Visualizing Kripke Models in LogiKEy: the Case of SDL

Luca Pasetto[1,*], Christoph Benzmüller[2,3]

[1]University of Luxembourg
[2]Otto-Friedrich-Universität Bamberg
[3]Freie Universität Berlin

Abstract

LogiKEy is a framework and methodology for the design and engineering of ethico-legal reasoners. It is based on semantical embeddings of logics and logic combinations in expressive classical higher-order logic (HOL). This meta-logical approach allows the use of off-the-shelf theorem provers and (counter-)model finders for HOL such as Nitpick. While these model finders produce precise model descriptions, their raw textual output can be hard to read and interpret for users. In this paper, we present a tool that converts a Kripke model description from Nitpick into a TikZ graph. We showcase the tool using an example in Standard Deontic Logic (SDL).

Keywords

Proof assistants, Model finders, Isabelle/HOL, Automated theorem proving, Semantical embedding, Higher-order logic

1. Introduction

LogiKEy [1] is a framework and methodology that can be used for the design, engineering, and experimentation of logics and logic combinations, with a focus on ethico-legal applications [2] and normative reasoning [3]. The formal framework of LogiKEy is based on *shallow semantical embeddings* (SSE) [4] of combinations of various logics in classical higher-order logic (HOL), and recently it has been extended to include also *deep embeddings* [5]. This meta-logical approach allows the use of off-the-shelf theorem provers and model finders for HOL such as *Nitpick* [6], so that designers of ethical intelligent agents can use existing technologies rather than build new tools from scratch. Continuous improvements in theorem proving also directly enhance the reasoning capabilities within LogiKEy with no need for extra adjustments. Figure 1 depicts this layered methodology: object logics are embedded in HOL, used to express domain theories, and then used to experiment with concrete applications and examples.

As part of the framework and methodology, in this paper we introduce a lightweight, self-contained tool that takes a Kripke model found by the HOL model-finder Nitpick [6] and converts it into a graph in the TikZ format [7]. By producing a visual rendering of possible worlds, propositional valuations, and accessibility relations, the tool fills an important gap: Nitpick's textual countermodels can be hard to interpret, even for technically trained users, and nearly impossible for legal experts or students without a background in formal methods. For encodings of logics used in normative reasoning, the tool has the potential to support (1) *interpretable normative reasoning*, as users can visually trace how obligations and facts interact, revealing information otherwise buried in symbolic output; and (2) *logic teaching*, allowing students of law, philosophy, and logic to explore models and counter-models visually. This allows for a *smooth workflow*, as it integrates with existing workflows in LogiKEy without relying on external visualization tools to then produce diagrams easily embedded in LaTeX.

The rest of the paper is structured as follows: Section 2 describes the problem that we are tackling by means of an example of input of the tool and an example of intended output, Section 3 describes the

Joint Proceedings of the Workshops and Doctoral Consortium of the 41st International Conference on Logic Programming, September 9–13, 2025, Rende, Italy
*Corresponding author.
luca.pasetto@uni.lu (L. Pasetto); christoph.benzmueller@uni-bamberg.de (C. Benzmüller)
0000-0003-1036-1718 (L. Pasetto); 0000-0002-3392-3093 (C. Benzmüller)

