In D. Kapur (ed.), *Lecture Notes in Computer Science: 11th International Conference on Automated Deduction.*
New York: Springer-Verlag, 1992.

An Overview of FRAPPS 2.0: A Framework for Resolution-based Automated Proof Procedure Systems

Tomás E. Uribe,* Alan M. Frisch, and Michael K. Mitchell**

Dept. of Computer Science and Beckman Institute, University of Illinois,
405 North Mathews Avenue, Urbana, Illinois, 61801, USA

1 Introduction

The Framework for Resolution-based Automated Proof Procedure Systems (FRAPPS) is a set of functions written in Common Lisp that facilitate the construction of a wide variety of resolution-based deductive systems. FRAPPS offers the basic functionality necessary to build such systems, freeing users from low-level implementation concerns. It is not intended for use in the construction of high-performance theorem provers, but rather to provide a modular and customizable system useful for rapid prototyping and experimentation in teaching and research.

In addition to data structures that represent clauses and procedures that implement inference rules, FRAPPS provides a *derivation graph* that records all the inferences that have been performed and a *priority queue* that keeps track of potential future inferences. A number of common deletion and search strategies are predefined, and many more can be implemented with a minimum amount of effort. Proof procedures can be conducted automatically or interactively; FRAPPS offers a wide range of functions to access the derivation graph, so that users can examine and guide the progress of a particular search.

An extension to FRAPPS enables the introduction of specialized unification procedures and constraints, thus supporting the implementation of substitutional hybrid reasoners as defined by Frisch [1991]. The latest version of FRAPPS is fully documented in the FRAPPS 2.0 Users' Guide [Mitchell *et al.*, 1991], and a companion document [Uribe, 1991] describes the extension to constraint unification. Mitchell [1989] provides an in-depth description of the original design of FRAPPS.

2 Proof Procedures Using FRAPPS

A proof procedure is a combination of inference rules and deletion strategies, together with a search strategy that controls the order in which these operations are performed. Proof procedures using FRAPPS grow a derivation graph, where nodes correspond to retained clauses and directed arcs link nodes with their parents and children. New nodes are formed by applying inference rules to previously existing

* Current address: Dept. of Computer Science, Stanford University, Stanford, Ca. 94305
** Current address: IBM Corp., Internal Zip 9151, 11400 Burnet Rd., Autstin, Texas, 78758, USA.

nodes and then *integrating* the resulting clauses into the derivation graph. The inference rules provided by FRAPPS are factoring, binary and general resolution, and common resolution refinements such as SLD-resolution. Answer extraction is supported through the use of *answer literals*.

The derivation graph is organized by levels: The initial set of clauses is integrated at level 0, and each derived clause is integrated at the level following the maximum level of its parent(s). Nodes are assigned unique integer *node-ids*, reflecting the order in which they are created, and are indexed according to their level as well as the number and type of literals that their clauses contain. A set of functions allow for efficient retrieval of nodes according to various properties they might posses. Particularly useful are those functions that return all nodes whose clause contains a literal equal to, more general than, or an instance of, any given literal. Besides the clause itself, nodes contain additional information, including a *user field* that can be used to associate arbitrary extra information with each node.

FRAPPS offers a number of common deletion strategies, including length, complexity and tautology deletion. Back and forward subsumption procedures are also provided; these can be used to their full power or restricted to consider only instance or variant tests. Users can also decide whether answer literals, if present, should participate in subsumption tests. In general, the user can label nodes as *active* or *inactive*; inactive nodes cannot be used to generate new clauses and are not considered when updating the priority queue.

