Improved Visualization of Relational Logic Models

University of Waterloo Technical Report CS-2013-04

Atulan Zaman, Iman Kazerani, Medha Patki, Bhargava Guntoori, Derek Rayside
Electrical and Computer Engineering
University of Waterloo
drayside@uwaterloo.ca

***Abstract*—The Alloy Analyzer includes a visualizer tool for presenting counter-examples to the user. This visualizer tool contains a wide array of settings and a 'Magic Layout' feature to automatically infer values for these settings based on a static analysis of the specification being visualized. We improve both the visualizer itself and the Magic Layout feature. For example, expert users often use colour to distinguish changes of state when visualizing specifications of dynamic systems, but previously Magic Layout was not sophisticated enough to infer where state changes might be represented in the specification. We have also improved the way in which the visualizer distinguishes different but related types of atoms, as well improved the visual consistency between different frames of a projection. Finally, a quantitative evaluation is done to compare how much better the new inferred theme compares to the default theme, and a qualitative evaluation of how the inferred theme compares to the expert made themes.**

I. Introduction

One of the main features of the Alloy Analyzer is its ability to produce examples and counter-examples to illustrate the user's specifications. The Analyzer includes a visualization facility that is commonly used to inspect these instances. The visualizer has a number of settings that can be customized by the user for a particular specification and saved in a *theme* file for later use.

The visualizer settings include basics such as node and edge labels, visibility, colour, and shape. The user can also choose whether to display a relation as edges between nodes or node labels. Perhaps the most sophisticated feature of the visualizer is *projection*. Projection's most common use is for dynamic models — models of systems that change state over time. When an instance of such a model is projected over time the visualizer shows a separate frame (image) for each tick of the clock. By viewing these frames in sequence the user can see how the state of the system evolved.

This main use case for projection is sometimes confounded by the layout engine, however. The layout engine treats each frame separately, and so nodes often change positions frame to frame as the edges (and nodes) visible on each frame changes. To address this concern we have developed a new meta-layout scheme to find node positions that can remain stable throughout all frames of a projection while still giving a reasonable layout on each individual frame.

The visualizer has many settings. Each signature in the model has eight different settings that can be adjusted, whereas each relation has nine. The graph as a whole has twenty one settings (including options for setting defaults). A simple toy example such as the Farmer/Fox/Chicken/Grain puzzle has six user signatures and three user relations, for a total of 96 adjustable settings.

The Magic Layout [1] feature was developed to help the user customize these settings. When the user presses the Magic Layout button a static analysis is performed on the model and the results of that analysis are used to adjust the visualizer settings. Magic Layout, as originally reported, was able to meet or exceed the default settings in almost all cases.

In this paper we improve Magic Layout so that it can meet or exceed the human expert settings in almost all cases — for 22 of the 24 models that come bundled with Alloy 4.2. The remaining two models have visualizer settings that require a deep semantic understanding of the concepts in the model, which is beyond the aspirations of Magic Layout.

This paper is told in three acts. First, new visualizer features are introduced. These include stability through projection (mentioned above) and a new visual dimension on nodes called *accents*. Second, improvements to the Magic Layout inference are described, with a focus on projection over multiple types and using colour to express changes of state (as expert users do, but which the original Magic Layout did not do). Finally, these new features and inferences are evaluated both quantitatively and qualitatively on the set of standard models that come bundled with Alloy 4.2. The conclusion is that the new Magic Layout, coupled with the new visualizer features, represents a significant improvement to Alloy's already powerful visualizer.

II. New Visualizer Features

The new features that are implemented for this version of alloy are *Projection Over Subtypes, Colouring to Express State Change, Shapes to Define Supertypes, Hierararchy Expression Using Modified Edges (Haircuts), Static Node Positioning through Projection Frames*. The features can be broadly categorized into two categories: features that are implemented using model inference, and more general features that make the model visualizations more intuitive.

Some features require static analysis to decide what scheme to implement for certain models, because it has been noticed that different visualization schemes for different features are better for models with certain characteristics. Among the features that are discussed in the paper, the following features use intelligent static analysis for scheme selection: *Projection, Coloring, Static Node Positioning*. The two other features complement each other by together addressing a new dimension of expression for the models. While they require some static analysis in applying the feature to models, they differ from

the other features because they do not have to decide between different schemes depending on the static analysis results.

A. Shapes to define supertypes

As briefly mentioned in the previous section on the coloring scheme, the scheme for Shape assignment is used to represent type distinction in the new magic layout. In distinguishing types, there are two inferable dimensions in which they can be expressed: *Depth in Hierarchy* & *Breadth in top level types*. In the previous magic layout, both forms of distinction was dealt with using shapes, which sometimes proved more confusing than helpful. In the new magic layout, the shapes scheme is only used to distinguish all *visible* top level types and not the depth in hierarchy.

