An Interactive Derivation Viewer

Steven Trac, Yury Puzis, Geoff Sutcliffe [1]

Department of Computer Science
University of Miami
Coral Gables, USA

Abstract

This work describes the *Interactive Derivation Viewer* (IDV) tool for graphical rendering of derivations that are written in the TPTP language. IDV provides an interactive interface that allows the user to quickly view various features of the derivation. A particularly novel feature of IDV is its ability to provide a synopsis of a derivation by identifying *interesting* lemmas within a derivation, and hiding less interesting intermediate formulae. IDV is deployed online as part of the SystemOnTPTP interface, thus providing ready access via any web browser.

Key words: Derivation viewer, Proof synopsis

1 Introduction

The proofs output by automated reasoning systems provide useful information to system users, e.g., the proof structure, lemmas that may be useful in future proofs, which axioms are most used, etc. Even derivations that do not form completed proofs are of interest, as they may provide insights leading to changes in the problem formulation or the system application, that result in a proof being found - automated reasoning systems are often debugged in this way. However, the proofs output by automated reasoning systems are often unsuitable for human consumption. For first-order automated theorem proving (ATP) systems, the reasons include:

- The conversion of problems stated in "natural" first-order form (FOF) to clause normal form (CNF).
- The use of proof by contradiction, which introduces formulae that are not logical consequences of the axioms.
- The use of fine grained inference steps, such as binary resolution, that exaggerate the size of a proof.

[1] Email: strac@mail.cs.miami.edu

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

Several types of tools have been developed to make the output from ATP system easier for humans to understand. These include graphical renderings of derivations [10], structuring of proofs by identification of lemmas [1], translation of resolution refutation proofs to natural deduction proofs [4], and full translation of proofs to natural language form [2]. This work describes the *Interactive Derivation Viewer* (IDV) tool for graphical rendering of derivations that are written in the TPTP [13] language [12]. IDV provides an interactive interface that allows the user to quickly view various features of the derivation. A particularly novel feature of IDV is its ability to provide a synopsis of a derivation by identifying *interesting* lemmas within a derivation, and hiding less interesting intermediate formulae. IDV is deployed online as part of the SystemOnTPTP interface [11], thus providing ready access via any web browser.

Section 2 describes the basic IDV tool and it's rendering process. Section 3 describes the production of proof synopses. Section 4 explains how IDV is deployed on the web, and provides an illustrative application. Section 5 concludes and discusses future developments planned for IDV.

2 Basic IDV

A derivation is a directed acyclic graph (DAG) whose leaf nodes are formulae (possibly derived) from the input problem, whose interior nodes are formulae inferred from parent formulae, and whose root nodes are the final derived formulae. For example, a CNF refutation proof is a derivation whose leaf nodes are the clauses formed from the axioms and the negated conjecture, and whose root node is the $false$ clause. The information required to record a derivation is, minimally, the leaf formulae, and each inferred formula with references to its parent formulae. More detailed information that may be recorded and useful includes: the role of each formula, e.g., axiom, conjecture, plain derived, etc; the name of the inference rule used in each inference step; sufficient details about each inference step to deterministically reproduce the step; and the semantic relationship of each inferred formula with respect to its parents, e.g., logical consequence, counter theorem, etc. The TPTP language is sufficient for recording all this, and more.

A derivation written in the TPTP language is a list of annotated formulae. Each annotated formula contains a name, a role, the logical formula, a source record, and a field for recording arbitrary useful information, as required for user applications. The source of each inferred formula is an `inference` record containing the inference rule name, a `status` record containing the semantic relationship of the formula to its parents as an SZS ontology value [14], and a list of references to its parent formulae.

IDV takes a derivation in the TPTP language and renders the DAG using Java's `Swing` components. IDV can run as a standalone application, or as a web browser applet; this description focuses on the web option, because it

provides ready (remote) access to IDV without any installation required.

2.1 Interface

Figure 1 shows the rendering of the derivation output by the ATP system EP 0.91 [9] for the TPTP problem `PUZ001+1`.[2] The IDV window is divided into three panes: the top control strip pane provides control buttons and sliders, the main middle pane shows the rendered DAG, and the bottom pane gives the text of the annotated formula for the node pointed to by the mouse.

The buttons and sliders in the control strip pane are, from left to right:

- Zoom in - zooms in 50%
- Fit vertical - scales the rendering to fit the height of the middle pane
- Fit horizontal - scales the rendering to fit the width of the middle pane
- Zoom out - zooms out 50%
- Display height - sets the number of text lines in the bottom pane
- Synopsis level - sets the minimal interestingness level for display - see Section 3.
- Redraw - redraws the derivation. This is typically used after extracting a synopsis - see Section 3.
- Synopsis undo - sets the minimal interestingness level to its previous value.
- Synopsis redo - sets the minimal interestingness level to its next value, after any undo steps.
- About button

The rendering of the derivation DAG uses shape and color to visually provide information about the derivation. Each node corresponds to a formula in the derivation, with FOF nodes outlined in black and CNF nodes outlined in orange. The role of the formulae is indicated by the shape of the node: triangle for axioms, hexagons for lemmas, inverted trapezium for hypotheses, house for conjectures, inverted house for negated conjectures (as done when converting a FOF problem to CNF), circle for plain derived formulae, and square for $false$ formulae. A node may be annotated above with a = sign in a circle to indicate that equality reasoning was used in its inference, e.g., a paramodulation inference. A node may be annotated inside at the top with a red circle to indicate that the formula is not a logical consequence of its parents, e.g., in Skolemization and splitting inferences, as indicated by the SZS status. A node may be annotated below with a red triangle to indicate that it is the parent of a splitting inference, e.g., an explicit split as implemented by SPASS [15] or a pseudo-split as implemented by Vampire [7,8] or E [9].

