978-3-030-29436-6_29.

[3] A. Riazanov, A. Voronkov, The design and implementation of VAMPIRE, AI Commun. 15 (2002) 91–110. URL: http://content.iospress.com/articles/ai-communications/aic259.

[4] S. Schulz, M. Möhrmann, Performance of clause selection heuristics for saturation-based theorem proving, in: N. Olivetti, A. Tiwari (Eds.), Automated Reasoning - 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 - July 2, 2016, Proceedings, volume 9706 of *Lecture Notes in Computer Science*, Springer, 2016, pp. 330–345. URL: https://doi.org/10.1007/978-3-319-40229-1_23. doi:10.1007/978-3-319-40229-1_23.

[5] F. Bártek, M. Suda, How much should this symbol weigh? A gnn-advised clause selection, in: R. Piskac, A. Voronkov (Eds.), LPAR 2023: Proceedings of 24th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Manizales, Colombia, 4-9th June 2023, volume 94 of *EPiC Series in Computing*, EasyChair, 2023, pp. 96–111. URL: https://doi.org/10.29007/5f4r. doi:10.29007/5F4R.

[6] J. Denzinger, S. Schulz, Learning domain knowledge to improve theorem proving, in: M. A. McRobbie, J. K. Slaney (Eds.), Automated Deduction - CADE-13, 13th International Conference on Automated Deduction, New Brunswick, NJ, USA, July 30 - August 3, 1996, Proceedings, volume 1104 of *Lecture Notes in Computer Science*, Springer, 1996, pp. 62–76. URL: https://doi.org/10.1007/3-540-61511-3_69. doi:10.1007/3-540-61511-3_69.

[7] J. Jakubuv, J. Urban, ENIGMA: efficient learning-based inference guiding machine, in: H. Geuvers, M. England, O. Hasan, F. Rabe, O. Teschke (Eds.), Intelligent Computer Mathematics - 10th International Conference, CICM 2017, Edinburgh, UK, July 17-21, 2017, Proceedings, volume 10383 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 292–302. URL: https://doi.org/10.1007/978-3-319-62075-6_20. doi:10.1007/978-3-319-62075-6_20.

[8] J. Jakubuv, J. Urban, Enhancing ENIGMA given clause guidance, in: F. Rabe, W. M. Farmer, G. O. Passmore, A. Youssef (Eds.), Intelligent Computer Mathematics - 11th International Conference, CICM 2018, Hagenberg, Austria, August 13-17, 2018, Proceedings, volume 11006 of *Lecture Notes in Computer Science*, Springer, 2018, pp. 118–124. URL: https://doi.org/10.1007/978-3-319-96812-4_11. doi:10.1007/978-3-319-96812-4_11.

[9] J. Jakubuv, J. Urban, Hammering mizar by learning clause guidance (short paper), in: J. Harrison, J. O'Leary, A. Tolmach (Eds.), 10th International Conference on Interactive Theorem Proving, ITP 2019, September 9-12, 2019, Portland, OR, USA, volume 141 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 34:1–34:8. URL: https://doi.org/10.4230/LIPIcs.ITP.2019.34. doi:10.4230/LIPICS.ITP.2019.34.

[10] M. Suda, Improving enigma-style clause selection while learning from history, in: A. Platzer, G. Sutcliffe (Eds.), Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings, volume 12699 of *Lecture Notes in Computer*

Science, Springer, 2021, pp. 543–561. URL: https://doi.org/10.1007/978-3-030-79876-5_31. doi:10.1007/978-3-030-79876-5_31.

[11] G. Bancerek, C. Bylinski, A. Grabowski, A. Kornilowicz, R. Matuszewski, A. Naumowicz, K. Pak, J. Urban, Mizar: State-of-the-art and beyond, in: M. Kerber, J. Carette, C. Kaliszyk, F. Rabe, V. Sorge (Eds.), Intelligent Computer Mathematics - International Conference, CICM 2015, Washington, DC, USA, July 13-17, 2015, Proceedings, volume 9150 of *Lecture Notes in Computer Science*, Springer, 2015, pp. 261–279. URL: https://doi.org/10.1007/978-3-319-20615-8_17. doi:10.1007/978-3-319-20615-8_17.

[12] J. Alama, T. Heskes, D. Kühlwein, E. Tsivtsivadze, J. Urban, Premise selection for mathematics by corpus analysis and kernel methods, J. Autom. Reason. 52 (2014) 191–213. URL: https://doi.org/10.1007/s10817-013-9286-5. doi:10.1007/S10817-013-9286-5.

[13] G. Irving, C. Szegedy, A. A. Alemi, N. Eén, F. Chollet, J. Urban, Deepmath - deep sequence models for premise selection, in: D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, R. Garnett (Eds.), Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, Barcelona, Spain, 2016, pp. 2235–2243.

