Visualizing HS-Tree-Based Abductive Reasoning: An Educational Tool for Algorithm Exploration

Michaela Tóthová[1], Janka Boborová[1], Júlia Pukancová[1], Jakub Kloc[2] and Martin Homola[1]

[1]Comenius University in Bratislava, Mlynská dolina, 842 41 Bratislava, Slovakia
[2]SOFTEC, Einsteinova 33, 851 01 Bratislava, Slovakia

Abstract

We present the HS-Tree Visualizer application, a visualization tool under development that currently supports displaying MHS and MHS-MXP trees from input JSON files produced by an abduction solver. The application supports step-by-step simulation of the algorithms, interactive manipulation of tree nodes and edges, and the ability to display or hide additional details, including branches that do not lead to explanations. These features make it a useful tool for learning, algorithm analysis, and debugging the underlying solver. In the future, the application will be extended to support additional abductive algorithms, and it also has the potential to be used by other abduction solvers. With further updates, it could even be adapted for other processes that can be represented using a tree/graph structure, as long as a suitable JSON file in the required format is provided.

Keywords

ABox abduction, abductive reasoning, description logics, visualization, educational tool

1. Introduction

Visualization is a useful tool that plays a key role in the educational process, as well as in algorithm analysis and debugging. One example is EVONNE [1], a tool focused on explaining the derivation of logical consequences in ontologies. In addition to simply providing an answer, it enables interactive proof visualization, step-by-step tracing of the reasoning process, debugging of unwanted consequences, and analysis of possible ontology repairs. Other examples of visualization tools include SOVA [2], a Protégé plugin designed for graphical visualization of ontology structure, and SIVA [3], an educational tool for simulating the tableau reasoning algorithm, which allows step-by-step construction of the completion tree and backtracking to any previous state. Another notable tool is EVEE [4, 5], a Java library and a collection of Protégé plugins that, in addition to explaining inferred entailments using proofs, also supports the explanation of missing entailments via counterexamples and abduction [6]. Counterexamples are presented as interactive graph-based visualizations of ontology models violating the expected entailment, with support for zooming, dragging, highlighting relevant elements, and dynamic updates. Abduction provides hypothetical explanations in the form of axioms that, when added to the ontology, would make the missing entailment hold. EVEE presents these hypotheses as a list of possible ontology repairs, which can be further examined and justified using proof-based methods.

Although EVEE employs abductive reasoning, it only presents the resulting hypotheses and does not provide insight into the process by which they are computed. For educational purposes, algorithm analysis, and debugging, it would be highly beneficial to have a tool that, similarly to EVONNE for logical consequence proofs, visualizes the step-by-step execution of an abduction algorithm. To the best of our knowledge, no such visualization tool currently exists for description logic (DL) abduction. This motivates our work, in which we aim to visualize the internal reasoning process of our ABox abduction solver CATS [7], providing users with insight into how hypothetical explanations are constructed.

DL 2026: 39th International Workshop on Description Logics, July 17–19, 2026, Lisbon, Portugal
tothova424@uniba.sk (M. Tóthová); boborova@fmph.uniba.sk (J. Boborová); pukancova@fmph.uniba.sk (J. Pukancová); kloc4@uniba.sk (J. Kloc); homola@fmph.uniba.sk (M. Homola)
https://dai.fmph.uniba.sk/~pukancova/ (J. Pukancová); https://dai.fmph.uniba.sk/~homola/ (M. Homola)
0009-0002-7487-9962 (J. Boborová); 0009-0001-3505-0716 (J. Pukancová); 0000-0001-6384-9771 (M. Homola)

We present a web-based application called *HS-Tree Visualizer*. It operates on a JSON representation of the computation produced by the solver and allows users to explore both the resulting HS-tree and the step-by-step execution of the algorithm. To support this, we have extended the CATS solver [7] to generate compatible JSON files for the MHS [8] and MHS-MXP [9] algorithms. These files capture the final tree structure as well as detailed information about intermediate steps. The main features of the tool include: (1) *interactive exploration:* zooming, node manipulation, and collapsing sub-trees or hiding pruned nodes to manage large structures; (2) *step-by-step simulation:* showing the algorithm's progress and how the solver explores the search space and prunes branches; (3) *visual feedback:* highlighting successful explanations and providing an information panel with details on nodes, axioms, and the underlying abduction problem.