An illustration of this can be noticed in reference to the farmer model example in figure 4. There the old magic layout decides to illustrate the objects using four different shapes. However in the in the new magic layout, all four types are assigned the same shape, but with different "haircuts" since they all belong to the same super type.

This resulted in the models becoming visually simpler and more intuitive to understand because lesser number of shapes were being used to the express the type families in each model. The reduction in the usage space for different shapes in a certain model allowed us to prevent some of the shapes from being readily used in the graph visualization, because some shapes are terrible for encapsulating labels inside them. Shapes such as Triangles and Ellipses are examples of such shapes. They still exist in the vocalbulary of the visualizer and are can possibly be utilized for manual customization, however they are prevented from being used during inference of models.

B. Subtyping Using Node Accents

Haircuts were required in order to help differentiate between different families of shapes. The previous Alloy segregates hierarchies by shapes, and then uses double or triple outlines, depending on the depth of the tree. This caused problems as trees grew deeper because it was harder to see the differences between subtypes of the same supertype.

In order to rectify this, we implemented haircuts and reduced a number of shapes used by alloy to achieve optimal clarity. Viewing hierarchies as families, we decided that each supertype and its subtypes should maintain the same shape throughout, and be differentiated through haircuts or hairstyles in a systematic, dynamic manner. For examples, in the previous version, if the supertype was a rectangle, its subtypes would be of shapes from the rectangle family. Here, the user would have to intuitively recognize the subtype hierarchy from shape similarities, which got blurry as models increased in depth due to the limited number of shapes and an unclear definition of shape families. Instead, we decided that a supertype and all its subtypes should be of the same shape. This way, user does not have to intuitively recognize shape similarities, and it is more clear that all these nodes are under the same supertype. The haircuts would be a far easier way for the user to easily understand any variations that were present.

Haircuts are modifications to the top edge of a node shape, mainly imitating various waveforms, including: Sin, Triangular, Sawtooth, Absolute Sin, Straight, and Slanted. These haircut methods took in as inputs the left and right corners of the node shape, the distance between these points and drew the haircuts with a predetermined frequency.

In order to accommodate for the nature of haircuts, several shapes were removed. Shapes that did not have flat upper edges (Ellipse, Circle, Egg, Triangle, Diamond, House, Double Circle, MDiamond, MCircle) were eliminated as they could not support haircuts using the methods we had outlined. As well, the number of shapes became redundant once haircuts were put into effect. An illustration of how haircuts improve subtype visualization in the new alloy is visible in figure 4 for the farmer model and figure 1 for the firewire model.

An empirical study was conducted on examples provided with Alloy, which indicated that the maximum depth in the hierarchies was usually within five levels. Although the haircut system improves upon what Alloy currently does, it may not be as effective for models of a larger depth. In those cases, the new system faces a bottle neck where the same haircuts are repeated for types with higher depth.

C. Static node positioning throughout projections

The Alloy visualizer currently optimizes the layout of each frame of a projection individually. This results in the best possible positioning of nodes on each frame, but makes it difficult to compare frames with each other. Projection is often used to visualize different states of a system as it evolves over time, and having the nodes change position makes it challenging to quickly identify the meaningful changes between the frames. Consequently, some users have requested that the node positions remain stable throughout all frames of a projection even if that comes at the cost of making the node placement sub-optimal on each individual frame.

We have developed three strategies for computing node placement in projections. The best strategy for a given model is selected dynamically when the user requests a projection. The selection objective is to minimize the number of edge crossings on each frame of the projection, while also trying to minimize the maximal number of crossings on any individual frame. The objective is realized using the following metric: $min(\sum_{\forall f \in P} c_f^2)$, where P is the projection, f is a frame, and c_f is the number of crossings on frame f. The strategies are:

1) *Arrange by Most Complex Frame.* Apply the normal (usually Sugiyama) layout algorithm to the frame with the most nodes and edges, and then use those node positions on every other frame.
2) *Arrange by Composite Frame.* Construct an artificial composite frame in which every node and every edge from every frame of the projection appears. Apply the normal layout algorithm to this composite frame, and then use those node positions on every frame.
3) *Arrange by Node Type.* Assign a rank (row) to each type of node in the model. Within each rank sort nodes lexicographically. Ranks with a high number of edges between them are placed adjacent to each other.

Figure 2 shows an example of a series of three frames from a Towers of Hanoi simulation. We can understand the example

Fig. 1: Firewire

a) Firewire with Expert Theme. Projected over 1 Sig.

b) Firewire with Magic Layout Theme. Projected over 2 Sigs.