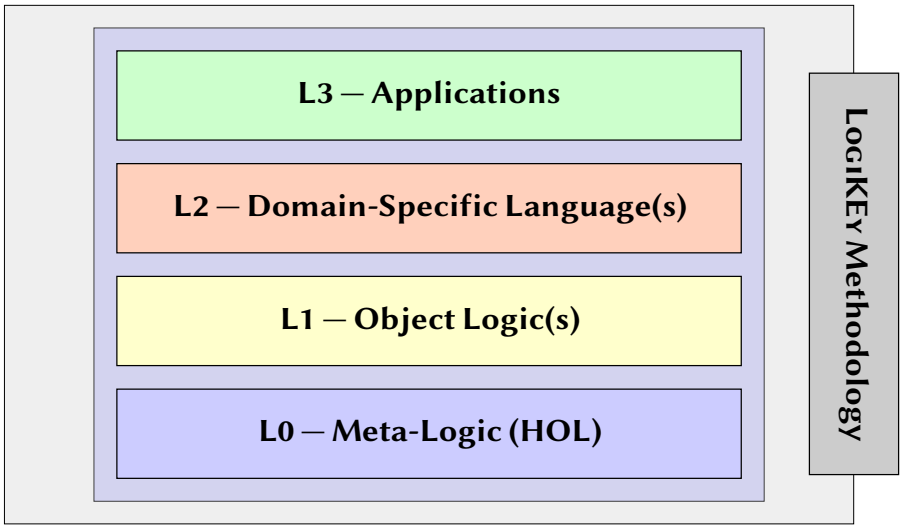

Figure 1: LogiKEy development methodology

visualizer tool, Section 4 demonstrates the tool by applying it to Standard Deontic Logic (SDL), and Section 5 discusses the relevant results and the remaining challenges.

2. Problem Statement

In this section we outline the specific task our tool addresses. Given a textual model or countermodel produced by the Nitpick model finder, we wish to generate a fully self-contained TikZ diagram that visualizes the possible worlds, their propositional valuations, and the accessibility relation between them.

Example Input Below is a representative snippet of the kind of Nitpick output our tool consumes:

```
Nitpicking formula...
Nitpick found a counterexample for card i = 6:
  Skolem constant:
    w = i1
  Constants:
    go = (\lambda x. _)(i1 := True, i2 := True, i3 := True, i4 := True, i5 := True,
     ↪  i6 := True)
    tell = (\lambda x. _)(i1 := True, i2 := True, i3 := True, i4 := True, i5 := True,
     ↪  i6 := True)
    (R) = (\lambda x. _)
      ((i1, i1) := False, (i1, i2) := False, (i1, i3) := False, (i1, i4) := True,
       ↪  (i1, i5) := False, (i1, i6) := False,
      (i2, i1) := False, (i2, i2) := False, (i2, i3) := False, (i2, i4) := False,
       ↪  (i2, i5) := False, (i2, i6) := True,
      (i3, i1) := False, (i3, i2) := False, (i3, i3) := False, (i3, i4) := True, (i3,
       ↪  i5) := False, (i3, i6) := False,
      (i4, i1) := False, (i4, i2) := False, (i4, i3) := False, (i4, i4) := False,
       ↪  (i4, i5) := False, (i4, i6) := True,
      (i5, i1) := False, (i5, i2) := False, (i5, i3) := False, (i5, i4) := False,
       ↪  (i5, i5) := False, (i5, i6) := True,
      (i6, i1) := False, (i6, i2) := False, (i6, i3) := False, (i6, i4) := False,
       ↪  (i6, i5) := False, (i6, i6) := True)
```

Intended Output The tool should produce TikZ code that, when compiled, yields a graphical rendering of the frame as a directed graph (rendered in Figure 2, where *start* indicates the current world[1]):

```
\begin{tikzpicture}
  \graph[spring electrical layout,node distance=20mm,spring constant=0.5,electric
  ↪  charge=1,cooling factor=0.5,convergence tolerance=1e-5,nodes=world] {
    w0/"$w_0:go, tell$" -> { w3/"$w_3:go, tell$" };
    w1/"$w_1:go, tell$" -> w5/"$w_5:go, tell$";
    w2/"$w_2:go, tell$" -> w3;
    w3 -> w5;
    w4/"$w_4:go, tell$" -> w5;
  };
% self loops
\path[->,loop right,looseness=8] (w5) edge (w5);
% initial state
\node[draw=none,left=of w0,xshift=-5mm] (init) {\small start};
\draw[->,thick] (init) -- (w0);
\end{tikzpicture}
```

