An Experimental Pipeline for Automated Reasoning in Natural Language

Tanel Tammet$^{1[0000-0003-4414-3874]}$, Priit Järv$^{1[0000-0001-7725-543X]}$, Martin Verrev$^{1[0000-0003-4890-9283]}$, and Dirk Draheim$^{2[0000-0003-3376-7489]}$

[1] Applied Artificial Intelligence Group,
Tallinn University of Technology, Tallinn, Estonia
{tanel.tammet,priit.jarv1,martin.verrev}@taltech.ee
[2] Information Systems Group, Tallinn University of Technology, Tallinn, Estonia
dirk.draheim@taltech.ee

Abstract. We describe an experimental implementation of a logic-based end-to-end pipeline of performing inference and giving explained answers to questions posed in natural language. The main components of the pipeline are semantic parsing, integration with large knowledge bases, automated reasoning using extended first order logic, and finally the translation of proofs back to natural language. While able to answer relatively simple questions on its own, the implementation is targeting research into building hybrid neurosymbolic systems for gaining trustworthiness and explainability. The end goal is to combine machine learning and large language models with the components of the implementation and to use the automated reasoner as an interface between natural language and external tools like database systems and scientific calculations.

1 Introduction

Question answering and inference using natural language is a classic A.I. area, with a long history of little success using symbolic methods, able to solve only small problems with a limited structure. The recent machine learning (ML) systems, in particular, the Large Language Model (LLM) implementations of the BERT and GPT families are, in contrast, often able to give satisfactory answers to nontrivial questions.

However, the current LLMs are neither trustworthy nor explainable. They have a well-known tendency of “hallucinating”, i.e. giving wrong answers and inventing actually nonexistent entities and facts. The problems of explicitly controlling the output and giving explanations for the solutions appear to be very hard for LLMs. An optimistic view of LLMs suggests that end-to-end learning can be improved to overcome these issues, while a more pessimistic view suggests that the problems are inherent and stem from the lack of an internal world model. The proponents of the latter view propose to build hybrid neurosymbolic systems, combining machine learning and symbolic methods of various kinds. Indeed, the research in the field of neurosymbolic systems has become quite active.

The recent survey [12] points to a wider interest in connecting natural language systems to external software like databases and scientific calculations.

Using logic for natural language inference (NLI) in combination with ML may potentially alleviate the problems with LLMs and provide a glue to connect external systems to natural language interfaces. However, using logic directly for processing natural language is hard, for a number of reasons:

– Semantic parsing, i.e. translating natural language to logic, is extremely hard due to the highly complex and exception-rich nature of natural language.
– Existing knowledge bases of "common sense" do not cover a critical mass of the basic understanding of the world even a small child possesses.
– Classical first order reasoning itself cannot cope with contradictory knowledge items, probabilistic or uncertain information and exceptions to rules.
– Finding logic-based proofs often requires long proofs and the huge knowledge base causes a quick combinatorial explosion of the search space.

The motivation behind the research described in the paper is the following hypothesis: all the main problems described above can be alleviated by using ML techniques tailored separately for each particular problem. The current paper does not introduce any ML techniques for the problems above. The goal of our system is to serve as a backbone for research into combining the symbolic methods with ML. Our hypothesis is that by gradual improvement and combination of the existing symbolic subsystems with ML techniques it is possible to eventually build a question answering system which has enough power, trustworthiness and explainability to be practically useful in various application areas.

2 Related Work

Here we will only consider projects building a full NLP inference system. The performance of older pure symbolic or logic-based methods like LogAnswer [6] remained at the level of specific toy examples and never achieved capabilities required for wider applicability. The long-running CYC project [18], although having several successes, did not succeed with its original stated goals, which is often used as an argument against symbolic systems.