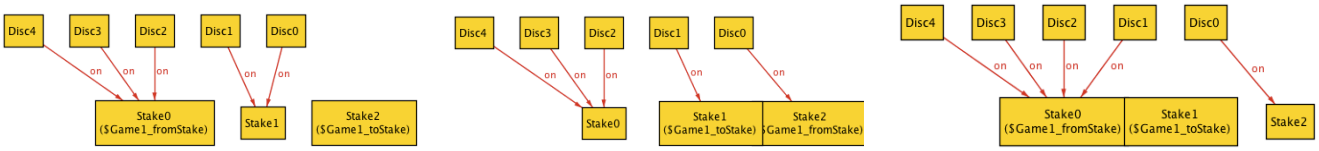


Fig. 2: Three frames from a Towers of Hanoi puzzle simulation. The top row of small boxes represent the discs. The bottom row of larger boxes represent the stakes that the discs are placed on. An edge from a disc to a stake indicates that the disc is on that stake. From frame *(a)* to frame *(b)* we see that the last disc is moved from the second to the third stake. From frame *(b)* to frame *(c)* we see that the penultimate disc is moved from the second to the first stake. We can have these insights about what happens frame to frame without reading the labels on the nodes because the relative position of the nodes is the same in each frame.

even though we cannot read the node labels because the relative node positions remain stable across the frames.

III. Improved Inference

A. Projection Over Multiple Types

The projection feature is designed to simplify the model instances in case of models with temporal expressions which have state changes. In the work of [2] a formal notion of the application of projection in relation graph models is illustrated in much technical detail and much of the work on projection in this context is inspired from that work.

In the previous version of alloy, only one dimensional projection was allowed in which case the static analysis engine takes each type declaration of the model and decides which is a proper candidate for projection based on a ranking system. The subsequent work, takes a more ambitious approach and attempts to produce more intuitive models by using certain model characteristic to infer situations where projecting of multiple types produce better graphs.

The following example illustrates how the application of multiple projections made the visualization for the `Hotel` model more intuitive.

The difference between the two themes is that in the previous version of magic layout, the model was only projected over the temporal type, which expressed the notion of state change. In the new version of magic layout it projects over both the `State` type, as well as the type `FrontDesk` because this is ternary type that is not changing throughout the frames. Therefore projecting over that type made the visualization simpler and more intuitive without compromising the general semantics of the model. The two visualizations also differ in the coloring scheme implemented. The previous magic layout colored the nodes using type definitions, while the new magic layout colored the instance using their relation to the state change. This feature is discussed in more detail in the following sections.

Now a closer look is taken at the static analysis algorithm used in deciding chosing whether the projection scheme want to apply multiple projection or one dimensional projection in figure 1.

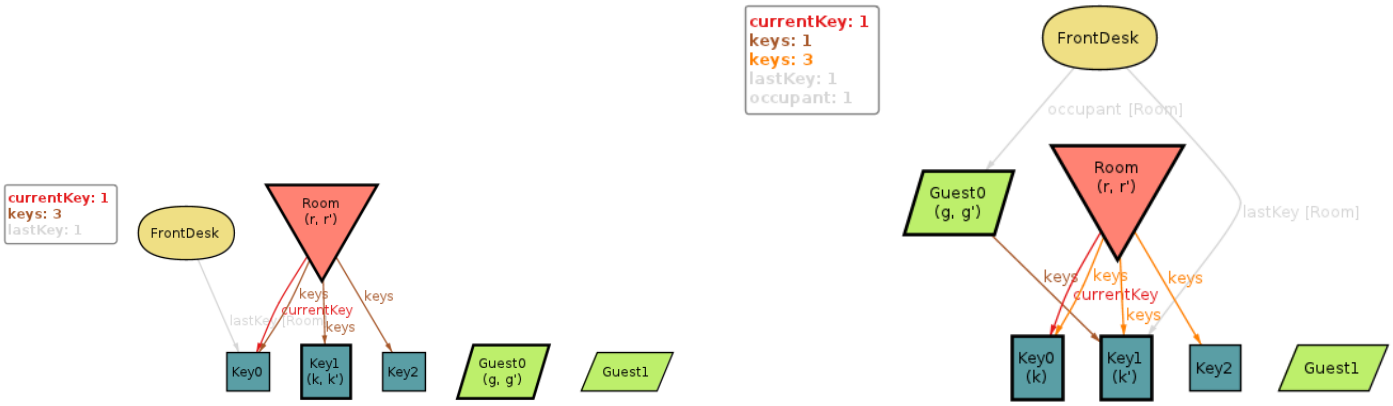
In this context, the following terms are defined as such:

a) LoneSingleton.: A `LoneSingleton` in Alloy is a type in alloy that it *quantified* to be one. Therefore if there exists such a type and it does not have any subtypes or belong to any super types and is a ternary type, then it qualifies as a LoneSingleton.

