# Monitoring Algebra Validation System

* Truth-Value Statements expressed in Boolean Algebra with variables capable of having a trivalent state (T, F, or N).
* Break into components that can be either T or F
* Execution plan for optimizing the solution of the statement
* Boolean Values for each Variable in the Boolean Statement are determined independently of one another but in parallel processes
* As variables values are known, a validator process will try to solve the algebraic statement with the available values based
* For each variable in the Algebra Statement and trivalent cell is created and is given an anonymous (lambda) function. The beginning state of the cell is always N. The anonymous function will return T or F.
* Anonymous functions may perform complex calculations such as look back operations over large data sets by using by passing variables that are objects that can perform operations on demand with the function.
* The trivalent cells are scanned continuously until they statement can be proved true or false, or all cells have been updated.

**Ternary Truth-Value Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***A*** | ***B*** | ***A* OR *B*** | ***A* AND *B*** | **NOT *A*** |
| *T* | *T* | *T* | *T* | *F* |
| *T* | *N* | *T* | *N* | *F* |
| *T* | *F* | *T* | *F* | *F* |
| *N* | *T* | *T* | *N* | *N* |
| *N* | *N* | *N* | *N* | *N* |
| *N* | *F* | *N* | *F* | *N* |
| *F* | *T* | *T* | *F* | *T* |
| *F* | *N* | *N* | *F* | *T* |
| *F* | *F* | *F* | *F* | *T* |

**Absolute Truth Function or SAT4J (optional)**

To evaluate whether a truth value of a statement can be known early (with incomplete data), the following function can be used:

* For statement *X*, evaluate using all available current values (uncalculated cells will have *N*) to produce *V1.*
* *Ft* (*V1*) = *V2 -* For all variables in *X* having the value *N,* replace *N* with *T* and evaluate statement to produce value *V2.*
* *Ff (V1*) = *V3 -* For all variables in *X* having the value *N,* replace *N* with *F* and evaluate statement to produce value *V3.*
* Add *V1, V2,* *V3* to Set *S1.*
* *IF* (*N* not in *S1*)*AND* (*V1 = V2* ) *AND* (*V2, = V3* ) *THEN* *X* = *V1*

# Monitoring Rules

Rules are described in terms of simple relations; *i.e.*, properties that assign truth values to combinations of n arguments. For example, in the expression “nodeName Like MICROWAVE”, **Like** is the property that assigns a truth value according to whether or not the value of the card transaction variable named nodeName matches the pattern MICROWAVE.' Since there are two arguments involved in this relation, nodeName and MICROWAVE, the **Like** relation is binary.

Most of the relations for monitoring are binary relations. **Eq, Ge, Gt, Le, Like, Lt** and **Match** are all binary relations. For monitoring, to declare the two rules: “nodeName **Like** MICROWAVE” And “isActive=True”, the following entries would be added to the rules table in the monitoring database using the binary relations **Like** and **Ge**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **rule\_id** | **rule\_name** | **relation** | **arguments** | **monitor\_id** |
| 1 | A | **Like** | nodeName MICROWAVE | 1 |
| 2 | B | **Eq** | isActive TRUE | 1 |

The **In** relation, as expressed in the rule: “nodeName In MICROWAVE,COFFEE POT,BEDROOM LIGHT” is also a binary relation. Although it appears that it takes more than two arguments it is better thought of as describing the connection between one argument (the value of the nodeName variable in this case) and another argument that happens to be a set of values. A set of values *is* comma-separated. Hence, the following rule entry would express the **In** relation:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **rule\_id** | **rule\_name** | **relation** | **arguments** | **monitor\_id** |
| 3 | C | **In** | nodeName [MICROWAVE,COFFEE POT,BEDROOM LIGHT] | 1 |

The **Bewteen** relation is ternary and therefore assigns a truth value to combinations of three arguments. The rule: “lastNodeOnTime **Between** 6:30am and 10:30am” can be expressed this way:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **rule\_id** | **rule\_name** | **relation** | **arguments** | **monitor\_id** |
| 4 | D | **Between** | lastNodeOnTime [06-11-2012:6:30am:CST 06-11-2012:6:30am:CST CST] | 1 |

Notice again that the first argument is always the monitoring variable name. Also, date values in general must be in one of the following formats: yyyy-MM-dd, dd/MM/yyyy, or MM/dd/yyyy.

Finally, to express these rules together in the form of a monitor algebra the following entry would be updated in the monitors table in the monitoring database:

|  |  |  |
| --- | --- | --- |
| **monitor\_id** | **monitor\_name** | **monitor\_algebra** |
| 1 | Example Monitor | **A and B and D (A Λ B Λ D)** |

The above monitor says: “The Microwave node is Active (workling) and the last time that it was turned on was between 6:30am CST and 10:30am CST on 06-11-2012” This statement is either True or False. Note that rules are bound to variables in the monitor algebra by their rule\_name in the rules table. Also, much more sophisticated algebras are possible with the additional operators **or**, **not** and parenthesis **( )**. For example, for a different set of rules, a monitor algebra might be: “(A and B) or (A and C and not B)”.

While the algebra statements themselves are simple, the calculation of a particular variable may be complex. For instance a nodeStatistic could be a probability statistic representing whether some other variables with a time dimension fall within a norm.

**Monitoring Variables**

|  |
| --- |
| nodeName (String) |
| nodeDescription (String) |
| nodeOn (boolean) |
| nodeOff (boolean) |
| isActive (boolean) |
| lastNodeOffTime (DateTime) |
| lastNodeOnTime (DateTime) |
| nodeStatistic (list) |
| sitePattern (map) |

**Monitoring Variable Relations**

|  |  |  |
| --- | --- | --- |
| **name** | **meaning** | **Data types recognized** |
| **Between** | Between | Numeric, dates |
| **Eq** | Equals | Numeric, dates or text |
| **Ge** | Greater than or equal | Numeric, dates |
| **Gt** | Greater than | Numeric, dates |
| **In** | In | Numeric, dates or text |
| **Le** | Less than or equal | Numeric, dates |
| **Like** | Like | Numeric, text |
| **Lt** | Less than | Numeric, dates |
| **Match** | Match (regex) | Text only |

**Boolean Algebra Operators**

|  |  |
| --- | --- |
| **Operator** | **Example** |
| **and (Λ)** | A **and** B **and** C |
| **Or (V)** | A **or** B **or** C |
| **Not (!)** | A or B and **not** C |
| **( )** | **(**A and B**)** or **((**A or C**)** and not B**)** |