[2] `PUZ001+1` is the "Aunt Agatha" problem, which describes a scenario in which three people live in a mansion, and Aunt Agatha is killed. The goal is to prove that Aunt Agatha killed herself. All TPTP problems, their solutions, and IDV renderings of the solutions, are available online via `http://www.tptp.org/` - follow the Problems link to reach the problems, the TSTP link to reach the solutions, and the View IDV Tree link at the top of any solution page (that has the solution in TPTP format) to generate the IDV rendering.

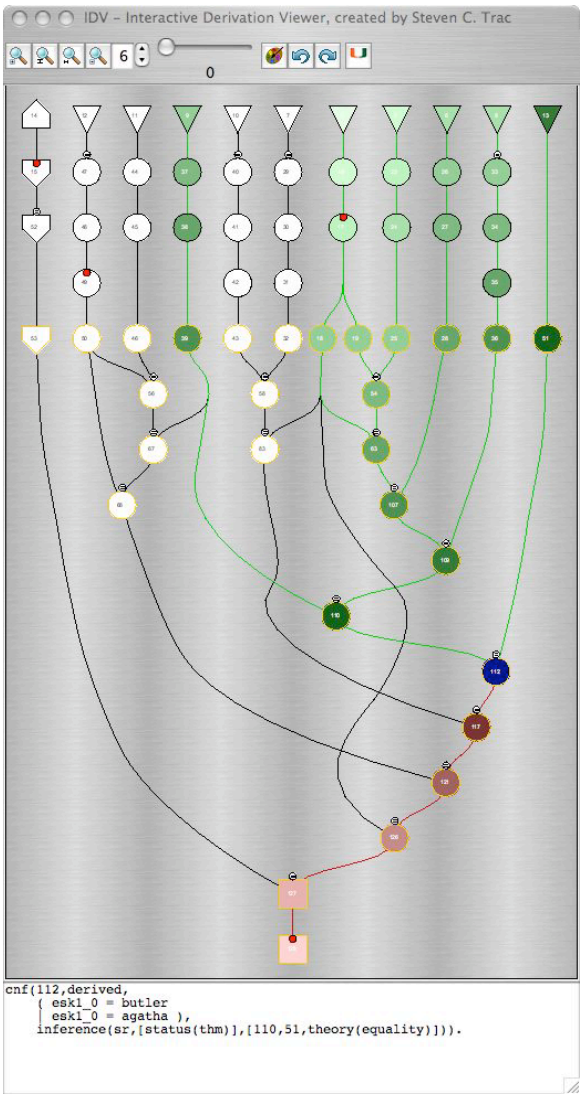

Fig. 1. EP's Proof by Refutation of PUZ001+1

The user can interact with the derivation rendering in two ways. First, moving the mouse over any node causes the annotated formula corresponding to the node to be shown in the bottom pane. At the same time, the moused-over node is highlighted in blue, all nodes leading down from leaf nodes into the moused-over node are highlighted in green, and all nodes leading down from the moused-over node to root nodes are highlighted in red. The effect is evident in Figure 1. The green highlighting shows from which formulae the moused-over node is derived, and the red highlighting shows which formulae are derived from the moused-over node. The intensity of the highlighting decreases according to the minimal path length from the moused-over node to the highlighted node. This allows easy differentiation between closer and more distant ancestors and descendants. A particularly useful effect is to identify which axioms (leaf nodes) are the closest ancestors. The second form of in-

teraction is to click on any node. This creates a pop-up window containing the annotated formulae of the clicked node and its parents, as shown in Figure 2. The annotated formula of the clicked node is shown twice once above the parents and again below, allowing for bidirection reading of the inference step.

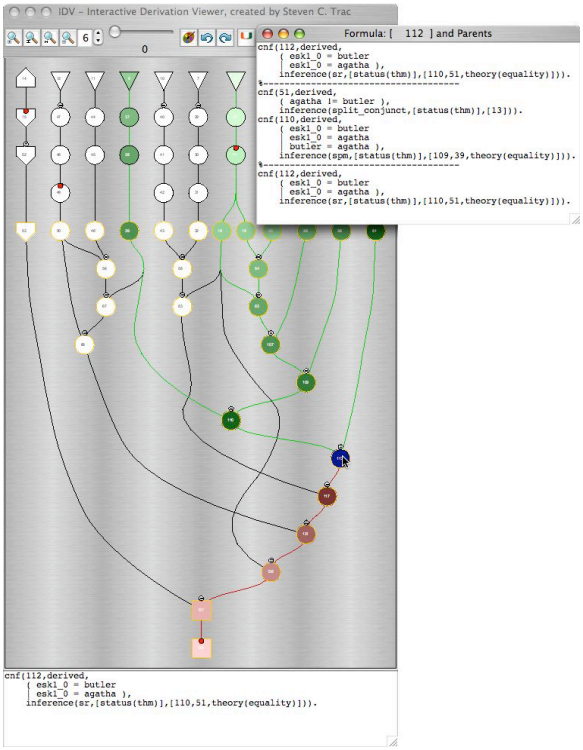

Fig. 2. Pop-up in EP's Proof by Refutation of `PUZ001+1`

2.2 Implementation

IDV reads in a derivation in TPTP format. It is sensitive to the form of the formulae, either FOF or CNF. The rendering is performed in two phases: the first phase determines the layout of the DAG nodes and edges, and the second phase implements the graphical display of the derivation DAG.