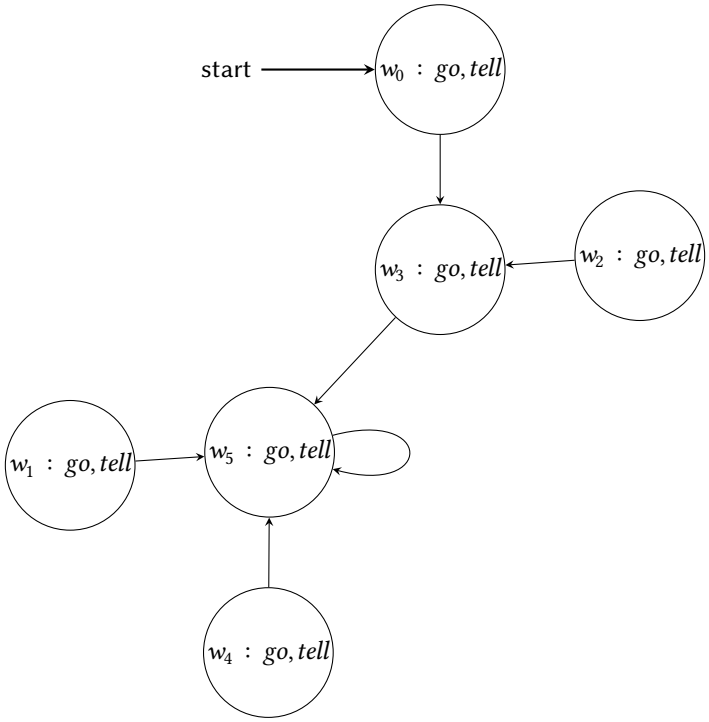

Figure 2: Intended output, visualized

Note that we use the Spring-Electrical Layout (see Section 32.3 of [8] and [9]) from the graph drawing features of TikZ [10]: each edge behaves like a *spring*, pulling its two endpoint nodes toward each other; simultaneously, every pair of distinct nodes experiences a small *electrical repulsion* [9]. The combined effect is that connected nodes cluster at a natural distance (set by `node distance`), while unrelated nodes space themselves apart, avoiding overlaps. Iterating these forces until equilibrium yields an aesthetically pleasing, roughly evenly spaced graph [9]. The advantage for us is generality: this approach yields clear, automatically arranged visualizations of Nitpick models (e.g., with varying numbers of worlds), suitable for inclusion in papers or presentations with minimal manual adjustment [8].

[1]While worlds in the Nitpick output are numbered starting from 1, in the TikZ graph we start from 0 (i.e., `i1` corresponds to w_0).

3. The Visualizer

The conversion tool automates the process of turning a Nitpick countermodel (text output from Isabelle/HOL) into a clean, self-contained TikZ diagram. In Algorithm 1 we show the pseudo-code of the tool[2]. Conceptually, it performs two main tasks: (1) Model Extraction, reading and extracting information from the Nitpick dump; and (2) Graph Rendering, producing TikZ code to draw a graph corresponding to the extracted information.

Data: Lines of a Nitpick (counter)-model dump
Result: TikZ `graph` rendering of the model
begin
  Read nonempty lines into `lines`;
  Parse the number n of worlds, the index m of the current world, and list `atoms` of proposition names;
  Build an $n \times |\texttt{atoms}|$ truth-table `values`;
  Extract all True edges into `edges`;
  Split `edges` into `self_loops` and `inter_edges`;
end
begin
  Print TikZ header and `graph` settings;
  Define Label(i):
    Return the formatted label for world i using `atoms` and `values`[i];
  Group `inter_edges` by source into `by_src`;
  Initialize empty set `defined`;
  foreach *source s in ascending order* **do**
    Let successors $\leftarrow$ `by_src`[s];
    if $s \notin$ *`defined`* **then**
      Define node w_s with Label(s);
      Add s to `defined`;
    end
    else
      Refer to existing node w_s;
    end
    if *multiple successors* **then**
      Emit an edge from w_s to the set of successors;
    end
    else
      Emit an edge from w_s to its single successor;
    end
    Mark any newly introduced successors in `defined`;
  end
  Print closing of the `graph` block;
  foreach *pair* (i, i) *in `self_loops`* **do**
    Print a separate self-loop edge for w_i;
  end
  Print arrow to the current world `m`;
  Print end of TikZ picture;
end

Algorithm 1: Pseudocode for Nitpick-to-TikZ conversion

[2]The python code is available online at logikey.org/Nitpick2TikZ.