A popular area for language processing is converting human queries to SQL or SPARQL queries. These systems typically do not handle rules expressed in natural language. The projects closest to ours use reasoners with a relatively limited capacity, like BRAID [10], which uses extended SLD+ reasoner with probabilistic rules and fuzzy unification, CASPR [15], which uses an ASP reasoner incorporating default logic, NatPro [1],[2], which uses a Natural Logic prover. The latter is the only such project we know to be publicly available: https://github.com/kovvalsky/prove_SICK_NL

The majority of research in neurosymbolic reasoning for natural language combines ML with weak forms of symbolic systems, typically taxonomies and triple graph knowledge bases like ConceptNet[20]. However, there are a few research projects combining ML with reasoning in quantified first order logic, although we are not aware of any such systems being publicly available. Noteworthy projects involving quantified logic are SQuARE [4], BRAID [10] and STAR [17].

We are not aware of any current projects except ours using high-performance full first order reasoners for the NLI task.

3 Natural Language Inference and Question Answering

The described pipeline is able to handle both the natural language inference (NLI) tasks (given a premise, determine whether a given hypothesis is true, false or indeterminate) and the closely related question answering tasks of finding a specific object matching a given criteria.

We will use a few simple examples throughout the paper. The expected answer to the first example *"If an animal likes honey, then it is probably a bear. Most bears are big, although young bears are not big. John is an animal who likes honey. Mike is a young bear. Who is big?"* is *"Likely John"*. The expected answer to the second example *"The length of the red car is 4 meters. The length of the black car is 5 meters. The length of the red car is less than 5 meters?"* is *"True"*.

It is worth noting that these examples are solved correctly by the current version of ChatGPT: moreover, ChatGPT is able to give a satisfactory explanation of the reasoning behind the answers. However, if we replace the known words in these questions above with invented words and insert additional irrelevant information, our system still finds the expected answer, while ChatGPT fails. The modified first example: *"If a greezer likes foozers, then it is probably a drimm. Greezers can eat frozen bread. Most drimms are red, although young drimms are not red. John likes bread. John is a nice greezer who likes foozers. Mike is a young drimm. Mike can eat a lot. Penguins are birds who cannot fly. Who is red?"*. The modified second example: *"The length of the barner is 200000000 meters. The length of the red foozer is 312435 meters. The length of the black foozer is 512000 meters. The length of the yellow foozer is 1000000 meters. The length of the red foozer is less than 312546 meters?"*. However, the answers given by ChatGPT vary over time, i.e. experiments with ChatGPT are not reproducible.

4 The Question Answering Pipeline

Our system is publicly available at http://github.com/tammet/nlpsolver. It requires Linux and should be easy to install. The implementation consists of four main software systems. The pipeline driver calls the external Stanza parser from Stanford, giving a Universal Dependencies (UD) graph, then runs the semantic parser on the UD graph, calls the reasoner, and finally builds a natural language answer along with the explanation built from the proofs given by the reasoner.

The pipeline driver, parser and answer construction components consist of over 400 Kbytes of Python code. Before running the solver, a small Python server component has to be started, to initialize the external UD parser Stanza and read a commonsense knowledge base into shared memory. For reasoning the pipeline calls our commonsense reasoner GK, written in C: this is the largest and the most complex part of the pipeline. There is a separate Python program for regression tests, along with several Python files containing sub-tests, currently over 1600 separate NLI tasks. The pipeline driver is called from a command line, with a natural language text and question as a command line argument, plus a number of optional arguments to control the behaviors like the amount of output.

4.1 Semantic Parsing

The parser takes English strings of natural language text as input and outputs extended clausified first-order logic formulas encoded in JSON-LD-LOGIC [24]. The main extension is adding numerical confidence to clauses and implementing default logic by including special literals to encode exceptions.

Parsing consists of a number of phases, each adding new structural details to the results of the previous phases. For the most part, the phases are implemented procedurally, without using explicit transformation rules: we found that the more complex aspects of translation cannot be easily expressed with the help of simple transformation rules. In particular, the correct interpretation of a sentence depends heavily on previous sentences and a collected database of objects which have been talked about.

Conversion to Universal Dependencies (UD) format We use the external Stanza parser to get the UD format dependency graphs from input sentences. Stanza itself uses pretrained neural models. We first preprocess English strings to avoid several typical mistakes of the Stanza conversion, and then interleave Stanza with simplifying transformations from the UD format to a simplified English text, which is then fed to Stanza to get the final result.

