Understanding Temporal Logic

Introducing

Coherent Object System Architecture (COSA)

By
Gordon Morrison, Author

*Breaking the Time Barrier*

www.vsmerlot.com
**Understanding Temporal Logic Introducing Coherent Object System Architecture (COSA)**

Presented at the 22nd Systems and Software Technology Conference (SSTC), 26-29 April 2010, Salt Lake City, UT. Sponsored in part by the USAF. U.S. Government or Federal Rights License.
The Challenge

- Using the traditional spatial If-Then-Else (ITE) approach
  - Produce a five-function calculator
  - add, subtract, multiply, divide, and percent
- The specification is at: www.vsmerlot.com
- Count the number of ITE and Case Statements
  - Count every logic decision point
  - Don’t use my temporal COSA approach
  - Did you improve on COSA?
Proper State Machine(1)

• In a proper state machine, the state transitions are all complete and orthogonal.
  – Complete Transitions: a transition is defined for every possible situation.
  – Orthogonal Transitions: none of the transitions have overlapping conditions.

• With a proper state machine
  – a next state is defined for every possible condition
  – the designated next state is unique

• My comment:
  – The proper state machine is spatial using ITE
  – The proper state machine doesn’t know where it’s working

(1)- © 2005 Carnegie Mellon University – PSP II Designing and Verifying State Machines- page 41
ITE Calculator Statechart

• The author’s implementation:
  – 112 ITE / Case
    • Ready – 6 case
    • Eval – 4 case + ?
    – 1,000+ LOC
• Arrows represent transitions
  – Transitions are events
• Minus is an ambiguous transition
  • Begin to negate
  • subtract
  • opEntered to negated2

“Practical Statecharts in C/C++, © CMP Books, Miro Samek, Ph.D.

www.vsmerlot.com
ITE Calculator Call Diagram

- No calls to trace display
- Very complex

www.vsmerlot.com
ITE State Machine

QHsm::CQSTATE Calc1::opEntered(QEvent const *e) {
    switch (e->sig) {
    case Q_ENTRY_SIG:
        dispState("opEntered");
        return 0;
    case IDC OPER:
        dispState("IDC Entered");
        if (((CalcEvt *)e)->keyId == IDC_MINUS) {
            clear();
            Q_TRAN(&Calc1::negated1);
        } return 0;
    case IDC POINT:
        dispState("IDC Point Entered");
        clear();
        insert(IDC_0);
        insert(((CalcEvt *)e)->keyId);
        Q_TRAN(&Calc1::frac1);
        return 0;
    case IDC_0:
        dispState("IDC 0 Entered");
        clear();
        Q_TRAN(&Calc1::zero1);
        return 0;
    case IDC_1_9:
        dispState("IDC 1-9 Entered");
        clear();
        insert(((CalcEvt *)e)->keyId);
        Q_TRAN(&Calc1::int1);
        return 0;
    case IDC_0:
        dispState("IDC 0 Entered");
        clear();
        Q_TRAN(&Calc1::zero1);
        return 0;
    }
    return QSTATE_SC(&Calc1::calc);
}
ITE With Debugging

- ITE trace
  - Red lines …
- Trace debug
  - Each function
  - Embedded
  - Side effects

www.vsmerlot.com
COSA vs. Traditional ITE

- Temporal domain
  - Time Indexed
- Reduces complexity
  - State diagrams
  - Call diagrams
  - Models
- Reduces code size
- Increases reuse
- Includes trace
- Preemptable

- Spatial domain
  - Find where last
- Increases complexity
  - State diagrams
  - Call diagrams
  - Models
- Increases code size
- Decreases reuse
- Manual trace
- ~Not Preemptable
Temporal vs. Spatial

- Imagine a CPU without a program counter (PC)
  - The hardware would need to save states continuously
  - After an interrupt determine where it was executing
  - Massive amount of logic as administrative overhead
  - This is spatial
- The PC is a temporal pointer
- Software does not have an equivalent PC
  - Until COSA was invented (see US Patent)
Proper COSA State Machine