We now go into more detail on what happens at the two distinct phases of the algorithm:

1. **Model Extraction (Parsing the Nitpick Output)**
   We read each non-empty line of the (counter)-model dump and extract:
   - The number n of worlds.
   - The index m of the current world.
   - The list `atoms` of all proposition names in the order they appear.
   - A truth-table `values` of size $n \times |\texttt{atoms}|$, recording which atoms hold in which worlds.
   - The list `edges` of all directed pairs (i, j) for which the relation is True.

   We then separate out any *self-loops* (i, i) so they can be handled after the automatic layout.

2. **Graph Rendering (Generating the TikZ Code)**
   We begin a `tikzpicture` using a force-directed `graph` layout. A helper function Label formats each world i as

   $$w_i: \ \text{atom}_1, \ \neg\text{atom}_2, \ \ldots.$$

   We group the non-loop edges by their source world, then iterate in ascending order. For each source s:
   - If s has not yet been printed, we introduce node w_s with its label.
   - We emit arrows from w_s to each of its successors, bundling multiple targets where appropriate.

   After closing the `graph` block, we draw each self-loop explicitly using TikZ's `\path[->,loop above]`, ensuring those loops remain visible, and we draw a special arrow indicating the current world. Finally we close the TikZ picture.

4. The Case of SDL

We demonstrate the application of the tool to Standard Deontic Logic (SDL). In particular, we present the well-known example of the Chisholm's contrary-to-duty paradox [11], which has been encoded in LogiKEy in [12]. For details on the Chisholm paradox in SDL, here we refer to Chapter 1 [13] of the *Handbook of Deontic Logic and Normative Systems*, Volume 1 [14].

Standard Deontic Logic is a formal system used to represent and reason about normative concepts such as obligation, permission, and prohibition. It is based on modal logic and typically corresponds to the modal system D (also known as KD), which includes the modal axioms of system K along with the seriality axiom D, ensuring that what is obligatory is at least permitted. In the language, **O** is the deontic modality for obligation. Semantically, the accessibility relation $R \subseteq W \times W$ is a serial binary relation over W. It is understood as a relation of deontic alternatives: sRt (or, alternatively, $(s, t) \in R$) expresses that t is an ideal alternative to s, or that t is a "good" successor of s. The first one is "good" in the sense that it complies with all the obligations true in the second one. Furthermore, the constraint of seriality means that the model does not have a dead end, a state with no good successor.

Chisholm's paradox is a classic contrary-to-duty scenario that exposes a key limitation of SDL. The scenario can be stated as follows with four sentences (see [13]):

F1 It ought to be that Jones goes to the assistance of his neighbors.

F2 It ought to be that if Jones goes to the assistance of his neighbors, then he tells them he is coming.

F3 If Jones doesn't go to the assistance of his neighbors, then he ought not tell them he is coming.

F4 Jones does not go to the assistance of his neighbors.

The problem is then how to formalize this: it is widely thought that all four could be true simultaneously, and none is a deductive consequence of the others. In Figure 3 we list four ways of representing Chisholm's paradox in SDL as in [13]. These are the four combinations depending on the obligation modality **O** having a wide or narrow scope with respect to implication, and each of them violates one or both of the requirements.

	WN	**WW**	**NN**	**NW**
F1	$\mathbf{O}g$	$\mathbf{O}g$	$\mathbf{O}g$	$\mathbf{O}g$
F2	$\mathbf{O}(g \rightarrow t)$	$\mathbf{O}(g \rightarrow t)$	$g \rightarrow \mathbf{O}t$	$g \rightarrow \mathbf{O}t$
F3	$\neg g \rightarrow \mathbf{O}\neg t$	$\mathbf{O}(\neg g \rightarrow \neg t)$	$\neg g \rightarrow \mathbf{O}\neg t$	$\mathbf{O}(\neg g \rightarrow \neg t)$
F4	$\neg g$	$\neg g$	$\neg g$	$\neg g$

Figure 3: Four formalizations WW, NN, NW and WN of Chisholm's paradox in SDL

Here, we do not address the technicalities of the paradox but only focus on interesting applications for the tool discussed in this paper. One example is checking whether a formula is not a deductive consequence of the others. Indeed, the fourth formula $F4$ is not a consequence of the other three in all of the formalizations, and this is something that we can find a countermodel for using Nitpick. More specifically, we use the SSE of SDL and Chisholm's paradox encodings from [12], and for each formalization WN, WW, NN and NW we check that formula $(F1 \wedge F2 \wedge F3) \rightarrow F4$ is not valid and find a countermodel using Nitpick from the Isabelle2025/HOL (March 2025) distribution. We then apply our tool to visualize such textual countermodels. Figures 4, 5, 6 and 7 show the countermodels found for formalizations WW, NN, NW and WN, respectively. Figure 2 above displays a larger countermodel found for the formalizations WN and NN. In all the frames, `start` is used to indicate the current world (for which the formula does not hold).

