Array-Based Logic for Realizing Inference Engine in Mobile Applications

Reggie Davidrajuh
Deaprtment of Electrical & Computer Engineering
University of Stavanger
PO Box 8002, N-4036 Stavanger, Norway
Tel: +47 51831700 Fax: +47 51831750 Email: reggie.davidrajuh@uis.no

Biographical notes:
Dr. Reggie Davidrajuh received a master's degree in Control Systems Engineering in 1994 and a PhD in Industrial Engineering in 2000, both from the Norwegian University of Science and Technology (NTNU). He is currently Associate Professor of Computer Science at the Department of Electrical and Computer Engineering at University of Stavanger, Norway. His current research interests include e-commerce, agile virtual enterprises, discrete event systems and modeling of distributed information systems.

Abstract

Mobile and wireless devices suffer from technological limitations such as limited battery life and limited memory size. Hence, use of technologies for mobile applications is confined to those technologies that are faster and take small footprint in memory. Firstly, this paper presents a survey of technologies that can be used for realization of inference engine, satisfying the qualities mentioned above. Secondly, this paper introduces a Scandinavian invention called Array-Based Logic that enables realization of inference engines for decision making that are compact and fast. Finally, a case study is presented to show how easy it is to use array-based logic for realizing inference engine in mobile applications.

Keywords: Logic modeling, array-based logic, mobile applications, and mobile ad-hoc network

1. INTRODUCTION

Mobile devices have become indispensable tools these days. Since mobile devices have limited resources, the research and application of technologies in these areas are confined to those technologies that are:

- **Faster**: in order to save battery life and to accommodate synchronous (blocking) communication,
- **Compact and memory friendly**: mobile devices have limited memory thus embedded code shouldn't take much memory.
- **Easy**: the tools used for development should make it easy to design the mobile applications so that development can be done faster.

This paper introduces a logic technology called array-based logic for realizing inference engines in mobile applications. The next section (section-2) presents a literature review of technologies that can be used for realization of inference engines. Section-3 introduces array-based logic.

As mentioned above, among other issues, battery life and memory size are critical issues in mobile devices. Battery life and memory size and two dependent variables of the independent variables like processing time and program code size, respectively. Thus, if array-based logic minimizes processing time, it also implies that the usage of array-based logic saves battery life. Similarly, if array-based logic minimizes program code size, it also means that the usage of array-based logic demands less memory. The case study given in section-4 proves minimization of the two independent variables processing time and program code size; Implication is that battery life is increased and less memory is needed.

The case study talks about developing an inference engine for evaluating a mobile host as the call manager in Mobile Ad-hoc wireless Network (MANET). This is a simple problem dealing with a small set of logic variables. Since the size of this problem is small, it is true that many logic technologies could be used to solve this problem, and the usage of array-based logic will not make any considerable difference. The benefits of array-based logic will be apparent when large and complex problems with many logic variables are considered. However, the case study is intentionally made small to give emphasis also to the modeling and simulation approach behind array-based logic; this modeling and simulation approach is unique and is based on the “theory of connection” (e.g. Davidrajuh, 2000).

2. LITERATURE REVIEW

The aim of logic in industrial applications is to develop a formal method for modeling problems so that decisions can be made out of the models, and it can be made automatically e.g. by an inference engine. In order to create models, a language is needed with which sentences can be created in such a way that forms the logical structure of the model.

2.1 Propositional Logic

The first language that can be used for logic modeling is the language of prepositional logic. It is based propositions, or declarative sentences which can be argued as being true or false; thus, propositional logic is concerned with the validation of an argument consisting of a set of propositions that are split up into a number of premises and conclusions. The Boolean logical variables describe the facts in the premises, and the premises themselves describe the system when combined together (Davidrajuh, 2000; Huth & Ryan, 2000).

2.1.1 Formal language

In propositional logic, symbols are used to compress large set of English declarative statements into compact logic model. Suppose a logic model consists of a set premises $\phi_1, \phi_2, \ldots, \phi_n$ and a conclusion φ, then the logic model is expressed by the sequent:

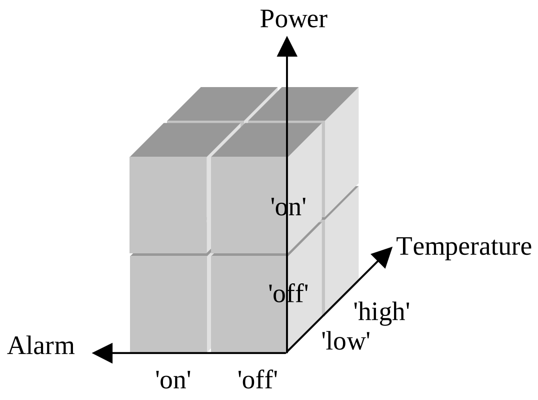
$$\phi_1, \phi_2, \ldots, \phi_n \vdash \varphi$$

By applying proof rules on these premises, the validity of the conclusion is found (Huth & Ryan, 2000).

