

# VMDV: A 3D Visualization Tool for Modeling, Demonstration, and Verification

Authors

No Institute Given

**Abstract.** In the setting of automated theorem proving, the output of an automated theorem prover is usually presented in text format, which is often too heavy to be understood. In the setting of model checking, it would be helpful if one can observe, at the same time, both the model structure under consideration and the verification procedure. To address these problems, a 3D visualization tool for modeling, demonstration and verification (VMDV for short) is proposed in this paper. The facilities of VMDV are illustrated by applying it to an automated theorem prover.

**Keywords:** 3D Visualization · Automated theorem proving · Model Checking

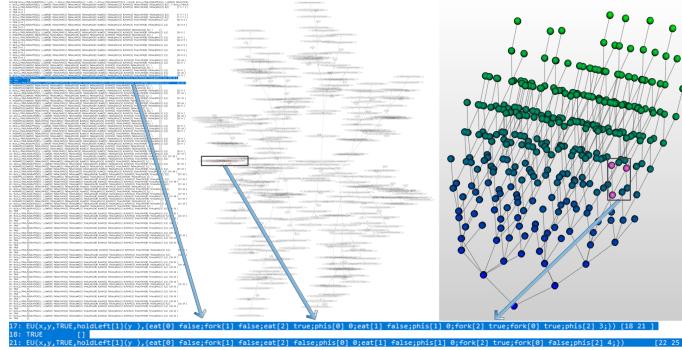
## 1 Introduction

In the field of mathematical logic, a formal proof is usually defined as a sequence of formulas, which are either axioms or logical consequences of the preceding formulas. In the most natural way, a proof is generally presented as a proof tree, where each node is labeled by a formula, and its children are labeled by its hypotheses. Nowadays, one can obtain proofs from computers automatically, thanks to the development of automated theorem provers. However, the output of a theorem prover is usually displayed in text format. This usually makes the proof tree difficult to understand (e.g., to observe the relative locations of some nodes in the same proof tree), especially when the proof tree contains a large set of nodes. The same problem also arises in the field of model checking, where counterexamples generated by model checkers are sometimes hard to read [1, 5, 4]. Moreover, text format is difficult to show the proof structures, especially when proof trees are dynamically updated. This is very common in a proving procedure, where the nodes may be dynamically created or deleted. Thus, a more readable form will be helpful to engineers who are not familiar with model checking or automated theorem proving. This leads to the following questions which motivate the present work:

1. How to make the output of a theorem prover to carry huge amount of information and, at the same time, easy to understand and to reason about?
2. How to observe the verification procedure in an intuitive manner when checking some properties of a given complex model?

To answer these questions, a 3D visualization tool for Modeling, Demonstration, and Verification (VMDV for short) is proposed<sup>1</sup>. VMDV represents proofs by 3D trees. As a matter of fact, VMDV employs a 3D renderer to plot proof trees in 3D space. As is well known, 3D renderer is based on the 3D information visualization techniques which take advantage of the human eyes' broad bandwidth pathway into the mind to allow users to capture large amounts of information at once. In this way, proof trees are easily organized and displayed, detailed local text information adheres to the node on demand.

In Fig. 1, it is shown that the 2D format is more convenient to reflect the structure of the proof tree than the text format. However, when a proof tree contains a large number of nodes, the information adhering to the nodes may be overlapped, and hence hard to understand, even to read. In contrast, there is enough space in 3D format to show, without confusion, both the structure of proof trees and the information adhering to the nodes. One of the reasons is that a 3D format space can be seen as a combination of infinitely many 2D plans, and VMDV allows to observe an output of an automated theorem prover, typically a proof tree, from different angles by rotating and zooming.



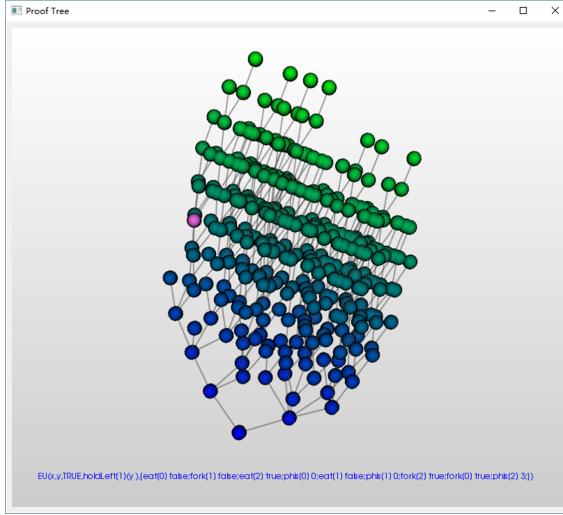
**Fig. 1.** Display of a proof tree in text format, 2D format, and 3D format, respectively.

The main facilities of VMDV are listed as follows. They are illustrated by applying VMDV to SCTLProV<sup>2</sup>, which can be seen both as an automated theorem prover and a model checker.