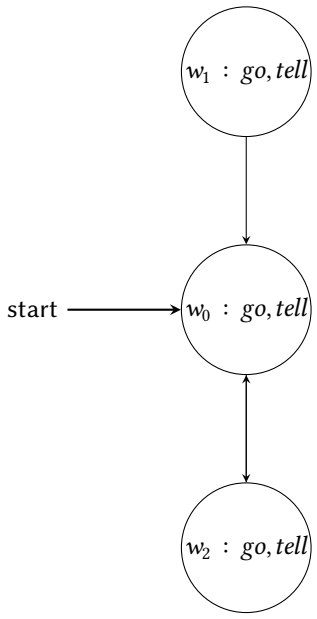

Figure 4: Countermodel for WW

5. Related Work and Conclusion

This paper contributes a self-contained tool for visualizing the Kripke models that are returned by Nitpick in a textual format. The tool complements the LogiKEy framework and methodology by

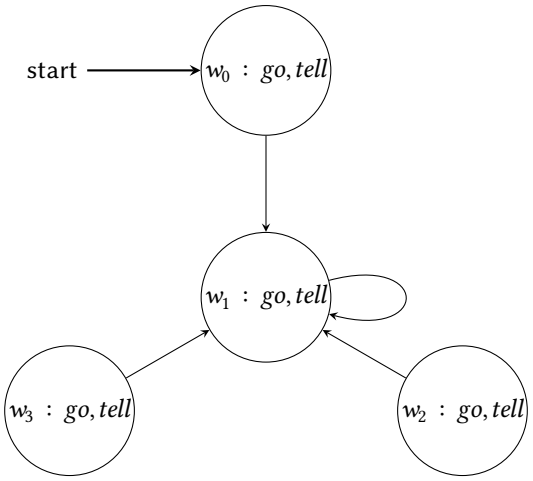

Figure 5: Countermodel for NN

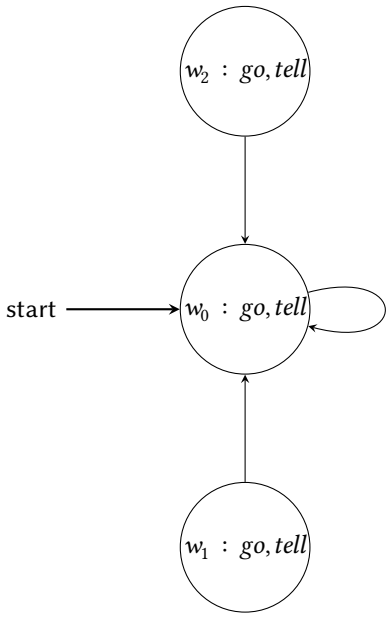

Figure 6: Countermodel for NW

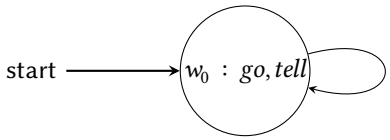

Figure 7: Countermodel for WN

facilitating the inspection of examples and counterexamples, which can be crucial in scenarios where such examples are not intuitive. Numerous systems support model or countermodel generation, e.g. Mace4 for first-order logic [15], or Nitpick and Quickcheck in Isabelle/HOL [6, 16, 17], yet these tools typically output raw text. Some exceptions are the model visualizer of Alloy [18], KripkeVis [19], or GoMoChe for Gossip Model Checking [20]. Our work brings visualization into the LogiKEy framework, which focuses on ethico-legal applications, bridging a gap between formal countermodels and human-readable explanations in such context.

Legal reasoning often involves detecting conflicts, conditional obligations, and exceptions within complex normative systems. By visualizing these structures, our tool has the potential to help LogiKEy

users *see* where obligations clash or where contrary-to-duty scenarios arise. This graphical "cognitive scaffold" not only speeds up case analysis and lets users explore *what-if* scenarios, but also enhances transparency and trust. Indeed, research in legal informatics suggests that systematically visualizing interactions of legal provisions can serve as a powerful decision-support tool [21]. With our demonstration we visualize models and countermodels as a valuable complement to formal proofs, for example for debugging and explaining.

In conclusion, our visualization tool extends the LogiKEy methodology by making countermodels accessible, can contribute to more intuitive legal-logic workflows and supports the broader goals of explainable AI in law. Future work will test how the tool scales to larger models and extend support to richer logics encoded in Isabelle/HOL in the LogiKEy framework (such as logics from the modal logic cube [5, 22], multimodal logics [23], Public Announcement Logic [24], the nested modalities for beliefs and desires of [25], value-oriented legal reasoning as in [2], or the dynamic logic for the right to know of [26]). Another future direction is to add interactive features to further enhance educational applications within LogiKEy [27].

Acknowledgments

This work was supported by the Luxembourg National Research Fund (FNR) through the project Logical methods for Deontic Explanations (INTER/DFG/23/17415164/LODEX) and the project Deontic Logic for Epistemic Rights (OPEN O20/14776480).

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] C. Benzmüller, X. Parent, L. van der Torre, Designing normative theories for ethical and legal reasoning: LogiKEy framework, methodology, and tool support, Artificial Intelligence 287 (2020). doi:10.1016/j.artint.2020.103348.

[2] C. Benzmüller, D. Fuenmayor, B. Lomfeld, Modelling value-oriented legal reasoning in LogiKEy, Logics 2 (2024) 31–78. doi:10.3390/logics2010003.

[3] X. Parent, C. Benzmüller, Automated verification of deontic correspondences in Isabelle/HOL – first results, in: C. Benzmüller, J. Otten (Eds.), ARQNL 2022, CEUR Workshop Proceedings, 2023, pp. 92–108. URL: https://ceur-ws.org/Vol-3326/.

[4] C. Benzmüller, Universal (meta-)logical reasoning: Recent successes, Sci. Comput. Program. 172 (2019) 48–62. doi:10.1016/j.scico.2018.10.008.

[5] C. Benzmüller, Faithful logic embeddings in HOL — deep and shallow, in: C. Barrett, U. Waldmann (Eds.), Automated Deduction – CADE-30 – 30th International Conference on Automated Deduction, Stuttgart, Germany, July 28-31, 2025, Proceedings, volume 15943 of *Lecture Notes in Computer Science*, Springer, 2025, pp. 280–302. doi:10.1007/978-3-031-99984-0_16, (preprint: arXiv:2502.19311).

[6] J. C. Blanchette, T. Nipkow, Nitpick: A counterexample generator for higher-order logic based on a relational model finder, in: M. Kaufmann, L. C. Paulson (Eds.), Interactive Theorem Proving (ITP2010), volume 6172 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 131–146. doi:10.1007/978-3-642-14052-5_11.

[7] T. Tantau, Graph drawing in TikZ, in: W. Didimo, M. Patrignani (Eds.), Graph Drawing (GD2012), volume 7704 of *Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, 2013, pp. 517–528. doi:10.1007/978-3-642-36763-2_46.

[8] T. Tantau, TikZ and PGF Manual (Version 3.1.5b), 2020. URL: https://mirrors.ctan.org/graphics/pgf/base/doc/pgfmanual.pdf.