The primary goal of the *HS-Tree Visualizer* is to improve the teaching of HS-tree-based abduction methods and to enable a more detailed and visual analysis and comparison of different abduction algorithms. By providing a clear view of the solver's steps, the tool also facilitates debugging of the underlying solvers. In the future, we plan to extend the application to support additional algorithms implemented in CATS and explore its potential for visualizing other tree-based structures, such as completion trees used in tableau-based reasoning.

2. Preliminaries

We assume the standard notions of a DL knowledge base $\mathcal{K} = \mathcal{T} \cup \mathcal{R} \cup \mathcal{A}$, where $\mathcal{T}$, $\mathcal{R}$, and $\mathcal{A}$ denote a TBox, RBox, and ABox, respectively, as well as the usual notions of interpretation, satisfaction, *consistency* and *models* of a knowledge base, and *entailment* ($\models$), for any DL up to $\mathcal{SROIQ}$ [10].

Definition 1 (ABox abduction problem [11, 12])**.** *An ABox Abduction problem is $\mathcal{P} = (\mathcal{K}, \mathcal{O}, Abd)$, where $\mathcal{K}$ is a DL knowledge base and $\mathcal{O}$ and Abd are finite sets of ABox assertions called* observations *and* abducibles*, respectively. An* explanation *for $\mathcal{P}$ is a finite set of ABox assertions $\mathcal{E} \subseteq Abd$ such that $\mathcal{K} \cup \mathcal{E} \models \mathcal{O}$.*

To prevent an infinite search space of explanations, abducibles should be meaningfully restricted. We restrict them to atomic and negated atomic concept assertions. Additionally, we are interested in explanations with suitable properties, referred to as *desired explanations* [12]: consistency ($\mathcal{E} \cup \mathcal{K}$ is consistent), relevance ($\mathcal{E} \not\models O_i \mid O_i \in \mathcal{O}$), being explanatory ($\mathcal{K} \not\models \mathcal{O}$), and minimality (there is no other explanation $\mathcal{E}'$ such that $\mathcal{E}' \subseteq \mathcal{E}$).

The MHS algorithm [8], which computes minimal hitting sets of a given collection of sets (Definition 2), can also be applied in our setting. Consider the collection $\{\mathcal{C}_{\mathcal{M}} \mid \mathcal{M} \models \mathcal{K} \cup \{\neg O\}\}$, where $\mathcal{C}_{\mathcal{M}} = \{\alpha \in Abd \mid \mathcal{M} \not\models \alpha\}$. $\mathcal{O}$ is either a single ABox assertion, in which case $\mathcal{O} = \{O\}$, or $\mathcal{O}$ is a set of ABox assertions. In the latter case, O is a single ABox assertion obtained from $\mathcal{O}$ by reduction [12]. In particular, the reduced ABox assertion is constructed as an assertion involving a new individual assigned to a complex concept formed from the concepts, roles, and individuals occurring in the original $\mathcal{O}$. Then, the minimal hitting sets of this collection correspond to the minimal explanations for $\mathcal{P} = (\mathcal{K}, \mathcal{O}, Abd)$ [8, 12]. Any remaining undesired explanations can be filtered out additionally. The algorithm constructs a structure called an HS-tree (Definition 3), from which all minimal hitting sets can be obtained as the sets of edge labels along paths from the root to leaf nodes labelled with ✓.

Definition 2 (Minimal hitting set)**.** *Let F be a collection of sets. A set H is called a* hitting set *for F if $H \cap S \neq \emptyset$ for every $S \in F$. It is* minimal *if no proper subset of H is a hitting set for F.*

Definition 3 (HS-tree for MHS with pruning)**.** *An HS-tree for a collection of sets F is a triple $T = (V, E, \mathcal{L})$, where (V, E) is a tree and $\mathcal{L}$ is a labelling function assigning elements of F to the nodes in V and elements of sets in F to the edges in E. For each node $n \in V$, let $H(n)$ denote the set of edge labels on the path from the root to n. (V, E) is a minimal tree constructed in breadth-first order with the following properties:*

- *For the root $r \in V$, $\mathcal{L}(r) = S$ for some $S \in F$ if $F \neq \emptyset$; otherwise $\mathcal{L}(r) =$ ✓.*

