Advanced State Machine Programming

15-494 Cognitive Robotics
David S. Touretzky &
Ethan Tira-Thompson

Carnegie Mellon
Spring 2014

Three Mechanisms for Communication Among States

1) SignalTrans allows one state to send a message to another as part of a transition, e.g., to send an int:

    state1 =S<int>=> state2

2) Variables defined in a parent state can be accessed by children using $provide and $reference.

3) Sketch and shape spaces are shared across all states, so sketches/shapes created by one state can be accessed by another using GET_SKETCH and GET_SHAPE.

1) State Signaling

Two principal uses:

- Transmit an arbitrary value, e.g., a float or struct
- Implement an n-way branch. In this case the signal is an enumerated type.

Both are implemented by posting a DataEvent<T> and using a SignalTrans<T> to test for the event.

Shorthand notation: =S<T>=> or =S<T>(v)=>

Transmit an Arbitrary Signal

```
$nodeclass TransmitDemo : StateNode {

  $nodeclass Pitcher : StateNode : doStart {
    float x = ...;   // some arbitrary computation
    postStateSignal<float>(x);
  }

  $nodeclass Catcher : StateNode : doStart {
    float val = extractSignal<float>(event);
    cout << "Message received: " << val << endl;
  }

  $setupmachine{
      startnode: Pitcher =S<float>=> Catcher
  }


}
```

The variable event is automatically defined for you and bound to the event that caused the transition into this state. The extractSignal call will return a default float value (0.0f) if event is not an instance of DataEvent<float>.

N-Way Branch

```
$nodeclass ChooseDemo : StateNode {
  enum choice {goLeft, goRight, goStraight};

  $nodeclass Chooser : StateNode : doStart {
    float x = rand()/(1.0f + RAND_MAX);
    if ( x < 0.1 ) postStateSignale<choice>(goLeft);
    else if ( x < 0.2 ) postStateSignal<choice>(goRight);
    else postStateSignal<choice>(goStraight);

  $setupmachine{
    startnode: Chooser
    startnode =S<choice>(goLeft)=> Turn(M_PI/2)
    startnode =S<choice>(goRight)=> Turn(-M_PI/2)
    startnode =S<choice>(goStraight)=> WalkForward(100)
  }

}
```

2) Parent-Defined Variables

```
$nodeclass SharedVarDemo : StateNode {
  $provide int counter;

  $nodeclass BumpIt : StateNode : doStart {
    $reference SharedVarDemo::counter;
    ++counter;
  }

  $nodeclass Report : StateNode : doStart {
    $reference SharedVarDemo::counter;
    cout << “Counter = “ << counter << endl;
  }

  virtual void doStart {
    counter = 0;  // can't rely on constructor if called twice
  }

  $setupmachine{
      startnode: BumpIt =N=> BumpIt =N=> BumpIt =N=> Report
  }

}
```

More State Signaling

- postStateCompletion()
  - Use the =C=> transition
  - Indicates normal completion of the state's action.
- postStateFailure(), postStateSuccess()
  - Use =F=> for abnormal completion, e.g., search failed.
  - Use =S=> for a third outcome if =C=> already used
- postParentCompletion(), postParentFailure()
  - Can be used to trigger a transition out of the parent node.
  - This is how nested state machines can “return” to the parent state machine.

Nested State Machine (1)

```
$nodeclass LookForIt {

  $nodeclass TakeImage : MapBuilderNode : doStart {
    mapreq->addObjectColor(ellipseDataType,"green");
  }

  $nodeclass CheckResult : doStart {
    if ( camShS.allShapes.size() > 0 )
      postStateSuccess();
    else
      postStateFailure();
  }

  $setupmachine{
    startnode: TakeImage =C=> check

    check: Checkresult
    check =F=> startnode
    check =S=> PostMachineCompletion

  }


}
```

Nested State Machine (2)

```
$nodeclass Trample : VisualRoutinesStateNode {
  $nodeclass GoToIt :
          PilotNode(PilotTypes::goToShape) : doStart {
    NEW_SHAPEVEC(ellipses, EllipseData,
                 select_type<EllipseData>(camShS));
    if ( ellipses.size() > 0 &&
         ellipses[0]->getSemiMajor() > 10 )
      pilotreq.targetShape = ellipses[0];
    else
      cancelThisRequest();

  }

  $setupmachine{
    startnode: FindIt =C=> goto

    goto: GoToIt
    goto =F=> startnode
    goto =C=> SpeechNode("Trampled!")
  }
}
```

When You Must Use =C=>

Completions are important when motion is involved:

```
straight: HeadPointerNode[getMC()->setJoints(0,0,0)]
   =RND=> {left, right}

left: HeadPointerNode[getMC()->setJoints(0,0.5,0)]
   =T(500)=> straight

right: HeadPointerNode[getMC()->setJoints(0,-0.5,0)]
   =T(500)=> straight
```

What's the problem? The =RND=> transition won't wait for the head motion to complete. Same for =T(...)=> transition. Can only use =C=> here.

3) Accessing Sketches, Shapes

```
$nodeclass State1 : VisualRoutinesStateNode : doStart {
  NEW_SHAPE(myline, LineData,
            new LineData(camShS,
                         Point(50,50),
                         Point(100,200)));
}
```

Variable myline goes out of scope upon exiting state1::doStart, but the shape it points to persists in camShS.

```
$nodeclass State2 : VisualRoutinesStateNode : doStart {
  GET_SHAPE(myline, LineData, camShS);
  if ( myline.isValid() )
    myline->setColor("blue");
}
```

GET_SHAPE retrieves the shape from camShS and binds a new local variable with that name so we can access it.

Examine Generated C++ Code

- Calling the stateparser directly:

  > stateparser MyDemo.cc.fsm –

- Examining the .cc file generated by make:

  > cat ../build/PLATFORM_LOCAL/TGT_CALLIOPE5KP/MyDemo-fsm.cc