Reactive Planning of Hidden States in Large State Spaces Through Decomposition and Serialization

Brian C. Williams
Joint with Seung H. Chung

16.412/18.834J
March 16th, 2005

Outline

• The need for model-based reactive planning
• The Burton model-based reactive planner

How do we reconfigure a valve?

• Device modes are changed through indirect commanding.
• Communication paths are established by reconfiguring other devices.
• The task of reconfiguring devices in the proper order generalizes state-space planning to handle indirect effects.
• To achieve reactivity all possible plans for all possible goal states should be pre-compiled (a generalization of universal plans).
  • To achieve compactness we decompose these universal plans according to a goal/sub-goal hierarchy.

DS 1 Attitude Control System

Mode Reconfiguration

• Device modes are changed through indirect commanding.
• Communication paths are established by reconfiguring other devices.
• The task of reconfiguring devices in the proper order generalizes state-space planning to handle indirect effects.
• To achieve reactivity all possible plans for all possible goal states should be pre-compiled (a generalization of universal plans).
  • To achieve compactness we decompose these universal plans according to a goal/sub-goal hierarchy.
Example: Driver Valve Command Sequence

Goal: No thrust

<table>
<thead>
<tr>
<th>Commands</th>
<th>Driver State</th>
<th>Valve State</th>
</tr>
</thead>
<tbody>
<tr>
<td>ME: dr = off, vlv = open</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GI: dr = off, vlv = closed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MRP dcmdin = on ME: dr = on, vlv = open</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MRP dcmdin = close ME: dr = reset failure, vlv = open</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MRP dcmdin = reset ME: dr = on, vlv = open</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MRP dcmdin = off ME: dr = off, vlv = open</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

To achieve reactivity we eliminate all forms of search.

Model-based Reactive Planning

Achieved by:

1. Eliminate Indirect Control 
   . . . through Compilation
2. Eliminate Search for Goal Ordering 
   . . . through Reversibility and Serialization
3. Eliminate Search to find Suitable Transitions
   . . . by Constructing Hierarchical Policies

To Handle Indirect Control . . . 

. . . Compile Out Constraints
To Compile Out Constraints

- Eliminate intermediate variables.
- Transitions are conditioned on mode and control variables.
- Generate transitions as prime implicates:
  \[ \Phi_i \models \text{next}(y_i = e_i) \]
  where \( \Phi_i \) is a conjunction of mode and control variable assignments.
- Prime implicates for transitions enumerated using OpSAT:
  - 40 seconds on SPARC 20 for 12,000 clause spacecraft model.

Model-based Reactive Planning

Achieved by:

1. Eliminate Indirect Effects
   - through Compilation
2. Eliminate Search for Goal Ordering
   - through Reversibility and Serialization
3. Eliminate Search to find Suitable Transitions
   - by Constructing Hierarchical Policies

Why Search is Needed

1) An achieved goal can be clobbered by a subsequent goal.

\[
\text{command} \quad \text{Driver} \quad \text{Valve}
\]

- Example
  - Current State: driver = on, valve = closed
  - Goal State: driver = off, valve = open
  - Achieving (driver = off) and then (valve = open) clobbers (driver = off)
  \( \Rightarrow \) Achieve Valve goal before Driver goal

Observation: Engineered systems tend not to have loops

- Define: Causal Graph \( G \) of compiled transition system \( S \)
  - vertices are state variables.
  - edge from \( v_i \) to \( v_j \) if \( v_j \) transition is conditioned on \( v_i \).
- Requirement: The causal graph is acyclic.
  \( \Rightarrow \) Work conjunctive goals upstream from outputs to inputs

Solution

- The only variables used to set some variable \( y_i \) is its ancestors,
  \( \Rightarrow y_i \) can be changed without affecting its descendants.

\[
\text{Affected} \quad \text{Unaffected}
\]

- Safe to achieve goals in an upstream order.
- Simple check:
  - Number causal graph depth first
  - achieve goals in order of increasing depth first number.
Why Search is Needed

2) Two goals can compete for the same variable in their subgoals.

- Example
  - Latch1 and Latch2 compete for the position of Switch if latch goals achieved concurrently.

Solution: Mark Allowed Transitions/Assignments

- Mark all control variable assignments allowed:
  - For each mode variable v, in decreasing order of DF number:
    - Select each transition of v whose guard has only allowed assignments.

Why Search is Needed

3) A state transition of a subgoal variable has irreversible effect.

- Example
  - Assume Switch can be used once,
  - Then Latch1 must be latched before Latch2.
  - But irreversible effects aren’t desirable for reactive planners
    - Don’t allow irreversible actions
    - . . . Except to repair failure modes

Solution: Mark Allowed Transitions/Assignments

- Mark all control variable assignments allowed:
  - For each mode variable v, in decreasing order of DF number:
    - Select each transition of v whose guard has only allowed assignments.