- *For each node* $n \in V$*:*
  - *either* $\mathscr{L}(n) = \times$ *(*n *is pruned*[1]*) if there is another node* n' *such that*
    1. $H(n') \subseteq H(n)$ *and* $\mathscr{L}(n') = \checkmark$*, or*
    2. $H(n') = H(n)$ *and* $\mathscr{L}(n') = S \in F$*;*
  - *or* $\mathscr{L}(n) = S$ *for some* $S \in F$ *such that* $S \cap H(n) = \emptyset$ *if such* S *exists;*
  - *otherwise* $\mathscr{L}(n) = \checkmark$*.*
- *Each* $n \in V$ *with* $\mathscr{L}(n) \in F$ *has a successor* n_σ *for each* $\sigma \in \mathscr{L}(n)$ *such that* $\mathscr{L}(n, n_\sigma) = \sigma$*.*

MHS-MXP [9] is an algorithm for ABox abduction that employs a hybrid approach combining MHS with the MergeXplain method (MXP) [14]. MXP is based on a divide-and-conquer strategy for computing a set of minimal explanations. It prioritizes efficiency over completeness; therefore, it does not guarantee finding all minimal explanations, but it typically terminates faster. The idea behind the hybrid approach is to improve the efficiency of the minimal hitting set algorithm. In particular, it aims to speed up the computation of explanations and enables earlier termination.

3. HS-Tree visualizer application

We present an interactive tool for visualizing HS-trees that are produced by abductive reasoning algorithms. The purpose of the application is to provide a clearer view of both the structure of the generated tree and the process by which it is constructed. The final output of abduction solvers, such as CATS [7], is typically in textual form and does not clearly show the execution of the algorithm. The visualizer offers a graphical and interactive alternative.

3.1. Implementation

The application is implemented as a web-based tool using JavaScript, with the Cytoscape.js library [15] used for rendering and manipulating graph structures. The HS-tree is displayed as a directed graph where each node represents a partial solution determined by the hypotheses selected along the path from the root, and is labelled with a set of candidate hypotheses. Edges are labelled with elements of the underlying sets used in the hitting set computation.

The layout of the graph is automatically computed using hierarchical layout algorithms provided by Cytoscape.js. For smaller trees, the *Dagre* layout [16] is used, as it arranges nodes into layers and places child nodes beneath their parent nodes, which produces a clear tree structure and improves its readability.

However, for larger trees, this approach often leads to very wide layouts due to the increasing number of nodes at each level. To address this, the *ELK (Eclipse Layout Kernel)* [17] layout is used for larger trees, as it produces more compact layouts that are better suited for larger structures. Visualizing large HS-trees introduces additional challenges, including visual clutter, increased width of hierarchical layouts, and limited readability when the structure exceeds the available screen space [18].

To mitigate these issues, the implementation combines the use of different layout algorithms with additional techniques such as depth limitation, subtree collapsing, and filtering of selected elements, which reduce visual complexity and improve navigation.

The visualization is designed to operate on data exported from abductive solvers. To obtain the data, we use the event system provided by the CATS solver, which reports selected steps of the algorithm execution. During solving, events are published for key operations of the tree construction process. We implement a custom JSON export subscriber within CATS that listens to these events and records them into a structured format. This approach allows us to capture both the structure of the HS-tree and the order in which it is constructed.

[1] Although Reiter introduces three pruning conditions, combining all of them may lead to the loss of explanations due to a known bug [13]. To preserve completeness, we use only the first two.

The application takes as input a JSON representation of the HS-tree. This representation captures the main structure of the tree while also capturing additional information about the order of execution of the algorithm. Each node contains structural attributes such as identifiers, depth, labels, and path information. Every edge contains labels corresponding to applied axioms and references to the parent and child nodes. The JSON file may also include information about the abduction problem itself, such as the ontology, in particular its TBox, and the observation.

Including event information in the JSON data is a key component of the system, enabling reconstruction of the algorithm's execution. Nodes are annotated with main events such as creation, processing, closing, and detection of explanations. Edges contain similar information, including their creation as well as events related to pruning with their corresponding reasons. Each event is associated with a step number that reflects the order in which the algorithm performs individual operations.

The JSON data is first converted into graph elements used by the rendering engine. During this step, additional attributes are assigned to nodes and edges, including visibility conditions, explanation status, and pruning information. These attributes are then used to update the graph when representing different stages of the computation.

