Model-based Programming

Outline

• Introduction & Overview
• Model-based Programming
  – Control Programs: RMPL & HCA
  – System Models: CCA
  – Model-based Program Semantics
• Execution of Model-based Programs
• Fundamentals of Model-based Reasoning
• Modeling via State Analysis
• Advanced Methods
• Conclusion
Model-based Program

Control Programs

Code specifying reactive control mechanisms:
- Concurrent Task Management
- Contingency Handling
- Resource Management
- Goal Achievement
- Constraint Maintenance

Control Programs refer to hidden states as if they were directly observable & controllable;

(Let the Deductive Controller worry about hidden state…)
Example

OrbitInsert()::
  (do-watching ( (EngineA = Firing) OR (EngineB = Firing) )
   (parallel
     (EngineA = Standby)
     (EngineB = Standby)
     (Camera = Off)
     (do-watching (EngineA = Failed)
       (when-donext ( (EngineA = Standby) AND
                       (Camera = Off) )
                     (EngineA = Firing) )
       (when-donext ( (EngineA = Failed) AND
                       (EngineB = Standby) AND
                       (Camera = Off) )
                     (EngineB = Firing) )
     )
   )
  )

Directly monitors state

Issues goals on state

Example

OrbitInsert()::
  (do-watching ( (EngineA = Firing) OR (EngineB = Firing) )
   (parallel
     (EngineA = Standby)
     (EngineB = Standby)
     (Camera = Off)
     (do-watching (EngineA = Failed)
       (when-donext ( (EngineA = Standby) AND
                       (Camera = Off) )
                     (EngineA = Firing) )
       (when-donext ( (EngineA = Failed) AND
                       (EngineB = Standby) AND
                       (Camera = Off) )
                     (EngineB = Firing) )
     )
   )
  )

Allows for pre-emption…

Parallel execution…

Conditional execution…

And many other features…
### Reactive Model-based Programming Language (RMPL)

#### RMPL Constructs (Reactive Combinators)

<table>
<thead>
<tr>
<th>Construct</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint (goal) assertion</td>
<td>( g )</td>
</tr>
<tr>
<td>Maintenance constraint</td>
<td>( \text{do-maintaining } , c , \text{A} )</td>
</tr>
<tr>
<td>Conditional execution</td>
<td>( \text{if-thennext } , c , \text{A} )</td>
</tr>
<tr>
<td>Guarded transition</td>
<td>( \text{unless-thennext } , c , \text{A} )</td>
</tr>
<tr>
<td>Full concurrency</td>
<td>( \text{parallel } , A , \text{B} )</td>
</tr>
<tr>
<td>Sequential composition</td>
<td>( \text{sequence } , A , \text{B} )</td>
</tr>
<tr>
<td>Iteration</td>
<td>( \text{always } , A )</td>
</tr>
<tr>
<td>Preemption</td>
<td>( \text{do-watching } , c , \text{A} )</td>
</tr>
<tr>
<td>Delay</td>
<td>( \text{next } , A )</td>
</tr>
<tr>
<td>Conditional execution with default behavior</td>
<td>( \text{if-thennext-elsenext } , A , \text{B} )</td>
</tr>
<tr>
<td>Temporally extended conditional execution</td>
<td>( \text{when-thennext } , c , \text{A} )</td>
</tr>
<tr>
<td>Iterated conditional execution</td>
<td>( \text{whenever-thennext } , c , \text{A} )</td>
</tr>
</tbody>
</table>

#### RMPL – Alternative Syntax

### Basic Constructs

<table>
<thead>
<tr>
<th>Construct</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint (goal) assertion</td>
<td>( g )</td>
</tr>
<tr>
<td>Maintenance constraint</td>
<td>( \text{do-maintaining } , c , \text{A} )</td>
</tr>
<tr>
<td>Conditional execution</td>
<td>( \text{if-thennext } , c , \text{A} )</td>
</tr>
<tr>
<td>Guarded transition</td>
<td>( \text{unless-thennext } , c , \text{A} )</td>
</tr>
<tr>
<td>Full concurrency</td>
<td>( \text{parallel } , A , \text{B} )</td>
</tr>
<tr>
<td>Sequential composition</td>
<td>( \text{sequence } , A , \text{B} )</td>
</tr>
<tr>
<td>Iteration</td>
<td>( \text{always } , A )</td>
</tr>
<tr>
<td>Preemption</td>
<td>( \text{do-watching } , c , \text{A} )</td>
</tr>
<tr>
<td>Delay</td>
<td>( \text{next } , A )</td>
</tr>
<tr>
<td>Conditional execution with default behavior</td>
<td>( \text{if-thennext-elsenext } , A , \text{B} )</td>
</tr>
<tr>
<td>Temporally extended conditional execution</td>
<td>( \text{when-thennext } , c , \text{A} )</td>
</tr>
<tr>
<td>Iterated conditional execution</td>
<td>( \text{whenever-thennext } , c , \text{A} )</td>
</tr>
</tbody>
</table>
Compiling RMPL to HCA

