Argument analysis using Answer Set Programming and Semantics-guided Large Language Models

Abhiramon Rajasekharan[1], Yankai Zeng[1] and Gopal Gupta[1]

[1]*Department of Computer Science*
The University of Texas at Dallas
Richardson, Texas 75080, USA

Abstract

Argumentation is used to make important decisions based on the information available. We tackle the question of whether it is possible to automatically extract argumentation from a given set of documents to justify a given claim. We propose a framework that (i) extracts an argumentation structure from a given set of documents using Large Language Models (LLMs), (ii) represents it as an answer set program, (iii) and then uses the answer set program to prove the claim. We use a semantics-guided approach that leverages the FrameNet lexical database to generate sub-claims that are anchored to the concepts related to the claim. We demonstrate the efficacy of our method with an example.

1. Introduction

In *argumentation*, *premises* (i.e. *evidence* or *assumptions*) along with *arguments* allow us to prove asserted *claims* [1, 2, 3, 4]. Arguments help connect premises to the claim. Justifying claims is fundamental to philosophy, rhetoric, and sciences. Argumentation is also crucial for decision-making in areas such as systems assurance. Recent progress in AI raises the question: how well can we automate such decision-making by analyzing argumentation?

Argumentation has been an active area of research in many research communities, including Natural Language Processing (NLP), knowledge representation, and system assurance. Previous works on argument mining [2] and argument generation [5] typically focus on extracting or generating argument structures from documents where an argument has a claim and several premises. Each premise can be linked together in different ways to ultimately support or attack the claim. A common method that has been used in prior research to produce premises has been by searching through supporting documents. However, such a search for information is not motivated by any specific direction. Hence, the arguments tend to be only as strong as the evidence found. The domain for arguments considered also tended to be limited in previous research since there was a need to procure and train the machine learning models on enough data. Recent advances in LLMs provide a unique opportunity to develop better solutions to these problems.

Argumentation structures can potentially be complex [2]. Given a claim, several premises

3rd Workshop on Goal-directed Execution of Answer Set Programs (GDE'23), July 10, 2023
abhiramon.rajasekharan@utdallas.edu (A. Rajasekharan); yankai.zeng@utdallas.edu (Y. Zeng); gupta@utdallas.edu (G. Gupta)
© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
CEUR Workshop Proceedings (CEUR-WS.org)

may support or attack it. Each of these premises may be either supported or attacked by other premises, in turn. The relations between these premises and claims might be of several types (linked, divergent, convergent, etc.). This chain of reasoning may extend to multiple levels and the entire argumentation structure needs to be processed accurately to determine if the claim holds.

Let's consider how a human expert would go about analyzing the argumentation in a report. The person would first extract the main claim in the report and think of ways or ideas that can make the claim either true or false (false meaning erroneous). Based on these ideas, the expert would then go through supporting documents to search for evidence, counter-evidence, inferences, assumptions, etc. Based on this exploration, he/she might go through deeper levels of similar examination. The expert finally reasons using all the information found to assess if the claim holds.

Inspired by this observation, we propose a methodology for automated argument analysis. Given a report and its supporting documents, our approach extracts the argumentation structure using a semantics-guided approach. We employ Large Language Models (LLM) [6] for sub-claim generation and evidence extraction. We anchor the sub-claims generated to the relevant concepts using frame elements extracted from FrameNet [7]. This ensures that the LLM covers all the different aspects related to the claim. The resulting argumentation structure is encoded as an answer set program [8] that can be processed to reliably return the truth value of the claim in the argumentation. We demonstrate our method using a well-known controversy: 'Culpability of Russia in the downing of MH17 Malaysian Airlines flight'.