The implementation also supports filtering based on structural and semantic properties of the tree, such as limiting the displayed depth, hiding pruned branches, or restricting the view to explanation paths. These operations are applied directly to the graph representation, which allows the system to handle larger trees efficiently.

The JSON representation serves as a general interface between the solver and the visualizer, allowing different systems to provide compatible input data.

3.2. Application Usage

The application is designed to be used together with abductive reasoning solvers that export HS-trees in a predefined JSON format. The user begins by loading such a file, after which the corresponding HS-tree is automatically rendered as an interactive graph. To assist users with the input format, the interface provides a *Help* button that links to the project documentation (README[2]), which contains a detailed description of the expected JSON file structure.

The visualization is fully interactive and allows users to explore the structure of the generated HS-tree. Users can navigate the graph using zooming (via mouse wheel or control buttons), panning, and automatic centering of the canvas. Individual nodes can be expanded or collapsed to focus on relevant parts of the tree.

Nodes are labelled with sets $\mathscr{C}_{\mathscr{M}}$ as defined in Section 2, while the path from the root to a given node determine the corresponding candidate explanation. Pruned branches are visually distinguished by edges that point to special nodes highlighted in red. These elements represent pruning events, and the corresponding information is available in the node information panel. The nodes corresponding to successful explanations are highlighted in green, and the path from the root to this node represents a computed minimal explanation.

The visualizer provides two modes of interaction. In the *Tree mode*, the entire HS-tree is displayed at once, allowing the user to explore its structure. The complexity of the visualization can be reduced by collapsing subtrees of selected nodes or by limiting the maximum depth of the displayed tree. It is also possible to hide selected elements, such as pruned branches, or to remove labels from nodes. These features help reduce visual complexity, especially for larger trees, and are available through the top toolbar.

In the *Step mode*, the tree is constructed incrementally according to the recorded sequence of events. At each step, the user can observe how the algorithm progresses, including node creation, processing, pruning of branches, and detection of explanations. This mode makes it possible to follow the exploration of the search space and better understand how individual explanations are derived.

The interface also supports interactive tree inspection. Using the *Info* button, a collapsible panel is displayed, providing details about selected nodes as well as information about the underlying abduction

