AN EQUATIONAL LOGIC APPROACH FOR MAPPING MULTIPLE-VALUED RULE-BASED EXPERT SYSTEMS INTO HARDWARE SPECIFICATION RULES

David C. Rine
George Mason University
Fairfax, Virginia 22030

Abstract

Epstein introduced equational approaches to the study of Post algebras. Epstein and Rine presented various studies of implication operators pertaining to multiple-valued logic. Rine presented a way that higher level production rules can be transformed to equivalent lower level hardware design notations in multiple-valued logic. This paper expands upon that work by introducing a different way of transforming rules into equational logic as a way to represent rules in multiple-valued logic design notation.

1 Introduction

Virtually all rule-based expert systems are executed on general purpose Von Neumann machines or on machines that are at least designed to run LISP or PROLOG. However, these machines only perform between 1,000 and 10,000 LIPS (Logical Inferences Per Second). The current symbolic processing for the DARPA (Defense Advanced Research Projects Agency) Strategic Computing Program requires 1,000,000 and 1,000,000,000 LIPS [33]. Current conventional architectures are only achieving about one percent of this goal. What is generally needed are new advances in the area of accelerating rule-based expert systems that hold great promise for increasing the LIPS rate into the 1,000,000 range with a single chip. This research extends new techniques for mapping rule-based expert systems into VLSI hardware design notation and provides design procedures for performing the mapping from expert systems' production rules to hardware specification rules. Results from this work enhance the applicability of the rule-based expert system approach to a larger class of real-time and control applications.

2 Pipelined Architectures

In the world of high-speed computer architectures, there are two major design techniques for achieving processing concurrency: parallelism and pipelining. Parallelism achieves processing concurrency, thus high speed, by duplicating a particular functional unit many times and providing each duplicate with data. Pipelining takes a function and partitions each piece into many independent but interconnected subfunctions. Of these two approaches to concurrency, parallelism is the most frequently discussed in the literature [24]. The techniques of pipelining are virtually as old as electronic computers, but the overall subject of pipelining has received nowhere near the
attention in the literature as has parallelism. The term pipelining herein refers to the design technique that introduces concurrency by taking a basic function and partitioning it into several subfunctions.

Most programmable architectures in the literature that use pipelining generally use it for a concurrency speedup in a Von Neumann architecture [9,20,24,25].

Earlier Rine [3] showed how both sequential and parallel production rules can be transformed into a multiple-valued logic design notation that is familiar to hardware logic designers. In the sections that follow we will demonstrate a different approach to this problem of logic transformation based on equations.

3 Multiple-Valued Equation Solver

New techniques for dramatically improving the performance of rule-based expert systems through the use of VLSI have been advanced by mappings from expert system production rules, both sequential and parallel, to hardware executable rules [30, 31]. Extensions of these powerful advances bear upon improvements for the following:
(a) automatic compilation of production systems, as represented by Huang and Rine [22]
(b) applications research
(c) control strategies and architectural design.

3.1 The Post Equation Logic Transform Architecture

Very early Post [27] introduced a theory for expanding earlier work in boolean algebra to multiple-valued algebras. Later Epstein [10] improved upon this work on Post algebras, as well as introducing equational approaches [11] to the study of Post algebras. In this paper we use, as background, this Post approach to multiple-valued algebras to justify a technique for transforming logic expressions by equations.

The Post Equation Logic Transform (PELT) architecture is applicable to any set of problems that can be mapped into a set of Post logic equations. Moreover, the PELT system can be used to emulate a synchronous finite state machine.

3.2 Expert System Architecture

The field of artificial intelligence spans a very broad spectrum. It involves such diverse techniques and applications as speech recognition, advising, and process control applications. There is no single accepted definition of artificial intelligence, but a useful boundary to draw around the concept is that of techniques to search for solutions to problems with large problem spaces [19, 29]. The specific area of the field of artificial intelligence that lends itself to some applications of PELT is the area of expert systems. The major parts of an expert system will be discussed, and an example of mapping a small rule-based expert system into the PELT architecture will be given.