- VMDV allows the observation either from a global view or from a local view. The global view shows the shape, or the topological structure of the concerned output, while the local view shows the detailed information of nodes or sub-structures under consideration (Fig. 2).
- VMDV also provides us a variety of ways to observe or interact with the proof tree. The zooming and rotation operations effectively change the viewport,

<sup>1</sup> <https://github.com/ProveVis/vmdvpy>

<sup>2</sup> [https://github.com/sctlprov/sctlprov\\_code](https://github.com/sctlprov/sctlprov_code)



**Fig. 2.** A typical screenshot of VMDV.

making the presentation of the overall structure of the proof tree clear via different angles of view (Fig. 6).

- When the output, typically a proof tree, is very large and only a subtree is of interest, this subtree can be selected and focused, while other parts of the tree can be hidden; also, nodes with specific pattern can be searched and highlighted on the proof tree etc. (Fig. 5). This allows us to find, among others, all the formulas from which one specific formula can be inferred or deduced.
- In some proof systems, there may be some auxiliary structures emerging along with the construction of proof trees, such as Kripke models or inductively defined terms. VMDV visualizes both proof trees and auxiliary structures. Visualizing the interaction between proof trees and the auxiliary structures may help us to better understand the proving procedure.

As a visualization tool, VMDV is designed and implemented as a stand-alone program, which is independent of the automated theorem prover to be applied. Indeed, automated theorem provers and VMDV communicate via TCP sockets (Fig. 4). This facilitates the extensibility of VMDV to different automated theorem provers.

*Related Work.* As far as we know, many efforts have been made to the visualization of the output of automated theorem provers. For instance, in [3, 13, 18], and [20], proofs are presented in 2D format instead of text format, and colors are used to highlight crucial parts of proof trees. In [13], the authors proposed several criteria for the visualization of proof trees, such as distinguishing different kinds of rules, following the progress of subproofs, and focusing on different aspects of the proof, etc. On the other hand, there exist indeed a few visual-

ization tools with 3D libraries ([9, 2]), and with the facility for visualizing both proof trees and other data structures such as term expressions [2], however the layout algorithms of these tools are only limited to graphs with simple structures. Our work is different from the existing visualization tools: First, we render the graphs in 3D format, instead of 2D format as in most of the existing visualization tools. Rendering in 3D space enables the visualization of graphs with much more complicated structures that are usually difficult to understand in 2D space. Secondly, with the employing **VTK** [19] as the underlying visualization library of VMDV, we can use more complicated layout algorithms to render the 3D graphs. VMDV is capable of handling the layout of proof trees and directed graphs smoothly, where nodes may be created or deleted dynamically during the proving procedure of the theorem prover.

*Outline.* The rest of the paper is organized as follows: Section II presents the preliminaries, including some basic notions of information visualization, and the brief introduction of a proof system, called **SCTL**. Section III introduces our visualization tool VMDV. Section IV involves the applications of VMDV to the prover of **SCTL**. Section V concerns conclusion and future works.

## 2 Preliminaries

In this section, we first present the concept of information visualization, and then present the basic definitions of a proof system called **SCTL** which is a sequent calculus for Computation Tree Logic (CTL) [7, 8], and also an automated theorem prover **SCTLProV** that implements the **SCTL** system.

### 2.1 Information Visualization

Information visualization is the study of visual representation of abstract data that focus on the creation of approaches for humans to capture abstract information intuitively. With the visual representations of data, it is easier for humans to get a deeper understanding, and gain the essentials over the massive data-sets. As opposed to digital numbers which are more readable for computers, information visualization systems are more friendly to human beings for providing both concrete and abstract inspirations. For instance, plotted charts are better understandable than bare data-sets. It assists in uncovering the trends, reveal insights, or even tell stories. Information visualization provides a easier way for users to capture the abstract patterns of the massive data by applying a graphical presentation. There are many libraries used to build information visualization applications, among which **OpenGL** and **VTK** are widely used.

*OpenGL* Benefit from the study of computer graphics, information presentation using visualization has been enjoying popular support. The Open Graphics Library (**OpenGL**) provides a language independent and cross-platform application programming interface (**API**) for the rendering of three dimensional graphics. As

for the hardware, the enhancing of the computational power of Graphics Processing Units (GPU) has made the efficient rendering of complex graphics a reality.

**VTK** VTK is a freely available framework for information visualization systems, which provides a library for the visualization of 3D graphics. VTK can be seen as a wrap of OpenGL, in the sense that VTK uses OpenGL as its underlying rendering API. Built upon OpenGL, VTK provides a more useful API for the layout and interactions with 3D graphics.

We apply the ideas of information visualization to the development of VMDV, for which we use VTK as its underlying visualization library. VMDV provides:

- The presentation of data in 3D space, where the data points are encoded as 3D solid spheres, and the structure of data are encodes as lines between solid spheres;
- Visual interaction with data, such as highlighting the search results for specific data, or controlling the progress of producing new data.

## 2.2 SCTL and SCTLProV

Model checking [5, 7, 8] and automated theorem proving [10, 14] are two major pillars of formal verification methods. The proof system SCTL, introduced in [6], is a sequent calculus for CTL, taking Kripke models as parameters. SCTL performs verification directly on a given Kripke model from the perspective of automated theorem proving, and produces formal proofs when the verification procedure terminates.