The priority queue maintained by FRAPPS contains pairs of nodes ordered by a user-defined *cost-function*, and provides a general mechanism for implementing search strategies. A typical proof procedure is conducted as follows:

1. Set the derivation graph and the priority queue to be empty, and let $\mathcal{S}$ be the initial set of clauses.
2. For each clause $c \in \mathcal{S}$, do:
   If c is not deleted by any of the deletion strategies in effect, then integrate c as a new node in the derivation graph. If back subsumption is in effect, label as *inactive* all other nodes whose clause is subsumed by c. Retrieve all active nodes whose clause can be resolved with c, and add to the priority queue the set of pairs that contain one of these nodes and the new node for c, ordered by the user-defined *cost function* currently in effect.
3. If the priority queue is empty or a termination condition is satisfied (for example, a given number of answer or empty clauses is found or a maximum time or space limit is reached), stop.
4. Pop the priority queue, removing the "cheapest" node pair, and repeat, if necessary, until the queue is empty or a pair is obtained such that both nodes are active. If a pair of active nodes is found, let $\mathcal{S}$ be the result of applying the chosen inference rule(s) to the clauses in the two nodes,[3] and go to Step 2.

This basic proof procedure can be implemented in less than a single page of Lisp code, using the higher-level functions provided by FRAPPS. The default dele-

[3] Note that, in general, more than one group of literals can be resolved upon at a given resolution step. Thus, a *set* of resolvents is obtained from a pair of nodes. Given two clauses, users can control which literals are factored or resolved upon.

tion, inference and control strategies can be easily overridden to obtain new proof procedures.

The node-pair cost function is defined as the weighted sum of an arbitrary number of individual components. In this way, a large number of complex search strategies can be implemented, starting out from a small set of primitive cost function components. New components are simple to write; each should simply return a numerical value given a pair of nodes. Node-pairs exceeding a maximum cost set by the user are not added to the queue.

Note that when the above procedure adds a new node to the graph, all possible resolvent pairs that can be formed using the new node are added to the priority queue. Efficiency can be improved by delaying the formation of these node pairs; if an underestimate of their cost were available, these pairs should be added to the queue only when the cost of all other pairs exceeds the given estimated cost. FRAPPS implements this scheme if given a user-defined function of single nodes that computes the cost-function estimate.

To further improve the efficiency of a search, users can redefine the node-pair creation process. By default, a node is paired up with *all* the previously created nodes that can resolve with it. Modifying this procedure to suit a particular resolution refinement such as unit, input or SLD resolution, for example, can exclude a large number of pairs from consideration based on indexed properties of nodes.

3 Integrating Constraint-solving Unification

FRAPPS has been extended to support the introduction of unification procedures that incorporate constraint-processing mechanisms [Uribe, 1991]. The user-defined unification procedure can take into account, and generate, constraint fields that are associated with each clause.

This extension of FRAPPS can be used to construct *hybrid reasoners* in accordance to the *substitutional framework* [Frisch, 1991]. The extended system could, for example, incorporate specialized equational reasoning through E-unification, taxonomic reasoning by using sorted logic and sorted unification, or even modal deduction, using constraints on terms that denote possible worlds [Frisch and Scherl, 1991].

Users are responsible for implementing unification of pairs of terms, used in binary resolution, and, optionally, unification of sets of terms, used in factoring and general resolution. (This distinction is due to efficiency considerations. If the more general procedure is not provided, the unification procedure for pairs of terms is used instead.) The new unification procedure can produce a finite number of unifiers; each should be identified by a substitution and a new set of constraints. If back and forward subsumption are to be done, users should also provide functions that implement subsumption tests under the new constraints: In general, there is no way of implementing a subsumption algorithm given only an arbitrary unification procedure.

Other than the points mentioned above, all of the functionality present in FRAPPS 2.0 is included in this extension, which we call *Hooked-on*-FRAPPS.

4 Strengths and Weaknesses of FRAPPS

The major strengths of FRAPPS are its flexibility and ease of use. Being written in Common Lisp, FRAPPS is highly portable[4] and relatively simple to use and modify, thereby providing a practical tool for quickly implementing and experimenting with resolution-based systems. As such, it has proved to be particularly useful to students and researchers in automated deduction. Starting with Version 1.0 in 1988, FRAPPS has been used on three separate occasions in a graduate-level introductory course on automated deduction at the University of Illinois. Students are usually able to design their own proof procedures within a few of days of exposure to the system. This hands-on experience provides a better understanding of the ideas covered in the course.