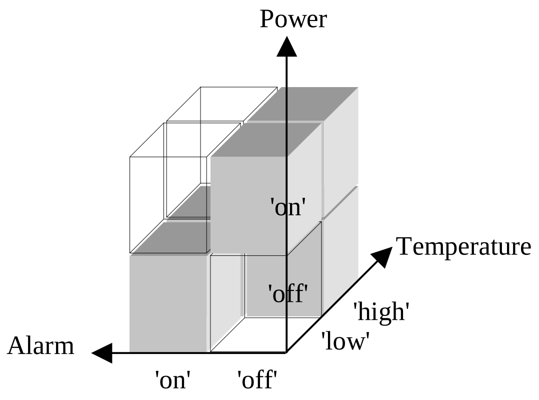
2.1.2 Mathematical reasoning approach

By the use of propositional logic, modeling a logic system can be done exactly like modeling a physical system (Bjørke, 2000). 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.

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').



5.2a:
The space spanned by the primitive logic variables Power, Alarm, and Temperature



5.2b:
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

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, and it was Pierce who applied these procedures (constraint satisfaction) to logical problems (Møller, 1995).

2.1.3 Advantages & disadvantages of propositional logic

This logic representation is useful in providing formal proofs as it offers clarity. Logic systems modeled with propositional logic is well defined and easily understood (Kusiak, 1997). Also, by the mathematical approach for modeling logic systems, a Cartesian axis is assigned to each logic variable in the system, generating subspaces spanning all possible states of all the variables, thus providing a *complete representation*. However, there are two serious shortcomings of propositional logic that disqualify itself as the technology for realizing inference engine:

1) *Exponential growth*: Though propositional logic offers complete systems, the representation is huge; this means, for *M* Boolean logic variables, the resulting space of *M* axes will contain 2^M subspaces. This exponential growth (also known as 'combinatorial explosion') of the subspaces with increasing number of variables makes the modeling and simulation slower. Thus, propositional logic is not suitable for realizing the inference engine.
2) Lack of quantifiers: Though propositional logic uses simple Boolean connectives like negation (‘not’), conjunction (‘and’), disjunction (‘or’), if-then (‘direct implication’), it lacks quantifies like ‘all’, ‘among’, ‘only’, ‘at least one’, etc. This limitation is restored in predicate logic.

2.2 Predicate Logic

This is much like propositional logic, but with its quantifiers, it is possible to express all arguments occurring in natural language. In other words, precise symbolic logic model equivalent of a set of English language statements is possible.

2.2.1 Formal language

A predicate logic formula has three entities: variables, functions that describe relationships between variables, and terms that are expressions consisting of constants, variables, and functions. Because of the power of predicate logic, the language is much more complex than that of propositional logic; interested reader is referred to Huth & Ryan (2000).

2.2.2 Mathematical reasoning approach

In the mathematical approach for modeling predicate logic systems is similar to that of propositional logic systems; a Cartesian axis is assigned to each logic variable in the system, generating subspaces spanning all possible states of all the variables, thus providing a complete representation making a huge representation if large number of logic variables are involved. Thus, advantages and disadvantages of predicate logic are similar for that of propositional logic.

2.3 Production Rules

Production rules are in effect subsets of predicate calculus with an added prescriptive component indicating how the information in the rules is to be used in reasoning. A production rule has the following form (Kusiak, 1997):

IF (condition)
THEN (conclusion)

2.3.1 Mathematical reasoning approach

The basic reasoning approach employed for production rule is searching: starting with a set of facts and look for those rules in which the IF clause matches the facts; if such rules are found ('hit'), then proceed to the THEN clause. This reasoning is known as 'forward reasoning'. In 'backward-reasoning', searching starts with a set of desired goals and to look for those rules in which the THEN clause (conclusion) matches the goals. Figure-2 shows an example with 6 rules, using forward reasoning (or bottom-up search). As shown in figure-2, it is usual to use AND/OR tree to illustrate the inference process.