Converting UD to Logic One of the strengths of UD representation given by Stanza is a high level of detail. The first subphase of conversion is restructuring the UD graph to a semi-logical representation explicating the outward logical structure around the subject/verb, object/verb or subject/verb/object tuples. The following subphases attach different kinds of properties to words. For example, the outmost structure constructed for the sentence "Most bears are big, although young bears are not big." is
`[and, svo[bear,be,big], svo[bear,be,big]]` which is then extended to
`[and, svo[bear,be,big], svo[[props,young,bear],be,big]]`. The words in these structures are key-value objects containing both the initial UD information and additional details added during the phases.

The next subphase results in the extended logic in a non-clausified form, i.e. using explicit quantifiers. The conversion uses the previous structure recursively, taking into account the details of the original UD structure to find additional critical information like articles, negation, different kinds of quantifiers etc. We follow the approach of Davidsonian semantics, introducing event identification variables, while not taking the neo-Davidsonian path of splitting all relations to their minimal components.

For the coreference resolution we calculate the weighted heuristic scores for all candidate words, using also taxonomies of Wordnet. Another inherently complex task is determining whether a noun stands for a concrete object or should be quantified over. Importantly, any object detected is stored in a special data structure with new information about the object possibly added as the parsing process proceeds.

Let us consider an example sentence "John is a nice animal who likes honey." It would be first converted to a conjunction of three formulas

$$\begin{array}{l}
\mathtt{isa}(\mathtt{animal}, \mathtt{c1\_John}) \\
\mathtt{prop}(\mathtt{nice}, \mathtt{c1\_John}, \mathtt{generic}, \mathtt{generic}, \mathtt{ctxt}(\mathtt{Pres}, \mathtt{1})) \\
\mathtt{def0}(\mathtt{c1\_John}) \\
\forall \mathtt{S}\ (\mathtt{def0}(\mathtt{c1\_John}) \leftrightarrow \\
\quad \exists \mathtt{X}\ \mathtt{isa}(\mathtt{honey}, \mathtt{X}) \,\&\, (\exists \mathtt{A}\ \mathtt{do2}(\mathtt{like}, \mathtt{c1\_John}, \mathtt{X}, \mathtt{A}, \mathtt{ctxt}(\mathtt{Pres}, \mathtt{S}))))
\end{array}$$

The system determined that in this sentence "John" refers to a concrete object and immediately created a Skolem constant `c1_John`, storing it for possible later use and extension. Here it also created a new definition `def0` for encoding the complex property of "John": liking honey. The properties of objects like given in the second formula above also encode the intensity of the property (slightly/very) and the comparative class: for example, saying "John is a very large animal ..." would create $\mathtt{prop}(\mathtt{large}, \mathtt{c1\_John}, 3, \mathtt{animal}, \mathtt{ctxt}(\mathtt{Pres}, \mathtt{1}))$. The constant `generic` indicates that intensity is not known or that the property is not comparative, i.e. does not relate to a specific class. The term $\mathtt{ctxt}(\mathtt{Pres}, \mathtt{1})$ encodes contextual aspects: the present tense and a concrete situation number in a possible sequence of situations created by different actions. The variable `A` in the last formula is an identifier of an action, which can be given additional properties, like place, time or assistive objects of an action, in the Davidsonian style.

The system is also able to handle simpler questions involving sizes of sets, like "An animal had two strong legs. The animal had a strong leg?", "John has three big nice cars. John has two big cars?", and measures, like "The length of the red car is 4 meters. The length of the black car is 5 meters. The length of the red car is less than 5 meters?". We use terms encoding the sets and measures: for example, the first sentence of the last question is translated to a formula containing a standard equality predicate, an integer and several properties involving the measure term, including the main statement $4 = \mathtt{count}(\mathtt{measure1}(\mathtt{length}, \mathtt{c1\_car}, \mathtt{meter}, \mathtt{ctxt}(\mathtt{Pres}, \mathtt{1})))$

Instance Generation In order to answer questions without indicating concrete objects, like "Adult bears are large animals. Cats are small animals. Who is a large animal?" we need constants representing an anonymous instance of a class, essentially a "default adult bear", a "default bear" and a "default cat". For each such object the system generates a constant along with the formulas indicating its class and properties, enabling the system to produce an answer "An adult bear".