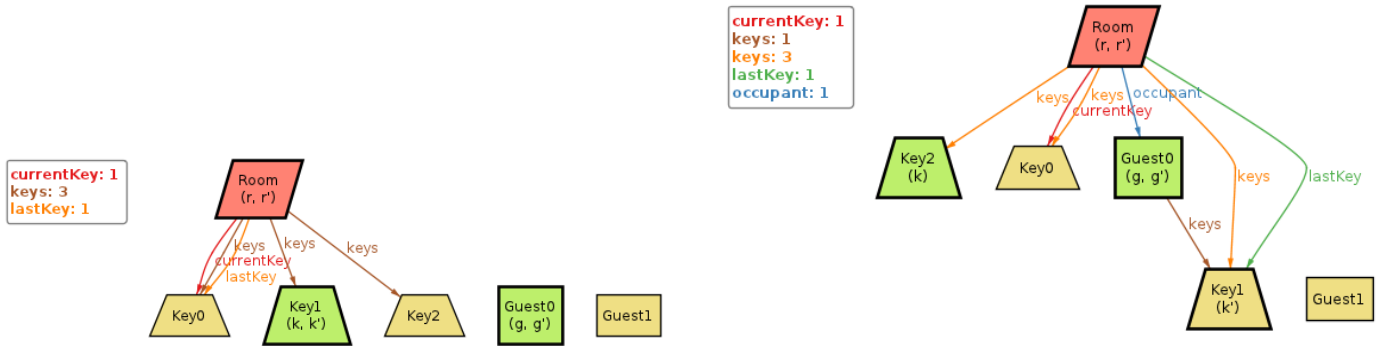
b) Temporal Type.: Examples of `Temporal` Types are State, Time or Tick which represent the non static aspect of a model.

Fig. 3: Hotel

Hotel Model drawn using previous magic layout



Hotel Model drawn using current magic layout



Algorithm 1 Projection algorithm for multiple types

Get the list of all types in the model
for types in the model **do**
  if Type is `LoneSingleton` **then**
    Add Type to projectable types
  else if Type is `Temporal Type` **then**
    Add Type to projectable types
  else
    if Type is `Ternary Type` **then**
      Add type to list of ternary types with a score `Ternary Wrapper` Types get a higher score
    end if
  end if
end for
Project over all `LoneSingleton` and `Temporal Types`
Project only over ternary type with highest score unless it exists in the same relation as the Temporal Type.
In case of tie, use the first ternary type available

c) Ternary Type: A ternary type is a type that is involved in a relation of arity ¿ 2. This means that such types influence the model strongly, and projecting of them usually makes the model simpler. A ternary wrapper type is the most dominant type in a ternary relation.

B. Coloring to Express State Change

In the previous magic layout version of alloy, both the coloring scheme as well as the scheme for assigning shapes was used to distinguish different types within a model. This is a redundant use of available features, and therefore the new inferred theme uses the coloring option to address a different dimension of expression. Instead of redundantly coloring the different types, which is alread distinguished using shapes and modified edges in the new inferrence method, coloring is instead used to highlight types that change through the frames of a projected model. This makes it much intuitive to follow the changes taking place in a model in the case of a projection. In case of models without projection, the original scheme of coloring over types is followed.

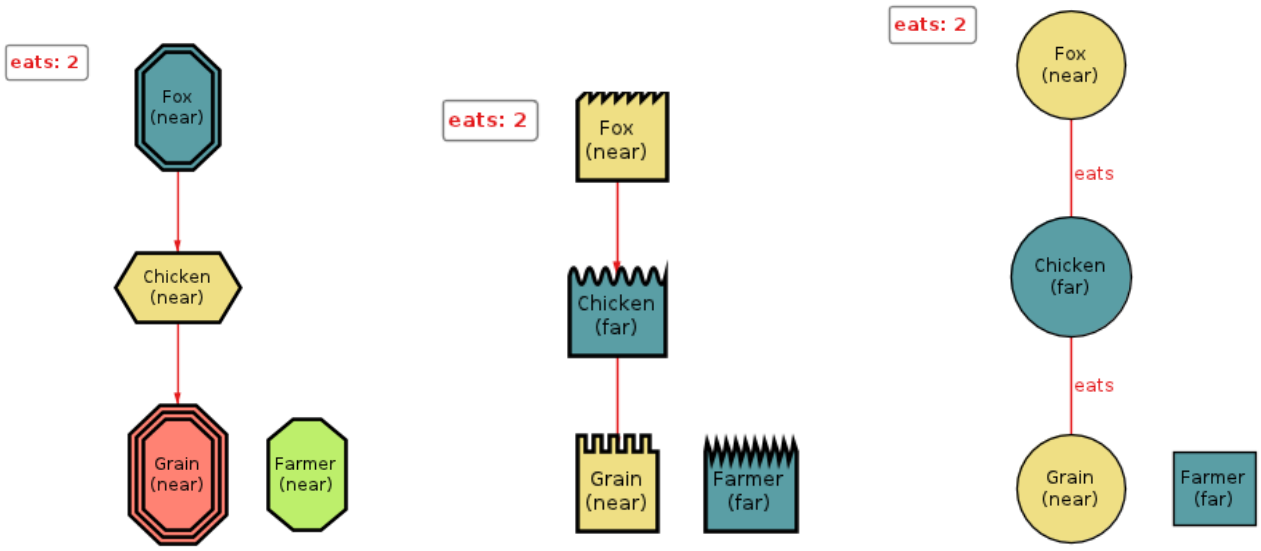
An illustration is provided with respect to the farmer model in magic layout in figure 4. .

