Logic Machine Architecture: Inference Mechanisms

Ewing L. Lusk
Northern Illinois University
Argonne National Laboratory

William McCune
Northwestern University

Ross A. Overbeek
Northern Illinois University

ABSTRACT

Logic Machine Architecture (LMA) is a layered implementation of theorem-proving tools. The kernel of formula storage, retrieval, and manipulation primitives (layers 0 and 1) is described elsewhere[2]. The layer described here (layer 2) contains resolution- and equality-based inference rules, subsumption, and demodulation. It is designed to provide all of the tools required to create a theorem-prover with minimal effort. Although LMA is currently implemented in Pascal, an interface to LISP will be included in the original release. This paper includes the design principles and techniques used in layer 2, as well as two simple theorem provers which illustrate the services of layer 2 - one written in LISP and the other in Pascal.

1. Introduction.

Research in automated theorem proving has been hindered by a lack of powerful, widely available theorem-proving systems. The time and effort required to create useful programs has proven to be substantial. Many researchers simply do not have access to a system suitable for experimentation. Logic Machine Architecture (LMA) has been designed to provide a varied set of software tools suitable for use by everyone from researchers in theorem proving to developers of application systems based on theorem provers. The tools themselves are written in Pascal, but an interface to LISP will be included in the original distribution.

The purpose of LMA is to make it possible to easily and quickly prduce diverse theorem-proving programs, either to test new ideas in theorem-proving or to implement application systems based on theorem-proving principles. As an example of how easy it is to construct a theorem prover using the layer 2 primitives, we give a two-page Pascal theorem prover in Appendix B. The corresponding LISP version of the same theorem prover is in Appendix C.

Throughout the history of automated theorem proving researchers have expressed the desire for tools that would allow comparisons between varied approaches within a common environment. To achieve this level of experimentation will require a system that is well layered, easily modifiable, and well supported. It is hoped that LMA will evolve into such a system and will offer the environment in which comparisons between clause-based formalisms, natural deduction, and higher-order logics can meaningfully occur.

LMA has been conceived and implemented as a layered architecture. Each layer has an independent function and provides a set of well-defined services. This type of architecture allows a user to select the tools appropriate to his task. A user of theorem provers can implement a powerful theorem-prover in a few days (or simply use one of the standard programs included in the release). On the other hand, someone wishing to construct a system based on a new formalism will use only the tools provided in the lower layers to construct a set of new tools (which, if they follow the architectural guidelines of the higher layers, can be added to the services provided by the higher layers).

Layer 0 of LMA implements several abstract data types required in outer layers. Layer 1 contains the functions required to create and maintain a database of formulas. These two layers have been described in detail in [2].

Layer 2 is a package of routines that define the abstract data types *list*, *clause*, *literal*, and *term*. In addition it provides a package of inference rules, demodulation, and subsumption. These provide the basic services from which a theorem prover can be easily constructed.

Layer 3, when it is completed, will provide uniprocessing modules from which a multiprocessing theorem prover could be configured. Layer 4 will implement the logic required to configure and manage cooperating multiple processes.

This paper discusses the details of layer 2. It is broken into the following sections:

a) a discussion of the implementation of the basic data types - list, clause, literal and term - and the inference services provided for clauses,

b) an overview of how subsumption is implemented,

c) an overview of a generalized approach to implementing inference rules, and

d) a discussion of the interface to LISP.

The last section attains its significance from the fact that LMA is currently written in Pascal. It is included to accentuate our desire to provide a set of tools usable by AI researchers and accessible from any major language.

2. The Abstract Data Types of Layer 2

2.1. The Fundamental Role of Objects

Layer 1 of LMA was designed to be general enough to support almost any research requiring the manipulation of logical formulas. It provides the storage, retrieval, and manipulation services required to manage a large database of formulas. The "query" capability - that is, the ability to rapidly retrieve all formulas satisfying a specified set of properties - is particularly critical. The initial release of layer 2 of LMA will provide only the abstract data types required to support clause-based theorem-proving systems. However, it is designed with the specific goal that this set can be enriched, using the powerful services of layer 1, to include the features required to support other formalisms.

One explicit goal of LMA is flexibility based on a layered architecture. Throughout the implementation an attempt has been made to hide the underlying data structures through a well-defined set of services provided to the user. Hence, layer 2 will contain the services required to hide the exact structure of the data types list, clause, literal, and term.

While the internal structure of any abstract data type defined by layer 2 is hidden from the user of layer 2, one aspect of their implementation is not hidden - they must all be instances of the general data type *object*. Objects are defined in [2], and are the central data type supported by layer 1. Briefly, an object is a structural concept which allows a natural representation of logical formulas, among other things. Objects are of three types: names, variables, and applications. Applications are trees with names and variables for leaves. (For example, the term $f(x,a)$ can be represented by an application object with three subobjects: the name f, the variable x, and the name a.) The definition of lists, clauses, literals, and terms as objects has the following immediate implications, because of the object-oriented services provided in layer 1: