MODELING, IDENTIFICATION AND CONTROL, 2003, VOL. 24, NO. 1, 27–35
doi:10.4173/mic.2003.1.3

Modeling Logic Systems with Structured Array-based Logic‡

REGGIE DAVIDRAJUH* and BASSAM HUSSEIN†

Keywords: *Array-based logic, logic modeling, logic programming*

First this paper introduces an efficient tool called array-based logic for modeling complex logic systems. The main concepts of array-based logic are explained. Second, the logic functions of this tool are presented with the help of six definitions. Finally, an application example on manufacturing system is worked out.

1. Introduction

Structured Array-Based Logic is a toolbox of logic functions for modeling logic systems. In the fields like manufacturing, mechatronic, control systems, and for embedded processors, there is a need for a logic programming tool that gives complete solutions, compact code, as well as fast computation (for real-time applications). Structured array-based logic is such as logic programming tool written in MATLAB language, that guarantees the requirements mentioned above, namely complete solutions, compact, and fast.

1.1. *Propositional logic*

Modeling a logic system can be done exactly like modeling a physical system (Bjørke, 1995). First, the fundamental logic variables (also called primitive logic elements) are identified and each logic variable is assigned an axis; thus the logic variables span the whole universe of discourse (total space), see Figure 1a. Then the logic variables are connected into premises, thus creating a subspace of the total space, see Figure 1b. Finally, the premises are combined to form the logic system, connecting subspaces spanned by the premises. There are some differences between the space span by the physical systems and logical systems; logical spaces are always linear and discrete.

By connection, spaces that do not satisfy the constraints are removed, leaving a smaller space that represents the feasible solution (Figure 1); this is after Lagrange, who in analytical mechanics developed the free variational method. Thus Lagrange developed the mathematical foundation for the basic procedures for logic modeling discussed in this paper, and it was Pierce who applied these procedures (constraint satisfaction) to logical problems (Møller, 1995).

1.2 *Array-based logic*

By the mathematical approach for modeling logic systems, a Cartesian axis is assigned to each logic variable in the system, generating subspaces spanning all

*Department of Electrical and Computer Engineering, Stavanger University College, Post Box 8002, 4068 Stavanger, Norway. Email: Davidrajuh@hotmail.com

†Department of Production and Quality Engineering, Norwegian University of Science and Technology (NTNU), Valgrinda, N-7491 Trondheim, Norway. Email: Bassam.Hussein@ipk.ntnu.no

‡An early version of this article was presented at Nordic Matlab Conference 2001, October 2001, Oslo, Norway.

Lets say that a logic system consists of three primitive logic variables, Temperature (with domain values 'low', 'high'), Alarm ('off', 'on'), and Power ('off', 'on').

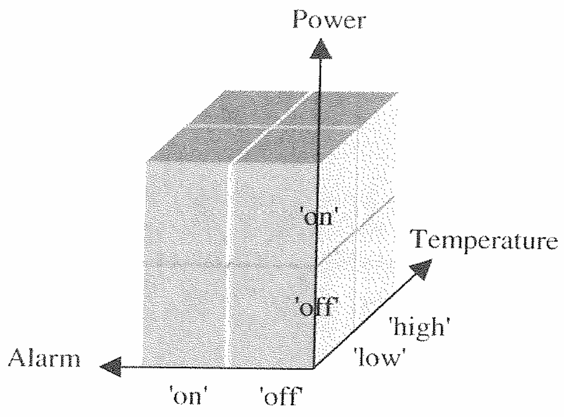

Figure-1a:
The space spanned by the primitive logic variables Power, Alarm, and Temperature

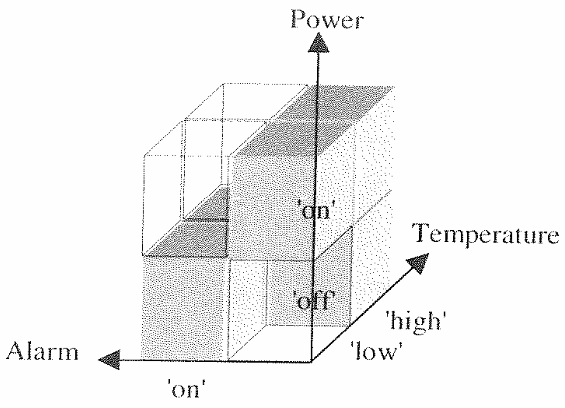

Figure-1b:
The subspace spanned by the combination
((((Temp is 'low') AND (Alarm is 'off')) OR (Power is 'on')) => (Power is 'on')) AND
(((Alarm is 'on') OR (Power is 'off')) => (Power is 'off'))