As can be seen in the visualization, the coloring scheme used in the new magic layout decides to color the types using the relations `far` and `near` because these are relations that are changing through the projection on `State` which is inferred by the fact that they are Ternary relations related to the projected type.

IV. Evaluation

To evaluate the property inference features, we applied our techniques to a collection of Alloy models that come bundled

Fig. 4: Using colour to distinguish state changes in the Farmer puzzle



(a) Old magic layout *(b)* New magic layout *(c)* Expert theme

in the Alloy distribution. The distribution also contains a theme file for each model, prepared by an expert user.

We used a quantitative analysis to compare our inferred theme to the expert theme and to the standard default theme (*i.e.*, no model-specific customizations). The distance in this analysis serves as basis for numerically justifying how much better the *magic-layout* theme is compared to the *default* theme without any customizations. We also performed a qualitative comparison of the inferred and expert themes, to evaluate how close the intelligence of the inference techniques is to human intuition for customization.

A. *Quantitative Evaluation*

We measure the distance between two visualization themes by assigning weights to each visualization property according to Table I. Many properties are set-valued. For example, a theme may project over multiple signatures. In these cases we assign a different weight depending on whether these sets are disjoint, intersecting, or subsets. We consider that the empty set is disjoint from all other sets, so by 'subset' we mean 'non-empty subset'.

Projection and node visibility are the properties with the greatest impact on how the diagram looks, so they are assigned the highest weights. We are not concerned with the particular values of node colour and shape but, rather, whether there is an isomorphism between the colourings (shape choices, respectively).

TABLE I: Weights assigned to property differences between two themes

Property	**Weight**		
	Disjoint	Intersection	Non-empty Subset
Projection	8	6	5
Node Visibility	8	7	6
Attributes	5	4	3
Edge Labels	3	2	1
Node Colour (non-isomorphic differences)	6		
Node Shape (non-isomorphic differences)	3		
Node Style	3		
Spine	2		

TABLE II: Distances of the Default and Inferred Themes from the Ideal Theme

Model	**Quantitative Evaluation**			**Qualitative Evaluation**	
	magic *vs* **expert**	**default** *vs* **expert**	Δ	**magic**	*vs* **expert**
birthday	0	11	11	Better	
ringElection	3	14	11	Better	
stable_ringlead	0	11	11	Better	
hanoi	0	11	11	Better	
filesystem	3	12	9	Better	
ringlead	11	20	9	Better	
life	3	11	8	Better	
firewire	13	20	7	Better	
hotel	17	20	3	Better	
p300_hotel	17	20	3	Better	
geneology	0	0	0	Better	
stable_orient_ring	0	20	20	Same	
op_spantree	6	20	14	Same	
dijkstra's	3	14	11	Same	
stable_mutex_ring	3	11	8	Same	
ceiling_and_floors	0	3	3	Same	
grandpa	0	0	0	Same	
farmer	3	11	8	Worse	Model
railway	14	17	3	Worse	Model
messaging	17	20	3	Worse	Model
handshake	3	3	0	Worse	Edge/Labels
lists	3	3	0	Worse	Attribute

The first numeric column of Table II lists the distances measured between the inferred (magic) theme and the expert theme. For example, in the first row we see a zero in the first numeric column, indicating that for the birthday model there were no measurable differences between the inferred theme and the expert theme. By contrast, for the hotel model we see that there are large differences between the inferred and expert themes (measuring 17).

The second numeric column of Table II lists the distances measured between the default theme (no model-specific customizations) and the expert theme. This could be considered a complexity metric for the expert theme: how many changes to the default did the expert have to make? For the geneology model the expert made no measurable changes from the default theme.

The third numeric column of Table II lists the difference of the first two columns, to measure whether the inferred theme is closer to the expert theme than the default is: positive indicates inferred is closer to expert; negative indicates default is closer to expert; zero indicates equidistant. There are no negative values, indicating that performing the inference is never worse than doing nothing (sticking with the default).

