Graphical Programming Languages in Model-Based Design

Vijay Raghavan
Director of Engineering
Design Automation
“When I step into a new car these days, I don’t smell leather anymore, I smell software”
Autonomous systems  Industry standards

Software

Everywhere and in everything
And growing exponentially!
Scaling up doesn’t just mean raising your game
Scaling up means playing a different game
Abstraction

Transform a hard problem to an easier problem

High-level languages
High-level languages at MathWorks

Simulink

Stateflow

MATLAB

% Predicted state and covariance
x_pred = A * x_est;
p_pred = A * p_est * A' + Q;

% Estimation
C = H * p_pred';
H = H * p_pred';
Klm_gain = (H' * H)^-1 * H';

% Estimated state and covariance
x_est = x_pred + Klm_gain * (y - H * x_pred);
p_est = p_pred - Klm_gain * H * p_pred;

% Compute the estimated measurements
y = H * x_est;
MATLAB

The leading environment for technical computing

- The industry-standard, high-level programming language for algorithm development
- Numeric computation
- Parallel computing, with multicore and multiprocessor support
- Data analysis and visualization
- Toolboxes for signal and image processing, statistics, optimization, symbolic math, and other areas
- Tools for application development and deployment
- Foundation of MathWorks products
Simulink

- Block-diagram environment
- Model, simulate, and analyze multidomain systems
- Design, implement, and test:
  - Control systems
  - Signal processing systems
  - Communications systems
  - Other dynamic systems
- Platform for Model-Based Design
Stateflow

- Model and simulate decision logic for reactive systems:
  - supervisory control
  - task scheduling
  - fault management

- Develop mode-logic using state machines and flow charts

- See how the logic behaves with diagram animation and integrated debugger
What are State Machines?

- Represent reactive systems that have states or modes
- States change based on defined conditions and events

What are Flow Charts?

- Represent an algorithm or process
Why state machines ?
Abnormal region identification: hysteresis

I want to identify large contiguous regions in my data vector that satisfy a certain property (say a threshold)

- my abnormal region starts when my sequence falls below “t1”
- my normal region starts again when my sequence rises above “t2”
Abnormal region identification

I want to identify large contiguous regions in my data vector that satisfy a certain property (say a threshold)

- Can be done in a few lines of MATLAB code and a simple state machine

```matlab
for i=1:length(inData)
    if (inData(i)>=t)
        outData(i) = inData(i);
    else
        outData(i) = 0;
    end
end
%outData = inData.*(inData<t);
```
Abnormal region identification: hysteresis

I want to identify large contiguous regions in my data vector that satisfy a certain property (say a threshold)

- my abnormal region starts when my sequence falls below "t1"
- my normal region starts again when my sequence rises above "t2"
Abnormal region identification: hysteresis

I want to identify large contiguous regions in my data vector that satisfy a certain property (say a threshold)

- my abnormal region starts when my sequence falls below “t1”
- my normal region starts again when my sequence rises above “t2”

```matlab
inNormalRegion = true;
for i=1:length(inData)
    if(inNormalRegion && (inData(i)<t1))
        inNormalRegion = false;
    elseif(~inNormalRegion && (inData(i)>=t2))
        inNormalRegion = true;
    end
    if(inNormalRegion)
        outData(i) = inData(i);
    else
        outData(i) = 0;
    end
end
```
Abnormal region identification: hysteresis

I want to identify large contiguous regions in my data vector that satisfy a certain property (say a threshold)
- my abnormal region starts when my sequence falls below “t1”
- my normal region starts again when my sequence rises above “t2”

```matlab
inNormalRegion = true;
for i=1:length(inData)
    if(inNormalRegion && (inData(i)<t1))
        inNormalRegion = false;
    elseif(~inNormalRegion && (inData(i)>=t2))
        inNormalRegion = true;
    end
    if(inNormalRegion)
        outData(i) = inData(i);
    else
        outData(i) = 0;
    end
end
```
Abnormal region identification: hysteresis + debouncing

I want to identify large contiguous regions in my data vector that satisfy a certain property (say a threshold)