Question Handling Actual questions like "Who is big?" or "The length of the red car is less than 5 meters?" require special handling. The automated reasoner GK used in the pipeline employs the well-known *answer predicate* technique to construct and output the required substitution term. All the variables in the question formula will be instantiated and output, potentially resulting in a large combination of different answers. The "Who is big?" question will be first translated to $\exists \mathtt{X}, \mathtt{Y}, \mathtt{Z}\, \mathtt{prop}(\mathtt{big}, \mathtt{X}, \mathtt{generic}, \mathtt{Y}, \mathtt{Z})$ indicating that we are not restricting the "bigness" or context in the question. However, we do not want to enumerate different "bigness" values or contexts in the answer, thus we wrap the formula into a definition (say, `def2`) over a single variable `X`, and search for different substitutions into `def2(X)` only. Asking questions about location and time is implemented by constructing a number of questions over relations "near", "on", ' "at", etc.

Clausification and Simplification The system contains a clausifier skolemizing the formulas and converting these to a conjunctive normal form. The clausification phase also performs several simplifications, some of which are possible due to the known properties of the constructed formulas. Since nontrivial formulas may be converted into several clauses, the clausifier decides how to spread the numeric confidence of the formula and the exception literals in the formula into the clauses.

4.2 Integration with Knowledge Bases

The knowledge base provides the world model of our reasoning system. To answer the query "Tweety is a bird. Can Tweety fly?", the system needs to have the background knowledge that birds can fly. We construct the knowledge base (KB) using default logic rules augmented with numeric confidences. A small part of the knowledge base forms a core world model and is built by hand, while the bulk of the knowledge is integrated automatically from existing common sense knowledge (CSK) sources as described in [9].

We have integrated eight published knowledge graphs: ConceptNet [20], WebChild [25], Aristo TupleKB [13], Quasimodo [19], Ascent++ [14], UnCommonSense [3], ATOMIC$^{20}_{20}$ [8] and ATOMIC^{10x} [27]. These CSK sources are collections of relation triples. The majority of the sources contain natural language clauses or fragments in the triple elements. We have built a specialized pattern matching semantic parser to convert the relations to first order logic rules with the default logic extensions and estimated numeric confidence. The full knowledge base contains 18.5 million rules, with over 15 million of those are related to taxonomy: inferring a property or an event from the class of an entity.

4.3 Automated Reasoning

We use our automated reasoner GK to solve the problems generated by semantic parser. The reasoner uses both the parser output and a selected subset of the world knowledge to solve the questions. Wordnet taxonomies are used to solve the precedence problem of exceptions. Large datasets are parsed, indexed and kept in shared memory for quick re-use. GK is built on top of a conventional high-performance resolution-based reasoner GKC [21] for conventional first order logic. Thus GK inherits most of the capabilities and algorithms of GKC. The main additional features of GK are following:

– Using a well-known answer clause mechanism for finding a number of different answers, with a configurable limit.
– Finding expected proofs even if a knowledge base is inconsistent. Basically, GK only accepts proofs which contain a clause originating from the question.
– Searching for both a proof of the question and a negation of the question / negation of each concrete answer.
– Estimating the numeric confidence in the statements derived from knowledge bases containing uncertain contrary and supporting evidence obtained from different sources.
– Handling exceptions by implementing default logic via recursively deepening iterations of searches with diminishing time limits.
– Performing reasoning by analogy via employing known similarity scores of words along with exceptions.

The first four features are covered in our previous paper [23] and the following two are covered in [22]. The word similarity handling is currently in an experimental phase: the initial experiments show that a naive implementation creates an unmanageable search space explosion, and thus a layered approach is necessary.

