INSTITUT FÜR INFORMATIK
Lehr- und Forschungseinheit für
Programmier- und Modellierungssprachen
Oettingenstraße 67, D–80538 München

Ludwig
Maximilians–
Universität
München

The Tableau Browser SNARKS (System Description)

Mathias Kettner and Norbert Eisinger

appeared in *Proc. 14th Int. Conf. on Automated Deduction (CADE)*, Springer LNAI 1997
http://www.pms.informatik.uni-muenchen.de/publikationen
Forschungsbericht/Research Report PMS-FB-1997-1, März 1997

The Tableau Browser SNARKS

Mathias Kettner and Norbert Eisinger

Ludwig-Maximilians-Universität München, Institut für Informatik,
Oettingenstr. 67, D-80538 München, Germany

{Mathias.Kettner | Norbert.Eisinger}@informatik.uni-muenchen.de
http://www.pms.informatik.uni-muenchen.de/software/

SNARKS is a graphical tool for the analysis of tableau proofs as generated by SATCHMO and similar deduction systems [2, 1, 3]. SNARKS comes with a built-in inference engine, but it can also be hooked to external inference engines. It is accessible through the World Wide Web.

Given a *specification* such as the sample to the right, i.e., a list of clauses in implication form, the built-in inference engine of SNARKS develops a tree whose structure corresponds to a PUHR tableau [1]. Alternatively, SNARKS can import a tree that reflects an external inference engine's deductive process on the specification, provided that this engine can produce a *trace*, a protocol of its deduction steps in a defined format. Regardless whether a tree is internally produced or imported, SNARKS offers a rich functionality to display, inspect, rearrange, and manipulate the tree interactively, including means to influence the "granularity" with which it is displayed and to compact what is not in the user's current focus of interest.

```
true ---> r(a) ; r(b) ; r(c).
r(X) ---> r(f(X)) ; p.
r(f(a)) ---> q.
r(f(f(X))) ---> false.
```

Sample Specification

SNARKS was developed as a support tool for several projects in the context of SATCHMO (e.g., compilation, generation of minimal models, generation of finite models), where developers of inference engines needed to debug their systems. But it can serve a variety of other purposes: Developers of specifications can use SNARKS to debug a specification with unexpected consequences. Users of inference engines can use SNARKS to understand and to compare the behaviour of different inference engines. Anyone can use SNARKS to give illustrative demonstrations of inference engines. Teachers and students can use SNARKS for educational purposes.

Although intended for systems of the SATCHMO family, SNARKS was designed such that its coupling with other tableau based deduction systems ought to be fairly easy. We plan experiments to extend existing inference engines such that they produce traces that enable SNARKS to import their trees.

SNARKS is implemented in Java with a Prolog server, and is not restricted to a specific platform. It can be called via the URL above.

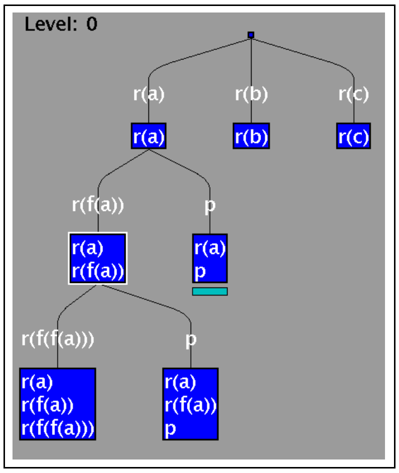
The rest of this description gives an overview of the functionality of SNARKS.

1 Creation and Expansion of Trees

One of the components associated with a node is its *database*, the set of formulae derived on the path from the root to the node. For SATCHMO-like inference engines each database is a set of ground atoms that can be understood as representing a Herbrand interpretation of the specification.

Initially, the tree consists only of a root node, whose database is empty.

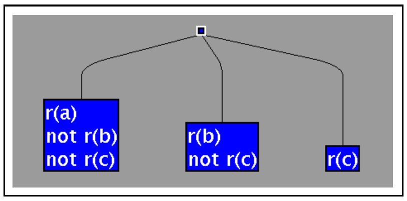
SNARKS provides an operation *"Create sons"*, which uses the built-in inference engine to expand the tree by applying clauses applicable to a leaf node. For the sample specification above, the first clause is applicable to the empty database, and its disjunctive head results in three branches of the tree. In cases where more than one clause (or clause instance) is applicable, it is possible to apply all applicable clauses simultaneously, or a subset chosen by the user, or a subset determined by a selection strategy that can be influenced by the user. Expansion of the tree need not proceed in a step-by-step fashion. The operation *"Create subtrees"* iterates tree expansion under a user-defined condition.



Example 1: Tree Expansion

Example 1 shows the tree resulting from *"Create sons"* activated for the root followed by *"Create subtrees"* activated for the first son of the root using the condition *"Expand up to level 2"*. The nodes in the example are labelled with their databases, the edges with the database increments. Some other possible labellings are described in Section 4 below.

The behaviour of the built-in inference engine can be controlled by some options. Among others, it can apply clauses with *complement splitting* [1], thus producing additional negative literals. Example 2 shows the tree (without edge labels) when the first clause is applied with complement splitting.



Example 2: Complement Splitting

Expansion steps rely on the built-in inference engine to compute additional nodes of the tree. Nodes of a tree imported from an external inference engine may also be expanded in this way, but then the resulting tree represents a mixture of the deductive processes of the two engines.