[14] B. Piotrowski, J. Urban, Stateful premise selection by recurrent neural networks, in: E. Albert, L. Kovács (Eds.), LPAR 2020: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Alicante, Spain, May 22-27, 2020, volume 73 of *EPiC Series in Computing*, EasyChair, 2020, pp. 409–422. URL: https://doi.org/10.29007/j5hd. doi:10.29007/J5HD.

[15] K. Prorokovic, M. Wand, J. Schmidhuber, Improving stateful premise selection with transformers, in: F. Kamareddine, C. S. Coen (Eds.), Intelligent Computer Mathematics - 14th International Conference, CICM 2021, Timisoara, Romania, July 26-31, 2021, Proceedings, volume 12833 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 84–89. URL: https://doi.org/10.1007/978-3-030-81097-9_6. doi:10.1007/978-3-030-81097-9_6.

[16] M. Mikula, S. Antoniak, S. Tworkowski, A. Q. Jiang, J. P. Zhou, C. Szegedy, L. Kucinski, P. Milos, Y. Wu, Magnushammer: A transformer-based approach to premise selection, CoRR abs/2303.04488 (2023). URL: https://doi.org/10.48550/arXiv.2303.04488. doi:10.48550/ARXIV.2303.04488. arXiv:2303.04488.

[17] S. de Rooij, W. Beek, P. Bloem, F. van Harmelen, S. Schlobach, Are names meaningful? quantifying social meaning on the semantic web, in: ISWC (1), volume 9981 of *Lecture Notes in Computer Science*, 2016, pp. 184–199.

[18] U. Furbach, T. Krämer, C. Schon, Names are not just sound and smoke: Word embeddings for axiom selection, in: CADE, volume 11716 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 250–268.

[19] C. Schon, Associative reasoning for commonsense knowledge, in: D. Seipel, A. Steen (Eds.), KI 2023: Advances in Artificial Intelligence - 46th German Conference on AI, Berlin, Germany, September 26-29, 2023, Proceedings, volume 14236 of *Lecture Notes in Computer Science*, Springer, 2023, pp. 170–183. URL: https://doi.org/10.1007/978-3-031-42608-7_14. doi:10.1007/978-3-031-42608-7_14.

[20] C. S. Oliver Jakobs, Context-specific selection of commonsense knowledge using large language models, in: to appear in KI 2024, Lecture Notes in Computer Science, Springer, 2024.

[21] J. R. Firth, Papers in Linguistics 1934 - 1951: Rep, Oxford University Press, 1991.

[22] G. A. Miller, W. G. Charles, Contextual correlates of semantic similarity, Language and Cognitive Processes 6 (1991) 1–28. URL: http://eric.ed.gov/ERICWebPortal/recordDetail?accno=EJ431389.

[23] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, J. Dean, Distributed representations of words and phrases and their compositionality, in: NIPS, 2013, pp. 3111–3119.

[24] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, CoRR abs/1301.3781 (2013). URL: http://arxiv.org/abs/1301.3781. arXiv:1301.3781.

[25] F. M. Suchanek, G. Kasneci, G. Weikum, Yago: A large ontology from Wikipedia and WordNet, Web Semant. 6 (2008) 203–217. URL: http://dx.doi.org/10.1016/j.websem.2008.06.001. doi:10.1016/j.websem.2008.06.001.

[26] C. Schon, Selection strategies for commonsense knowledge, 2022. URL: https://arxiv.org/abs/2202.09163. doi:10.48550/ARXIV.2202.09163.

[27] J. Álvez, P. Lucio, G. Rigau, Adimen-sumo: Reengineering an ontology for first-order reasoning, Int. J. Semantic Web Inf. Syst. 8 (2012) 80–116.

[28] R. Speer, J. Chin, C. Havasi, Conceptnet 5.5: An open multilingual graph of general knowledge, in: AAAI, AAAI Press, 2017, pp. 4444–4451.

[29] J. Álvez, I. Gonzalez-Dios, G. Rigau, Applying the closed world assumption to sumo-based FOL ontologies for effective commonsense reasoning, in: G. D. Giacomo, A. Catalá, B. Dilkina, M. Milano, S. Barro, A. Bugarín, J. Lang (Eds.), ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020), volume 325 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2020, pp. 585–592. URL: https://doi.org/10.3233/FAIA200142. doi:10.3233/FAIA200142.

[30] K. Hoder, A. Voronkov, Sine qua non for large theory reasoning, in: N. Bjørner, V. Sofronie-Stokkermans (Eds.), Automated Deduction – CADE-23, volume 6803 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2011, pp. 299–314. URL: http://dx.doi.org/10.1007/978-3-642-22438-6_23. doi:10.1007/978-3-642-22438-6_23.