As a simple example of the basic features, consider sentences “John is nice. John is not nice. Mike is nice. Steve is not nice.” GK output to the parsed versions of the following questions will directly lead to these answers: “John is nice?”: “Unknown”, “Mike is nice?”: “True”, “Mike is not nice?”: “False”, “Who is nice?”: “Mike”, “Who is not nice?”: “Steve”. For a slightly more complex example, consider the earlier “If an animal likes honey, then it is probably a bear. Most bears are big, although young bears are not big. John is an animal who likes honey. Mike is a young bear. Who is big?”. GK will output the following proof in JSON, where we have removed quotation marks and a number of steps:

```
{result:answer found,

answers:[
{
```

```
answer:[[$ans,some_bear]],
blockers:[[$block,[$,bear,1],[$not,[prop,big,some_bear,$generic,$generic,[$ctxt,Pres,1]]]]],
confidence:0.85,
positive proof:
[
...,
[7,[mp,[5,1],6,fromgoal,0.85],
  [[$block,[$,bear,1],[$not,[prop,big,some_bear,$generic,$generic,[$ctxt,Pres,1]]]],
  [$ans,some_bear]]]
]},
{
answer:[[$ans,c1_John]],
blockers:[[$block,[$,bear,1],[$not,[prop,big,c1_John,$generic,$generic,[$ctxt,Pres,1]]]],
         [$block,[$,animal,3],[$not,[isa,bear,c1_John]]]],
confidence:0.765,
positive proof:
[
[1,[in,frm_10,axiom,0.85],
   [[$block,[$,bear,1],[$not,[prop,big,?:X,$generic,$generic,[$ctxt,Pres,1]]]],
    [prop,big,?:X,$generic,$generic,[$ctxt,Pres,1]],
    [-isa,bear,?:X]]],
[2,[in,frm_9,axiom,0.9],
   [[$block,[$,animal,3],[$not,[isa,bear,?:X]]],
    [-do2,like,?:X,?:Y,?:Z,[$ctxt,Pres,1]],
    [-isa,honey,?:Y],[-isa,animal,?:X],[isa,bear,?:X]]],
...,
[18,[mp,[1,2],[17,1],fromaxiom,0.765],
  [[$block,[$,bear,1],[$not,[prop,big,c1_John,$generic,$generic,[$ctxt,Pres,1]]]],
  [$block,[$,animal,3],[$not,[isa,bear,c1_John]]],
  [prop,big,c1_John,$generic,$generic,[$ctxt,Pres,1]]]],
...,
[21,[in,frm_30,goal,1],[[-$def2,?:X],[$ans,?:X]]],
[22,[mp,[20,2],21,fromgoal,0.765],
  [[$block,[$,bear,1],[$not,[prop,big,c1_John,$generic,$generic,[$ctxt,Pres,1]]]],
   [$block,[$,animal,3],[$not,[isa,bear,c1_John]]],
   [$ans,c1_John]]]
]}
]}
```

Observe that we get two answers. The following NLP pipeline step removes the generic `[[$ans,some_bear]]`, since the more informative `[[$ans,c1_John]]` is available. Here both proofs contain only positive parts, although in the general case we may find both a positive and a negative proof, each with their own confidences. GK will throw away both the clauses produced during search and the final answers which have a summary confidence below a configurable threshold. GK will also throw away proofs which do not contain a goal clause.

The final answers contain blocker literals, which have been recursively checked by separate proof searches before the final proof is accepted by GK. The details of these failed searches are not shown in the final proof. Had we included the sentence "John is not big" in our example, then the proof of the first blocker of the main answer would have been found, thus disqualifying the proof and leaving us with the final answer "Likely a bear.".