[2] https://github.com/m1chaelaT/HSTreeVisualizer/blob/main/README.md

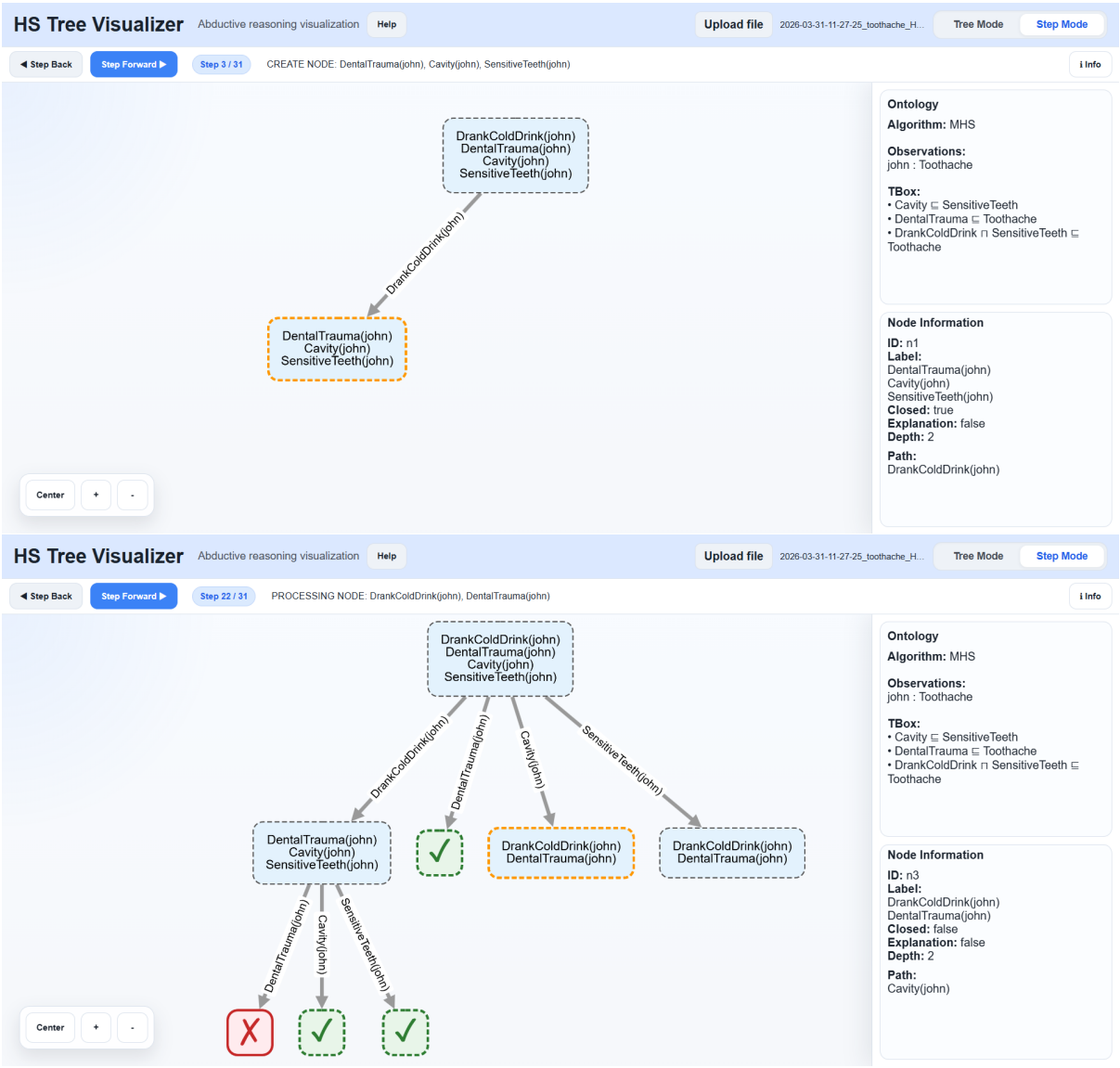

Figure 1: Step-by-step visualization of the HS-tree construction. Top: early stage of the algorithm (step 3). Bottom: later stage (step 22) showing a more developed tree with pruning and identified explanations.

problem, including the observation and the ontology. By selecting a node, the user can view its properties, such as its label, depth, and associated path.

The visualizer is useful in several scenarios. In education, it helps students follow how HS-tree based algorithms progress, as the step-based view makes individual steps of the computation visible. It can also support debugging and analysis, for example by revealing unnecessary branching or incorrect pruning that are difficult to detect in textual outputs. The tool is currently used with the CATS solver [7], which provides the required JSON export, but it is not limited to this system and can be used with any solver that produces data in the specified format.

The application is publicly available as an open-source project on GitHub[3]. A live web version of the tool can be accessed via GitHub Pages[4].

[3]Source code: https://github.com/m1chaelaT/HSTreeVisualizer
[4]Live demo: https://m1chaelat.github.io/HSTreeVisualizer/

