Logic Machine Architecture:
Kernel Functions

Ewing L. Lusk
Northern Illinois University
Argonne National Laboratory

William W. McCune
Northwestern University

Ross A. Overbeek
Northern Illinois University

ABSTRACT

In this paper we present an attempt to abstract from the great diversity of approaches to automated deduction a core collection of operations which are common to all of them. Implementation of this kernel of functions provides a software platform upon which a variety of theorem-proving systems can be built. We outline the architecture for a layered family of software tools to support the development of theorem-proving systems and present in some detail the functions which comprise the two lowest layers. These are the layer implementing primitive abstract data types not supported by the host language and the layer providing primitives for the manipulation of logical formulas. This layer includes the implementation of efficient unification and substitution application algorithms, structure sharing within the formula database, and efficient access to formulas via arbitrary user-defined properties. The tools are provided in a highly portable form (implemented in Pascal) in order that a diverse community of users may build on them.

1. Introduction

There are currently a variety of theorem-proving research efforts, each oriented around a particular theorem-proving approach or application. Some of these approaches are clause-based resolution, natural deduction, equality-oriented systems, and Gentzen systems, as well as higher-order logics. Application systems include program verification, problem-solving, automated circuit design, and piping or wiring network analysis. Diverse as these approaches are, they all require solutions to certain common subproblems. Furthermore, implementation of solutions to these common subproblems constitutes a significant overhead in the implementation of systems designed to study new approaches and applications in theorem proving. The question we have attempted to answer in this paper is the following: Is it possible to define a kernel of functions dealing with the storage, retrieval, and manipulation of logical formulas which is common to a wide variety of theorem-proving environments?

In this paper we present such a collection of functions. They respond to the requirements outlined in [4] for any general-purpose system capable of dealing with very large numbers of formulas. They have all been implemented, and form the kernel of a new theorem-proving system being developed at Argonne National Laboratory and Northern Illinois University. We outline here the architecture of our new implementation and present in detail the two lowest layers of the system, the one which implements primitive abstract data types and the one which provides for the manipulation of logical formulas. The entire system we refer to as Logic Machine.
Architecture (LMA), which is not a single theorem prover, but rather a carefully engineered set of layered functions from which many theorem provers can be constructed. We believe that this approach will be increasingly useful as more and more application-oriented theorem-proving systems are developed. We are currently using this package of tools to construct a variety of inference mechanisms — multiple resolution-based inference rules, equality-based inference mechanisms, the inequality resolution system of Bledsoe[1], and case analysis techniques. We further believe that the layers of LMA presented in this paper provide an integrated foundation for the implementation of several theorem-proving techniques (such as Gentzen and natural deduction systems) which we do not intend to implement immediately.

The LMA package of tools is designed for a wide range of users. The distribution package will include several distinct theorem provers, some of which are oriented towards specific applications. These are for users who wish to do no programming at all. It will allow such people to perform research in application areas without being forced to develop a substantial amount of software. In this regard it is worth noting that the open problems solved by Wos and Winker used only a standard general-purpose theorem prover. No extra programming was done to focus on peculiarities of any of the mathematical systems that were being studied.

Those who wish to utilize the many tools that exist in a large theorem-proving system (such as subsumption, demodulation, and the normal inference rules), but also wish to experiment with new strategies and inference mechanisms, can easily construct new theorem provers from the components included in the package. The layered architecture and the supporting tools included in the distribution package should dramatically reduce the effort required to subject new ideas to experimental verification.

Five distinct software layers have been defined within LMA:

- Primitive Abstract Data Types (Layer 0)
- Formula Manipulation Primitives (Layer 1)
- Inference Mechanisms (Layer 2)
- Theorem-Proving Processes (Layer 3)
- Interprocess Communication (Layer 4)

Layer 0 implements three primitive abstract data types not provided by the host language (which is Pascal). These are character strings of unbounded size, integer vectors of unbounded length, and long integers in a range fixed at compilation time but indefinitely extendible. This layer is discussed below.

Layer 1 implements the data type "object," which is used for representing and storing logical formulas, among other things. This layer provides a variety of operations which can be performed on objects, including unification, application of substitutions, and efficient access to objects via arbitrary boolean combinations of user-defined properties. This latter facility will become increasingly significant as theorem provers are applied to problems involving the very large sets of formulas typical of many applications. A discussion of the concepts implemented in Layer 1 provides the bulk of this paper.

Layer 2 provides mechanisms for generating new formulas from old and for detecting relationships among formulas. In particular, most resolution-based inference rules, paramodulation, demodulation, subsumption, and inequality inference rules are included in the initial set of functions provided at this level. These functions operate on objects which are clauses, but will not assume that all existing objects are clauses. Thus non-clausal inference mechanisms can eventually be integrated into the system in a harmonious fashion. Layer 2 will be described in detail in another paper.

Layer 3 defines a set of components from which a complete theorem prover can be configured. Each component will represent an independent theorem-proving process utilized in