Figure 1. Configuration space spanned by the logic variables.

possible states of all the variables, thus providing a *complete representation*. Though complete, this representation is huge; this means, for M multi-valued logic variables with a domain of N values, the resulting space will contain M^N subspaces. This exponential growth of the subspaces with increasing number of variables makes the modeling and simulation slower. Array-Based Logic developed by O. I. Franksen and G. L. Møller avoids this exponential problem by compressing M^N subspaces into $M \times N$ liner representation (Møller, 1995; Franksen, 1979). Array-based logic also provides operations, which operates on the compressed representation in linear time.

Array-based logic was written in APL language; we ported array-based logic to MATLAB environment with some additional functions, and named structured array-based logic.

2. The Toolbox Function

Propositional logic functions are for basic mathematical treatment of the logic system after Lagrange and Pierce. By using the propositional logic functions, though the configuration spaces will be large (exponential growth with increasing number of variables), it will be complete; that is, the configuration space includes all possible combinations of the logic variables. Array-based logic functions are enhanced logic functions for modeling and simulation of logic systems using a compression technology that provides compact representation of configuration space and faster simulation, without loosing completeness.

2.1 *Propositional logic functions*

All the logic variables (primitive elements) that are used in a system are to be declared first; it is the function `element` that is used for declaration. Relevant to the function `element` is the function `assign`; this function changes the values of a logic variable.

E.g. To define a multi-valued logic variable 'Color' with a domain of three values 'red', 'green', and 'blue', we use,

Color = element('n',{'red','green','blue'},{'green'}, 'Color');

The first argument 'n' indicates that the variable is multi-valued (or boolean). The second group of input argument is values (of domain), the third group is the default values selected at the time of declaration (in this example, default value is 'green'), and the final input argument is the label or name of the variable. After declaring a logic variable, we could change the values of the variable with the function `assign`;

ColorRED = assign({'red'}, Color);

Definition 1: Basic operations
A logic system can be built by applying the following four basic operations on variables: disjunction (V), direct-implication ($\Rightarrow$), nand, and converse-implication. These four operations are known as the Klein four group. Other logic operations can be derived from these four basic operations. The functions for these four operations are, `disjunct`, `dimp`, `nand`, *and* `cimp` *respectively.* ■

E.g. If Premise1 = (ColorRED $\Rightarrow$ AlarmON), then we write,

Premise1 = dimp(ColorRED, AlarmON);

Definition 2: Colligation
If the same variable occurs more than once in a premise or in a combination of premises, then duplicate axes will be found in the configuration space. The process of removing superfluous axes without losing any information is called Colligation. The function that performs colligation is `fuse`. ■

E.g. if System = disjunct(Premise1,Premise2), where

Premise1 = dimp(ColorRED, AlarmON), and
Premise2 = dimp(ColorGREEN, AlarmOFF)

Then, the System contains two copies of the logic variables Color and Alarm (or mathematically, two axes each for Color and Alarm). Duplicates of Color and Alarm must be removed (or the axes are fused together) by,

System = fuse(System);

2.2. *Array-based logic functions*

The following definitions present the main functions for array-based logic.

Definition 3: Compressed representation
Compressed representation is to keep the relation (premises, subsystems, or system—see Figure 2) to a minimal size without losing any information. The function used for compression is `compress`. ■

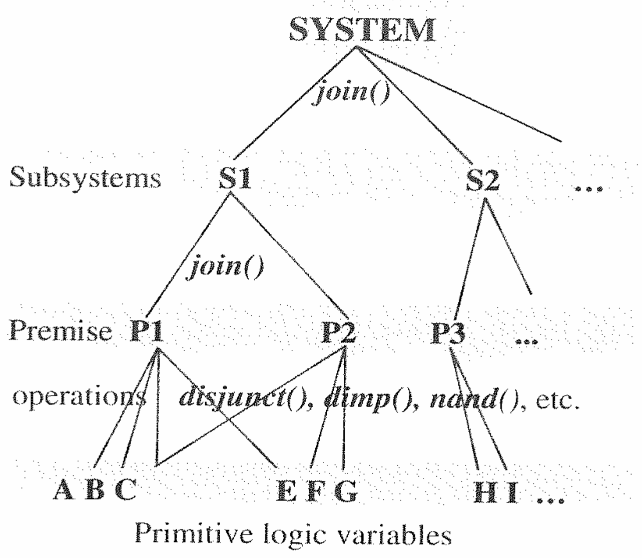

Figure 2. System perspective of modeling a logic system.