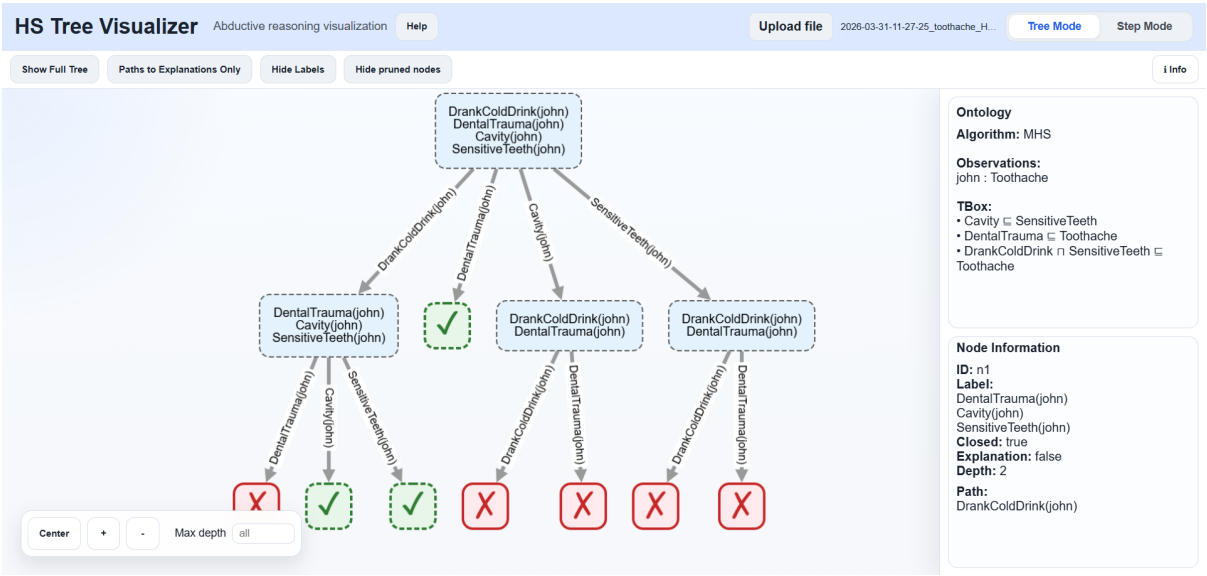

Figure 2: Overview of the HS-Tree Visualizer application showing a generated tree for an example input.

4. Potential Other Uses

The implemention can possibly be used to visualize similar tree-shaped data structures produced by other algorithms. The closest and most directly related problem is the visualization of completion trees (CTrees) generated by tableau-based algorithms (TA) for DL [19]. This is particularly relevant as the CATS solver itself utilizes DL reasoners based on these tableau methods. For future development, it would be highly beneficial to allow the interactive rendering of the CTree corresponding to the model from which a specific node label in the HS-tree was derived.

In its current state, the application is capable of visualizing a CTree in its final state. Formally, the CTree representation is complete; however, for pedagogical and research purposes, several additional features are desired. These include: (1) clash highlighting, (2) visual marking of nodes that are ”blocked”, (3) set-based edge labels (currently only a single symbol is supported), (4) striking out the concepts in union leading to a clash.

We demonstrate CTree visualization using the following example. Let us consider a knowledge base $\mathscr{K} = \{\text{Coffee} \sqsubseteq \text{Beverage} \sqcap \text{Hot}, \text{Cold} \sqsubseteq \neg\text{Hot}, \text{Student} \sqsubseteq \exists\text{eats.Chocolate}, \text{Student} \sqsubseteq \exists\text{drinks.}(\text{Coffee} \sqcap \text{Cold})\}$. The concept Student is satisfiable w.r.t. $\mathscr{K}$ if and only if there exists a complete clash-free CTree. As shown in Figure 3, the visualized CTree contains a clash. Furthemore, all unexplored branches of the $\sqcup$-rules lead to an immediate clash as well. Consequently, it is impossible to construct a clash-free tree for this input, proving that the concept Student is unsatisfiable w.r.t. $\mathscr{K}$.

5. Conclusions

We presented the HS-Tree Visualizer, an interactive, web-based tool designed to bridge the gap between abstract abductive reasoning and its practical understanding. While existing tools focus primarily on visualizing ontology structures or logical proofs, our application provides a unique insight into the internal execution of HS-tree-based algorithms, such as MHS and MHS-MXP.

The tool’s core strength lies in its dual-mode interaction: the *Tree mode* for structural analysis and the *Step mode* for granular execution tracing. By implementing advanced layout techniques (Dagre and ELK) and interactive features such as subtree collapsing and hiding pruned branches, we have addressed the challenges of visualizing large-scale search spaces. Furthermore, the use of a standardized JSON interface ensures that the visualizer is not limited to the CATS solver but can be integrated with any abduction system that follows the specified format.

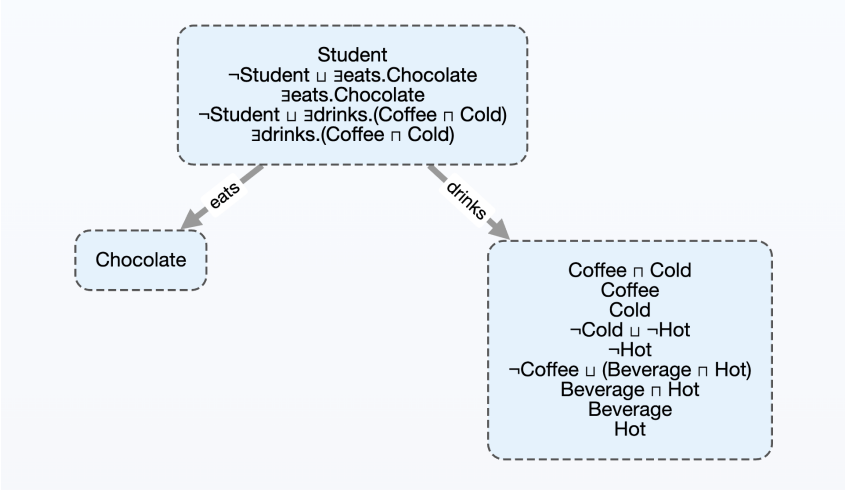

Figure 3: CTree proving the concept Student is unsatisfiable w.r.t. $\mathscr{K}$.