A conventional model of a rule-based expert system consists of four major blocks. These are the Knowledge Base, the Inference Engine, the Knowledge Acquisition System, and the User Interface. In this paper the rules for such a system are post logic valued.
A rule-based expert system is a specific type of production rule system [29, 33].

In order to solve a problem using the production rule system approach, one must reduce it to a precise problem statement. The problem statement in this case will be IF THEN rules. The problem can be solved by searching for a path through the space to a given solution state. The word 'state' means a specific set of premises and conclusions having Post logic values (e.g. in the two-valued Post logic case, true or false).

3.3 Design Procedure for Mapping Rules Sets into PELT

In mapping rule-based expert systems into the PELT architecture, one must clarify the types of production rule sets that are being considered. The general model for these rules is the IF THEN rule which logically implements the type of reasoning called 'Modus Ponens' [19]. The inference rule, IF A THEN B, says if A is true then this produces the conclusion that B is true. Another property of modus ponens is 'Modus Tollens', which states that if B is false (two-valued Post logic), then from the rule, IF A THEN B, it is valued to conclude that A is false. Not all possible inferences from a rule base can be drawn from the applications of these rules.

The PELT architecture solves Post logic equations in the forward chaining mode as the input variables are placed on the input lines and then the solution is commenced producing the outputs. It should be noted at this point, however, that the class of production rules sets that the PELT can address are limited. Values not easily mapped to Post logic variables, reasoning with uncertainty, and complex knowledge structures all are not easily mapped to the PELT. Commonly in rule-based expert systems based upon two-valued Post logic, conclusions are not just true and false, but rather true, false, or unasserted (meaning no information one way or the other). This can be emulated by PELT, at the cost of transforming each conclusion variable into a two variable representation where the valid values for this two variable set is true, false, and unasserted. However, systems with many numeric bounded value inputs, complex rules with many terms, and high solution speed requirements are all attributes of problems that map well to PELT.

3.4 Producing Post Logic Equations from Rules

In order to map differing types of rule sets into Post logic equations, it is necessary to translate production rules into equations of Post algebra expressions. The first step of this mapping is to translate the rules into a form of Post algebra expression equation.

3.4.1 Algebraic Rules: For translating the form of IF <premise> THEN <conclusion>, the following procedure is used.

Procedure 1.
(1) The <premise> of the rule becomes the independent side of a Post logic equation of the form f = <Post algebra expression>.
(2) The <conclusion> becomes the dependent side of the Post logic equation. If a rule has multiple conclusions separate conclusions with identical premises can be
formed.

(3) Separate rules with identical conclusions may be merged with OR's between the premises.

Example 1. Applying the above procedure.
Let A, B, C, D, E, F, and G be Post logic valued variables, and let * and + denote the two binary operations from this logic which are commonly termed MIN and MAX, or sometimes AND and OR. Then consider the two-valued Post logic expression
IF (A + B)*C*D + F THEN G
which maps into the two-valued equation of a Post algebra expression,
G = (A + B)*C*D + F.
This equation can then be manipulated like any Post logic equation with theorems and algebraic operations.

Expert system problems are often stated in terms of rules that resemble English sentences. The first step in translating these rule statements into equations of Post algebra expressions is to break the sentence into phrases and associate a Post logic variable if the phrase can be given a Post logic value (e.g. in the two-valued Post logic case, true or false). If a correct rule for a production rule system is constructed, all of the phrases will be able to be given a value.

3.5 Combinational Rule Sets
A combinational rule set is a set of production rules where the conclusions are a function only of the present premises. An inference engine must scan the rules in such a manner that all the rules and all the dependencies from the rules are taken into account. This can be done in a manner similar to the way a logic simulator must evaluate gates in a combinational logic system. Either the simulator can do an evaluation scanning algorithm [16] or use a more sophisticated event-driven algorithm to improve efficiency and to insure proper evaluation of the gates. The problem can be shown with a simple example. Consider the following set of equations comprised of Post logic variables where CO is one of the two unary logic operators in the two-valued Post logic [10], the other being C1:

(1) G1 = A*CO(B) + G2
(2) G2 = A*CO(E) + G3
(3) G3 = B*CO*E + A*CO*E.
To properly solve these equations in one evaluation pass, they must be evaluated in the order G3, G2, G1. Any other single pass through these equations may not produce the proper value for G1. A solution used for some inference engines is to repeatedly evaluate the rule set until no conclusions change, i.e. four passes in this example, if the order of evaluation is as shown.