- Hierarchical Constraint Automata (HCA): graphical specification language for control programs, in the spirit of StateCharts
- Writable, inspectable by systems engineers
- Directly executable by Control Sequencer

```
OrbitInsert():
(do-watching ((EngineA = Firing) OR (EngineB = Firing))
(parallel (EngineA = Standby)
(EngineB = Standby)
(Camera = Off)
(do-watching (EngineA = Failed)
(when-done ( (EngineA = Standby) AND (Camera = Off) )
(EngineA = Firing)))
(when-done ( (EngineA = Failed) AND (EngineB = Standby) AND (Camera = Off) )
(EngineB = Firing)))
```

LEGEND:
- EAS (EngineA = Standby)
- EAF (EngineA = Failed)
- EAR (EngineA = Firing)
- EBS (EngineB = Standby)
- EBF (EngineB = Failed)
- EBR (EngineB = Firing)
- CO (Camera = Off)

```
MAINTAIN (EAR OR EBR)

MAINTAIN (EAF)
```

```
(EAS AND CO)
EAS

(EAF AND EBS AND CO)
EAF AND EBS

EBR

```

Compiling RMPL to HCA

OrbitInsert()::
(do-watching ((EngineA = Firing) OR (EngineB = Firing))
(parallel
(EngineA = Standby)
(EngineB = Standby)
(Camera = Off)
(do-watching (EngineA = Failed)
(when-done next ((EngineA = Standby) AND (Camera = Off))
(EngineA = Firing))))
(when-done next ((EngineA = Failed) AND (EngineB = Standby) AND (Camera = Off))
(EngineB = Firing))))

➢ conditioned on state constraints

MAINTAIN (EAR OR EBR)

LEGEND:
EAS (EngineA = Standby)
EAF (EngineA = Failed)
EAR (EngineA = Firing)
EBS (EngineB = Standby)
EBF (EngineB = Failed)
EBR (EngineB = Firing)
CO (Camera = Off)

MAINTAIN (EAF)

LEGEND:
EAS (EngineA = Standby)
EAF (EngineA = Failed)
EAR (EngineA = Firing)
EBS (EngineB = Standby)
EBF (EngineB = Failed)
EBR (EngineB = Firing)
CO (Camera = Off)

➢ compact encoding: multiple locations can be simultaneously marked
Compiling RMPL to HCA

OrbitInsert():

(do-watching ((EngineA = Firing) OR
 (EngineB = Firing))

(parallel
 (EngineA = Standby)
 (EngineB = Standby)
 (Camera = Off)

(do-watching (EngineA = Failed)
 (EngineB = Standby)
 (Camera = Off)

(when-donext ((EngineA = Standby) AND
 (Camera = Off))
 (EngineA = Firing)))

(when-donext ((EngineA = Failed) AND
 (EngineB = Standby) AND
 (Camera = Off))
 (EngineB = Firing)))

MAINTAIN (EAS AND CO)
EAS AND CO
MAINTAIN (EAF)
EAF
EAF AND EBS AND CO
EAF AND EBS AND CO
MAINTAIN (EAS OR EBR)
EAS
EAS (EAS AND CO)
EAS AND CO
EAF
EAF AND EBS AND CO
EAF AND EBS AND CO

LEGEND:
EAS (EngineA = Standby)
EAF (EngineA = Failed)
EAR (EngineA = Firing)
EBS (EngineB = Standby)
EBF (EngineB = Failed)
EBR (EngineB = Firing)
CO (Camera = Off)

➤ act on hidden state
OrbitInsert():

(\texttt{do-watching} \((\text{EngineA} = \text{Firing}) \\text{OR} \ (\text{EngineB} = \text{Firing}))\)

(\texttt{parallel} \n
(\text{EngineA} = \text{Standby}) \n
(\text{EngineB} = \text{Standby}) \n
(\text{Camera} = \text{Off}) \n
(\texttt{do-watching} \text{EngineA} = \text{Failed}) \n
(\texttt{when-donext} \ ((\text{EngineA} = \text{Standby}) \ \text{AND} \ (\text{Camera} = \text{Off}) ) \n
(\text{EngineA} = \text{Firing}))) \n
(\texttt{when-donext} \ ((\text{EngineA} = \text{Failed}) \ \text{AND} \ (\text{EngineB} = \text{Standby}) \ \text{AND} \ (\text{Camera} = \text{Off}) ) \n
(\text{EngineB} = \text{Firing}))))