The layout of the DAG is determined in a five-pass algorithm, similar to that in [3]. The first pass assigns a rank to each node. The rank is the level of the node in the rendered tree, with the top row containing the nodes at level 1, and increasing downwards. The rank is used in the third pass to determine the Y coordinate of each node. The first row of FOF nodes (leaf nodes) are placed in the first rank. A depth first search (DFS) is then used to assign increasing rank to the rest of the FOF nodes. Next the first row of CNF nodes are placed in the rank above the maximum FOF node rank. Finally, the DFS algorithm is run again to assign increasing rank to the rest of the CNF nodes. After ranks are assigned to all nodes, the edges are partitioned as

follows: If an edge connects two nodes that are more than one rank apart, the edge is replaced by a chain of virtual nodes and edges. The virtual nodes are given incremental ranks between the two end nodes' ranks. If a non-virtual node has more than one chain of virtual nodes leading down from it, the chains are combined as far as possible, dividing immediately above the end nodes of the chains.

The second pass directly follows the algorithm from [3], setting the left-to-right vertex order within ranks by an iterative heuristic incorporating a weight function and local transpositions to reduce edge crossings. The introduction of the virtual nodes at each rank guarantees that edge crossings can only occur between adjacent ranks.

The third pass sets an initial X coordinate and final Y coordinate for each node. The rank with the largest number of nodes determines the maximum width of the graph. With the left-to-right vertex ordering within ranks from the second pass, equidistant X coordinates are given to the nodes in each rank, between 0 and maximum width of the graph. The final Y coordinate is based linearly on the nodes' ranks.

The fourth pass finds the optimal X coordinate for each node. For this pass spring embedding is used.[3] Spring embedding is a graph drawing technique that models a graph as a system of springs and then uses energy minimization to space the nodes. The following forces are balanced: edge spring force for keeping edges at a certain length, node-to-node repulsive forces to keep nodes from being too close, gravity force that keeps all edges pointing downwards, and repulsive boundary forces to keep the nodes from spreading too far apart horizontally. After the fourth pass the X and Y coordinates are fixed - the nodes cannot be moved by user interaction.

The fifth pass generates Bezier curves to draw edges between nodes. If two non-virtual nodes are connected by a chain of virtual nodes, then the chain of virtual nodes is used to plot the points of the Bezier curve.

The layout determined by the first phase does not guarentee that nodes will not overlap (or hence, given the use of virtual nodes to guide edge generation, that edges will not pass through nodes). The extent to which node overlaps are avoided is determined by the number of iterations in the spring embedding. The number of iterations in the current implementation has been found to be sufficient to avoid most overlaps. After the layout has been determined, the interface and DAG are rendered.

IDV is implemented in Java, mainly using basic `Swing` components. The TPTP formulae are read in using `StreamTokenizer`. The IDV window is a `JFrame`, and the rendering is a `JPanel`. A `MouseMotionListener` is used in the `JPanel` to detect when the mouse moves over a node, to implement the node coloring feature. A `MouseListener` is used in the `JPanel` to detect when a node is clicked, to implement the pop-up window feature. The `JFrame` is

[3] Thanks to Christian Duncan for providing the original spring embedding code.

implemented with `ActionListener` and `ChangeListener` to detect the user's manipulations in the control strip.

3 Derivation Synopses

As mentioned in Section 1, one of the features of derivations output by ATP systems is the use of fine grained inference steps such as binary resolution, which exaggerate the size of a proof. Derivations output by humans typically use coarser grained inference steps, leaving intermediate steps "to the reader". The inferred formulae of such coarser grained inference steps are logical consequences of their leaf ancestors, at various levels of saliency - humans often single out certain of the logical consequences to be specifically designated as lemmas. By considering only those logical consequences above a certain level of saliency (hiding those below that level), a synopsis of the detailed derivation is formed. In a synopsis the lowest visible ancestors of a hidden formula become the parents of the highest visible descendents. A synopsis hides the fined grained inference steps and the intermediate formulae, thus making is easier for a user to grasp an overview of the proof. The user may later choose to examine the details. Synposes may similarly be used to summarize extremely large derivations.

IDV is able to form a synopsis of a proof by CNF refutation. This is achieved by rating the *interestingness* of inferred CNF formula, and hiding the nodes whose formula rating is below a user specified threshold. The interestingness rating of inferred CNF formulae is computed by the AGInT system [6] - see Section 3.3, and the user sets a threshold using the slider in the control strip pane.

3.1 Interface

The interestingness of each formula is a value in the range 0.0 to 1.0. Some formulae have a preset interestingness rating: leaf formulae are set at 1.0, the topmost CNF formula are set at 1.0, the intermediate formulae between leaf FOF formulae and topmost CNF formulae are set at 0.0, all formulae derived from the negation of a conjecture are set at 1.0, and root formulae are set at 1.0. The interestingness of values for the other formulae, i.e., the internal CNF formulae of the derivation, are computed by the AGInT system, as described in Section 3.3. Initially the threshold slider in the top pane is set to an interestingness value of 0.0, and all nodes are displayed in the rendering. As the slider is moved up the interestingness threshold increases, and nodes whose formula rating is below the threshold are hidden. Figure 3 shows the derivation in Figure 1, with a interestingness threshold of 0.5.