4.4 Answers and Explanations in Natural Language

Answers and explanations are generated from the proof, with additional details taken from the database of objects along with their properties as detected during semantic parsing. While some of the principles were described in the previous section, there are two major tasks to perform: give a suitably detailed representation of objects in a proof (say, select between "a car", "a red car", "the red car", "Mike's car" etc) and create a grammatically correct and easy-to-understand textual representation of clauses. The system translates clauses in a proof one-to-one to English sentences, as exemplified by the explanation generated from the previously presented proof:

```
Likely john:
Confidence 76%.
Sentences used:
(1) If an animal likes honey, then it is probably a bear.
(2) Most bears are big, although young bears are not big.
(3) John is an animal who likes honey.
(4) Who is big?
Statements inferred:
(1) If X is a bear, then X is big. Confidence 85%. Why: sentence 2.
(2) If X does like Y and Y is a honey and X is an animal, then X is a bear.
    Confidence 90%. Why: sentence 1.
(4) If John has a property def1, then John does like cs4. Why: sentence 3.
...
(18) John is big. Confidence 76%. Why: statements 1, 17.
...
(21) If X matches the query, then X is an answer. Why: the question.
(22) John is an answer. Confidence 76%. Why: statements 20, 21.
```

5 Performance and the Test Set

The system has miserable performance on most well-known natural language inference or question answering benchmarks, the majority of which are oriented towards machine learning. As an exception, the performance on the anti-machine-learning question set HANS [11] is ca 95%, in contrast to the ca 60% performance of LLM systems before the GPT3 family (random choice would give 50% performance). The loss of 5% of HANS is due to the wrong UD parses chosen by Stanza.

However, the system is able to solve almost all of the demonstration examples of the Allen AI ProofWriter system
https://proofwriter.apps.allenai.org/ and is able to solve inference problems the current LLM systems cannot, like the examples presented in the introduction. For regression testing we have built a set of ca 1600 simple questions with answers, structured over different types of capabilities. This test set may be of use for people working towards similar goals.

The runtime for the small examples presented in the paper is ca 0.5 seconds on a Linux laptop with a graphics card usable by Stanza. Of this time, Stanza UD parsing takes ca 0.17 seconds, UD to logic takes ca 0.04 seconds, and the rest is spent by the reasoner. For more complex examples the reasoner may spend unlimited time, i.e. the question is rather how complex questions can be solved in a preconfigured time window. In case the size of the input problem is relatively small and a tiny world model suffices for the solution, the correct answer is found in ca 1-2 seconds. However, in case the system is given a large knowledge base (KB) with a size of roughly one gigabyte, and the answer actually depends on the KB, then the search space may explode and the system may fail to find answer in a reasonable time. Efficiently handling a very large knowledge base clearly requires suitable heuristics based on the semantics and interdependence of rules / facts in the KB.

6 Towards a Hybrid Neurosymbolic System

Although the scope of the sentences successfully parsed and questions answered could be improved by adding more and more specialized cases to the current system, the cost/benefit ratio of this work would rapidly decrease. We'll describe the most promising avenues of extending the system with ML hybridization as we currently see them.

Semantic parsing. The two main approaches would be (a) end-to-end learning from sentences directly to extended logic as exemplified in [26], and (b) using existing LLMs or training specialized LLMs to perform simplification of sentences to the level where a hand-made semantic parser is able to convert the sentence to logic. Our initial experiments with ChatGPT have shown that using a suitable prompt causes the LLMs system to successfully split and simplify complex sentences.

Automated reasoning. Despite being optimized for large knowledge bases and performing well in reasoning competitions on such problems, our system often fails to find nontrivial proofs in reasonable time in case a large knowledge base is used. The main approaches here would be (a) learning to find a proof, based on the experience of previous proofs (see [16] for an example), (b) using machine learning along with measures of semantic relatedness of formulas to the assumption and the question (see [5]) for an example), (c) using LLMs to predict intermediate results or relevant facts and rules. A significant boost in the terms of usability could be achieved by integrating external systems like databases and scientific computing with the automated reasoners.

The knowledge base. Publicly available knowledge bases do not focus on formalizing a basic world model, arguably critical for common-sense reasoning. It is possible that a core part needs to be built by hand. On the other hand, the existing knowledge bases along with large text corpuses can be extended by creating crucial new uncertain rules using both simpler statistical methods and more complex ML techniques: see [7] for a review.