B. Qualitative Evaluation

In this section the second column of figure II is addressed. Here we use human judgement by reviewers to categorize and justify how the magic layout visualizations compare to the original expert created visualizations. The different criteria in which we categorize the models are *Better, Same*, & *Worse*. Among these categories, *Same* does not deserve a lot of attention because we modelled our inference by extracting the intuition behind customization of experts, and in those cases the inference feature was able to reproduce the expert theme. Instead we take a closer look at some of the models in the other categories.

In the following discussion we focus on models that had interesting (usually large) measured differences between the expert and the inferred themes.

d) Better.: First we look at the model `Hotel` where the magic-layout inference did a subjectively better job at creating an intuitive visualization than the expert. In this Alloy counter-example, it tries to prove that according to the current model, no intrusion can happen from key control of a hotel. The model is illustrated in figure 5.

The first criteria in which the inferred theme does a better job than the expert theme in this model, is in terms of projection. The expert theme only projects over the temporal type `Time`, whereas the magic-layout inference projects both over `Time` and `FrontDesk` types. According to the projection inference discussed above, the model projects over Time because it is a temporal type, and it also projects over FrontDesk type because it qualifies as a `LoneSingleton` which does not change over projection frames. This simplifies the model by removing a type from being visible. Conversely, the expert decided to simplify the model by viewing a lot of the types and relations using node attributes. It is uncommon for experts to design themes that project over multiple types because in the general case it's not clear what the intuitive meaning of that would be. However, in the special case of LoneSingleton types these extra projections can make the visualization clearer. This discovery was an unexpected result of our experiments with the magic layout inference algorithm.

The expert theme for hotel was designed to be printed in black and white and hence did not use colour. The inferred theme uses colour to show changes in state, which is helpful if the user has a colour display.

Now we take closer look at a model where the inferred outcome is better than the expert only due to the new features that were not available to the original experts. One such model is the the `firewire` model. In this model, firewire protocol of connecting consumer electronics is simulated. In the model declaration, a lot of consumer devices are instantiated in the model to simulate the environment, however only a single election procedure is simulated in the run clause of the model to simulate the behavior, hence most of the device instances are unused.

In this case since most of the devices are not affected by the projection frames, the expert theme decided to hide the device types and only view the types that are affected by the temporal type. However, the magic-layout theme infers that the devices are all subtypes of the same supertype and therefore assign it the same shape with different haircuts, which help in distinguishing them from the ternary types such as `Node` and `Stutter`. Arguably this is a better representation than the expert theme for the sake of completeness. Although the expert theme is simpler to view than the magic-layout theme, which is generally better, in this case since there is a feature to differentiate type hierarchy it is arguably a better idea to retain the completeness of the model while making it intuitive to the user what the type distinction is.

Similarly haircuts are used in the farmer model as shown in figure 4 to highlight the different subtypes of object which is a dimension the expert ignores in expressing.

e) Worse.: In this section we describe why some of the models faired worse than the expert in our subjective judgement. In the farmer model, arguably the expert theme is slightly more intuitive than the magic layout theme, because in the expert theme farmer was identified as a seperate object than the *Possession* elements such as fox, chicken and grain which makes semantic sense. However this contextual knowledge was unavailable in the model because farmer was not declared as member of a seperate type family than the other objects. If this information was available in the model declaration then the inference tool would infact do a better job than the expert model because it would both assign farmer a seperate shape according to the new shape assignment scheme, as well as distinguish the other objects using *haircuts*.

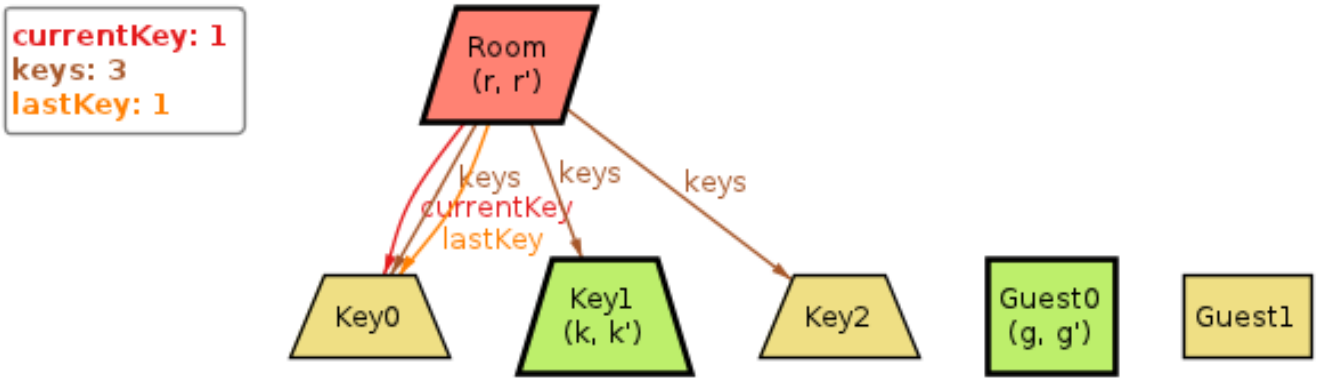
In the Messaging model the expert chose to project over the Tick type. The model also contains types NodeState and MsgState. 'Tick' and 'State' are special keywords for our projection inference, and so magic layout projected over all three of these types. If NodeState and MsgState were renamed to something not containing the 'State' keyword then magic layout would have inferred the same theme as the expert.