Other available operations delete nodes from the tree or reset it to the initial tree consisting of just the root node.

2 Navigation in the Tree

Many of the SNARKS commands apply to a selected node, the so-called *active* node, which is visually highlighted by a white borderline. There are several cursor key commands, mouse clicks, or menu items for moving the status of being active to another node.

In order to focus on interesting parts of the tree it is possible to restrict the display to a subtree and, within this subtree, to a specifiable range of levels.

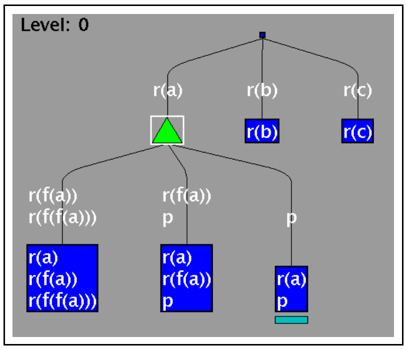
Even so the displayed parts of the tree often do not fit into a window. Snarks provides scrollbars, a standard technique for selecting the visible part of geometrically arranged data in such cases. But its horizontal scrollbar has a non-standard functionality: If all nodes of a layer are displayed inside the window, horizontal scrolling does not change their position. Only the nodes of layers that are too wide will move, and their edges will adapt accordingly. Thus, scrolling intelligently alters the geometry of the tree.

3 Hiding Nodes

Typical trees are too complex to be handled conveniently by the techniques described so far. They require means to change (temporarily) their structure, such that the user sees a compacted version that abstracts from details that are currently irrelevant. Snarks offers the concept of *hiding* a node in its parent.

A node forms a unit with all nodes it hides and all nodes the hidden nodes hide and so on. Nodes that are neither hidden nor hiding others are displayed as illustrated in the previous examples. The remaining units are displayed as triangles.

The user has a wide range of possibilities to hide and "unhide" nodes explicitly, and to specify conditions to hide and unhide nodes automatically. For example, two mouse clicks are sufficient to transform the tree such that all its leaf nodes appear under a single root that hides the other nodes.



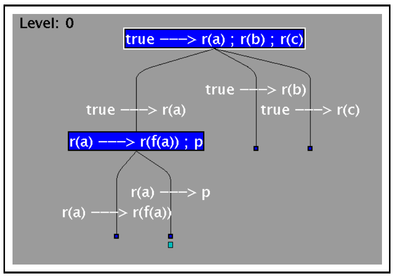
Example 3: Hiding Nodes

Example 3 shows the tree from Example 1 after the node with database {`r(a), r(f(a))`} has been hidden. Note the adjustments of the edge labels.

4 Layout

4.1 Labelling

Nodes and edges can be labelled independently with various types of information about the databases and the derivation. Example 4 shows a tree in which the nodes are labelled with the clause instances used to create their sons. The edges are labelled with the *reasons* for the database increments.



Example 4: Labelling

The display part of SNARKS can handle any format for reasons given by an external inference engine. In Example 4 each reason has the form of a definite subclause of the applied clause instance, i.e., it consist of the whole body plus the single selected head atom corresponding to the edge. This format supports some advanced functionality of SNARKS (Section 5 below).

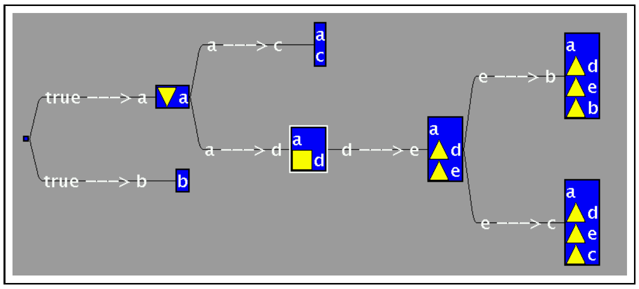
In addition to the information displayed as node labels in the tree, more detailed information about the active node is available in separate windows: a scrollable database window, a window for applicable or applied clause instances or reasons, and table representations of predicates in the node's database. These windows update their contents whenever another node becomes active.

4.2 Landscape Mode

If a level of the tree contains many nodes or if the labels are long, *landscape* mode may result in a better layout, because any text remains written from left to right. Example 5 below shows a tree displayed in landscape mode.

5 Marking Dependencies

Application of inference rules induces a dependency relationship between formulae in databases of different nodes. This relationship can be displayed by *marking* one or more formulae with a coloured box. Downward pointing triangles of the same colour then indicate the formulae on which the marked ones depend; the triangles for formulae depending on the marked ones point upward.



Example 5: Marks and Landscape Mode

Example 5 illustrates such dependency marks with a tree for a propositional problem. A typical use is to find the causes of a contradiction.

Acknowledgements We thank François Bry, Tim Geisler, and Heribert Schütz for useful comments on an earlier draft.

References

1. Bry, F., Yahya, A.: Minimal model generation with positive unit hyper-resolution tableaux. *5th Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, LNAI 1071. Springer (1996) 143–159.
2. Manthey, R., Bry, F.: SATCHMO: A theorem prover implemented in Prolog. *9th Int. Conf. on Automated Deduction (CADE)*, LNCS 310. Springer (1988) 415–434.
3. Schütz, H, Geisler, T.: Efficient model generation through compilation. *13th Int. Conf. on Automated Deduction (CADE)*, LNAI 1104. Springer (1996) 433–447.