Our preliminary results demonstrate that the visualizer is a valuable asset for educational purposes, allowing students to follow complex reasoning steps, as well as for research and debugging, where it helps identify inefficiencies in branching or pruning logic that are often obscured in textual outputs. The tool will be evaluated within the Knowledge representation and reasoning course during lab sessions. Students will use the CATS solver to compute explanations for ABox abduction problems and will become familiar with MHS-tree construction. To assess the tool's usability, a dedicated questionnaire will be prepared.

Future work will focus on expanding the tool's capabilities to support a broader range of algorithms and reasoning tasks. While our current focus is on abduction, the underlying HS-tree structure is a fundamental component used across various logical tasks, including diagnosis, ontology debugging, and repair. We aim to generalize the visualizer to support diverse algorithm families, from classical HS-tree constructions to modern hybrid approaches such as MHS-MXP [9]. By incorporating more versatile features – such as enhanced node annotations, sophisticated branch filtering, and closer integration with underlying reasoners – we aim to evolve the HS-Tree Visualizer into a comprehensive framework for exploring a wide variety of tree-based search spaces, including those generated by dynamic pruning strategies (e.g., RCT [20]), and tableau-based simulations (TA [19]) in description logics.

Acknowledgments

This research was sponsored by Slovak Republic under the SRDA grant APVV-23-0292 (DyMAX) and VEGA grant no. 1/0630/25 (eXSec) and by the EU NextGenerationEU through the Recovery and Resilience Plan for Slovakia under the project No. 09I05-03-V02-00064 (EMA). M. Tóthová was funded by DFG grant 389792660 as part of TRR 248 – CPEC, see https://perspicuous-computing.science. M. Tóthová and J. Boborová were supported by a DL student grant.

Declaration on Generative AI

During the preparation of this work, the authors used GPT-5.3, Gemini 3 Flash, and Microsoft Copilot (powered by GPT-5) in order to: Text Translation, Improve writing style, and Grammar and spelling check. After using these services, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] J. Méndez, C. Alrabbaa, P. Koopmann, R. Langner, F. Baader, R. Dachselt, Evonne: A visual tool for explaining reasoning with OWL ontologies and supporting interactive debugging, Comput. Graph. Forum 42 (2023). doi:10.1111/CGF.14730.

[2] P. Kunowski, T. Boiński, SOVA – protege wiki, https://protegewiki.stanford.edu/wiki/SOVA, 2016. Accessed: 2026-04-09.

[3] P. Paulovics, J. Pukancová, M. Homola, SIVA: an educational tool for the tableau reasoning algorithm, in: M. Ortiz, T. Schneider (Eds.), Proceedings of the 31st International Workshop on Description Logics co-located with 16th International Conference on Principles of Knowledge Representation and Reasoning (KR 2018), Tempe, Arizona, US, October 27th - to - 29th, 2018, CEUR Workshop Proceedings, CEUR-WS.org, 2018. URL: https://ceur-ws.org/Vol-2211/paper-29.pdf.

[4] C. Alrabbaa, S. Borgwardt, T. Friese, P. Koopmann, M. Kotlov, Why not? explaining missing entailments with evee, in: O. Kutz, C. Lutz, A. Ozaki (Eds.), Proceedings of the 36th International Workshop on Description Logics (DL 2023) co-located with the 20th International Conference on Principles of Knowledge Representation and Reasoning and the 21st International Workshop on Non-Monotonic Reasoning (KR 2023 and NMR 2023)., Rhodes, Greece, September 2-4, 2023, CEUR Workshop Proceedings, CEUR-WS.org, 2023. URL: https://ceur-ws.org/Vol-3515/paper-1.pdf.

[5] C. Alrabbaa, S. Borgwardt, T. Friese, A. Hirsch, N. Knieriemen, P. Koopmann, A. Kovtunova, A. Krüger, A. Popovic, I. S. R. Siahaan, Explaining reasoning results for OWL ontologies with Evee, in: P. Marquis, M. Ortiz, M. Pagnucco (Eds.), Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning, KR 2024, Hanoi, Vietnam. November 2-8, 2024, 2024. doi:10.24963/KR.2024/67.