The syntax of SCTL is stipulated as follows. The properties of a Kripke model are expressed in a language tailored for this model. The language contains, for each state  $s$  of the model, a constant also written  $s$ ; for each relation  $P$  over the model, a predicate symbol also written  $P$ . Formulas are built in the usual way with the connectors  $\top, \perp, \wedge, \vee$  and  $\neg$ , to which we add modalities  $AX, EX, AF, EG, AR$ , and  $EU$ . If  $\phi$  is a formula, and  $t$  is either a constant or a variable, then  $AX_x(\phi)(t)$ ,  $EX_x(\phi)(t)$ ,  $AF_x(\phi)(t)$ , and  $EG_x(\phi)(t)$  are formulas. Like quantifiers, modalities bind the variable  $x$  in  $\phi$ . If  $\phi_1$  and  $\phi_2$  are formulas and  $t$  is either a constant or a variable, then  $AR_{x,y}(\phi_1, \phi_2)(t)$  and  $EU_{x,y}(\phi_1, \phi_2)(t)$  are formulas. These modalities bind the variable  $x$  in  $\phi_1$  and  $y$  in  $\phi_2$ , respectively.

The semantics of a SCTL formula is defined as follows, the proof rules for the system SCTL is depicted in Fig. 3.

**Definition 1 (Valid formula).** Let  $\mathcal{M}$  be a model and  $\phi$  be a closed formula, the set of valid formulas  $\models \phi$  in the model  $\mathcal{M}$  is defined by induction on  $\phi$ :

- $\models P(s_1, \dots, s_n)$ , if  $\langle s_1, \dots, s_n \rangle \in P$ ,
- $\models \neg P(s_1, \dots, s_n)$ , if  $\langle s_1, \dots, s_n \rangle \notin P$ ,
- $\models \top$  is always the case,
- $\models \perp$  is never the case,

- $\models \phi_1 \wedge \phi_2$ , if  $\models \phi_1$  and  $\models \phi_2$ ,
- $\models \phi_1 \vee \phi_2$ , if  $\models \phi_1$  or  $\models \phi_2$ ,
- $\models AX_x(\phi_1)(s)$ , if for each state  $s'$  in  $\text{Next}(s)$ ,  $\models (s'/x)\phi_1$ ,
- $\models EX_x(\phi_1)(s)$ , if there exists a state  $s'$  in  $\text{Next}(s)$  such that  $\models (s'/x)\phi_1$ ,
- $\models AF_x(\phi_1)(s)$ , if for all infinite paths  $s_0, s_1, \dots$  starting from  $s$ , there exists a natural number  $i$ , such that  $\models (s_i/x)\phi_1$ ,
- $\models EG_x(\phi_1)(s)$ , if there exists an infinite path  $s_0, s_1, \dots$  starting from  $s$ , such that for all natural numbers  $i$ ,  $\models (s_i/x)\phi_1$ ,
- $\models AR_{x,y}(\phi_1, \phi_2)(s)$ , if for all infinite paths  $s_0, s_1, \dots$  starting from  $s$ , and for all  $j$ , either  $\models (s_j/y)\phi_2$  or there exists an  $i < j$  such that  $\models (s_i/x)\phi_1$ ,
- $\models EU_{x,y}(\phi_1, \phi_2)(s)$  if there exists an infinite path  $s_0, s_1, \dots$  starting from  $s$  and a natural number  $j$  such that  $\models (s_j/y)\phi_2$  and for all  $i < j$ ,  $\models (s_i/x)\phi_1$ .

An automated theorem prover **SCTLProV** is developed to implement **SCTL**. The interested reader is referred to [6] and [12] for further details of the proof system **SCTL** and the automated theorem prover **SCTLProV**.