After extracting a synopsis it is possible to zoom in, rendering only the visible nodes. This is done in IDV with the redraw button in the control strip. Figure 4 shows the synopsis derivation rendering of Figure 3. After a

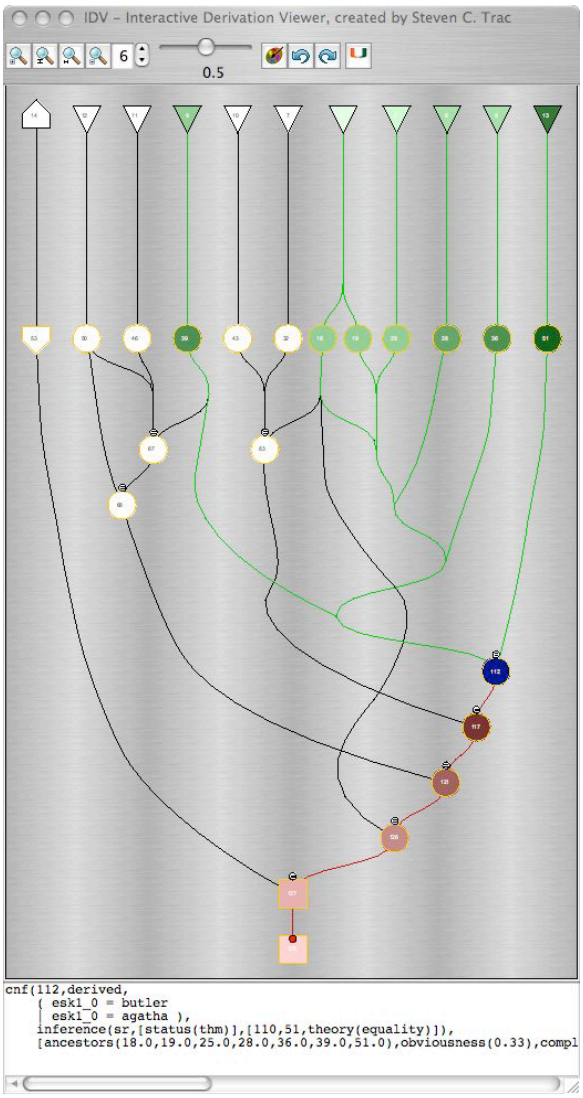

Fig. 3. Interesting nodes of EP's Proof by Refutation of PUZ001+1

redraw the threshold slider may be moved and the derivation redrawn again, to produce a different level of synopsis. Note that after a redraw, if the threshold is moved to below the interestingness level used for the redraw, the hidden nodes are do not immediately become visible - another redraw is required. The user is warned of this state by the threshold value being shown in red. Sequences of redraws can be undone and redone using the synopsis undo and redo buttons.

While using the slider to adjust the interestingness level, the layout of the nodes does not change - simply more or less of the nodes are hidden. This provides a identity mental map of the derivation (a *mental map* is the user's memory of the rendering [5]). When redrawing a synopsis it important to maintain the mental map as far as possible. To this end, all nodes that are not hidden in a synopsis are kept in the same order as in the original. The

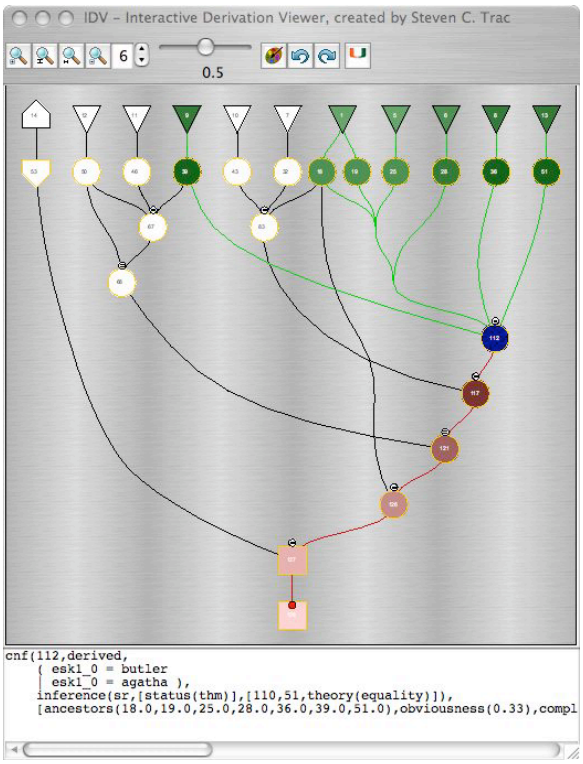

Fig. 4. Synopsis of EP's Proof by Refutation of `PUZ001+1`

Bezier curves that connect the visible nodes are recomputed, but maintain the same form as in the original.

As mentioned in Section 2.1, when a node is clicked a pop-up window appears containing the annotated formula of the node and its parents. After a reddraw, the parents shown in a pop-up window are the parents of the formula in this rendering, i.e., they might not be the formulae's original parents. If some parent information is different than the original, then the pop-up window informs the user of this.