7 Summary and Future Work

We have described an implementation of a full natural language inference and question answering pipeline built around an extended first order reasoner. The system is capable of understanding relatively simple sentences and giving reasonable answers to questions, including the types currently out of scope of the capabilities of LLMs. We plan to enhance the capabilities of the system by incorporating machine learning techniques to the components of pipeline, while keeping the overall architecture, including the semantic parser, word knowledge and a reasoner.

References

1. Abzianidze, L.: Solving textual entailment with the theorem prover for natural language. Applied Mathematics and Informatics **25**(2), 1–15 (2020), http://www.viam.science.tsu.ge/Ami/2020_2/8_Lasha.pdf
2. Abzianidze, L., Kogkalidis, K.: A logic-based framework for natural language inference in dutch. CoRR abs/2110.03323 (2021), https://arxiv.org/abs/2110.03323
3. Arnaout, H., Razniewski, S., Weikum, G., Pan, J.Z.: Uncommonsense: Informative negative knowledge about everyday concepts. In: Hasan, M.A., Xiong, L. (eds.) Proceedings of the 31st ACM International Conference on Information & Knowledge Management, Atlanta, GA, USA, October 17-21, 2022. pp. 37–46. ACM (2022). https://doi.org/10.1145/3511808.3557484, https://doi.org/10.1145/3511808.3557484
4. Basu, K., Varanasi, S.C., Shakerin, F., Gupta, G.: Square: Semantics-based question answering and reasoning engine. CoRR abs/2009.09158 (2020), https://arxiv.org/abs/2009.10239
5. Furbach, U., Krämer, T., Schon, C.: Names are not just sound and smoke: Word embeddings for axiom selection. In: Fontaine, P. (ed.) Proc. of CADE'2019 – the 27th Intl. Conf. on Automated Deduction. LNCS, vol. 11716, pp. 250–268. Springer (2019)
6. Furbach, U., Glöckner, I., Pelzer, B.: An application of automated reasoning in natural language question answering. Ai Communications **23**(2-3), 241–265 (2010)
7. Han, X., Gao, T., Lin, Y., Peng, H., Yang, Y., Xiao, C., Liu, Z., Li, P., Zhou, J., Sun, M.: More data, more relations, more context and more openness: A review and outlook for relation extraction. In: Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing. pp. 745–758 (2020)
8. Hwang, J.D., Bhagavatula, C., Le Bras, R., Da, J., Sakaguchi, K., Bosselut, A., Choi, Y.: (comet-) atomic 2020: On symbolic and neural commonsense knowledge graphs. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 35, pp. 6384–6392 (2021)
9. Järv, P., Tammet, T., Verrev, M., Draheim., D.: Knowledge integration for commonsense reasoning with default logic. In: Proceedings of the 14th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - KEOD. pp. 148–155. INSTICC, SciTePress (2022). https://doi.org/10.5220/0011532200003335
10. Kalyanpur, A., Breloff, T., Ferrucci, D.A., Lally, A., Jantos, J.: Braid: Weaving symbolic and statistical knowledge into coherent logical explanations. CoRR abs/2011.13354 (2020), https://arxiv.org/abs/2011.13354
11. McCoy, T., Pavlick, E., Linzen, T.: Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. pp. 3428–3448. Association for Computational Linguistics (2019)
12. Mialon, G., Dessì, R., Lomeli, M., Nalmpantis, C., Pasunuru, R., Raileanu, R., Rozière, B., Schick, T., Dwivedi-Yu, J., Celikyilmaz, A., Grave, E., LeCun, Y., Scialom, T.: Augmented language models: a survey. CoRR abs/2302.07842 (2023), https://arxiv.org/abs/2302.07842
13. Mishra, B.D., Tandon, N., Clark, P.: Domain-targeted, high precision knowledge extraction. Trans. Assoc. Comput. Linguistics **5**, 233–246 (2017). https://doi.org/10.1162/tacl_a_00058, https://doi.org/10.1162/tacl_a_00058