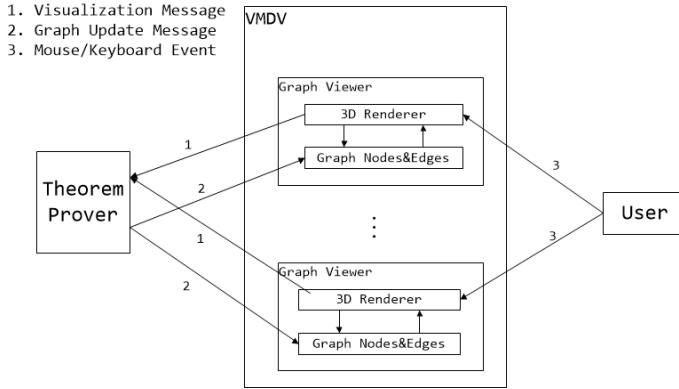
$\frac{}{\vdash P(s_1, \dots, s_n)} \text{ atom-R}$	$\frac{}{\vdash \neg P(s_1, \dots, s_n)} \neg\text{-R}$	$\frac{}{\vdash \top} \top\text{-R}$
$\frac{\vdash \phi_1 \quad \vdash \phi_2}{\vdash \phi_1 \wedge \phi_2} \wedge\text{-R}$	$\frac{\vdash \phi_1}{\vdash \phi_1 \vee \phi_2} \vee\text{-R}_1$	$\frac{\vdash \phi_2}{\vdash \phi_1 \vee \phi_2} \vee\text{-R}_2$
$\frac{\vdash (s'/x)\phi \quad \text{EX-R}}{\vdash EX_x(\phi)(s)} \quad s' \in \text{Next}(s)$	$\frac{\vdash (s_1/x)\phi \quad \dots \quad \vdash (s_n/x)\phi}{\vdash AX_x(\phi)(s)} \quad \{s_1, \dots, s_n\} = \text{Next}(s)$	
$\frac{\vdash (s/x)\phi \quad \text{AF-R}_1}{\Gamma \vdash AF_x(\phi)(s)} \quad \text{AF-R}_1$	$\frac{\Gamma \vdash AF_x(\phi)(s_1) \quad \dots \quad \Gamma \vdash AF_x(\phi)(s_n)}{\Gamma \vdash AF_x(\phi)(s)} \quad \text{AF-R}_2$	$\frac{\Gamma \vdash EG_x(\phi)(s) \quad \text{EG-merge}}{\Gamma \vdash EG_x(\phi)(s)} \quad EG_x(\phi)(s) \in \Gamma$
$\frac{\vdash (s/x)\phi \quad \Gamma, EG_x(\phi)(s) \vdash EG_x(\phi)(s') \quad \text{EG-R}}{\Gamma \vdash EG_x(\phi)(s)} \quad s' \in \text{Next}(s)$		
	$\frac{\vdash (s/y)\phi_2 \quad \Gamma' \vdash AR_{x,y}(\phi_1, \phi_2)(s_1) \dots \Gamma' \vdash AR_{x,y}(\phi_1, \phi_2)(s_n) \quad \text{AR-R}_1}{\Gamma \vdash AR_{x,y}(\phi_1, \phi_2)(s)} \quad \{s_1, \dots, s_n\} = \text{Next}(s)$	$\frac{\Gamma \vdash AR_{x,y}(\phi_1, \phi_2)(s) \quad \text{AR-merge}}{\Gamma' = \Gamma, AR_{x,y}(\phi_1, \phi_2)(s)} \quad \Gamma' = \Gamma, AR_{x,y}(\phi_1, \phi_2)(s)$
$\frac{\vdash (s/x)\phi_1 \quad \vdash (s/y)\phi_2 \quad \text{AR-R}_2}{\Gamma \vdash AR_{x,y}(\phi_1, \phi_2)(s)} \quad \text{AR-R}_2$	$\frac{\Gamma \vdash AR_{x,y}(\phi_1, \phi_2)(s) \quad AR_{x,y}(\phi_1, \phi_2)(s) \in \Gamma}{\Gamma \vdash AR_{x,y}(\phi_1, \phi_2)(s)} \quad AR_{x,y}(\phi_1, \phi_2)(s) \in \Gamma$	
$\frac{\vdash (s/y)\phi_2 \quad \text{EU-R}_1}{\Gamma \vdash EU_{x,y}(\phi_1, \phi_2)(s)} \quad \text{EU-R}_1$	$\frac{\vdash (s/x)\phi_1 \quad \Gamma \vdash EU_{x,y}(\phi_1, \phi_2)(s') \quad \text{EU-R}_2}{\Gamma \vdash EU_{x,y}(\phi_1, \phi_2)(s)} \quad s' \in \text{Next}(s)$	

**Fig. 3.**  $\text{SCTL}(\mathcal{M})$

### 3 VMDV

#### 3.1 Architecture

In the design of VMDV, the theorem prover is not supposed to know exactly how VMDV renders graphs, and VMDV does not know the working procedure