In compressed form, functions like `join`, `deduct`, etc. make use of the compressed (compact) representation for faster computation. The function join connects premises together via the common variables they posses; the resulting relation (subsystem, or system) will be in compressed form. Compression technique is similar to the Karnaugh map (K-map) reduction done in digital electronics.

In addition to Boolean variables and multi-valued variables, array-based logic allows also quantitative (intervals, for example) to be treated as logic variables. There are three types of variables in array-based logic: the nominal logic variables (boolean and multi-valued), ordinal logic variables (e.g. Coordinate is [2,2], [4,2], or [3,3]) and intervals (e.g. Cost is between <50 and 100>).

Definition 4: Intervals as logic variables
Array-based logic facilitates intervals to be treated as logic variables too. An interval variable may contain many intervals, each of which may be true or false. ■

To create an interval, the function `interval` is used.
E.g.: LowerInterval = interval('ge', 85, 'lt', 98).
This means, the LowerInterval is greater than or equal to 85, and less than 98.

An interval variable is created using the function `element`.
E.g.: InputPrice = element('i', {LowerInterval, UpperInterval}, 'Input Price'), where the first argument 'i' indicates that the variable to be created is an interval variable, and the final argument is a label of the variable.

Definition 5: Deducing conclusions
Deduction (or inference) is to draw conclusion from a connected system. Deduction is performed by the function `deduct`, which makes the OR—projection of all the axes complementary to the variables concerned, on the axes of the variables. ■

The final definition is about the state of a system.

Definition 6: state of system
The state of a system is the information required of the system to uniquely determine an output for an input to the system. The output is a vector of output variables that is computed from the input vector of variables and the system (see Figure 2), using the function `state`. ■

Allowing quantitative variables to be treated as logic variables facilitates numerous advantages in modeling complex logic systems. Use of propositional and array-basic logic functions will become clear in the next section where we model a simple logic system from manufacturing engineering. See Davidrajuh (2000) for more elaborate explanation of the logic functions. Table 1 shows the main functions.

Table 1. The main logic functions of the toolbox

A. Propositional logic functions	
A1. Basic operations	
disjunct	OR operator
conjunct	AND operator
dimp	direct implication
bimp	binary implication (equivalence)
cimp	Converse implication
exor	Exclusive OR
nand	NOT-AND operator
invert	NOT operator
A2. Format change	
affirmative	to extract valid configuration space from whole space
unpack	Opposite of affirmative
A3. Deductions	
ared	Abductive (AND) reduction
dred	Deductive (OR) reduction
fuse	to remove superfluous axes (variables)
state	to find state of a system (or outputs) for a given inputs
B. Array-based logic functions	
B1. Compression techniques	
compress	to make compact model of a system
expand	Opposite of compress
comsize	Complement sizes of variables in a system
domsize	domain sizes of variables in a system
B2. Interval	
interval	to create an interval
union	union (combination) of intervals
complement	to complement an interval
intersection	Intersection of intervals
B3. Combination	
join	to combine relations (premises) through common variables
C. Utility functions	
element	to create a logic variable (or primitive logic element)
assign	to assign new values to a multi-valued logic variable
domain	to assign new domain to an interval variable
print	to print any relation on screen (variables, and their combinations)