- my abnormal region starts when my sequence falls below “t1” for at least N1 samples
- my normal region starts again when my sequence rises above “t2” for at least N2 samples
Abnormal region identification: hysteresis + debouncing

```matlab
inNormalRegion = true;
counter = 0;
for i=1:length(inData)
    if(inNormalRegion)
        if(inData(i)<t1)
            counter = counter+1;
            if(counter>=N1)
                inNormalRegion = false;
            end
        else
            counter = 0;
        end
    else
        if(inData(i)>=t2)
            counter = counter+1;
            if(counter>=N2)
                inNormalRegion = true;
            end
        else
            counter = 0;
        end
    end
    if(inNormalRegion)
        outData(i) = inData(i);
    else
        outData(i) = 0;
    end
end
```
Abnormal region identification: hysteresis + debouncing

```matlab
ingNormalRegion = true;
counter = 0;
for i=1:length(inData)
  if(inNormalRegion)
    if(inData(i)<t1)
      counter = counter+1;
      if(counter>=N1)
        inNormalRegion = false;
      end
    else
      counter = 0;
    end
  else
    if(inData(i)>=t2)
      counter = counter+1;
      if(counter>=N2)
        inNormalRegion = true;
      end
    else
      counter = 0;
    end
  end
  if(inNormalRegion)
    outData(i) = inData(i);
  else
    outData(i) = 0;
  end
end
```
As the problem gets more complex, the code gets complex

```python
for i in range(1, len(inData) + 1):
    if inData[i-1] >= t:
        outData[i-1] = inData[i-1];
    else:
        outData[i-1] = 0;
end

inNormalRegion = True;
for i in range(1, len(inData) + 1):
    if inNormalRegion and (inData[i-1] < t1):
        inNormalRegion = False;
    elif (~inNormalRegion and (inData[i-1] >= t2)):
        inNormalRegion = True;
    if inNormalRegion:
        outData[i-1] = inData[i-1];
    else:
        outData[i-1] = 0;
end

inNormalRegion = True;
counter = 0;
for i in range(1, len(inData) + 1):
    if inNormalRegion:
        if inData[i-1] < t1:
            counter = counter + 1;
            if counter >= N1:
                inNormalRegion = False;
        else:
            counter = 0;
    else:
        if inData[i-1] >= t2:
            counter = counter + 1;
            if counter >= N2:
                inNormalRegion = True;
        else:
            counter = 0;
    if inNormalRegion:
        outData[i-1] = inData[i-1];
    else:
        outData[i-1] = 0;
end
```
While the chart remains concise:

- **Normal**
  - If $u < t$:
    - $y = u$
  - If $u \geq t$:
    - If $u < t_1$:
      - $y = u$
    - If $u \geq t_2$:
      - $y = 0$

- **Abnormal**
  - If $u < t_1$:
    - $y = 0$
  - If $u \geq t_2$:
    - $y = 0$

- If $\text{count}(u < t_1) \geq N_1$:
  - $y = u$
- If $\text{count}(u \geq t_2) \geq N_2$:
  - $y = 0$
State Machines are everywhere
Is there a state machine in here?
Calculator state machine

Logic

button = ButtonPushed;

initialize
en
expr = "";
currNum = 0;
parenDepth = 0;

[isOperator(button)]

isNumber(button)

Number

[isOperator(button)]

[isNumber(button)]

Operator

button == 'C'

[lastOp === '=' || expr === "]")

button == '='

MathWorks
Logic for handling number entry
State Machines are everywhere
Symbol/Frame Detection in Communication Systems

We receive either a pulse (1) or not a pulse (0) every 10ms

MARKER - A 770-ms pulse (sequence of 77 pulses) is sent every 10 seconds for synchronization

MISS - No pulse (0) is sent at the beginning of each frame, to indicate the start of a new frame

A - A 170-ms pulse (sequence of 17 pulses) indicates symbol A

B - A 470-ms pulse (sequence of 47 pulses) indicates symbol B
Stateflow Diagram for Frame and Symbol Detection
State machines are in space

“Until Deep Space 1, state charts and automatic code generation technology had not been used on large systems for spacecraft avionics software. MathWorks tools made this approach possible.”

Dr. Wesley Huntress, NASA
"With great power comes great responsibility"
~Voltaire
Challenges for Graphical Programming Languages

- Simulation and Debugging
- Design Error Detection
- Automatic Code Generation
Demo
1997: Deep Blue defeats Gary Kasparov
Joel Benjamin: the chess grandmaster behind Deep Blue
Automation needs your help

- Computers are stupid
  - It is the human expertise that makes them smart

- World class automation tools needs world class researchers
  - Your expertise in creating efficient systems is critical to build automated tools

- Automation is the wave of the future
  - If we do not do this, someone else will
Model Based Design has many interesting problem areas

- **Language Design**
  - Efficient ways to denote graphical variants
  - Combining state charts and physical domains

- **Automatic Code Generation**
  - Balancing optimization with traceability
  - Targeting heterogeneous hardware (CPU, GPU, FPGA etc.)

- **Verification and Validation**
  - Deadlock detection in the presence of messages
  - Quality of Service estimates
  - Security and safety properties
Enabling Success in Academia with MATLAB

CURRICULUM DEVELOPMENT

AUTOGRADING ASSIGNMENTS

PROJECT-BASED LEARNING

RESEARCH COMPUTING

We are here to help.

Balaji Sharma, Customer Success Team, MathWorks
bsharma@mathworks.com | mathworks.com/academia