**Fig. 4.** The architecture of VMDV

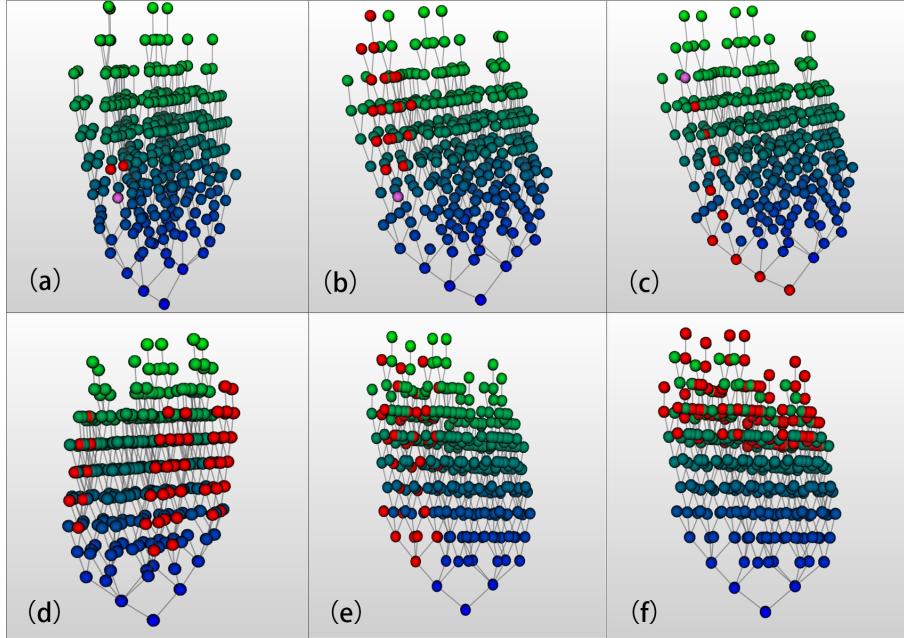
of the theorem prover. The theorem prover simply sends output information to VMDV, and receives feedbacks from it. As is shown in Fig. 4, there are two kinds of messages interchanged by a theorem prover and VMDV: the Visualization Messages sent by VMDV, and the Graph Update Messages sent by the theorem prover. Visualization Messages are control messages consist of requiring the related information of the current formula, such as the hypothesis, sub-formulas, or other specific information (e.g., the related state in an SCTL formula). Graph Update Messages are data messages consist of the responds to the control messages. VMDV serves as the interface to dynamically visualize the output of the theorem prover. The relationship of VMDV and a theorem prover is very similar to that of a web browser and a web server. In order to extend the applications of VMDV to other theorem provers, VMDV is designed and implemented as a stand-alone application, not as a part of specific theorem provers. Theorem provers and VMDV communicate via TCP sockets. Both control and data messages are wrapped as TCP packets. This way, VMDV can easily communicate with theorem provers implemented in different programming languages, or run in different computers in networks.

Note that VMDV allows multiple outputs to visualize multiple structures, proof trees, or, e.g., proof trees and Kripke models in SCTL system.

### 3.2 The Presentation of 3D Graphs

In VMDV, there are two kinds of graphs that can be visualized in 3D space: trees and directed graphs. Usually, a proof tree of a formula is represented as a 3D tree, while a Kripke model in the SCTL system is represented as a 3D directed graph. In the visualization of a proof tree, each node of the proof tree is visualized as a solid sphere, and it connects all its children (if any) via 3D lines. In the visualization of a Kripke model, each state is also visualized as a solid sphere, and it connects all its successors via 3D lines. The layout strategies of trees and directed graphs are different: trees are visualized as cones, while the layout of

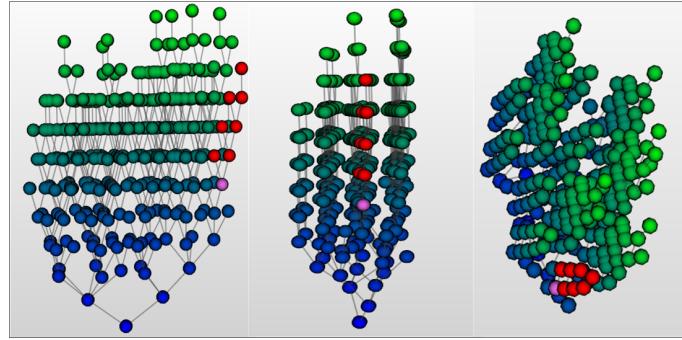
directed graphs uses forced-based layout algorithms [15, 11]. In this way, each node of a 3D graph is associated a location in the 3D space. However, in order to retrieve more information from visualization, it is tempting to associate more information to each node. For instance, Fig. 5 shows the visualization of different information of a proof tree: (a) shows the premises of a formula (i.e., the children of a node); (b) shows a sub-tree; (c) shows the ancestors of a node in a proof tree; (d) shows the proof trees of formulae starting with the same modality; (e) shows the sub-formula of a formula in its proof tree; and finally, (f) shows parts of a proof tree where all of them are proved using the same rule. For each node of the 3D tree, we call each kind of associated information a *dimension*. The first three dimensions (i.e., the coordinates on the X, Y, and Z axes of the 3D space) of a node form its location in the 3D space. Other dimensions are dynamically visualized as the highlighting of the nodes. For instance, Fig. 5 shows the visualization of 6 dimensions of nodes in a proof tree. Moreover, dimensions can also be dynamically added to the nodes. This is crucial to extend VMDV with more functionalities in order to integrate with more specific theorem provers. For instance, in Example 2, the number of aircraft in each zone of the SATS model can be viewed as a dimension of a node, and each such dimension can be visualized respectively. When adding dimensions to nodes, the following



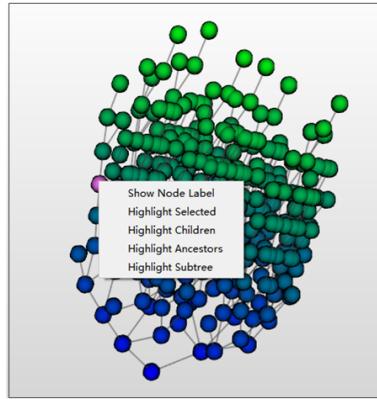
**Fig. 5.** Visualization of 6 kinds of information (dimensions) of proof trees

interfaces of controlling the 3D graphs are needed.