2. Background

2.1. Argumentation Structure

As mentioned earlier, argumentation research spans many areas. Different areas use different terminology. In this paper, we follow the terminology used in the Claim-Assurance-Evidence (CAE) framework as realized in Bloomfield and Rusbhy's Assurance 2.0 methodology [4]: a *claim* is a conclusion we wish to draw given *arguments* and *premises*. Arguments correspond to *rules* (conditional truths) and premises correspond to *facts* (unconditional truths) or assumptions. Counter evidence is called a *defeater*. A defeater can also be thought of as an *exception* to a *default* [9, 10]. An argument decomposes a claim into a series of *subclaims* and *defeaters*. Note that a subclaim or a defeater may have its own underlying logic similar to a claim. The argumentation terminology of Assurance 2.0 has close correspondence to answer set programming [8]: a claim corresponds to a query goal, an argument to a rule, premise to a fact or an *abducible* (assumption) [10], and a defeater to a negation-as-failure goal.

2.2. Large Language Models

Until recently, deep learning models have been applied to NLP tasks by fine-tuning them on task-specific datasets [11]. This fine-tuning process involves providing a model with a large number of input-output examples so that it learns to generate the correct output when given a new input at test time. In 2020, Brown et al. introduced a Large Language Model (LLM)

called GPT-3 [6] with approximately 175 billion parameters and trained on a massive corpus of filtered text from the internet. During training, the model learns to predict the next token given a sequence of input tokens. In order to perform this task correctly, it needs to learn the underlying language, and also the commonsense knowledge involved. This arms the model to generate human-like text. With the advent of Large Language Models, the training paradigm changed to teaching a language model *any arbitrary task* using just a few demonstrations, called *in-context learning*. GPT-3 is such an LLM that is able to perform competitively on several tasks such as question-answering, semantic parsing [12], and machine translation.

When further trained using reinforcement learning to align model outputs to human preferences, such models are able to follow instructions and perform natural language tasks even without any demonstrations [13]. This is the basis for the well-known chatbot, ChatGPT [14].

2.3. FrameNet

FrameNet is a lexical database of English that focuses on certain semantic concepts called 'frames' and the corresponding words (lexical units) and semantic roles (frame elements) involved. For example, the verb 'arrest' has two frames: *Arrest* and *Cause_to_end*. The $Arrest$ frame has elements such as *Suspect* (the person taken into custody), *Authorities* (the ones taking the suspect into custody), *Charge* (the category in the legal system under which the person is charged), and *Offense* (the reason for which the suspect is arrested). Hence, knowing the frames a sentence uses, helps to understand and connect the different concepts involved. Each frame element also contains an annotated sentence describing how it is used. In total, FrameNet contains 1,200 semantic frames, 13,000 word senses, and around 200,000 manually annotated sentences [7].

In our work, we use FrameNet to gather all the concepts in a sentence. The frame elements of the lexical units in a sentence tell us what aspects of the event we should look for. For example from the frame 'shoot down' that is extracted from the sentence 'Russia is responsible for shooting down the flight MH17', we can gather the frame elements 'Agent', 'Patient', 'Instrument', etc. Using these elements we can direct the search to look for evidence linking the group that launched the attack with Russia (Agent), evidence to show that the flight was indeed MH17 (Patient) and also that the missile used in the attack can be linked to Russia (Instrument).

2.4. Answer Set Programming

ASP is a logic-based paradigm for knowledge representation and reasoning. Answer set programming (ASP) [8, 9], a paradigm based on logic programming, incorporates reasoning with incomplete information, assumption-based reasoning, as well as integrity constraints. ASP extends logic programming with negation-as-failure based on the stable model semantics [15]. Negation-as-failure (NAF) allows ASP to perform non-monotonic reasoning. Non-monotonic reasoning is essential for modeling commonsense reasoning since as our knowledge changes, i.e., as more things become known from being unknown, the conclusions we can draw may change. For example, a conclusion that could have been drawn earlier may have to be retracted. ASP is a highly expressive paradigm that can elegantly express complex reasoning methods used by humans such as default reasoning, deductive and abductive reasoning, counterfactual reasoning, and constraint satisfaction [16, 9]. As mentioned earlier, there is a one-to-one mapping

between argumentation structures and ASP: a claim corresponds to a query, a premise to a fact or assumption, an argument to an ASP rule, and a defeater to a NAF goal. Assumption-based arguments can also be represented in ASP since ASP naturally supports abductive reasoning.