2.3.2 Advantages and disadvantages of production rules

The main advantage is that the simple rules are easy to understand, modify, and extend. However, there are some shortcomings: In production rules, a logic system is evaluated with a couple of 'if-then' statements, taking a linguistic view than a mathematical approach. This means, for M multi-valued logic variables with N values N^M 'if-then' statements are needed to span all combinations of the variables; missing any of these statements may cause unexpected results. For a large system of many logic variables, it will be impossible to write so many if-then statements to take care of all possible combinations of variables; thus creating a complete model is not easy and prone to errors. In addition to this shortcoming, there is another serious problem: the reasoning approach based on searching is slow.

The logic system is represented here by the following six rules:

Rule-1:	*IF ((Temp is 'low') AND (Alarm is 'off')*	*THEN (R1)*
Rule-2:	*IF (R1) OR (Power is 'on')*	*THEN (R2)*
Rule-3:	*IF (R2)*	*THEN (Power is 'on')*
Rule-4:	*IF ((Alarm is 'on') OR (Power is 'off'))*	*THEN (R3)*
Rule-5:	*IF (R3)*	*THEN (Power is 'off')*
Rule-6:	*IF ((Power is 'off') OR (Power is 'on')*	*THEN (Goal)*

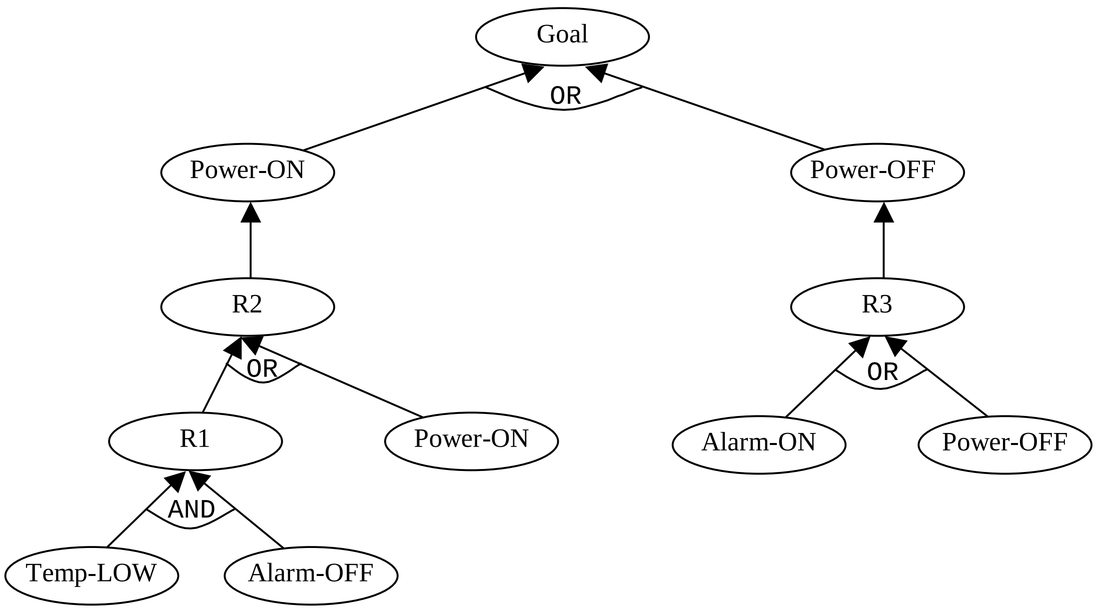


Figure-2: Forward reasoning inference process

2.4 Fuzzy Logic

In relation to classical logic, Fuzzy logic, in a narrow sense, can be considered as an extension and generalization of classical multi-valued logic (Klir & Yuan, 1995). Fuzzy logic is a promising technology to realize inference engines and it used in diverse industrial applications. For a detailed study about Fuzzy logic, see (Adcock, 1993; Meridian, 1997; Tsoukalas & Uhrig, 1997; Yager & Zadeh, 1991).

2.4.1 Formal language

Fuzzy logic is a methodology for expressing operational laws of a system in linguistic terms instead of mathematical equations. Systems that are too complex to model accurately using mathematics, can be easily modeled using fuzzy logic's linguistic terms. These linguistic terms are most often expressed in the form of logical implications, such as fuzzy if–then rules. For example, a fuzzy if-then rule (or simply a *fuzzy rule*) looks like:

IF delivery_time is LATE
THEN supplier_preference is LOW.