**Zooming and rotation.** The most obvious advantage of 3D visualization over 2D one is the capacity of observing a graph from different angles of view. Although there exist different ways to plot a 2D graph, it is still hard to match the 3D solution when the structure of the graph is too complicated to present in 2D space. 3D visualization techniques handle this easily using two operations, zooming and rotation: zooming in to see the details, zooming out to capture the overall shape, and rotating the tree to locate the sub-tree of interest, as are shown in Fig. 6.



**Fig. 6.** A proof tree observed from different angles of view.



**Fig. 7.** Extensible operations.

**Extended Operations.** In addition to the commonly used operations, some other operations that are specific to theorem provers are also needed, so that

VMDV can be adapted to different theorem provers. The controlling of the construction of both the proof tree and the Kripke model in the SCTL system is a good example. The way of defining extended operations in VMDV can be explained as follows: firstly, we define the operations of 3D graphics in VMDV as a set of Python objects, called the **Affect** objects; secondly, we define another set of Python objects, called the **Trigger** objects, to load and execute the code in the **Affect** objects, i.e., to trigger the operations of 3D graphics; finally, we associate each **Trigger** object to a pop-up menu item, such that one can select a pop-up menu item to activate the associated **Trigger** object when right clicked the mouse (Fig. 7). This way, defining operations for new theorem provers in VMDV can be implemented by defining new **Affect** objects, **Trigger** objects, and pop-up menu items, and VMDV exposes simple, but powerful interfaces to define them.

## 4 Applications

With the application of VMDV, we can show both proof trees and Kripke models in 3D format. We can also visualize, in 3D format, the verification procedure, revealing gradually the relation between a proof tree and the corresponding states of the Kripke model under consideration. The application of VMDV to the theorem prover SCTLProV is illustrated in the following two examples.

*Example 1 (The River Crossing Puzzle).* A farmer is trying to transport a wolf, a goat, and a cabbage from one side of a river to another, however, he can only carry at most one item each time. During the transportation, the goat cannot be left alone with the wolf, nor the cabbage can be left alone with the goat. The question is how can the farmer get across the river by bringing the wolf, the goat, and the cabbage.

We formalize this problem by defining a Kripke model, where each state is represented by an assignment of four boolean state variables **farmer**, **wolf**, **goat** and **cabbage**, and each transition between states is represented by the transformation of an assignment of these variables to another. The property to be verified is that whether there is a path that starts with the state

$$S_0: \{\text{farmer:false}, \text{wolf:false}, \text{goat:false}, \text{cabbage:false}\}$$

and end with the state

$$S: \{\text{farmer:true}, \text{wolf:true}, \text{goat:true}, \text{cabbage:true}\}.$$

This equals to verify if the sequent

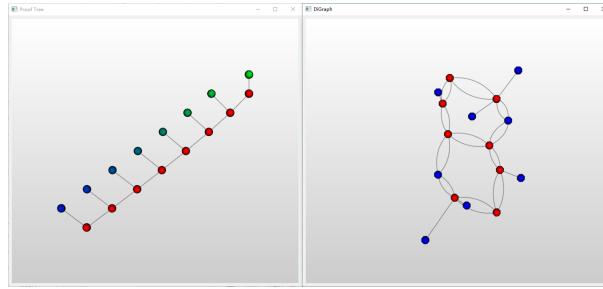
$$\vdash EU_{x,y}(\text{safe}(x), \text{complete}(y), S_0)$$

is provable in SCTL, taking the Kripke model above as parameter, where  $\text{safe}(x)$  denotes the atomic formula which specifies that, in state  $x$ , the goat is not alone with the wolf, and the cabbage is not alone with the goat; and  $\text{complete}(y)$  the atomic formula which specifies that, in state  $y$ , the farmer has carried all the

three items across the river. Using VMDV, we can show in 3D format the proof tree and the Kripke model both provided by SCTLProV (Fig. 8). In addition, along with the proof search of the sequent in progress, in the Kripke model, a path of states emerges from scratch, certifying this proof. This is because the formula

$$EU_{x,y}(safe(x), complete(y), S_0)$$

starts with the *EU* modality (called *EU*-formula), and each application of the rule **EU-R**<sub>2</sub> of SCTL (Fig. 3) corresponds to one unfolding step in the Kripke model. This process stops until the **EU-R**<sub>1</sub> is applied. This way, when we highlight all nodes with *EU*-formulas in the proof search tree, a path of states in the Kripke model is also highlighted, certifying this proof (Fig. 8).



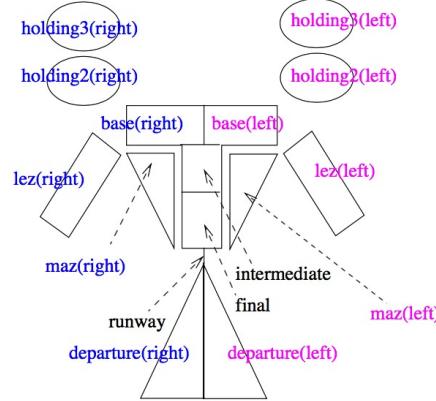
**Fig. 8.** The highlighting of all the *EU*-formulas in the proof tree (left), and a path in the model (right).