• Engine/Table relationship
  – Table contains 1 or more rules
  – Each rule has a single entry point
• Rules consist of steps
• Every step is a binary state
  – Each step has a test condition
    • a True Behavior / Next Rule/Step Transition
    • a False Behavior / Next Rule/Step Transition
    • and a Trace (tied to the specification)
COSA State Machine

• Event or non-event driven applications
• States are true or false
  – Transitions are next true or next false
• Three fundamental parts
  – Engine (temporal, trace, and control)
  – Logic-Flow / Rule Table (class)
  – Data-Flow / Reusable Members
• Logic-flow is orthogonal to data-flow
COSA Pattern

- One or more engine/table pairs
- Tracing
  - Duo Point
    - True Trace
    - False Trace
Engine / Table Pattern

**Engine**
- Engine is temporal ( iTIME )
- Preemption control
  - Test condition (state)
    - Dynamic bind **True** Behavior
      - Next True Rule/Step
      - True Trace
    - Dynamic bind **False** Behavior
      - Next False Rule/Step
      - False Trace
  - End preemption control loop

**Table**
- Each row is a temporal sequence
  - Rule/Step (name)
  - Test condition (state)
    - **True** Behavior
      - Next True Rule/Step
    - **False** Behavior
      - Next False Rule/Step
  - Trace (unique to app)
procedure TCOSAFrame.Run(intState integer);
begin
  bEngine := TRUE;
  iState := intState;
  while bEngineLocal AND bEngineGlobal do
begin
  if iState = rRule[iTime].iState then
begin
    rRule[iTime].pTrueRule;
    True_Trace(iTime);
    iTime := rRule[iTime].iTrueRule;
end
  else
begin
    rRule[iTime].pFalseRule;
    False_Trace(iTime);
    iTime := rRule[iTime].iFalseRule;
end;
end;
end;

www.vsmerlot.com
# 23-Steps of Calculator Logic


table

```c
// Rules          | Static State | True Behavior | Next True Rule | False Behavior | Next False Rule | Trace
pBRT(rOpr1,   | iNeg44,      | Negate,       | (0) rOpr1+1,  | Clr_Buf,       | (1) rOpr1+1,  | 100;
pBRT(rOpr1+1, | iDigit,      | Any_Number,   | (0) rOpr1+1,  | Ignore,        | (1) rOpr1+2,  | 101;
pBRT(rOpr1+2, | iDot59,      | One_Period,   | (0) rOpr1+3,  | Ignore,        | (1) rOpr1+4,  | 102;
pBRT(rOpr1+3, | iDigit,      | Any_Number,   | (0) rOpr1+3,  | Ignore,        | (1) rOpr1+4,  | 103;

// clear
pBRT(rOpr1+4, | iClEnt,      | Clear_Entry,  | (0) rOpr1,    | Ignore,        | (1) rOpr1+5,  | 104;
pBRT(rOpr1+5, | iClear,      | Clear,        | (0) rOpr1,    | Ignore,        | (1) rOpr1+6,  | 105;
pBRT(rOpr1+6, | iPush,       | Push_Disp,    | (1) rOpr8,    | Push_Disp,     | (1) rOpr8,    | 106;

// operations
pBRT(rOpr8,   | iAdd43,      | Addition,     | (1) rOpr2,    | Ignore,        | (1) rOpr8+1,  | 500;
pBRT(rOpr8+1,| iSub44,      | Subtraction,  | (1) rOpr2,    | Ignore,        | (1) rOpr8+2,  | 501;
pBRT(rOpr8+2,| iMul42,      | Multiply,     | (1) rOpr2,    | Ignore,        | (1) rOpr8+3,  | 502;
pBRT(rOpr8+3,| iDiv47,      | Division,     | (1) rOpr2,    | Ignore,        | (1) rOpr2,    | 503;

// next number
pBRT(rOpr2,   | iOff,        | Engine_Off,   | (0) rOpr2+1,  | Ignore,        | (0) rErr,     | 700;
pBRT(rOpr2+1,| iNeg44,      | Negate,       | (0) rOpr2+2,  | Ignore,        | (1) rOpr2+2,  | 701;
pBRT(rOpr2+2,| iDigit,      | Any_Number,   | (1) rOpr2+2,  | Ignore,        | (1) rOpr2+3,  | 702;
pBRT(rOpr2+3,| iDot59,      | One_Period,   | (0) rOpr2+4,  | Ignore,        | (1) rOpr2+5,  | 703;
pBRT(rOpr2+4,| iDigit,      | Any_Number,   | (1) rOpr2+4,  | Ignore,        | (1) rOpr2+5,  | 704;

// clear
pBRT(rOpr2+5, | iClEnt,      | Clear_Entry,  | (0) rOpr2+1,  | Ignore,        | (1) rOpr2+6,  | 705;
pBRT(rOpr2+6,| iClear,      | Clear,        | (0) rOpr1,    | Ignore,        | (1) rOpr2+7,  | 706;
pBRT(rOpr2+7,| iSave,       | Save_Disp,    | (0) rResu,    | Save_Disp,     | (1) rResu,    | 707;

// equals
pBRT(rResu,   | iPer37,      | Percent,      | (0) rOpr1,    | Ignore,        | (1) rResu+1,  | 900;
pBRT(rResu+1,| iEqual,      | Equals,       | (0) rOpr1,    | SetChain,      | (1) rResu+2,  | 901;
pBRT(rResu+2,| iChain,      | Operate,      | (0) rOpr1+6,  | Error,         | (0) rErr,     | 902;
pBRT(rErr,   | iErr86,      | Error,        | (0) rOpr1,    | Unknown,       | (0) rOpr1,    | 993;
```

end:

www.vsmerlot.com
The User Perspective  
(tends to be temporal)

• Calculator
  – Enter Operand 1        (optional sign)
  – Enter Operation       (  +  -  *  / )
  – Enter Operand 2      (optional sign)
  – Select Result Type  ( =  %  ( +  -  *  / ))

• The user perspective is generally temporal
• Enter ‘-’ ‘3’ ‘.’ ‘1’ ‘4’ ‘1’ ‘5’ ‘9’
Understanding the Time Index

<table>
<thead>
<tr>
<th>ENTER</th>
<th>Rule</th>
<th>State</th>
<th>True Action</th>
<th>Next True</th>
<th>False Action</th>
<th>Next False</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘-’</td>
<td>rOper1</td>
<td>= &lt;iNeg44&gt;?</td>
<td>Negate</td>
<td>rOper+1</td>
<td>Ignore</td>
<td>rOper+1</td>
</tr>
<tr>
<td>‘3’</td>
<td>+1</td>
<td>= &lt;iDigit&gt;*</td>
<td>Any_Number</td>
<td>rOper+1</td>
<td>Ignore</td>
<td>rOper+2</td>
</tr>
<tr>
<td>‘.’</td>
<td>+2</td>
<td>= &lt;iDot59&gt;?</td>
<td>One_Period</td>
<td>rOper+3</td>
<td>Ignore</td>
<td>rOper1+4</td>
</tr>
<tr>
<td>‘14159’</td>
<td>+3</td>
<td>= &lt;iDigit&gt;*</td>
<td>Any_Number</td>
<td>rOper1+3</td>
<td>Ignore</td>
<td>rOper1+4</td>
</tr>
<tr>
<td>Time→</td>
<td>+4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- I know where I am
- I know where I came from
- I know where I am going
- At iT ime+4 - Not a number from iT ime+3

www.vsmerlot.com
## Logic and Trace

<table>
<thead>
<tr>
<th>Rule</th>
<th>State</th>
<th>True Action</th>
<th>Next</th>
<th>False Action</th>
<th>Next</th>
<th>Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>rOper1</td>
<td>iNeg44</td>
<td>Negate</td>
<td>rOper1+1</td>
<td>Ignore</td>
<td>rOper1+1</td>
<td>100</td>
</tr>
<tr>
<td>+1</td>
<td>iDigit</td>
<td>Any_Number</td>
<td>rOper1+1</td>
<td>Ignore</td>
<td>rOper1+2</td>
<td>101</td>
</tr>
<tr>
<td>+2</td>
<td>iDot59</td>
<td>One_Period</td>
<td>rOper1+3</td>
<td>Ignore</td>
<td>rOper1+4</td>
<td>102</td>
</tr>
<tr>
<td>+3</td>
<td>iDigit</td>
<td>Any_Number</td>
<td>rOper1+3</td>
<td>Ignore</td>
<td>rOper1+4</td>
<td>103</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Time +4

<table>
<thead>
<tr>
<th>T</th>
<th>TR</th>
<th>DS</th>
<th>Behavior</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>44;</td>
<td>Negate;</td>
<td>N= -</td>
</tr>
<tr>
<td>2</td>
<td>101</td>
<td>1;</td>
<td>Any_Number;</td>
<td>N= -3</td>
</tr>
<tr>
<td>3</td>
<td>101</td>
<td>59;</td>
<td>Ignore;</td>
<td>N=</td>
</tr>
<tr>
<td>4</td>
<td>102</td>
<td>59;</td>
<td>One_Period;</td>
<td>N= -3</td>
</tr>
<tr>
<td>5</td>
<td>103</td>
<td>1;</td>
<td>Any_Number;</td>
<td>N= -3.1</td>
</tr>
<tr>
<td>6</td>
<td>103</td>
<td>1;</td>
<td>Any_Number;</td>
<td>N= -3.14</td>
</tr>
<tr>
<td>7</td>
<td>103</td>
<td>1;</td>
<td>Any_Number;</td>
<td>N= -3.141</td>
</tr>
<tr>
<td>8</td>
<td>103</td>
<td>1;</td>
<td>Any_Number;</td>
<td>N= -3.1415</td>
</tr>
<tr>
<td>9</td>
<td>103</td>
<td>1;</td>
<td>Any_Number;</td>
<td>N= -3.14159</td>
</tr>
<tr>
<td>10</td>
<td>103</td>
<td>44;</td>
<td>Ignore;</td>
<td>N=</td>
</tr>
</tbody>
</table>
Some ITE Logic

QHsm::CQSTATE Calc1::opEntered(QEvent const *e) {

    switch (e->sig) {
        case Q_ENTRY_SIG:
            dispState("opEntered");
            return 0;
        case IDC_1_9:
            dispState("IDC 1-9 Entered");
            if (((CalcEvt *)e)->keyId == IDC_MINUS) {
                Q_TRAN(&Calc1::int1);
                return 0;
            }
            return QSTATE_SC(&Calc1::calc);
        case IDC_0:
            dispState("IDC 0 Entered");
            clear();
            Q_TRAN(&Calc1::zero1);
            return 0;
        case IDC_OPER:
            dispState("IDC Entered");
            Q_TRAN(&Calc1::negated1);
            return 0;
        case IDC_POINT:
            clear();
            dispState("IDC Point Entered");
            Q_TRAN(&Calc1::frac1);
            return 0;
    }
    return 0;
    
""Practical Statecharts in C/C++, © CMP Books, Miro Samek, Ph.D.""
## Compare COSA Trace

### COSA Trace

<table>
<thead>
<tr>
<th>T</th>
<th>TR</th>
<th>DS</th>
<th>Behavior</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>44</td>
<td>Negate;</td>
<td>N= -</td>
</tr>
<tr>
<td>2</td>
<td>101</td>
<td>1</td>
<td>Any_Number;</td>
<td>N= -3</td>
</tr>
<tr>
<td>3</td>
<td>101</td>
<td>59</td>
<td>Ignore;</td>
<td>N=</td>
</tr>
<tr>
<td>4</td>
<td>102</td>
<td>59</td>
<td>One_Period;</td>
<td>N= -3.5</td>
</tr>
<tr>
<td>5</td>
<td>103</td>
<td>1</td>
<td>Any_Number;</td>
<td>N= -3.1</td>
</tr>
<tr>
<td>6</td>
<td>103</td>
<td>1</td>
<td>Any_Number;</td>
<td>N= -3.14</td>
</tr>
<tr>
<td>7</td>
<td>103</td>
<td>1</td>
<td>Any_Number;</td>
<td>N= -3.141</td>
</tr>
<tr>
<td>8</td>
<td>103</td>
<td>1</td>
<td>Any_Number;</td>
<td>N= -3.1415</td>
</tr>
<tr>
<td>9</td>
<td>103</td>
<td>1</td>
<td>Any_Number;</td>
<td>N= -3.14159</td>
</tr>
<tr>
<td>10</td>
<td>103</td>
<td>44</td>
<td>Ignore;</td>
<td>N=</td>
</tr>
<tr>
<td>T</td>
<td>Behavior</td>
<td>e-&gt;sig</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>---------------</td>
<td>--------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>O O O</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19, g-negated1, 2,</td>
<td>g-frac1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20, <strong>negated1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21, g-negated1, 1,</td>
<td>g-negated1, 1010,</td>
<td>-0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>O O O</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31, <strong>int1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32, g-int1, 1,</td>
<td></td>
<td>-3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>33, g-int1, 1101,</td>
<td>g-int1, 1101,</td>
<td>-3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>34, g-frac1, 0,</td>
<td>g-frac1, 0,</td>
<td>-3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>O O O</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37, g-frac1, 2,</td>
<td></td>
<td>-3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>38, <strong>frac1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>39, g-frac1, 1,</td>
<td></td>
<td>-3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40, g-frac1, 1010,</td>
<td></td>
<td>-3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>O O O</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>45, g-frac1, 1107,</td>
<td>g-frac1, 1107,</td>
<td>-3.14159</td>
<td></td>
<td></td>
</tr>
<tr>
<td>46, g-Oper1, 1107,</td>
<td>g-Oper1, 1107,</td>
<td>-3.14159</td>
<td></td>
<td></td>
</tr>
<tr>
<td>47, g-frac1, 3,</td>
<td>g-frac1, 3,</td>
<td>-3.14159</td>
<td></td>
<td></td>
</tr>
<tr>
<td>48, g-opEntered, 0,</td>
<td>g-opEntered, 0,</td>
<td>-3.14159</td>
<td></td>
<td></td>
</tr>
<tr>
<td>49, g-Oper1, 0,</td>
<td>g-Oper1, 0,</td>
<td>-3.14159</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50, g-Oper1, 3,</td>
<td>g-Oper1, 3,</td>
<td>-3.14159</td>
<td></td>
<td></td>
</tr>
<tr>
<td>51, g-opEntered, 2,</td>
<td>g-opEntered, 2,</td>
<td>-3.14159</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>52, opEntered</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>53, g-opEntered, 1,</td>
<td>g-opEntered, 1,</td>
<td>-3.14159</td>
<td></td>
<td></td>
</tr>
<tr>
<td>54, g-opEntered, 1107,</td>
<td>g-opEntered, 1107,</td>
<td>-3.14159</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
COSA

-3.14159 – ten steps to enter eight actions
  – 80% efficient
  – 20% of cost is overhead

-3.14159 - - 2.14195 = thirty steps for eighteen actions
  – 60% efficient
  – 40% of cost is overhead
ITE