The terms LATE and LOW are actually sets that define ranges of values known as *membership functions*. By choosing a range of values instead of a single discrete value to define the input parameter "*delivery_time*", we can compute the output value "*supplier_preference*" more precisely.

2.4.2 Inference mechanism

Inference mechanism in fuzzy logic is based on fuzzy rules that connect input and output parameters (fuzzy rule base), and the membership functions for input and output parameters. To create an inference engine, first the membership functions for input and output parameters are developed; both a range of values and a degree of membership define membership functions.

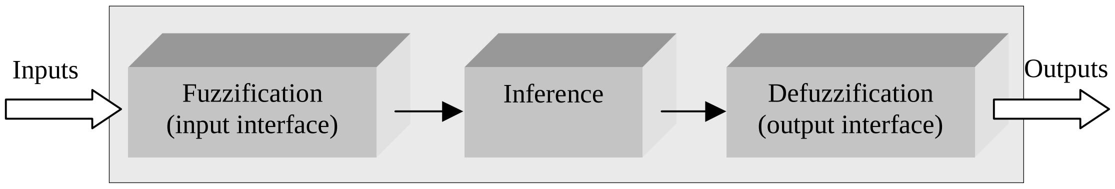

Inputs
Fuzzification (input interface)
Inference
Defuzzification (output interface)
Outputs

Figure-3: The three phases of inference mechanism in fuzzy logic

Inference mechanism in Fuzzy logic is implemented in three phases (see figure-3):

- Phase-1: Fuzzification phase (converting crisp input value into fuzzy value).
- Phase-2: Inference phase (computing fuzzy output value by the fuzzy rules base).
- Phase-3: Defuzzification phase (converting fuzzy output value into crisp value).

2.4.3 Advantages and disadvantages of fuzzy logic

Fuzzy logic offers fast inference, offers compact executable code that can be downloaded into micro-controllers for embedded applications. Fuzzy logic is also easy to learn and use. However, it has some limitations too.

The first limitation of fuzzy logic is tuning; if one wanted to change the pattern the output parameters that are computed from the input parameters, then in-addition to changes in the fuzzy rule base, the membership functions of the input and output parameters must be changed too. The second limitation is that fuzzy logic does not guarantee completeness; it is up to the designer to include all the fuzzy rules connecting all possible combinations between the input and output parameters.

The third limitation is the difficulty in generating fuzzy rule base. The fuzzy rules generated for an application must be consistent; they must properly adhere to the process dynamics with no contradictions between the rules. Generating the antecedent (the IF part) of a fuzzy rule is easy; but generating the consequent of a fuzzy rule (the THEN part) is not easy, as it demands deep understanding of the process dynamics (Davidrajuh, 2000).

2.5 Array-Based Logic

The previous subsections state that a complete representation of *M* multi-valued logic variables with a domain of *N* values contains 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 G. L. Møller avoids this exponential problem by compressing N^M subspaces into *M x N* linear representation (Møller, 1995). Array-based logic also provides mechanisms for operations to operate on the compressed representation in linear time.

2.5.1 Formal language

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>).

Structured Array-based Logic (SABL) is a formal language of array-based logic for modeling, logic programming, and implementation of logic systems; interested reader is referred to Davidrajuh (2000).

2.5.2 Inference mechanism

The inference mechanism used in array-based logic is geometry or topology of connections between the fundamental components of a system. A system consists of three fundamental components, namely *elements*, *connections*, and *sources*. Elements carry all the physical properties of the system; thus, elements are the fundamental building blocks of a system. Connections reflect how elements in a system influence each other; thus, connections represent the structure of a system. Finally, sources reflect the influence between a system and its environment. Sources are the environment's influence on a system.

The inference mechanism consists of three phases; see Davidrajuh (2000) for details:

- Phase-1: identifying the primitive system
- Phase-2: making the connected system
- Phase-3: applying the sources, and solving the connected system

2.6 Summary

Table-1 presents the summary of the literature review.