*Example 2 (The Small Aircraft Transportation System).* As an application to an engineering problem, we present a concept of operations for the *Small Aircraft Transportation System (SATS)* [16, 17] in SCTLProV<sup>3</sup>.

In this concept of operations, the airspace volume surrounding an airport facility, called *the self controlled area*, is divided into 15 zones (Figure 9). For instance, the zone **holding3(right)** is a holding pattern at 3000 feet on the right of the self controlled area. Each zone contains a list of aircraft and 24 transition rules specify different SATS-procedures. For instance, the rule **Vertical Entry (right)** specifies the vertical entry of an aircraft in the zone **holding3(right)**.

The model is non-deterministic, that is, for a given state, several transitions are possible and all must be considered. As there are no a priori bounds on the number of aircraft in each zone, the number of states in the model is potentially infinite. However, the number of states that are reachable from the initial state is finite: an enumeration of the model shows that there are 54221 such states (and around 3000 in the simplified model where departure operations are not considered [16]).

<sup>3</sup> [https://github.com/sctlprov/sctlprov\\_sats](https://github.com/sctlprov/sctlprov_sats)

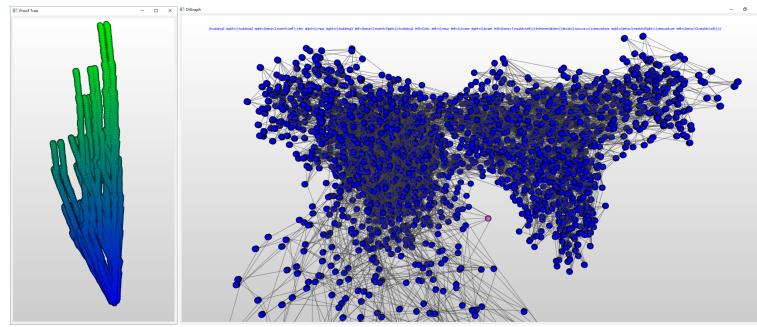


**Fig. 9.** SCA zones, where right and left are relative to the pilot facing the runway, i.e., opposite from the reader point of view [16].

There are eight properties of the model that we want to verify with SCTL-ProV, for instance that the SATS concept does not allow more than four simultaneous landing operations and none of the 15 zones contains too many aircraft (each zone is assigned a maximum number of aircraft and the actual number of aircraft is never higher than this number). The safety property is the conjunction of these eight properties.

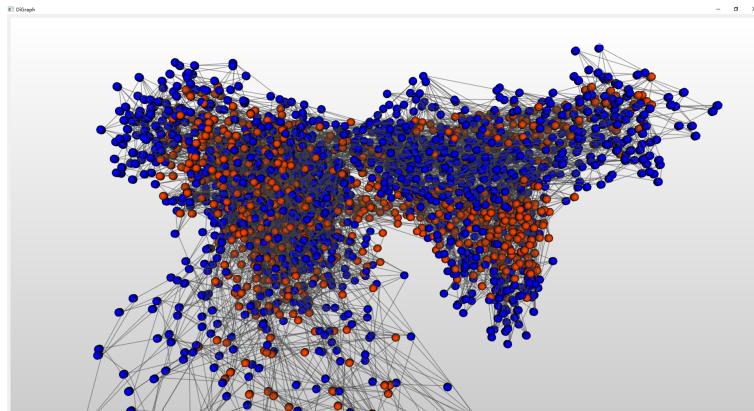
The verification problem is to check that this property holds on every reachable state from the initial state (the state where there are no aircraft on each zone of the self controlled area), so the formula to be checked is  $AG_x(\phi)(e)$  where  $\phi$  is the safety property and  $e$  is the initial state.

We have been able to check in SCTLProV that the safety property holds on every reachable state from the initial state in the model. The visualization of the proof tree and the Kripke model is depicted in Fig. 10.

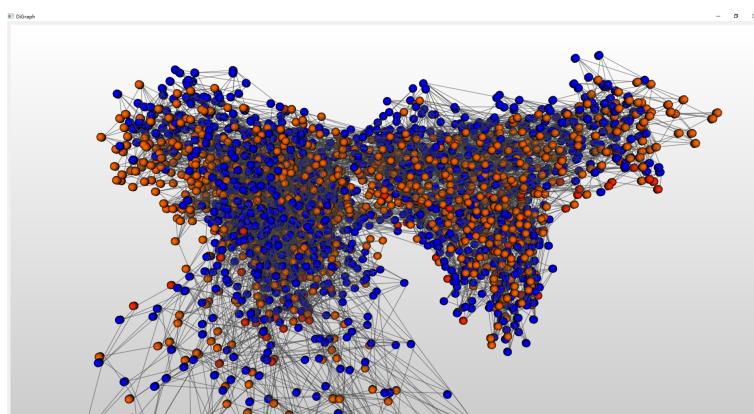


**Fig. 10.** The visualization of the proof tree and the Kripke model of the SATS system

Moreover, to have a better understanding of the Kripke model, VMDV can be used to visualize the distribution of aircraft in each zone of the SATS model. For instance, Fig. 11 and Fig. 12 show the distribution of aircraft in the zone **holding3(right)** and **departure(left)**, respectively, where we use the orange color to highlight the states where the given zone has at least one aircraft, and the more aircraft in the zone, the darker the color is.