We use the s(CASP) goal-directed ASP engine [17] for executing answer set programs. The justification tree [18] generated by s(CASP) is crucial to our efforts.

3. Related Work

There is significant prior research in mining arguments in text data [2]. It has been applied to domains ranging from student essays [19] to law [20] documents. The typical method to extract the argumentation structure employed is to identify the text span corresponding to the argument components and then link them together using a classifier that determines if each pair of these components are related and by which type of relation (support/attack). Our method differs entirely since we use a semantics-guided approach for generating sub-claims. The next step used to extract evidence using the sub-claims is also different since we use an LLM that is able to identify complex premises that may not even be contiguous in the document.

The line of research in generating arguments is also relevant to our work. Argument generation models usually use an LSTM [5] for argument component generation which may also make use of an additional content planning model aided by external knowledge (fetched from a database such as Wikipedia). In a closer approach, Schiller et al.[21] use a pre-trained language model called CTRL to generate arguments, controlled using control codes (this set of codes is limited and learned beforehand). Our approach for sub-claim generation uses an LLM that can be applied to any domain and is capable of using instruction directly to guide generation.

Our final step of using ASP to analyze arguments has precedent. ASP-based methods have been developed to implement arbitrary argument structures [1]. Since we use semantics-guided sub-claim generation, the ASP-based argumentation structure processing we employ is much simpler.

4. Methodology and Experiments

Our methodology has primarily two components:

1. Extracting the argument structure from the documents.
2. Encode the argument structure into an ASP program.

For extracting the argument structure (step 1), we use LLMs. This extraction is guided by the semantics of the concepts involved in the claim. An LLM is then used to generate sub-claims using these concepts. Using these, we use the LLM again for extracting evidence and defeaters from supporting documents. We use the s(CASP) system to process the argumentation constructed (step 2).

4.1. Creating the Argument Structure

Our approach works as follows. We first extract a claim from the document. Then we generate sub-claims that would support a given claim. We next find supporting evidence for these sub-

claims from the documents. Similarly, we also generate defeaters (exceptions) to the sub-claims using an LLM. Given all the supporting evidence found for sub-claims, we repeat the process for a few more levels (generating sub-sub-claims, etc., and finding supporting evidence for them). Likewise, we find a defeater to the defeater using an LLM. The entire process is also repeated for the negated claim to check if the counterclaim holds. Each evidence found to support the counterclaim is essentially a defeater for the main claim. Each of these steps is explained below in more detail. We conduct experiments on articles on the topic of Malaysian Airlines flight MH17 being shot down.

4.1.1. Claim Extraction

The claim can be extracted using an LLM from the document. We simply use the prompt: *What is the single main claim of the following article?* Using this prompt, we get the following claim :

> Russia is responsible for shooting down Malaysian Air Flight MH17 since the missile debris had a Russian signature on them.

4.1.2. Sub-claim Generation

Using the claim extracted, we need to generate sub-claims. Our experiments revealed that using LLMs directly for this task is difficult to control and might miss out on important directions to generate such sub-claims. Hence, we use FrameNet, a human-curated lexical database, to anchor the sub-claim to the concepts in the claim. The steps involved in support sub-claim generation are as follows:

1. Identify the main verbs in the claim. Here we use NLTK to identify all the verbs in the claim and remove stop verbs. Stop verbs are a pre-defined set of verbs, such as 'be', 'might', 'did', etc., which we believe are too general and do not carry much information.
2. The main verbs are then lemmatized and matched with verbs available on FrameNet.
3. Each verb on FrameNet has multiple frames. Each frame denotes a different way the verb may be used. Hence we need to identify which frame is the correct frame for the verb in the claim. Using the claim sentence and the set of frames, we use an LLM to determine the correct frame. Our experiments reveal that LLMs identify the correct frame with high accuracy.
4. Once we have the frame, we have multiple frame elements that represent all the semantic roles in the frame. We use these elements along with the original claim to generate the sub-claims.