Technology	*Property (Relation)*	*Inference mechanism*	*Inference cycle time*	*Complete system*	*Implementation*
Propositional logic	Boolean truth values	modus ponus etc.	slow	yes	not compact
Predicate logic	any predicate	same as propositional	slow	yes	not compact
Production Rules	IF - THEN	searching	slow	no	not compact
Fuzzy logic	fuzzy rules	membership functions & fuzzy rule base	fast	no	compact
Array-based logic	any predicate	geometry	fast	yes	compact

Table-1: Summary of the survey on approaches for modeling logic systems

Table-1 reveals that array-based logic satisfies all the requirements on the qualities for realizing an inference engine (such as high processing speed, and compact size) for mobile applications. However, this does not mean that array-based logic is the only or the best option for realization of every inference engines. On the contrary, the type of the inference engine or the modeling problem under scrutiny determines the technology for realization. For example, if the model could be best expressed by a set English statement that approximately describe the dynamics of the system, then fuzzy logic is perhaps the best technology for realization of the system.

3. STRUCTURED ARRAY-BASED LOGIC

Array-based logic guarantees complete solutions (explained below), compact code, as well as fast computation (for real-time applications). Array-based logic was written in APL language; APL is a primitive symbolic language that is hard to learn and use. Davidrajuh (2000) ported array-based logic to MATLAB environment with some additional functions, and named it "*structured* array-based logic".

Structured array-based logic toolbox consists of two types of functions:

1) Propositional logic functions, and
2) Array-based logic functions

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. The following subsections present these two types of logic functions.

3.1 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. Declaration of a multi-valued logic variable 'Color' with a domain of three values 'red', 'green', and 'blue':

```
                    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 are 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, values of the variable could be changed 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 (=>), 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 => AlarmON) then Premise-1 is declared as:

```
                    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):

```
                    System = fuse(System);
```

3.2 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 loosing any information. The function used for compression is compress. ■

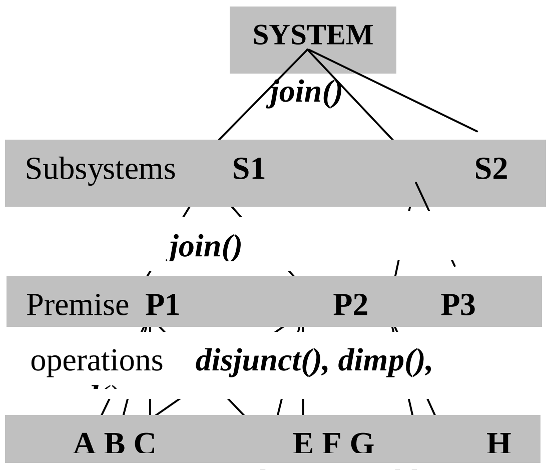
In compressed form, functions like join, deduct, etc. make use of the compressed (compact) representation for faster computation. The function join connects premises together via the common variables they posses; the resulting relation (subsystem, or system) will be in compressed form. Compression technique 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 declare 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.



Primitive logic variables

Figure-4: System perspective of modeling a logic system

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, which is computed from the input vector of variables and the system (see figure-4), using the function state. ■

Allowing quantitative variables to be treated as logic variables facilitates numerous advantages in modeling large logic systems. Use of propositional and array-basic logic functions will become clear in the next section where a case study is done on mobile platform. See Davidrajuh (2000) for more elaborate explanation of the logic functions.

4. CASE STUDY

The previous section proved that array-based logic provides fast computation and compact code. In this section, a case study is provided to show whether it is also easy to develop (program) an inference engine. Case study deals with an inference engine that is to be used to evaluate a mobile host (MH) as the call manager; the call manager is to manage MHs in a specified area in Mobile Ad-hoc Network (MANET). MANET is a mobile network where any mobile device (mobile phones, personal digital assistants, etc.) located inside a specified area can act as the call manager. Selecting a call manager is an important problem and is discussed widely in literature; see for example Yan et al (2004).

4.1 The Best MH

Yan et al (2004) proposes an algebraic equation for selection of the best MH as the call manager in a specified area in MANET. The equation computes the total cost of a MH that is under evaluation. After calculating the total costs of all the MHs, the one with the minimal cost is selected as the call manager in the specified area. The equation is:

$$c_i = (w_1 \times d_i) + (w_2 \times s_i) + (w_3 \times p_i)$$

Where *d* is the distance between the MH to the center of the specified area, *s* is the average of speed of the MH, and *p* is power cost; w_1, w_2, and w_3 are coefficients (weighting factors). Thus, the equation calculates total cost of an MH in terms of its distance from the center, its speed and its battery power. In summary: using the equation proposed by the Yan et al (2004) demands calculation of total cost of all the MHs in the specified area so that the best MH (the one with minimal cost) can be selected as the call manager.