**Fig. 11.** The distribution of aircraft in the zone **holding3(right)**



**Fig. 12.** The distribution of aircraft in the zone **departure(left)**

## 5 Conclusion and Future Works

We have proposed a visualization tool VMDV for visualising (dynamically) the output of various theorem provers in 3D space, where an automatic layout algorithm to manage the shape of the 3D graph is used. The focus of this paper is on the automated theorem prover SCTLProV, where the output is either a proof tree, or some auxiliary structure such as a Kripke model. VMDV provides us a variety of ways to observe or interact with the outputs of SCTLProV from a 3D perspective.

We have made our first step to the visualization analysis of proof trees produced by some automated theorem prover. One of the main objectives for our further work is to integrate VMDV with more sophisticated theorem provers such that **Coq**. Another objective is to improve VMDV so that it helps to guide proof procedures.

## Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable comments. This work was partially funded by the CAS-INRIA project VIP (GJHZ1844) and the French-Chinese project LOCALI (NSFC 61161130530 and ANR-11-IS02-00201).

## References

1. Baier, C., Katoen, J.: Principles of model checking. MIT Press (2008)
2. Bajaj, C., Khandelwal, S., Moore, J., Siddavanahalli, V.: Interactive symbolic visualization of semi-automatic theorem proving. Computer Science Department, University of Texas at Austin (2003)
3. Byrnes, J., Buchanan, M., Ernst, M., Miller, P., Roberts, C., Keller, R.: Visualizing proof search for theorem prover development. Electronic Notes in Theoretical Computer Science **226**, 23–38 (2009)
4. Clarke, E.M., Grumberg, O., McMillan, K.L., Zhao, X.: Efficient generation of counterexamples and witnesses in symbolic model checking. In: DAC. pp. 427–432 (1995)
5. Clarke, E.M., Grumberg, O., Peled, D.: Model checking. MIT Press, Cambridge, MA, USA (2001)
6. Dowek, G., Jiang, Y.: A logical approach to CTL (2013), <https://who.rocq.inria.fr/Gilles.Dowek/Publi/ctl.pdf>
7. Emerson, E.A., Clarke, E.M.: Using branching time temporal logic to synthesize synchronization skeletons. Sci. Comput. Program. **2**(3), 241–266 (1982)
8. Emerson, E.A., Halpern, J.Y.: Decision procedures and expressiveness in the temporal logic of branching time. J. Comput. Syst. Sci. **30**(1), 1–24 (1985)
9. Farmer, W.M., Grigorov, O.G.: Panoptes: An exploration tool for formal proofs. Electr. Notes Theor. Comput. Sci. **226**, 39–48 (2009)
10. Fitting, M.: First-Order Logic and Automated Theorem Proving, Second Edition. Graduate Texts in Computer Science, Springer (1996)

11. Hu, Y.: Efficient and high quality force-directed graph drawing. *The Mathematical Journal* **10**, 149–160 (2005)
12. Jiang, Y., Liu, J., Dowek, G., Ji, K.: SCTL: towards combining model checking and proof checking. arXiv (2017)
13. Libal, T., Riener, M., Rukhaia, M.: Advanced proof viewing in prooftool. In: Proceedings Eleventh Workshop on User Interfaces for Theorem Provers, UITP 2014, Vienna, Austria, 17th July 2014. pp. 35–47 (2014)
14. Loveland, D.W.: Automated Theorem Proving: A Logical Basis (Fundamental Studies in Computer Science). Elsevier (1978)
15. Mathieu, J., Tommaso, V., Sebastien, H., Bastian, M.: Forceatlas2, a continuous graph layout algorithm for handy network visualization designed for the gephi software. *PLOS ONE* **9**(6), e98679 (2014)
16. Muñoz, C.A., Dowek, G., Carreño, V.: Modeling and verification of an air traffic concept of operations. In: Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2004. pp. 175–182. ACM, USA, Boston, Massachusetts, USA (11-14 July 2004)
17. NASA/TM-2004-213006: Abstract Model of SATS Concept of Operations: Initial Results and Recommendations. NASA, USA (2004)
18. Sakurai, K., Asai, K.: Mikibeta: A general GUI library for visualizing proof trees. In: Logic-Based Program Synthesis and Transformation, pp. 84–98. Springer (2011)
19. Schroeder, W.J., Avila, L.S., Hoffman, W.: Visualizing with VTK: A tutorial. *IEEE Computer Graphics and Applications* **20**(5), 20–27 (2000)
20. Steel, G.: Visualising first-order proof search. In: Proceedings of User Interfaces for Theorem Provers. vol. 2005, pp. 179–189 (2005)