This is the simplest method and the least efficient. By ordering the rule G3, G2, G1, the number of passes can be reduced to two. By using an event driven algorithm, the number of evaluations can be reduced further by only evaluating rules if one of their premises changes and then evaluating any rule that has as a premise the first rule's conclusion. Both the scanning and the linking of rules for event driven evaluation requires significant amounts of preprocessing of the rules before execution.

Procedure 2 shows the method for reducing a set of rules to the
governing Post logic equation set for implementation into the PELT architecture.

Procedure 2. A set of combinational production rules may be mapped to the PELT architecture by using the following procedure.

Procedure
(1) Map the inputs and the outputs of the system to appropriate Post logic variables. Both inputs and outputs are classified as many-value or single-value outputs.
(2) Translate each rule into its Post logic equivalent by use of Procedure 1.
(3) Manipulate the equation set by use of Post algebra and theorems to produce a sum of products Post logic equation set.
(4) Map the sum of products equation set onto the PELT using defined mapping procedures.

The ability to map production rules to Post logic equations has several advantages. It allows one to use the PELT architecture to dramatically reduce the amount of time it takes to solve the set of equations. Furthermore, it allows one to use the theorems of Post algebra to eliminate redundancies and produce a reduced set of equations for computational operations.

3.5.1 Many-valued and Single-valued Variables: Each input and output variable must be classified into one of two types, single-valued or many valued. Single-valued input or output variables may only have one value at a time. This means that because only one of the possible values of the variable may appear at a time, one can use the most efficient mapping of the variable and all its possible values. In the case of two-valued Post algebra this corresponds to taking an input that has sixteen possible values and mapping it to four separate Post logic variables, with a unique value of the four Post logic variables representing each state. A many-valued variable is one that can take on more than one value at a time. Therefore, for a many-valued variable, there is a separate Post logic variable for each possibility.

Example 2. Applying the Above Procedure.
Consider the following set of production rules where once again the variables are two-value Post logic valued:

IF A is true and B is false
THEN F1 is true
IF A is true and B is false and C is true and D is true
THEN F1 is true
IF B is true and E is false
THEN F1 is true
IF A is true and E is false and I is true
THEN F1 is true
IF F2 is true THEN F1 is true
IF F3 is true THEN F1 is true
IF A is true and E is false and G is true
THEN F1 is true
IF A is true and C is true and G is true
THEN F2 is true
IF A is true and D is false and H is true
THEN F2 is true
IF B is true and C is true and E is true
THEN F3 is true
IF A is true and C is true and E is true
THEN F3 is true
IF A is true and B is true and H is true
THEN F3 is true.

The first step is to derive the following Post logic equations from these by use of Procedure 1.
(4) \[ F_1 = A \cdot CO(B) + A \cdot CO(B) \cdot C \cdot D + B \cdot E = A \cdot E \cdot I + F_2 + F_3 \]
(5) \[ F_2 = A \cdot CO(E) \cdot G + A \cdot C \cdot G + A \cdot D \cdot H \]
(6) \[ F_3 = B \cdot C \cdot E + A \cdot C \cdot E + A \cdot B \cdot H \]

Note that all the variables in this example must be asserted and that the only output that is desired is \( F_1 \).

In the English equivalents of these production rules, it is very difficult to see the methods for reducing the rules to a simpler form. In the Post logic equation domain, it is quite simple. The following steps are applied: The equations (5) and (6) are substituted into equation (4); this results in the equation:

\[ F_1 = A \cdot CO(B) + A \cdot CO(B) \cdot C \cdot D + B \cdot E + A \cdot E \cdot I + A \cdot CO(E) \cdot G + A \cdot C \cdot G + A \cdot D \cdot H + B \cdot C \cdot E + A \cdot C \cdot E + A \cdot B \cdot H. \]

Post algebra simplification theorems reduce the number of equations from three to one and the number of product terms from twelve to four. This is a significant reduction.