Since, calculating total costs of *all* the MHs in the area to find *the best* MH takes time, this paper proposes selection of *an optimal* MH (rather than the best MH) as the call manager. If an MH under evaluation satisfies the selection criteria, then it is selected as the call manager, and the selection process is terminated; this means the selection process does not evaluate all the MH in that area.

4.2 The optimal MH

This paper proposes a selection process that uses a logical equation rather than an algebraic equation. The logical equation is based Yan et al (2004) in the sense that the logical approach uses distance, speed, and power as input parameters for the selection process. In the logical approach, first the primitive logic variables (elements) 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.

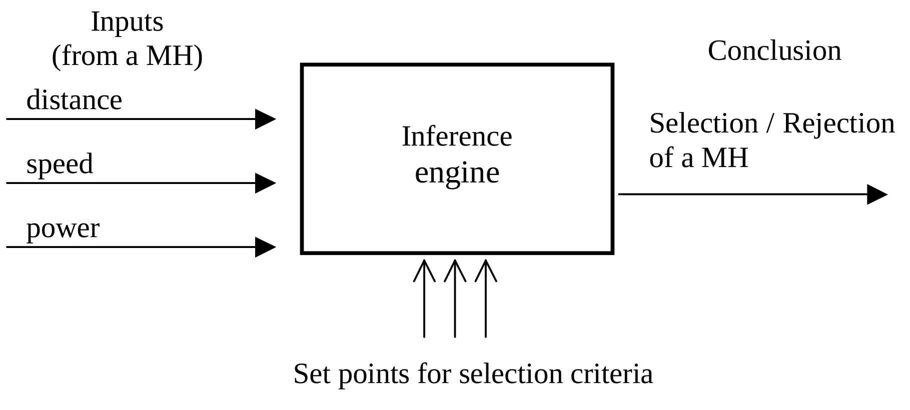
It is assumed that the inference engine receives information from MHs in the specified area about their distance from the area center, their speed, and about their power capacities (figure-5). For brevity, how the information is sent to the inference engine is not discussed here.

The selection is based on three data (figure-5): distance of MH from the center of the area, speed of the MH, and the power cost of the MH. To make decisions based on the data, the inference engine needs three set points (on for each input). These set points are fine-tuned to make the selection process agile; suppose, all MHs fail in the selection process, then the set points are relaxed a little, to make some MHs pass the selection process.

Figure-6 shows the logic variables and the premises that make up the complete system. The first three premises deal with the input values. The input (numeric) values for distance, speed, and power cost, are used to assign values to some auxiliary logic variables. In effect, the first three premises are about converting interval variables into nominal (Boolean or multi-valued) variables. Premises 4- to- 5 uses the auxiliary logic variables to compute the conclusion.

4.3 Premises 1-To-3: Dealing with the Input Values

Premises 1 to 3 deal with the input values named as inputDistance, inputSpeed, and inputPower. Premises 1-to-3 are to convert the input numerical values into auxiliary logic variables named distance, speed, and power respectively.



Inputs (from a MH)
distance
speed
power
Inference engine
Conclusion
Selection / Rejection of a MH
Set points for selection criteria

Figure-5: The inputs and outputs of the inference engine

4.3.1 Dealing with inputDistance

If the distance is greater than the set point for distance, then the distance is 'long'. If the distance is less than or equal to the set point for distance, then the distance is 'short'. To formulate this logic statement, two logic variables are needed: a multi-valued logic variable "distance" with the domain values of 'long' and 'short', and an interval logic variable "InputDistance" with two intervals, one interval between minimum possible distance to set point and the other interval between set point to maximum possible distance.

To declare the logic variable distance:

```
distance = element('n',{'short', 'long'},{}, 'distance');
```

Before declaring the interval variable InputDistance, a value should be assigned to the set point for distance. It is assumed that the given value for set point is 2 km, the minimum possible value for distance is 0 km, and the maximum possible value is 5 km.

```
DistanceSetPoint = 2; MinDistance = 0; MaxDistance = 5;
```

To declare two intervals, the lower interval and the upper interval:

```
LowerInterval = interval('ge', MinDistance, 'le', DistanceSetPoint);
UpperInterval = interval('gt', DistanceSetPoint,'le', MaxDistance);
```

Declaraing the interval variable InputDistance input distance:

```
InputDistance = element('i', {LowerInterval, UpperInterval},'Input Distance');
```