3.2 Implementation

Interestingness ratings are stored in a record in the `useful_info` fields of annotated formulae. There are two ways for formulae to have interestingness ratings. First, the annotated formulae input to IDV may already have interestingness values. Second, the input formulae do not have interestingness ratings, and the AGInT system has to be called by IDV. In this case AGInT is called as soon as the user uses the threshold slider in the control pane.

When the redraw button is clicked by the user, the derivation synopses is rendered as follows: First, non-virtual nodes in the DAG are set to be interesting if their interestingness rating is greater than the threshold value, and all virtual nodes are set to be uninteresting. Each rank is then checked to see if it contains any interesting nodes. If a rank contains at least one

interesting node the rank is retained, otherwise the rank is empty and all nodes in ranks below are moved up a rank (i.e., their rank is decremented). The original rank of each node is stored for redrawing purposes. After the ranks are updated the Y coordinates of the nodes in the retained ranks are updated, as in Section 2.2. Finally, the Bezier curves are updated to uniquely connect each interesting node to its closest interesting ancestors, which are found using a DFS search up the DAG. All uninteresting nodes remain hidden after a redraw.

When the synopsis undo/redo button is clicked, the current interesting threshold value changes to the last value pushed onto the undo/redo stack and above redraw procedure is called.

3.3 Interestingness Ratings

The interestingness ratings of derived CNF formulae in a derivation are computed by the *runtime filter* and *static ranker* components of the AGInT system. AGInT is a system that discovers interesting theorems of a given set of axioms. AGInT uses a deductive approach to discovery - it uses an ATP system to generate CNF logical consequences of the given set of axioms, filters the logical consequences to extract interesting theorems, and then computes an interestingness rating for each theorem. This basic process takes place in the context of an outer level control loop that regularly refocuses the generation of logical consequences, thus enabling AGInT to proceed deeply into the search space of logical consequences. Details are given in [6].

In the context of IDV, the derived CNF formulae of a derivation are given to AGInT as the logical consequences of the topmost CNF formulae (i.e., the topmost CNF formulae are considered to be the axioms from which the formulae are derived). AGInT's runtime filter and static ranker are used to compute interestingness values for the formulae. Figure 5 shows the combined architecture of these two components.

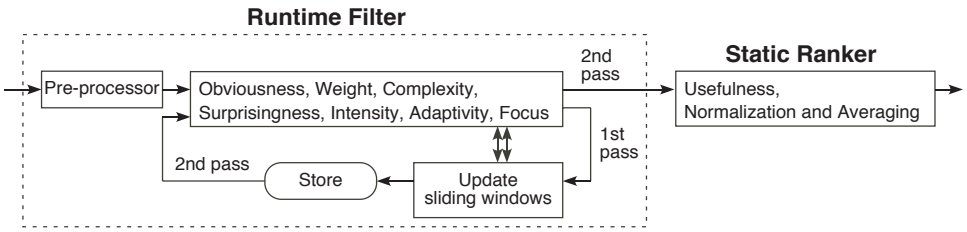

Fig. 5. Architecture of AGInT's Runtime Filter and Static Ranker

The task of the runtime filter is to aggressively filter out and discard boring formulae. Each formula must first pass the pre-processor, and must then pass the majority (i.e., at least four) of the seven filters: obviousness, weight, complexity, surprisingness, intensity, adaptivity, and focus. Each filter maintains a sliding window defined by the best distinct scores from the filter's evaluation of the formulae seen so far. The upper and lower bounds of each window are

initialized to the worst possible score for that filter. If an incoming formula is scored equal to or better than the lower bound, it passes the filter, and the score is used to update the window. Initializing the upper and lower bounds to the worst possible score allows all formulae through until the window starts sliding up. As a result some boring formulae early in the stream may pass the runtime filter. Therefore the formulae that pass the runtime filter in the first pass are stored, and after all formulae have been processed the stored formulae are filtered again, with the windows fixed from the first pass. This removes any that do not meet the final lower bounds.

The individual filters are as follows:

Pre-processor: The preprocessor detects and discards obvious tautologies, e.g., clauses that contain an atom and it's negation, and clauses containing a *true* atom.

Obviousness: Obviousness estimates the difficulty of proving a formula. The obviousness score of a formula is the number of inferences in its derivation. A higher score is better.

Weight: Weight estimates the effort required to read a formula. Formulae that contain very many symbols (variables, function and predicate symbols) are less interesting. The weight score of a formula is the number of symbols it contains. A lower score is better.

Complexity: Complexity estimates the effort required to understand a formula. Formulae that contain very many different function and predicate symbols, representing many different concepts and properties, are less interesting. The complexity score of a formula is the number of distinct function and predicate symbols it contains. A lower score is better.

Surprisingness: Surprisingness measures new relationships between concepts and properties. Formulae that contain function and predicate symbols that are seldom seen together in a formula are more interesting. The symbol-pair surprisingness score of a pair of symbols is the number of axioms that contain both symbols divided by the number of axioms that contain either symbol. The surprisingness score of a formula is the sum of the symbol-pair surprisingness scores, over all pairs of distinct symbols in the formula. A lower score is better.

Intensity: Intensity measures how much a formula summarizes information from the leaf ancestors in its derivation tree. The plurality score of a formula (or set of formulae) is number of function and predicate symbols in the formula divided by the number of distinct function and predicate symbols in the formula. The intensity score of a formula is the plurality of its leaf ancestors divided by the plurality of the formula itself. A higher score is better.

Adaptivity: Adaptivity measures how tightly the universally quantified variables of a formula are constrained. The adaptivity score of a clause is the number of distinct variables in the clause divided by the number of variable occurrences in the clause. A lower score is better.

Focus: Focus measures the extent to which a formula is making a posi-

tive or negative statement. Let FPL and FNL be the fractions of positive and negative literals in a clause. The focus score of a clause is $1 + FPL * log_2(FPL) + FNL * log_2(FNL)$. Clauses with up to three literals are considered to have perfect focus because their polarity distribution is limited. A higher score is better.

The formulae that pass the runtime filter are considered to be interesting. The task of the static ranker is to compute a final interestingness rating for the formulae. This is done in two phases: first a usefulness score is computed for each formula, and second, all the scores are individually normalized and then averaged.

Usefulness: Usefulness measures how much an interesting formula has contributed to proofs of further interesting formulae, i.e, its usefulness as a lemma. The usefulness score of a formula is the ratio of its number of interesting descendents (i.e., descendents that have passed the runtime filter) over its total number of descendents. A higher score is better.

Normalization and Averaging: The scores of the formulae, from each of the runtime filter and static evaluations, are normalized into the range 0.0 to 1.0. The formulae with the worst score are given a final score of 0.0, the formulae with the best score are given a final score of 1.0, and all other scores are linearly interpolated in between. If the worst and best score of a particular filter are equal, then that filter does not differentiate between the formulae, and those scores are removed. The remaining scores of each formula are averaged to produce a final interestingness rating.

4 Deployment and an Application

IDV is deployed online as part of the SystemOnTPTP interface at

`http://www.tptp.org/cgi-bin/SystemOnTPTPFormMaker`

The IDV code is wrapped as a web browser applet, and all computation for the rendering is done on the client side. The annotated formulae that constitute the derivation to be rendered may be passed to the applet as a parameter within the `<APPLET>` tags in the encompassing web page, or retrieved from a URL specified as a parameter within the `<APPLET>` tags. The AGInT code is deployed as a server side `cgi-bin` script, and is invoked by the IDV code via a `POST` call when interestingness ratings are required. Figure 6 shows the deployment architecture.

IDV has been used to analyze proofs of theorems, to identify key steps in the proofs. As an example EP's proof of the TPTP problem `SET615+3` is considered. This problem proves that for any three sets X, Y, and Z, $(X \cup Y) \backslash Z = (X \backslash Z) \cup (Y \backslash Z)$. Figure 7 shows EP's derivation DAG - clearly a hairy beast which is hard to comprehend as a whole. Figure 8 shows a synopsis of the derivation. It is very easy to see which nodes are key points in the synopsis, e.g., the one with the blue (darkest) coloring has the formula $X = (X \cup Y) \backslash (Y \backslash X)$. Another key node has the formula $X \cup (Y \backslash Z) =$

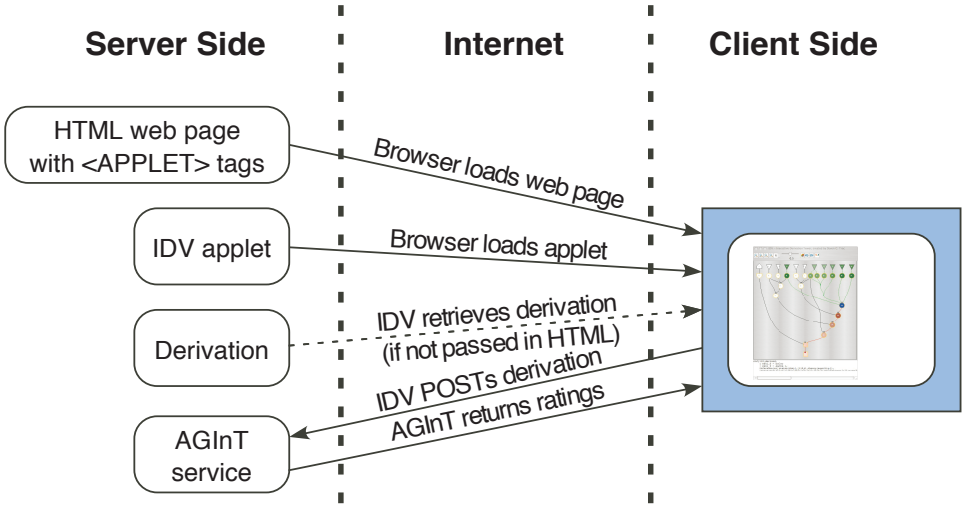

Fig. 6. Deployment of IDV

$(X \cup Y)\backslash(Z\backslash X)$.

5 Conclusion

This paper has presented the design, implementation, deployment, and application of an interactive derivation viewer, implemented as the IDV tool. IDV provides strong visual information showing the structure of a derivation, with original details available as text. IDV provides interactive features that enable a user to visually highlight and examine salient parts of a derivation. In particular, the ability to extract proof synopses sets IDV apart from other existing derivation viewers. The use of "interestingness ratings", which are artificially intelligently determined, to provide a sliding scale of proof synopsis, is particularly powerful and certainly highly novel. The online deployment makes IDV easily available to users (who use the TPTP language for their derivations), without any need for software installation.