\textbf{MAINTAIN (EAR OR EBR)}

\textbf{LEGEND:}

\begin{itemize}
\item \textit{EAS} (EngineA = Standby)
\item \textit{EAF} (EngineA = Failed)
\item \textit{EAR} (EngineA = Firing)
\item \textit{EBS} (EngineB = Standby)
\item \textit{EBF} (EngineB = Failed)
\item \textit{EBR} (EngineB = Firing)
\item \textit{CO} (Camera = Off)
\end{itemize}
Compiling RMPL to HCA

OrbitInsert()::
(\text{do-watching} \ ((\text{EngineA} = \text{Firing}) \ OR \ (\text{EngineB} = \text{Firing})))

(\text{parallel}
 (\text{EngineA} = \text{Standby})
 (\text{EngineB} = \text{Standby})
 (\text{Camera} = \text{Off})
 (\text{do-watching} \ (\text{EngineA} = \text{Failed})
  (\text{when-donext} \ ( (\text{EngineA} = \text{Standby}) \ AND \ (\text{Camera} = \text{Off}) )
   (\text{EngineA} = \text{Firing})))
 (\text{when-donext} \ ( (\text{EngineA} = \text{Failed}) \ AND \ (\text{EngineB} = \text{Standby}) \ AND \ (\text{Camera} = \text{Off}) )
  (\text{EngineB} = \text{Firing})))

\text{transition}  \ \\
\text{guard}

Compiling RMPL to HCA

OrbitInsert()::
(\text{do-watching} \ ((\text{EngineA} = \text{Firing}) \ OR \ (\text{EngineB} = \text{Firing})))

(\text{parallel}
 (\text{EngineA} = \text{Standby})
 (\text{EngineB} = \text{Standby})
 (\text{Camera} = \text{Off})
 (\text{do-watching} \ (\text{EngineA} = \text{Failed})
  (\text{when-donext} \ ( (\text{EngineA} = \text{Standby}) \ AND \ (\text{Camera} = \text{Off}) )
   (\text{EngineA} = \text{Firing})))
 (\text{when-donext} \ ( (\text{EngineA} = \text{Failed}) \ AND \ (\text{EngineB} = \text{Standby}) \ AND \ (\text{Camera} = \text{Off}) )
  (\text{EngineB} = \text{Firing})))

\text{transition}  \\
\text{guard}
Control Program Overview

- Control programs can be viewed as deterministic state transition systems, acting on the plant by asserting and checking constraints in propositional state logic.
- Propositions are assignments of state variables to values within their domains.
- Reactive combinators allow flexibility in expression of complex system behavior and dynamic relations.
- Similar to constructs in:
  - Concurrent Constraint languages (CC, TCC, PCCP, etc.),
  - Robotic execution languages (TDL, RAPs, ESL, etc.),
  - Synchronous programming languages (Esterel, Lustre, Signal, etc.)
  - Graphical specification representations (StateCharts, etc.)
Concurrent Constraint Automata

- Variant of Factored POMDP (component models, state not directly observable, next state probability distribution depends on current state and control actions)

**Engine Model**
- Off
  - \( (\text{eng Cmd} = \text{off}) \)
  - \( (\text{thrust} = \text{full}) \) & \( (\text{power in} = \text{nominal}) \) & \( (\text{flow in1} = \text{nominal}) \) & \( (\text{flow in2} = \text{nominal}) \)
- Standby
  - \( (\text{eng Cmd} = \text{standby}) \)
  - \( (\text{thrust} = \text{zero}) \) & \( (\text{power in} = \text{zero}) \)
- Firing
  - \( (\text{eng Cmd} = \text{standby}) \)

**Camera Model**
- Off
  - \( (\text{power in} = \text{zero}) \) & \( (\text{shutter} = \text{open}) \)
- Standby
  - \( (\text{power in} = \text{zero}) \) & \( (\text{shutter} = \text{closed}) \)
- Firing
  - \( (\text{power in} = \text{zero}) \) & \( (\text{shutter} = \text{open}) \)

**Modal Constraints**
- Fault modes
- Nominal modes
- Guarded probabilistic transitions

Concurrent Constraint Automata

- Variant of Factored POMDP (component models, state not directly observable, next state probability distribution depends on current state and control actions)