For the flight example, the main verb extracted is 'shoot down' and the frame is 'Downing'. The different frame elements involved are:

- Agent: The Agent forces the Patient to come to the ground.
- Patient: The Patient is the entity that is being forced to the ground.
- Depictive: Depictive phrase describing the actor of an action.
- Result: The Result of causing the Patient to come to the ground.

- Means: An act performed by an Agent that enables them to force the Patient to the ground.
- Place: The location at which the event takes place.
- Result: The Result of causing the Patient to come to the ground.
- Time: The time at which the event takes place.

Using these frame elements, we prompt the LLM to generate the following sub-claims below. To establish the claim, these sub-claims relating to agent, patient, depictive, etc., must be proved.

Agent: Evidence connecting Russian military or government officials to the ordering or execution of the missile strike, such as intercepted communications or witness testimony.

Patient: Evidence linking the missile debris to the downed aircraft, including forensic analysis of the wreckage and comparisons to other known missile fragments.

Depictive: Evidence of the type and model of missile used in the attack, and whether it matches any known Russian missile systems.

Result: Evidence of the damage caused by the missile strike, including autopsy reports and analysis of the debris field.

Means: Evidence of how the missile was transported to the launch site, such as satellite imagery or eyewitness accounts.

Manner: Evidence of any specific instructions or tactics used in the attack, which could point to a specific military unit or command structure.

Time_and_Place: Evidence of the exact time and location of the missile launch, including satellite imagery, radar data, and eyewitness testimony.

Cause: Evidence of the motive behind the attack, such as political tensions between Russia and Ukraine or revenge for previous incidents.

Instrument: Further analysis of the missile debris, including its manufacturing and maintenance history, to determine its origin and any connections to Russian military or government agencies.

4.1.3. Evidence Extraction for Sub-claims

Once we have the sub-claims, we simply use LLM to find if any evidence exists to support each one of them in the supporting documents. If such a match exists, we extract it. If not, the LLM returns the response that there is no close match. Testing this approach for the 'Agent' frame element sub-claim above returns the match:

> The article provides evidence connecting Russian military officials to the missile strike, such as intercepted mobile-phone conversations allegedly between two of the defendants and senior Russian officials.

Supporting evidence for other frame elements can be similarly generated.

4.1.4. Sub-claim Defeater Generation

Sub-claims can be generated for what might be defeaters or exceptions to the claim and sub-claims in a similar way. For this task, we use LLM by asking it to generate an exception that makes the claim false for a given support sub-claims. For the support sub-claim given above, we get the following exception sub-claim:

> However, the intercepted communications or witness testimony could provide evidence that suggests the involvement of higher-ups in the Russian government or military, even if the missile strike was carried out by a rogue faction or third party.

In a similar fashion, we can find evidence for the exception to the exception in the documents.

4.2. Converting Argument Structure to ASP

The argument structure is now put together as an ASP program. Each of the supporting sub-claim generated is a term in the program. The truth value of each term depends on if evidence has been found to corroborate it. After determining the truth value of each term, the program can be run to determine if the claim is true.

The general structure of the program is explained here.

$$\texttt{claim(T) :- } d_1\texttt{(}D_1\texttt{), } d_2\texttt{(}D_2\texttt{), .., } d_m\texttt{(}D_m\texttt{), not -claim(Tnot).}$$

where `claim(T)` represents the claim (`T` represents the text corresponding to the claim), each d_i represents a subclaim (D_i. We may be able to prove the counterclaim (`-claim`), in which case our positive claim cannot hold. Therefore, we must include `not -claim(Tnot)` in the body of the argument, where `Tnot` represents the text supporting the counterclaim. Next, we will recursively check the sub-claims pertaining to these frame elements. These sub-claims about the frame elements correspond to the d_i's. For each such sub-claim d_i, we will first find support for it (supporting evidence), then exceptions to it (contrary evidence), then, finally, we will find exceptions to the exceptions to it (evidence contradicting the contrary evidence). The exceptions to d_i's and exceptions to exceptions to d_i's will produce multiple pieces of additional evidence. In other words, we refine d_i's as follows:

d_i`(`D_i`) :- subclaim_frame_element`$_i$`(`D_i`).`

`subclaim_frame_element`$_i$`(`D_i`) :- subclaim_support`$_i$`(`D_i`), not exception`$_i$`(`E_i`).`

`exception`$_i$`(`E_i`) :- support_exception(`E_i`), not exception_to_exception(`X_i`).`

Note that we can think of the rule for sub-claim as a default rule [9], since `subclaim_support`$_i$`(`D_i`)` can be viewed as the default supporting-justification for the sub-claim and `support_exception`$_i$`(`E_i`)` as the exception to the default.