Future work planned for IDV includes finer grained synopsis of the FOF to CNF parts of derivations, which are currently considered to be not interesting at all. Future work on the implementation includes tighter integration with the SystemOnTPTP interface, so that interestingness ratings are computed in advance of their need, improving the performance on extremely large derivations, and improving the Bezier curve drawing in synopses of very large derivations. When the features have been optimized and implementation is stable, user evaluation will also be desirable.

References

[1] J. Denzinger and S. Schulz. Recording, Analyzing and Presenting Distributed Deduction Processes. In H. Hong, editor, *Proceedings of the 1st International*

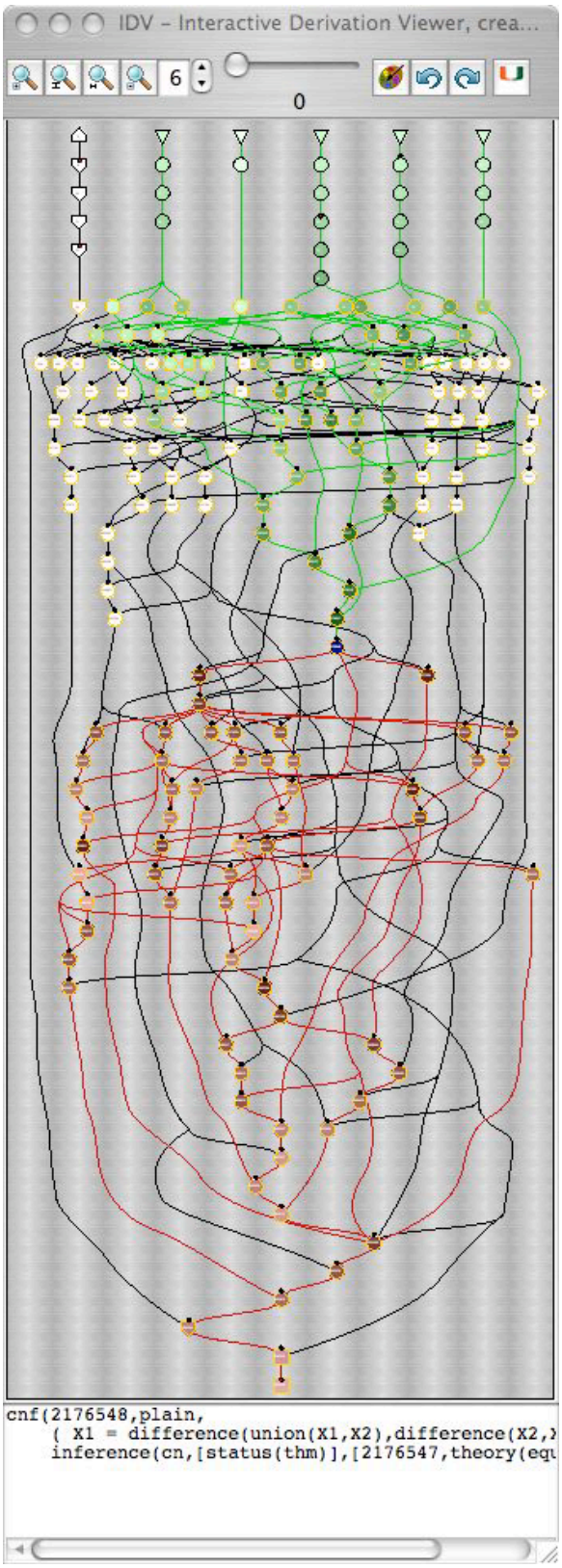

Fig. 7. EP's Proof by Refutation of SET615+3

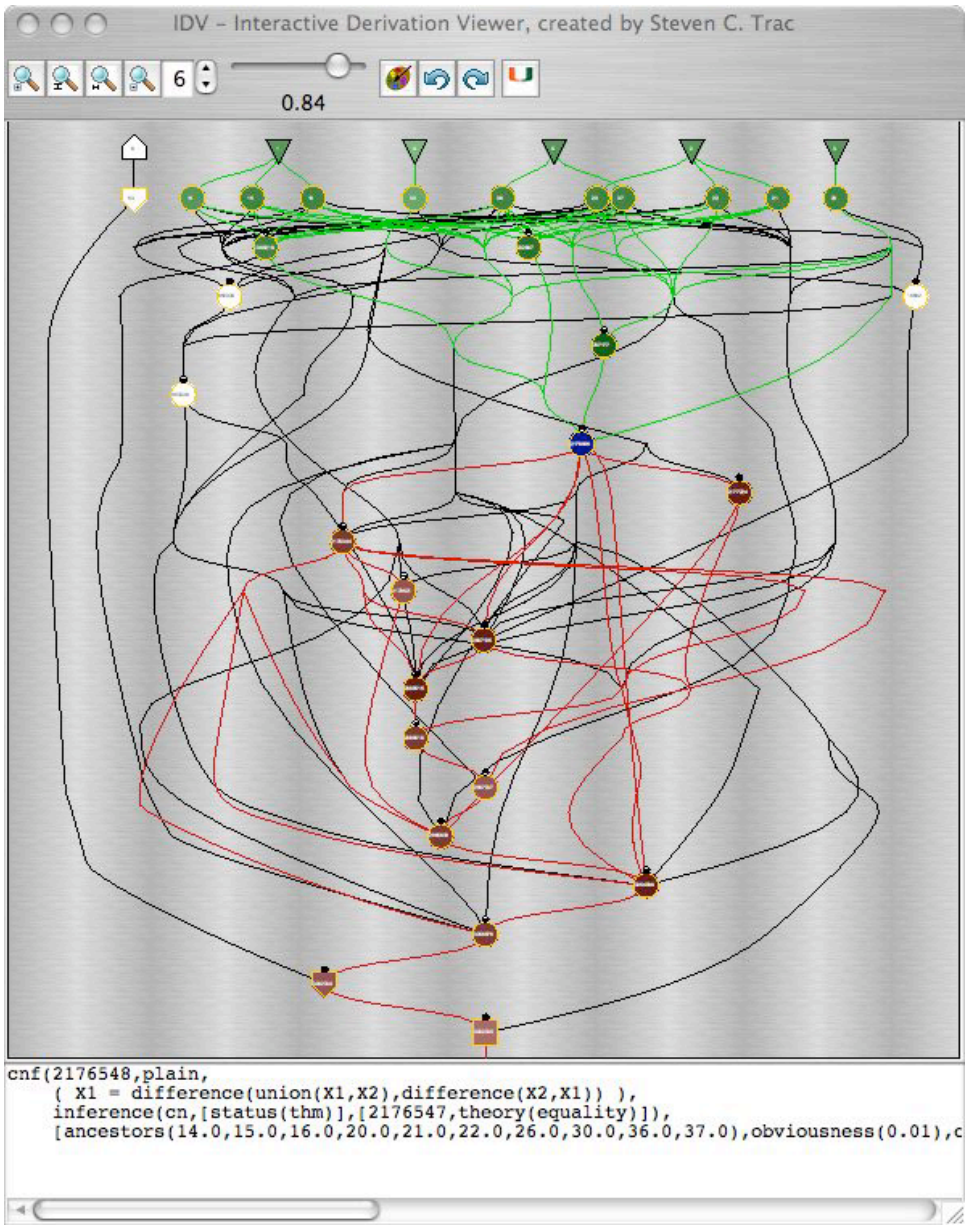

Fig. 8. Synopsis of EP's Proof by Refutation of `SET615+3`

Symposium on Parallel Symbolic Computation, number 5 in Lecture Notes Series on Computing, pages 114–123. World Scientific Publishing, 1994.

[2] A. Fiedler. P.rex: An Interactive Proof Explainer. In R. Gore, A. Leitsch, and T. Nipkow, editors, *Proceedings of the International Joint Conference on Automated Reasoning*, number 2083 in Lecture Notes in Artificial Intelligence, pages 416–420. Springer-Verlag, 2001.

[3] E. Gansner, E. Koutsofios, S. North, and K-P. Vo. A Technique for Drawing Directed Graphs. *Software Engineering*, 19(3):214–230, 1993.

[4] A. Meier. System Description: TRAMP - Transformation of Machine-

Found Proofs into Natural Deduction Proofs at the Assertion Level. In D. McAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction*, number 1831 in Lecture Notes in Artificial Intelligence, pages 460–464. Springer-Verlag, 2000.

[5] K. Misue, P. Eades, W. Lai, and K. Sugiyama. Layout Adjustment and the Mental Map. *Journal of Visual Languages and Computing*, 6(2):183–210, 1995.