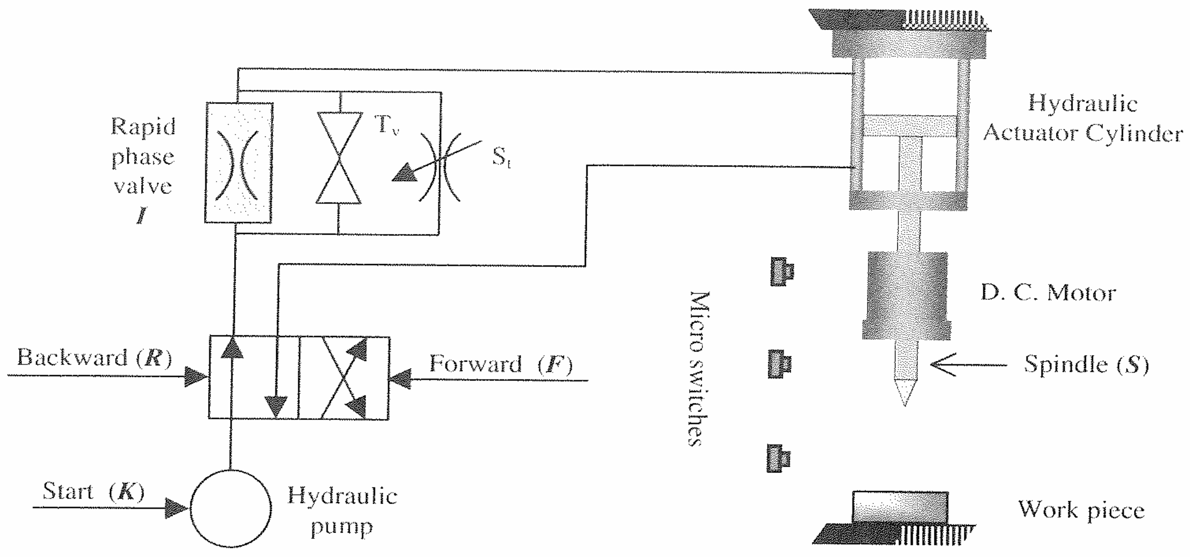

Figure 3. Application example—a manufacturing system.

3. Application Example

In this section, we shall model a manufacturing system as an application example. This example is taken from Hussein (1999). The example is simple, but the methodology given below can be used to model larger logic systems. In the modeling approach (see Figure 2), first the primitive logic variables are identified. Then these variables are grouped into premises using the logic operators like disjunct, dimp, etc. Finally, the premises are joined to make the compete system.

3.1. *The manufacturing system example*

The manufacturing system is shown in Figure 3. It consists of a boring spindle operated by a direct current servomotor. The linear motion of the boring spindle is carried out by means of a hydraulic linear actuator. A constant pressure hydraulic pump powers the hydraulic actuator. The volumetric flow is the hydraulic circuit is controlled by a servo valve S_t (Lien, 1995).

The operation of the system is governed by three sensors (micro-switches) B, M, and E, where,

1. Sensor B indicates that the boring spindle is at the rear position,
2. Sensor M indicates that the boring spindle has reached the feeding position, and
3. Sensor E indicates that the boring spindle has reached the final destination, and ready for backward motion.

The spindle is at the rear position initially, and the operator switches on the system by a very short signal K. The manufacturing system then will go through three modes of operation:

1. Starting from the initial state, hydraulic circuit will open rapid phase valve I, and the spindle will go forward by the opening the valve F.

Table 2. The logic variables of the system

Axis	Variable	Description of True state	Description of False state
1	F (output)	Forward motion is On.	—
2	K (input)	Start	Stop
3	E (input)	Breaker E is On	Breaker E is Off
4	S (output)	Spindle motor is On	Spindle motor is Off
5	M (input)	Breaker M is On	Breaker M is Off
6	I (output)	Rapid motion is On	Rapid motion is Off
7	R (output)	Backward motion is On	—
8	B (input)	Breaker B is On	Breaker B is Off

Table 3. Grouping logic variables into premises

Premises	Linguistic description	Symbolic description
P1	Precede forward motion if and only if start signal is On or spindle is already moving forward and it didn't reach breaker E.	$F = (K \text{ OR } F) \text{ AND } (\text{NOT } E)$
P2	Open rapid motion valve I if and only if spindle is moving forward and it has not reached breaker M.	$I = (F \text{ AND NOT } M)$
P3	Start spindle motor if and only if spindle is moving forward and it has reached breaker M.	$S = (F \text{ AND } M)$
P4	Start backward motion if and only if breaker B is Off and feed forward is Off.	$R = (\text{NOT } F) \text{ AND } (\text{NOT } B)$

2. At position M, the rapid phase valve I will be switched off to start a controlled feed forward motion. This motion is regulated manually by the servo valve S_t. At position M, the spindle motor will also be switched on by a signal S.
3. At position E, the backward motion R will begin. Simultaneously, the rapid phase valve I will be switched on.

3.2. *Modeling the logic system*

The manufacturing system consists of a physical system and a logical system; we shall model the logical system only.