14. Nguyen, T.P., Razniewski, S., Romero, J., Weikum, G.: Refined commonsense knowledge from large-scale web contents. IEEE Transactions on Knowledge and Data Engineering (2022). https://doi.org/10.1109/TKDE.2022.3206505
15. Pendharkar, D., Basu, K., Shakerin, F., Gupta, G.: An asp-based approach to answering natural language questions for texts. Theory and Practice of Logic Programming **22**(3), 419–443 (2022), https://arxiv.org/abs/2009.10239
16. Piepenbrock, J., Heskes, T., Janota, M., Urban, J.: Guiding an automated theorem prover with neural rewriting. In: Automated Reasoning: 11th International Joint Conference, IJCAR 2022, Haifa, Israel, August 8–10, 2022, Proceedings. pp. 597–617. Springer (2022)
17. Rajasekharan, A., Zeng, Y., Padalkar, P., Gupta, G.: Reliable natural language understanding with large language models and answer set programming. CoRR abs/2302.03780 (2023), https://arxiv.org/abs/2302.03780
18. Ramachandran, D., Reagan, P., Goolsbey, K.: First-orderized researchcyc: Expressivity and efficiency in a common-sense ontology. In: AAAI workshop on contexts and ontologies: theory, practice and applications. pp. 33–40 (2005)
19. Romero, J., Razniewski, S., Pal, K., Pan, J.Z., Sakhadeo, A., Weikum, G.: Commonsense properties from query logs and question answering forums. In: Zhu, W., Tao, D., Cheng, X., Cui, P., Rundensteiner, E.A., Carmel, D., He, Q., Yu, J.X. (eds.) Proc. of CIKM'19 – the 28th ACM Intl. Conf. on Information and Knowledge Management. pp. 1411–1420. ACM (2019)
20. Speer, R., Chin, J., Havasi, C.: ConceptNet 5.5: An open multilingual graph of general knowledge. In: Singh, S.P., Markovitch, S. (eds.) Proc. of AAAI'2017 – the 31st AAAI Conf. on Artificial Intelligence. pp. 4444–4451. AAAI (2017)
21. Tammet, T.: GKC: A reasoning system for large knowledge bases. In: Fontaine, P. (ed.) Proc. of CADE'2019 – the 27th Intl. Conf. on Automated Deduction. LNCS, vol. 11716, pp. 538–549. Springer (2019)
22. Tammet, T., Draheim, D., Järv, P.: Gk: Implementing full first order default logic for commonsense reasoning (system description). In: Blanchette, J., Kovács, L., Pattinson, D. (eds.) IJCAR 2022: Automated Reasoning. LNCS, vol. 13385, pp. 300–309. Springer (2022)
23. Tammet, T., Järv, P., Draheim, D.: Confidences for commonsense reasoning. In: Platzer A., S.G. (ed.) Automated Deduction – CADE 28. CADE 2021. LNCS, vol. 12699, pp. 507–524. Springer (2021)
24. Tammet, T., Sutcliffe, G.: Combining json-ld with first order logic. In: 2021 IEEE 15th International Conference on Semantic Computing (ICSC). pp. 256–261. IEEE (2021)
25. Tandon, N., de Melo, G., Weikum, G.: Webchild 2.0 : Fine-grained commonsense knowledge distillation. In: Bansal, M., Ji, H. (eds.) Proceedings of ACL 2017, System Demonstrations. pp. 115–120. Association for Computational Linguistics (2017). https://doi.org/10.18653/v1/P17-4020
26. Wang, C., Bos, J.: Comparing neural meaning-to-text approaches for dutch. Computational Linguistics in the Netherlands **12**, 269–286 (2022)
27. West, P., Bhagavatula, C., Hessel, J., Hwang, J.D., Jiang, L., Bras, R.L., Lu, X., Welleck, S., Choi, Y.: Symbolic knowledge distillation: from general language models to commonsense models. CoRR **abs/2110.07178** (2021), https://arxiv.org/abs/2110.07178