• -3.14159 – fifty-four steps for eight actions
  – 14.8 % efficient
  – 85% of cost is overhead

• -3.14159 - - 2.14195 = 107 steps for eighteen actions
  – 16.8 % efficient
  – 83% of cost is overhead
ITE Enter “-” Only

Trc=  1, g-calc, sig= 0 ;          Operand= ,            Trc= 16, g-ready, sig= 3 ;        Operand= 0,
Trc=  2, g-calc, sig= 0 ;          Operand= ,            Trc= 17, g-ready, sig= 0 ;        Operand= 0,
Trc=  3, g-calc, sig= 1 ;          Operand= ,            Trc= 18, g-negated1, sig= 2 ;     Operand= 0,
Trc=  4, g-clear;                  Operand= ,            Trc= 19, g-negated1, sig= 1 ;     Operand= -0,
Trc=  5, g-ready, sig= 0 ;         Operand= 0,            Trc= 20, g-negated1, sig= 100 ;    Operand= -0,
Trc=  6, g-ready, sig= 2 ;         Operand= 0,            Trc= 21, g-calc, sig= 100 ;        Operand= -0,
Trc=  7, g-ready, sig= 1 ;         Operand= 0,            Trc= 22, g-negated1, sig= 3 ;     Operand= -0,
Trc=  8, g-begin, sig= 0 ;         Operand= 0,            Trc= 23, g-final, sig= 0 ;        Operand= -0,
Trc=  9, g-begin, sig= 2 ;         Operand= 0,            - End of Analysis
Trc= 10, g-begin, sig= 1 ;         Operand= 0,
Trc= 11, g-begin, sig= 1107 ;      Operand= 0,            Trc= 24, g-calc, sig= 0 ;        Operand= -0,
Trc= 12, g-negated1, sig= 0 ;     Operand= 0,            Trc= 25, g-calc, sig= 3 ;        Operand= -0,
Trc= 13, g-begin, sig= 0 ;         Operand= 0,            Trc= 26, g-final, sig= 2 ;        Operand= -0,
Trc= 14, g-calc, sig= 0 ;         Operand= 0,            Trc= 27, g-final, sig= 1 ;        Operand= -0,
Trc= 15, g-begin, sig= 3 ;         Operand= 0,            - End of Analysis

- End of Analysis
A COSA State Diagram

- Simple state view
- True Behavior
  - One green arrow
- False Behavior
  - One red arrow
- Temporal
  - Trace
  - Specification
    - Compliance
Statechart Comparison

COSA

ITE

www.vsmerlot.com
Call Diagram Complexity

COSA

ITE
COSA and Time

• Understanding “Time” in software means not having to do an “if” to test where the program is executing and what has happened.
  – 23 Logic points in COSA calculator example
  – 112 IF statements in the ITE calculator example
Spatial Software

• Must leave a trail of “breadcrumb” states
• Must track down where it was
  – This is pure overhead
• Difficult to maintain
• Difficult to modify
Temporal Software

- Keeps a temporal pointer
- Reduces complexity
- Eliminates much of the overhead
- Easier to maintain
- Easier to modify
  - Add new rule
Software Quality

• Testing doesn’t improve quality
  – Testing fixes quality problems
  – Quality is still poor

• Temporal engineering
  – Improves quality
  – Reduces overhead logic
The End – Definitions

- **COSA** – Coherent Object System Architecture
  - U.S. Patent #6,345,387 abandoned by inventor
  - Available to the public in book: *Breaking the Time Barrier*
- **BNF** – Backus-Naur Format
  - Diagramming the logic of syntax
- **ITE** – If-Then-Else logic
  - commonly referred to as ‘spaghetti code’
- **CMU** – Carnegie Mellon University
- **SEI** – Software Engineering Institute at CMU
- **CPU** – Central Processing Unit
- **UML** – Unified Modeling Language

www.vсмерло.com 32