Identifying primitive logic variables The behavior of the logic system can be described by the logic variables shown in Table 2.

Establishing the premises The logic variables are grouped into premises as shown in Table 3 (Lien, 1995).

3.3. *Logic Programming*

In this subsection, we shall show how the logic system described above can be programmed using structured array-basic logic.

Declaring the logic variables First of all, the logic variables must be declared. MATLAB codes for creating the variables are shown below. MATLAB codes start with a MATLAB prompt '>'. In MATLAB, the text that follows the '%' mark is comments, that is, not executable.

First we create the logic variables for the inputs K and E.

```
> %declaring input K with domain
> % stop/start and default start
> K=element('n',{'stop','start'},
    {'start'},'K');
> %declaring E with domain off/on,
> % with default value on
> E=element('n', {'off','on'},
   {'on'}, 'E');
```

Similarly, other inputs M, B, and the outputs F, S, I, and R can be declared.

Grouping the logic variables into premises
Premise P1 is: $F=(K$ OR $F)$ AND (NOT E). This is programmed as follows:

```
> P1=bimp(F, conjunct(disjunct(K, F), invert(E)));
```

P1 contains two copies of variable F. Therefore, a duplicate of F is removed:

```
> P1=fuse(P1);
```

Premise P2 is: $I=(F$ AND NOT $M)$. This is programmed as follows:

```
> P2=bimp(I, conjunct(F, invert(M)));
```

Premise P3 is: $S=(F$ AND $M)$. The program code:

```
> P3=bimp(S, conjunct(F, M));
```

Premise P4 is: $R=$(NOT F) AND (NOT B). The program code:

```
> P4=bimp(R, conjunct(invert(F), invert(B)));
```

The combined system If we combine the four premises together using AND operator, there will be 2^8 values (or *tuples*) in the truth table, as there are 8 Boolean variables. Also, fusing the duplicate variables will be time consuming. Instead of AND'ing the variables together, we use the logic function `join`.
The system as the combination of the four premises,

```
> SYSTEM=join(P1, P2, P3, P4);
```

When we join the four premises, the function join removes duplicate variables when combining two premises together, and leaves the combined system in the compressed form by taking only the valid tuples. The combined system (SYSTEM) is very compact and complete. This is the core of the inference engine. Because it is compact, the inferences (deductions) made from it are fast.

3.4. *Simulations on the combined system*

Let us input some sample values to the inference engine.

```
> Input_K=assign(K, {'start'});
> Input_B=assign(B, {'off'});
> Input_M=assign(M, {'off'});
> Input_E=assign(E, {'off'});
> Test_Input_Vector=[Input_K, Input_B, Input_M, Input_E];
> output=state(Test_Input_Vector, SYSTEM);
> print(output);
```

The output printed on the screen is:

```
** F: ON S: OFF I: ON R: OFF **
```

This means, during the initial phase, when K is pressed, both F and I will be switched on for forward rapid movement, while spindle motor (S) and backward movement (R) will remain switched off.

4. Closing Remark

The core technology of the approach discussed in this paper, the array-based logic, is already used in many industrial applications, e.g. in TV sets. This paper presented an efficient logic programming toolbox called structured array-based logic, developed in MATLAB language with the aim of computing with words, in-addition to fast, complete solutions, and compact. Computing with words facility greatly enhances the modeler to concentrate on the logic modeling aspects rather than on the internal representations.

5. References

BJØRKE, Ø. (1995) Manufacturing Systems Theory, TAPIR Publishers, ISBN-82-519-1413-2.

MØLLER, G. L. (1995) On the Technology of Array-Based Logic, Ph.D. thesis, Technical University of Denmark.

FRANKSEN, O. I. (1979) *Group Representations of Finite Polyvalent Logic—a Case Study Using APL Notation.* In NIEMI, A. (ed.): A Link between Science and Applications of Automatic Control, Pergamon Press, Oxford and New York, 1979.

DAVIDRAJUH, R. (2000) An Introduction to SABL, http://www.hin.no/~rd/Projects/SABL.

HUSSEIN, B. (1999) Modeling the Physical and Logical Properties of Mechatronic Systems: A Manufacturing Theory Approach, Ph.D. thesis, Norwegian Univ. of Science and Technology, 1999.

LIEN, T. K. (1995) Digital Control for Mechatronic (in Norwegian), TAPIR publishers, Norway, 1995.