[6] Y. Puzis, Y. Gao, and G. Sutcliffe. Automated Generation of Interesting Theorems. In G. Sutcliffe and R. Goebel, editors, *Proceedings of the 19th International FLAIRS Conference*. AAAI Press, 2006.

[7] A. Riazanov and A. Voronkov. Splitting without Backtracking. In B. Nebel, editor, *Proceedings of the 17th International Joint Conference on Artificial Intelligence* , pages 611–617. Morgan Kaufmann, 2001.

[8] A. Riazanov and A. Voronkov. The Design and Implementation of Vampire. *AI Communications*, 15(2-3):91–110, 2002.

[9] S. Schulz. E: A Brainiac Theorem Prover. *AI Communications*, 15(2-3):111–126, 2002.

[10] G. Steel. Visualising First-Order Proof Search. In C. Aspinall, D. Lüth, editor, *Proceedings of User Interfaces for Theorem Provers 2005*, pages 179–189, 2005.

[11] G. Sutcliffe. SystemOnTPTP. In D. McAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction*, number 1831 in Lecture Notes in Artificial Intelligence, pages 406–410. Springer-Verlag, 2000.

[12] G. Sutcliffe, S. Schulz, K. Claessen, and A. Van Gelder. Using the TPTP Language for Writing Derivations and Finite Interpretations. In U. Furbach and N. Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, Lecture Notes in Artificial Intelligence, Submitted.

[13] G. Sutcliffe and C.B. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.

[14] G. Sutcliffe, J. Zimmer, and S. Schulz. TSTP Data-Exchange Formats for Automated Theorem Proving Tools. In W. Zhang and V. Sorge, editors, *Distributed Constraint Problem Solving and Reasoning in Multi-Agent Systems*, number 112 in Frontiers in Artificial Intelligence and Applications, pages 201–215. IOS Press, 2004.

[15] C. Weidenbach, U. Brahm, T. Hillenbrand, E. Keen, C. Theobald, and D. Topic. SPASS Version 2.0. In A. Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction*, number 2392 in Lecture Notes in Artificial Intelligence, pages 275–279. Springer-Verlag, 2002.