[9] Y. Hu, Efficient, high-quality force-directed graph drawing, The Mathematica Journal 10 (2006) 37–71.
[10] J. Pohlmann, Configurable Graph Drawing Algorithms for the TikZ Graphics Description Language, Master's thesis, 2011.
[11] R. M. Chisholm, Contrary-to-duty imperatives and deontic logic, Analysis 24 (1963) 33–36. doi:10.1093/analys/24.2.33.
[12] C. Benzmüller, A. Farjami, D. Fuenmayor, P. Meder, X. Parent, A. Steen, L. van der Torre, V. Zahoransky, LogiKEy workbench: Deontic logics, logic combinations and expressive ethical and legal reasoning (Isabelle/HOL dataset), Data in Brief 33 (2020) 1–10. doi:10.1016/j.dib.2020.106409.
[13] R. Hilpinen, P. McNamara, Deontic logic: A historical survey and introduction, in: D. M. Gabbay, J. Horty, X. Parent, R. van der Meyden, L. van der Torre (Eds.), Handbook of Deontic Logic and Normative Systems, College Publications, London, UK, 2013, pp. 3–136.
[14] D. M. Gabbay, J. Horty, X. Parent, R. van der Meyden, L. van der Torre (Eds.), Handbook of Deontic Logic and Normative Systems, College Publications, London, UK, 2013.
[15] W. McCune, Mace4 Reference Manual and Guide, Technical Report ANL/MCS-TM-264, Argonne National Laboratory, 2003. URL: https://www.mcs.anl.gov/research/projects/AR/mace4/July-2005/doc/mace4.pdf, technical Memorandum, Mathematics and Computer Science Division.
[16] S. Berghofer, T. Nipkow, Random testing in isabelle/hol, in: Proceedings of the Second International Conference on Software Engineering and Formal Methods, 2004. SEFM 2004., 2004, pp. 230–239. doi:10.1109/SEFM.2004.1347524.
[17] L. Bulwahn, The new quickcheck for isabelle, in: C. Hawblitzel, D. Miller (Eds.), Certified Programs and Proofs, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 92–108.
[18] D. Jackson, Alloy: a lightweight object modelling notation, ACM Trans. Softw. Eng. Methodol. 11 (2002) 256–290. doi:10.1145/505145.505149.
[19] M. Gattinger, Kripkevis — a haskell module to visualize kripke frames, 2014. URL: https://w4eg.de/malvin/illc/kripkevis/.
[20] M. Gattinger, Gomoche: Gossip model checking, 2022. URL: https://malv.in/2022/LAMASSR-GoMoChe.pdf.
[21] S. McLachlan, E. Kyrimi, K. Dube, N. Fenton, L. C. Webley, Lawmaps: enabling legal ai development through visualisation of the implicit structure of legislation and lawyerly process, Artificial Intelligence and Law 31 (2023) 169–194. doi:10.1007/s10506-021-09298-0.
[22] C. Benzmüller, M. Claus, N. Sultana, Systematic verification of the modal logic cube in Isabelle/HOL, in: C. Kaliszyk, A. Paskevich (Eds.), PxTP 2015, volume 186, EPTCS, Berlin, Germany, 2015, pp. 27–41. doi:10.4204/EPTCS.186.5.
[23] C. Benzmüller, L. C. Paulson, Quantified multimodal logics in simple type theory, Logica Universalis 7 (2013) 7–20. doi:10.1007/s11787-012-0052-y.
[24] C. Benzmüller, S. Reiche, Automating public announcement logic with relativized common knowledge as a fragment of HOL in LogiKEy, Journal of Logic and Computation 33 (2023) 1243–1269. doi:10.1093/logcom/exac029.
[25] L. Pasetto, C. Benzmüller, Implementing the fatio protocol for multi-agent argumentation in logikey, in: C. Benzmüller, J. Otten, R. Ramanayake (Eds.), ARQNL 2024, CEUR Workshop Proceedings, 2024.
[26] L. Lawniczak, L. Pasetto, C. Benzmüller, X. Li, R. Markovich, Reasoning with epistemic rights and duties: Automating a dynamic logic of the right to know in LogiKEy, in: I. Lynce, N. Murano, M. Vallati, S. Villata, F. Chesani, M. Milano, A. Omicini, M. Dastani (Eds.), Proceedings of the 28th European Conference on Artificial Intelligence (ECAI 2025), volume 413 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2025, pp. 1623 – 1630. doi:10.3233/FAIA250988.
[27] C. Benzmüller, D. Fuenmayor, Mathematical proof assistants for teaching logic: The LogiKEy methodology, in: Book of Abstracts — V Congress Tools for Teaching Logic, Madrid, Spain, 2023. URL: https://www.researchgate.net/publication/371044093. doi:10.13140/RG.2.2.24708.74888.