**Example 3.** Applying the above procedure to an example by Smith [35].

Consider the following set of production rules where the variables/terms are three-value Post logic valued:

Here \( A, B, C, D, E, \) and \( F \) are variables/terms, and \( F_1, F_2, \) and \( F_3 \) are variables.

IF \( A \) and \( C \) and \( D \) THEN \( F_1 \)
IF \( A \) and \( E \) and \( F \) THEN \( F_1 \)
IF \( A \) and \( C \) and \( F \) THEN \( F_1 \)
IF \( A \) and \( D \) and \( E \) THEN \( F_1 \)
IF \( B \) and \( C \) and \( D \) THEN \( F_2 \)
IF \( B \) and \( C \) and \( F \) THEN \( F_2 \)
IF \( F_3 \) THEN \( F_2 \)
IF \( B \) and \( D \) and \( E \) THEN \( F_3 \)
IF \( B \) and \( E \) and \( F \) THEN \( F_3 \)

The first step is to derive the following Post logic equations from these by use of Procedure 1.

(9) \[ F_1 = A \cdot C \cdot D + A \cdot C \cdot F + A \cdot D \cdot E + A \cdot E \cdot F + F_2 + F_3 \]
(10) \[ F_2 = B \cdot C \cdot D + B \cdot C \cdot F + F_3 \]
(11) \[ F_3 = B \cdot D \cdot E + B \cdot E \cdot F \]

Application of Procedure 2 leads to the single equation:

(12) \[ F_1 = A \cdot C \cdot D + A \cdot C \cdot F + B \cdot C \cdot D + B \cdot C \cdot F + A \cdot D \cdot E + A \cdot E \cdot F + B \cdot D \cdot E + B \cdot E \cdot F \]

From additional information about this example it is assumed that \( A \cdot D = A \) and \( B \cdot E = E \). This and a minimization algorithm [35] leads to a simplified equation with four prime implicants.

(13) \[ F_1 = A \cdot C + A \cdot E + B \cdot D + B \cdot F, \] a much more efficient representation.

The next step would be to map this final equation to the PELT pipeline by use of additional mapping.

### 3.6 Research Applications

#### 3.6.1 Automatic Compilation of Production Rules:
In any system that relies on numerous complicated specification structures that may be changed a method must be found that will allow the automatic conversion of these structures (e.g. production rules) into a format that is suitable for controlling the system. In the case of PELT, a compiler must be designed and written such that the expert system production rules can be automatically translated. While we have introduced a general procedure, many details need to be worked out.

#### 3.6.2 Applications Research:
In order to show the promise of the PELT system, new applications for real-time, rule-based expert systems must be examined. The major area that needs to be
explored is that of real-time control processes such as reactors, chemical plants, production plants, and robotics applications.

3.6.3 Improved Architectural Design: As additional applications are examined and understood, there will be modifications and improvements to the PELT architecture. Research is needed to determine the relative performance benefits of purely combinatoric production rule sets to those that incorporate state information for real-time applications. This will involve building a computer-based simulation to find out the interactions between systems and the controlling rule-based expert system.

4. Impact and Future Directions

Epstein [11] earlier introduced equational approaches to the study of Post algebras. Epstein and Rine [12] later presented various studies of implication operators as they pertain to multiple-valued logic. More recently Rine [30, 31] presented a means by which higher level production rules can be transformed to equivalent lower level hardware design notations in multiple-valued logic. This paper expanded upon that work by introducing a different way of transforming rules into equational logic as a conceptual way to efficiently represent rules in multiple-valued logic.

Design should be clear that there are many applications for dense rule-based expert system controllers capable of real-time operation. Various large, complex real-time controllers could potentially take advantage of the results of this work.

5. References

IEEE, pp. 20-32.