The railway model, which was of great interest in the preceding paper, presents a similar challenge. The expert bases

Fig. 5: Hotel

a) Hotel with Expert Theme

b) Hotel with Magic Layout Theme



the visualization on the *contains* relation, which is defined to be the inverse of the *on* relation. The *on* relation is suppressed in the expert's theme. All of the constraints in the model are written in terms of the *on* relation. If the model had only the *contains* relation, which would not be a difficult revision of the model, then magic layout would produce the same visualization as the expert.

In the *Handshake* model, the expert chose to suppress the edge labels due to the high number of edges in a typical instance. Magic layout performs only static analyses: it does not look at individual instances, and so does not make decisions on that basis.

In the *Lists* model the expert used deep semantic understanding of the model to choose to display the *prefixes* relation as attributes rather than as edges. There is no simple syntactic modification of this model that would allow magic layout to infer this setting.

In conclusion, the Handshake and Lists models are the only ones in this collection for which it is not, in principle, possible for magic layout to do the right thing. Humans will always have a better semantic understanding of Alloy models, but in most cases such a deep understanding is not necessary to produce a good visualization.

V. Related Work

The area most closely related to our work is counter-example visualization in model checking. Alloy is sometimes considered as a model checker. The main difference is that most model checking tools work with some temporal logic, whereas Alloy works with a relational logic. Temporal logic counter examples are sometimes visualized as state charts (*e.g.*, [3, 4]), or as message sequence charts [5], or in other ways (*e.g.*, [6]). The key difference here is that none of these visualization techniques for temporal logic counter examples involve settings that the end user must customize in order to get a reasonable visualization. Because Alloy is used to specify a wide variety of systems, each Alloy specification requires its own visualization settings. The purpose of the Magic Layout feature is to infer those settings from a static analysis of the specification. In this paper we have made significant improvements to the Magic Layout inference, as well as making other improvements to the Alloy visualizer, such as node stability through projection frames.

Our work is distinguished from imperative program visualization in two ways. First, our work is not about visualizing the syntax of the Alloy specification. Many program visualization systems are concerned with the source code of the program [7]. Second, our work is not concerned with visualizing the execution (or analysis process) of the Alloy specification. Our work is solely concerned with visualizing the output of the analysis/execution, but not the analysis/execution process itself. The output of an imperative program can be in many forms, and to our knowledge there is not a generic visualization technique for this.

Logic programs, such as those written in Prolog, are more similar to Alloy specifications than are imperative programs. But the different emphasis in visualization remains: systems such as The Transparent Prolog Machine [8, 9] or

Logicharts [10] are used to visualize the syntax and execution of Prolog programs, and we are concerned only with the results of the computation, neither the syntax of the input nor the process of execution.

There has been work in visualizing temporal logic formulas (*e.g.*, [11]). Our work, by contrast, is not concerned with visualizing the syntactic elements of Alloy specifications.

There is a rich literature on automatic layout algorithms for graphs that is complementary to our work. Our work assumes that the Alloy visualizer is using some graph layout algorithm — at present, it uses the classic Sugiyama algorithm [12].

VI. Conclusion

One of the main features of the Alloy Analyzer is its ability to produce examples and counter-examples to illustrate the user's specifications. The Alloy Visualizer, and its Magic Layout feature, are popular ways for users to inspect these examples and counter-examples. In this paper we have made improvements to both the base visualizer and to the magic layout inference algorithm.

We made three main enhancements to the base visualizer. First, we developed a dynamic scheme to compute node positions that remain stable throughout all frames of a projection. This was a feature strongly requested by Alloy users. Second, we added a new visual dimension to the visualizer, node *accents* or *haircuts*. This extra visual dimension is useful for identifying sibling types in a type hierarchy. In the past colour was sometimes used for this identification. Having this extra dimension frees colour to be used for highlighting changes in state, which is a way that experts often use colour in dynamic models. Finally, we added a number of new quadrilateral shapes to the visualizer's vocabulary. Quadrilaterals are good at displaying long labels and are compatible with the new haircut accents.