**Engine Model**
- Off
  - \( (\text{eng Cmd} = \text{off}) \)
  - \( (\text{thrust} = \text{full}) \) & \( (\text{power in} = \text{nominal}) \) & \( (\text{flow in1} = \text{nominal}) \) & \( (\text{flow in2} = \text{nominal}) \)
- Standby
  - \( (\text{eng Cmd} = \text{standby}) \)
  - \( (\text{thrust} = \text{zero}) \) & \( (\text{power in} = \text{zero}) \)
- Firing
  - \( (\text{eng Cmd} = \text{standby}) \)

**Camera Model**
- Off
  - \( (\text{power in} = \text{zero}) \) & \( (\text{shutter} = \text{closed}) \)
- Standby
  - \( (\text{power in} = \text{zero}) \) & \( (\text{shutter} = \text{closed}) \)
- Firing
  - \( (\text{power in} = \text{zero}) \) & \( (\text{shutter} = \text{open}) \)

**Modal Rewards**
- \( P_1 = 0.1\% \)
- \( P_1 = 99.9\% \)
Translating CCA to Propositional Logic

• System Model captured as CCA

\[
\text{Valve}
\]

\begin{align*}
\text{mode} = \text{open} & \Rightarrow (p_{in} = p_{out}) \land (f_{in} = f_{out}) \\
\text{mode} = \text{closed} & \Rightarrow (f_{in} = 0) \land (f_{out} = 0) \\
(mode = \text{open}) \land (cmd-in = \text{close}) & \Rightarrow (next (mode = \text{closed})) \\
(mode = \text{closed}) \land (cmd-in = \text{open}) & \Rightarrow (next (mode = \text{open})) \\
\ldots
\end{align*}

Translating CCA to Propositional Logic

• System Model captured as CCA
• CCA representation translates directly to clauses in propositional logic
• Logical representation is used by reasoning algorithm in Deductive Controller
Semantics of Model-based Programs

- The power of Model-based Programming lies in its rigorous underlying semantics
  - Defines the theoretical underpinnings of the approach
  - Allows us to derive useful properties of model-based programs
  - Allows us to make certain simplifying assumptions that enable tractable on-line deduction

- Formal computational model has its roots in:
  - Automata Theory
  - Markov Decision Theory
  - Control Theory

Control Program Semantics

Semantic Model: Deterministic Automaton
- program locations
  - "state" of program at time t
- transitions between locations
  - conditioned on plant states
- configuration goals associated with each program location
  - goal states for plant
Control Program Semantics

- Control program represented as a deterministic automaton:

\[ CP = \langle L_{cp}, \lambda_{cp}, \tau_{cp}, g_{cp}, \Sigma_s \rangle \]

- Set of program locations
- Set of all feasible system states
- Initial program location
- Transitions between locations, conditioned on state
- Config. goal \( g_{cp}(l) \subseteq \Sigma_s \)

System Model Semantics

- Model-based Program
  - Control Program
  - System Model
- Semantic Model: Factored POMDP
  - Variables (state/control/observable)
  - State (assignments to variables)
  - Transition functions
  - Transition probabilities
  - Observation probabilities
  - Rewards on state
System Model Semantics

- Variables:
  \[ \Pi = \{ \Pi^s, \Pi^c, \Pi^o \} \]
  - State vars \( \downarrow \)
  - Control vars \( \downarrow \)
  - Obs vars \( \downarrow \)

- Factored POMDP:
  \[ SM = < \Sigma, T, P_{\theta}, P_{\mu}, P_{o}, R > \]
  - Transitions \( \tau : \Sigma \rightarrow \Sigma \)
  - Initial state prob \( P_{\theta}(s_0) \)
  - Transition prob \( P_T(s' | s, \mu) \)
  - Obs prob \( P_o(o | s) \)
  - State reward \( R(s) \)

Model-based Program Semantics

- Can view all possible evolutions of system state, given a model-based program, in the form of a trellis diagram:
Summary

- Model-based program:
  - specification of state intent (Control Program)
  - specification of state behavior (System Model)

- Control Program:
  - Textual (procedural) and graphical representations
  - Hierarchical constructs provide flexibility & expressivity
  - Semantically represented as a deterministic automaton

- System Model:
  - Textual (logical) and graphical representations
  - Compositional description of system provides modularity
  - Semantically represented as a Factored POMDP

Up Next…

- Introduction to Model-based Execution:
  - Control Sequencer
  - Deductive Controller

![Diagram](image-url)