[6] C. S. Peirce, Illustrations of the logic of science VI: Deduction, induction, and hypothesis, Popular Science Monthly 13 (1878) 470–482.

[7] J. Kloc, J. Boborová, M. Homola, J. Pukancová, CATS solver: The rise of hybrid abduction algorithms, in: L. Tendera, Y. Ibáñez-García, P. Koopmann (Eds.), Proceedings of the 38th International Workshop on Description Logics - DL 2025, Opole, Poland, September 3-6, 2025, volume 4091 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2025. URL: https://ceur-ws.org/Vol-4091/paper34.pdf.

[8] R. Reiter, A theory of diagnosis from first principles, Artificial intelligence 32 (1987) 57–95. doi:10.1016/0004-3702(87)90062-2.

[9] M. Homola, J. Pukancová, J. Boborová, I. Balintová, Merge, Explain, Iterate: A combination of MHS and MXP in an ABox abduction solver, in: Logics in Artificial Intelligence - 18th European Conference, JELIA 2023, Dresden, Germany, September 20-22, 2023, Proceedings, volume 14281 of *Lecture Notes in Computer Science*, Springer, 2023, pp. 338–352. doi:10.1007/978-3-031-43619-2_24.

[10] I. Horrocks, O. Kutz, U. Sattler, The even more irresistible $\mathcal{SROIQ}$, in: Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, AAAI, 2006, pp. 57–67. URL: http://www.aaai.org/Library/KR/2006/kr06-009.php.

[11] C. Elsenbroich, O. Kutz, U. Sattler, A case for abductive reasoning over ontologies, in: Proceedings of the OWLED*06 Workshop on OWL: Experiences and Directions, Athens, GA, US, volume 216 of *CEUR-WS*, CEUR-WS.org, 2006. URL: https://ceur-ws.org/Vol-216/submission_25.pdf.

[12] J. Pukancová, M. Homola, ABox abduction for description logics: The case of multiple observations, in: Proceedings of the 31st International Workshop on Description Logics, Tempe, Arizona, US, volume 2211 of *CEUR-WS*, CEUR-WS.org, 2018. URL: https://ceur-ws.org/Vol-2211/paper-31.pdf.

[13] R. Greiner, B. A. Smith, R. W. Wilkerson, A correction to the algorithm in Reiter's theory of diagnosis, Artif. Intell. 41 (1989) 79–88. doi:10.1016/0004-3702(89)90079-9.

[14] K. M. Shchekotykhin, D. Jannach, T. Schmitz, MergeXplain: Fast computation of multiple conflicts for diagnosis, in: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015, AAAI Press, 2015, pp. 3221–3228. URL: http://ijcai.org/Abstract/15/454.

[15] M. Franz, C. T. Lopes, G. Huck, Y. Dong, O. Sumer, G. D. Bader, Cytoscape.js: a graph theory library

for visualisation and analysis, Bioinformatics 32 (2016) 309–311. doi:10.1093/bioinformatics/btv557.

[16] C. Pettitt, Dagre: Graph layout for JavaScript, https://github.com/dagrejs/dagre, 2024.

[17] M. Kasperowski, S. Domrös, R. von Hanxleden, The Eclipse Layout Kernel, in: Proceedings of the International Symposium on Graph Drawing and Network Visualization (GD 2024), LIPIcs, 2024. doi:10.4230/LIPIcs.GD.2024.56.

[18] I. Herman, G. Melancon, M. S. Marshall, Graph visualization and navigation in information visualization: A survey, IEEE Transactions on Visualization and Computer Graphics 6 (2000) 24–43. doi:10.1109/2945.841119.

[19] F. Baader, I. Horrocks, C. Lutz, U. Sattler, An Introduction to Description Logic, Cambridge University Press, 2017.

[20] I. Pill, T. Quaritsch, RC-Tree: A variant avoiding all the redundancy in Reiter's minimal hitting set algorithm, in: 2015 IEEE International Symposium on Software Reliability Engineering Workshops, ISSRE Workshops, Gaithersburg, MD, USA, November 2-5, 2015, IEEE Computer Society, 2015, pp. 78–84. doi:10.1109/ISSREW.2015.7392050.