Finally, declaring the premise-1: (DistanceIsFair) if and only if (FairDistanceRange)

```
ShortDistanceRange = assign(InputDistance, LowerInterval);
DistanceIsShort = assign(distance, {'short'});
Premise_1 = bimp(ShortDistanceRange, DistanceIsShort);
```

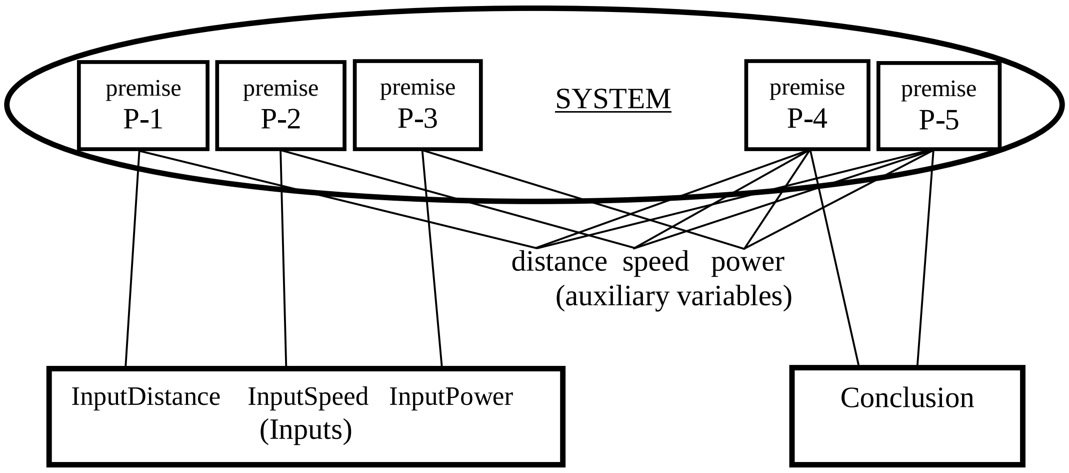


Figure-6: Logic model of the inference engine

4.3.2 Dealing with speed

If the speed is between the minimum possible speed and the first set point for speed, then the speed is 'slow'. If the speed is between the first and second set points, then the speed is 'moderate'. On the other hand, if the speed is between the second set point and the maximum speed, then the speed of the MH is 'fast'. To formulate this logic statement, again two logic variables are needed: a multi-valued logic variable “speed” with the domain values of 'slow', 'moderate', and 'fast', and an interval logic variable “InputSpeed” with three intervals. The first interval (LowerInterval) is between the minimum possible speed and set point-1, the second interval (MiddleInterval) is between the set points, and the third interval (UpperInterval) is between set point-2 and anticipated maximum speed.

Formulating the premise-2 that deals with the inputSpeed is very similar to premise-1 for inputDistance. The only difference is that, speed has three intervals whereas distance has two intervals. For brevity, detailed formulations are not shown here.

4.3.3 Dealing with power cost

Premise-3 for power is formulated very similar to that of premise-1.

4.4 Premise - 4 and 5: Accepting or Rejecting a MH

The auxiliary logic variables distance, speed, and cost are used to compute premise-4. Premise-4 is about the conditions for accepting a MH as the call manager. A MH should be selected if and only if all three inputs values are within the acceptable regions, like distance is 'short', speed is 'moderate' or 'fast', and power is 'moderate' or 'superior'.

First the logic variable Conclusion is declared:

```
Conclusion = element('n', {'reject', 'select'}, {},'Conclusion');
```

Now the acceptable conditions:

```
AcptDIS = assign(distance, {'short'});
AcptSPE = assign(speed, {'moderate', 'fast'});
AcptPOW = assign(power, {'moderate', 'superior'});
AcptCondtion = conjunct(AcptDIS, AcptSPE, AcptPOW);
```

For these acceptable inputs, the conclusion is 'select':

```
Action = assign(Conclusion, {'select'});
```

Finally, the premise-4 is for accepting a MH (selecting a MH): Conclusion is 'select' if and only if ((distance is 'short') AND (speed is 'moderate'/'fast') AND (power is 'moderate'/'superior')):

```
Premise_4 = bimp(AcptCondition, Action);
```

4.4.1 Premise - 5: Rejecting a MH

A MH should be rejected if any one of the following conditions is met: either distance is 'long', or speed is 'slow' or power is 'inferior'.