Finally, we determine if the counterclaim (represented as `-claim(Tnot)`) holds, where `Tnot` is the text corresponding to the counterclaim. The counterclaim is defined as:

`-claim(Tnot) :-` dc_1`,` dc_2`, ...,` dc_p`.`

The counterclaim will be established in a manner similar to the claim. For extracting the text corresponding to support for the sub-claim, the exception to the sub-claim, the exception to the exception to the sub-claim, and the counterclaim, we make use of a large language model [22].

Once all the available evidence has been found, we can run the program to determine if the claim (or the counterclaim) is true. If the claim is true, then s(CASP)'s justification facility can be used to generate the whole argumentation case for proving the claim. If the counterclaim is true, then s(CASP)'s justification facility can be similarly used to generate the whole counter-argument case. If neither the claim, nor the counterclaim is provable, then the claim is unsupported, and proof of the lack of support for the claim can be generated by generating the justification for the negated claim as well as the negated counterclaim. We show a fragment of the argument generated to prove the claim "Russia is responsible for shooting down Malaysian Airline MH17 over Ukraine." We only show the argument development for one frame element (agent) due to lack of space. The document used for generating this argument fragment consists of an Economist magazine article about the downing of Malaysian Airlines flight MH17 (Mar 8th, 2020 issue). The argument contains the supporting premise, the exception to the premise, and the exception to the exception. Arguments for other frame elements can be likewise automatically generated.

> The main action is shoot, which in this case means downing. Downing has the following semantic concepts associated with it: Agent, Patient, Depictive, Result, Means, Place, Result, Time, Cause, and Instrument.
>
> **Agent**: Is there evidence connecting Russian military or government officials to the ordering or execution of the missile strike, such as intercepted communications or witness testimony? The article provides evidence connecting Russian military officials to the missile strike, such as intercepted mobile-phone conversations allegedly between two of the defendants and senior Russian officials. It is possible that a third party or a rogue faction within the Russian military or government was responsible for shooting down the plane without the knowledge or authorization of higher-ups in the Russian government. However, the intercepted communications or witness testimony could provide evidence that suggests the involvement of higher-ups in the Russian government or military, even if the missile strike was carried out by a rogue faction or third party.

5. Conclusion

In this paper, we propose a method to automatically prove claims where the claim and the arguments for proving the claim are embedded in natural language documents. Our approach uses LLMs to extract both the claim and the argument structure from these documents. We use a semantics-guided process to achieve this goal. Once the argument structure is extracted, it is represented using ASP. The ASP code can then be run under the s(CASP) system to generate the whole proof (in a natural language) for establishing the claim. We demonstrate our method using an example related to the downing of flight MH17 over Ukraine. Our experiment shows that the approach works reasonably well for such problems. Our future plan is to apply our

method to more complex examples, as well as develop a tool to automatically generate the argumentation case for a claim, if it is proved or, for the counterclaim if it is disproved.

References

[1] G. Charwat, W. Dvořák, S. A. Gaggl, J. P. Wallner, S. Woltran, Methods for solving reasoning problems in abstract argumentation – a survey, Artificial Intelligence 220 (2015) 28–63. URL: https://www.sciencedirect.com/science/article/pii/S0004370214001404. doi:https://doi.org/10.1016/j.artint.2014.11.008.

[2] J. Lawrence, C. Reed, Argument mining: A survey, Computational Linguistics 45 (2019) 765–818. URL: https://aclanthology.org/J19-4006. doi:10.1162/coli_a_00364.