The very same virtues mentioned above are also the source of FRAPPS' main weakness, namely, its relative inefficiency. Even though FRAPPS includes features that make proof procedures more efficient (such as structure sharing and literal indexing), it is *not* intended for use in building high-performance systems. Though its speed could certainly be improved, it appears that some efficiency must always be sacrificed if the flexibility and modularity currently provided by FRAPPS are to be retained. Theorem provers achieve high performance in part by tightly integrating their components; in FRAPPS, these are only "loosely" coupled.[5] On the other hand, added flexibility can enhance overall performance by allowing the implementation of special-purpose proof procedures that are not offered by other systems; thus, raw performance comparisons may not always be a good indicator of a system's success.

FRAPPS' overall efficiency can be improved by adding more support for specialized node-retrieval functions. Ideally, derivation graph nodes would be cross-indexed according to an arbitrary number of pre-determined properties (including the usual level, literal and length indexing), so that search procedures could *efficiently* identify all the nodes that satisfied any arbitrary combination of such properties. Currently, nodes can only be retrieved according to one property at a time, so set intersection must be used to answer conjunctive queries.

A related problem is that specialized inference rules to handle equality, such as paramodulation and demodulation, are currently not available in FRAPPS; its indexing capabilities would have to be further extended in order to support the efficient implementation of these rules.

There are many other ways in which FRAPPS can be extended. The most outstanding one is to generalize the priority queue mechanism, so that not only pairs but arbitrary *tuples* of nodes can participate in the queue. This capability would be used together with inference rules that are a function of more than two clauses, such

[4] Identical code has been run under Allegro, Kyoto and Lucid Common Lisps.

[5] Given the many proof procedures that can be implemented using FRAPPS as a starting point, as well as the different ways of implementing each, how to determine its real "speed" is unclear. On an IBM RISC 6000 machine running Allegro Common Lisp, the compiled code can generate (and integrate) on the order of 10-20 clauses per second using binary resolution in a breadth-first search, while a theorem prover like OTTER [McCune, 1990] under equivalent circumstances can generate around 200. FRAPPS' relative performance degrades as the derivation graph grows and when subsumption tests are used.

as *UR-resolution* and *hyperresolution*. Currently, the priority queue mechanism can be simply turned off if a particular proof procedure does not use it.

5 System Acquisition

The entire FRAPPS and Hooked-on-FRAPPS systems, including commented source code, demonstration examples, and documentation in postscript, is publicly available. It can all be found in the tar file `/pub/frapps` obtained by anonymous FTP to `a.cs.uiuc.edu` (128.174.252.1).

6 Acknowledgments

This work was partially supported by NASA, under grant NAG 1-613, and by the University of Illinois. We thank the many students who have used the system and provided valuable feedback.

References

[Frisch and Scherl, 1991] Alan M. Frisch and Richard B. Scherl. A general framework for modal deduction. In James Allen, Richard Fikes, and Erik Sandewall, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference*, pages 196–207. Morgan Kaufman, San Mateo, CA, 1991.

[Frisch, 1991] Alan M. Frisch. The substitutional framework for sorted deduction: Fundamental results on hybrid reasoning. *Artificial Intelligence*, 49:161–198, 1991.

[McCune, 1990] William W. McCune. OTTER 2.0 users' guide. Technical Report ANL-90/9, Argonne National Laboratory, Argonne, Illinois, March 1990.

[Mitchell *et al.*, 1991] Michael K. Mitchell, Tomás E. Uribe, and Alan M. Frisch. *The Framework for Resolution-based Automated Proof Procedure Systems* – FRAPPS *2.0 Users' Guide*. University of Illinois, Urbana, Illinois, September 1991.

[Mitchell, 1989] Michael K. Mitchell. The anatomy of a framework for resolution-based automated proof procedure systems. Master's thesis, University of Illinois, Urbana, Illinois, July 1989.

[Uribe, 1991] Tomás E. Uribe. *Constraints in* FRAPPS *2.0*. University of Illinois, Urbana, Illinois, September 1991.