The conditions for rejection:

```
RejtDIS = assign(distance, {'long'});
```

```
RejtSPE = assign(speed, {'slow'});
RejtPOW = assign(power, {'inferior'});
RejtCondtion = disjunct(RejtDIS, RejtSPE, RejtPOW);
```

For these inputs, conclusion is 'reject':

```
Action  = assign(Conclusion, {'reject'});
```

Finally, the premise-5 for rejecting a MH: (Conclusion is 'reject') if and only if ((distance is 'long') OR (speed is 'slow') OR (power is 'inferior')):

```
Premise_5 = bimp(RejtCondition, Action);
```

4.4.2 The Connected System

The system is the combination of the five premises. That is,

```
System = join(Premise_1,Premise_2,Premise_3,Premise_4,Premise_5);
```

When the five premises are joined using the function join, it removes duplicate variables in the connected system, and leaves the connected system in compressed form; the three auxiliary variables (distance, speed, and power) are only to help compute the conclusion from the input numeric values, thus in the final system. Thus, they must be removed.

```
Inputs  = [InputDistance InputSpeed InputPower];
SYSTEM_F = deduct([Inputs Conclusion], System);
```

The final system (SYSTEM_F) is compact and complete. This is the core of the inference engine. Because it is operates in linear time, the decision made by the inference engine is also fast.

4.4.3 Simulations on the connected system

Some sample input values are input to the inference engine:

```
InputDIS = assign(InputDistance, 1);
InputSPE = assign(InputSpeed, 4);
InputPOW = assign(InputPower, 8.2);
```

Making a source vector of sample inputs:

```
TestInputVector =  [InputDIS InputSPE InputPOW];
```

Applying the source (vector of sample inputs) to the system, the outputs are generated:

```
output = state(TestInputVector, SYSTEM_F);
```

Using the print system, the output is echoed on the screen:

```
print(output);
```

The output of the system (printed on the screen) is:

```
** Conclusion ** : select :
```

This means, for the given input values and for the given set points, the MH is selected as the call manager.

5. Managerial Implications

The most difficult aspect of developing applications for the mobile platform is that the applications must satisfy at least three basic criteria: they must be memory friendly (compact code), they must run fast (take minimal execution time; for example - to save battery), and the tools for development must facilitate fast and easy development. This paper presents a logic technology called array-based logic that guarantees applications developed by this technology fulfill the three criteria; array-based logic operates on a linear (compact) space thus the code size is small, the operations on it are faster (takes linear time), and it is also easy to use this tool.

This paper also presents a structured language of array-based logic called ‘structured array-based logic’, which is a toolbox of functions written in MATLAB language. This toolbox can be used for modeling and simulation of logic programs as shown in the case study; the case study deals with developing an inference engine for selection of a mobile device as the call manager in MANET mobile wireless network. The case study is intentionally kept small just to give emphasis to the modeling and simulation approach behind array-based logic; this approach is unique and is based on the theory of connection.

REFERENCE

Adcock, T. (1993). *What is Fuzzy Logic: An overview of the latest control methodology*. Texas Instruments, SPRA028

Bjørke, Ø. (1995). *Manufacturing Systems Theory*. TAPIR Publications, Trondheim, Norway

Davidrajuh, R. (2000). *Automating Supplier Selection Procedures*. PhD Thesis, The Norwegian University of Science and Technology

Huth, M. and Ryan, M. (2000). *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge University Press.

Klir, G. and Yuan, B. (1995). *Fuzzy Sets and Fuzzy Logic: Theory and Applications*. Prentice Hall PTR, New Jersey

Kusiak, A. (1997). Knowledge-Based Systems. *Nordic-Baltic Summer School on Applications of AI to Production Engineering, Ed. K. Wang, Kaunas University of Technology Press*

Meridian Marketing Group (1997). *Fuzzy Logic Newsletter*, Vol.2, No.1, 1997.

Møller, G. (1995). *On the Technology of Array-Based Logic*. PhD thesis, Technical University of Denmark

Tsoukalas L. and Uhrig, R. (1997). *Fuzzy and Neural Approaches in Engineering*. John Wiley and Sons

Yager and R. and Zadeh, L. (1991). *An Introduction to Fuzzy Logic Applications in Intelligent Systems*. Kluwer Academic Publishers

Yan, K., Wang, S., and Chou, Y. (2004). A Power-Aware Appling to Hierarchical Cellular-Based Management. *The 15th International Conference of Information Management*, New Orleans, USA