[3] C. M. Holloway, The friendly argument notation, 2020. URL: https://ntrs.nasa.gov/citations/20205002931.

[4] R. Bloomfield, J. Rushby, Assurance 2.0: A manifesto, 2020. ArXiv:2004.10474v2.

[5] X. Hua, Z. Hu, L. Wang, Argument generation with retrieval, planning, and realization, in: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Florence, Italy, 2019, pp. 2661–2672. URL: https://aclanthology.org/P19-1255. doi:10.18653/v1/P19-1255.

[6] T. Brown, B. Mann, Others, Language models are few-shot learners, in: NeurIPS, volume 33, Curran Associates, Inc., 2020, pp. 1877–1901. URL: https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

[7] C. F. Baker, C. J. Fillmore, J. B. Lowe, The Berkeley FrameNet project, in: 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume 1, Association for Computational Linguistics, Montreal, Quebec, Canada, 1998, pp. 86–90. URL: https://aclanthology.org/P98-1013. doi:10.3115/980845.980860.

[8] G. Brewka, T. Eiter, M. Truszczynski, Answer set programming at a glance, Commun. ACM 54 (2011) 92–103.

[9] M. Gelfond, Y. Kahl, Knowledge Representation, Reasoning, and the Design of Intelligent Agents: An Answer Set Programming Approach, Cambridge Univ. Press, 2014.

[10] G. Gupta, Automating common sense reasoning with ASP and s(CASP), 2022. Technical Report, https://utdallas.edu/~gupta/csr-scasp.pdf.

[11] S. Casola, I. Lauriola, A. Lavelli, Pre-trained transformers: an empirical comparison, 2022. URL: https://www.sciencedirect.com/science/article/pii/S2666827022000445. doi:https://doi.org/10.1016/j.mlwa.2022.100334.

[12] R. Shin, B. Van Durme, Few-shot semantic parsing with language models trained on code, in: Proc. ACL-HLT, 2022, pp. 5417–5425.

[13] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, R. Lowe, Training language models to follow instructions with human feedback, 2022. arXiv:2203.02155.

[14] OpenAI, Optimizing language models for dialog, 2022. https://openai.com/blog/chatgpt/.

[15] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming., in: ICLP/SLP, volume 88, 1988, pp. 1070–1080.
[16] C. Baral, Knowledge representation, reasoning and declarative problem solving, Cambridge University Press, 2003.
[17] J. Arias, M. Carro, E. Salazar, K. Marple, G. Gupta, Constraint Answer Set Programming without Grounding, TPLP 18 (2018) 337–354.
[18] J. Arias, M. Carro, Z. Chen, G. Gupta, Justifications for goal-directed constraint answer set programming, in: Proceedings 36th International Conference on Logic Programming (Technical Communications), volume 325 of *EPTCS*, 2020, pp. 59–72.
[19] C. Stab, I. Gurevych, Parsing argumentation structures in persuasive essays, Computational Linguistics 43 (2017) 619–659. URL: https://aclanthology.org/J17-3005. doi:10.1162/COLI_a_00295.
[20] P. Poudyal, J. Savelka, A. Ieven, M. F. Moens, T. Goncalves, P. Quaresma, ECHR: Legal corpus for argument mining, in: Proceedings of the 7th Workshop on Argument Mining, Association for Computational Linguistics, Online, 2020, pp. 67–75. URL: https://aclanthology.org/2020.argmining-1.8.
[21] B. Schiller, J. Daxenberger, I. Gurevych, Aspect-controlled neural argument generation, in: Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Association for Computational Linguistics, Online, 2021, pp. 380–396. URL: https://aclanthology.org/2021.naacl-main.34. doi:10.18653/v1/2021.naacl-main.34.
[22] H. W. Chung, L. Hou, S. Longpre, B. Zoph, Y. Tay, W. Fedus, E. Li, X. Wang, M. Dehghani, S. Brahma, et al., Scaling instruction-finetuned language models, arXiv preprint arXiv:2210.11416 (2022).