We improved the Magic Layout inference algorithm to more closely match what expert Alloy users do. In some cases Magic Layout now exceeds the experts. One of the main differences between the old Magic Layout and the expert was the use of colour: old Magic Layout used colour to distinguish types, whereas experts often use colour to distinguish changes in state. Our improved Magic Layout uses colour like the experts do, to distinguish changes in state. It was not obvious to us at the outset of this project that this goal could be achieved. We also improved the inference algorithms for determining projection and when to display relations as attributes rather than as edges.

The improvements to the projection inference taught us something that we had not previously known about visualizing Alloy models: projecting over multiple types can be useful. It is not clear what such a projection means in the general case, but projecting over singleton types can make the visualization clearer and more obvious to the viewer. That was a surprising result that we were not expecting.

In the original Magic Layout paper [1] the standard of comparison was against the default theme (*i.e.*, doing nothing). That first version of Magic Layout was better than the default in almost, but not all, cases. Our improved version of Magic Layout now meets or exceeds the default in all cases.

Through this work Magic Layout has matured to the point that it, in principle, meets or exceeds the expert themes on all but two of the models that come bundled with the Alloy Analyzer. (There are three models for which Magic Layout requires some simple syntactic changes to the models for it to be able to match the experts.) It is surprising how few cases actually require a deep human semantic understanding of the model in order to visualize well.

There are still more possibilities for improvement in the Alloy visualizer. For example, Zave's models of the Chord distributed hashtable protocol (*e.g.*, [13]) do not display well with the hierarchical layout algorithm currently used by the visualizer. A circular layout algorithm would be better. We are investigating alternative layout algorithms — and the inferences necessary for Magic Layout to help the user select the most appropriate algorithm.

References

[1] D. Rayside, S. Chang, S. Dennis, S. Seater, and D. Jackson, "Automatic visualization of relational logic models," in *First Workshop on the Layout of (Software) Engineering Diagrams (LED'07)*, 2007.

[2] K. Androutsopoulos, D. Binkley, D. Clark, N. Gold, M. Harman, K. Lano, and Z. Li, "Model projection: simplifying models in response to restricting the environment," in *Software Engineering (ICSE), 2011 33rd International Conference on*, may 2011, pp. 291 –300.

[3] Z. Manna, A. Browne, H. Sipma, and T. E. Uribe, "Visual abstractions for temporal verification," in *AMAST*, ser. Lecture Notes in Computer Science, A. M. Haeberer, Ed., vol. 1548. Springer, 1998, pp. 28–41.

[4] M. Ben-Ari, "Video demonstration of the Erigone/Spin visualizer," 2011. [Online]. Available: http://screencast.com/t/HNQ5l1lijXsJ

[5] S. Merz, "Model checking: A tutorial overview," in *Proceedings of the 4th Summer School on Modeling and Verification of Parallel Processes*, ser. MOVEP '00. London, UK, UK: Springer-Verlag, 2001, pp. 3–38. [Online]. Available: http://dl.acm.org/citation.cfm?id=646410.692520

[6] A. Gurfinkel and M. Chechik, "Proof-like counter-examples," in *TACAS*, ser. Lecture Notes in Computer Science, H. Garavel and J. Hatcliff, Eds., vol. 2619. Springer, 2003, pp. 160–175.

[7] P. Caserta and O. Zendra, "Visualization of the Static Aspects of Software: A Survey," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 7, pp. 913–933, Jul. 2011.

[8] M. Eisenstadt and M. Brayshaw, "The Transparent Prolog Machine (TPM): an execution model and graphical debugger for logic programming," *Journal of Logic Programming*, vol. 5, no. 4, pp. 277–342, 1988.

[9] M. Eisenstadt, M. Brayshaw, and J. Payne, "The Transparent Prolog Machine: visualising logic programs," 1991.

[10] Y. Adachi, K. Tsuchida, T. Imaki, and T. Yaku, "Logichart — Intelligible Program Diagram for Prolog and its Processing System," in *Tenth Workshop on Logic Programming Environments (WLPE)*, ser. Electronic Notes in Theoretical Computer Science, vol. 30, no. 4, 1999, pp. 276–288.

[11] A. D. Bimbo, L. Rella, and E. Vicario, "Visual specification of branching time temporal logic," in *Proceedings of the 11th International IEEE Symposium on Visual Languages (VL'95)*, 1995.

[12] K. Sugiyama, S. Tagawa, and M. Toda, "Methods for visual understanding of hierarchical system structures," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 11, no. 2, pp. 109 –125, feb. 1981.

[13] P. Zave, "Using lightweight modeling to understand chord," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 2, pp. 49–57, Mar. 2012. [Online]. Available: http://doi.acm.org/10.1145/2185376.2185383