Solution: Mark Allowed Transitions/Assignments

- For each mode variable v, in decreasing order of DF number:
  - Select each transition of v whose guard has only allowed assignments.
  - Given current assignment v = I for v:
    - Find strongly connected component of selected transitions that contains I.
    - Mark assignments and transitions in SCC allowed.
Model-based Reactive Planning

Achieved by:

1. Eliminate Indirect Effects
   . . . through Compilation

2. Eliminate Search for Goal Ordering
   . . . through Reversibility and Serialization

3. Eliminate Search to Find Suitable Transitions
   . . . by Constructing Hierarchical Policies

Plan by passing sub-goals up causal graph

Goal: Driver = off, Valve = closed
Current: Driver = off, Valve = open
Plan by passing sub-goals up causal graph

Goal: Driver = off, Valve = closed
Current: Driver = off, Valve = open

Plan by passing sub-goals up causal graph

Goal: Driver = off, Valve = closed
Current: Driver = off, Valve = open

Plan by passing sub-goals up causal graph

Goal: Driver = off, Valve = closed
Current: Driver = resettable, Valve = open

Plan by passing sub-goals up causal graph

Goal: Driver = off, Valve = closed
Current: Driver = resettable, Valve = open

Plan by passing sub-goals up causal graph

Goal: Driver = off, Valve = closed
Current: Driver = on, Valve = open

Plan by passing sub-goals up causal graph

Goal: Driver = off, Valve = closed
Current: Driver = on, Valve = open
Plan by passing sub-goals up causal graph

Goal: Driver = off, Valve = closed
Current: Driver = on, Valve = closed

Send cmd = off

Driver 2

Valve 1

Goal
On
Off

Current
On
Off

Resettable

Open
Closed
Stuck

Send cmd = off

Driver 2

Valve 1

Goal
On
Off

Current
On
Off

Resettable

Open
Closed
Stuck

Hierarchical, Model-based Reactive Planning

• Compile-time Analysis:
  – Compile-out interactions
  – Confirm schematics are loop free.
  – Depth first number variables.
• Periodic, Run-time Analysis:
  – Given initial state
    • Identify allowed transitions and assignments
  – Given autonomous jump to failure state
    • Identify allowed transitions and assignments
• Run-time Plan Execution:
  – Work conjunctive goals from outputs to inputs.
  – Achieve goals serially.
  – Only perform reversible transitions.
  – Lookup control actions and sub-goals in policies

Complexity of Reactive Planning

• Worst Case per action: Depth * Sub-goal branch factor
• Average Cost per action: Sub-goal branch factor

What If Plan is Not Serializable?

• What if causal graph G contains cycles?
• Solution:
  – Isolate the cyclic components (compute SCCs)
  – Compose each cycle into a single component.
  – Create a new causal graph G' that is acyclic.
  – Goals of G' are serializable

Composing Cyclic Components
Policy for Composed Components

- Problem: Composition grows exponential in space usage.
- Solution: Use BDD encoding (Seung Chung SM).

Goal

<table>
<thead>
<tr>
<th>Current</th>
<th>OnT, OnA</th>
<th>OnT, OffA</th>
<th>OffT, OffA</th>
<th>OffT, OnA</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>cmdT = off</td>
<td>idle</td>
<td>cmdT = off</td>
<td>idle</td>
</tr>
<tr>
<td>cmdT = on</td>
<td>idle</td>
<td>cmdT = off</td>
<td>idle</td>
<td>idle</td>
</tr>
<tr>
<td>idle</td>
<td>cmdT = on</td>
<td>idle</td>
<td>cmdT = off</td>
<td>idle</td>
</tr>
<tr>
<td>idle</td>
<td>idle</td>
<td>idle</td>
<td>idle</td>
<td>idle</td>
</tr>
</tbody>
</table>

Model-based Reactive Planning

1. Compile away constraints from the model
2. Compile away cyclic components
3. Plan serially pursuing causal components upstream
4. Generate actions using hierarchical policies

Only performs reversible actions
Responds to failure at each step
Average cost per step = subgoal branching factor

Next Challenge: Mars Smart Lander (2009)

Mission Duration: 1000 days
Total Traverse: 3000-69000 meters
Meters/Day: 230-450
Science Mission: 7 instruments, sub-surface science package (drill, radar), in-situ sample lab
Technology Demonstration: (2005).

Model-based Programming of Embedded Systems

- To survive decades embedded systems orchestrate complex regulatory and immune systems.
- Future systems will be programmed with models, describing themselves and their environments.
- Runtime kernels will be agile, deducing and planning by solving optimization problems with propositional constraints.
- Model-based reactive planners respond quickly to failure, while using compile-time analysis of structure to respond quickly and concisely to indirect effects.

Appendix 1
Handling Cycles In The Causal Graph
Slides by Seung Chung

Telecommunication Subsystem Example
• Turning the transmitter on or off can generate a noise (i.e. transient signal).
• The transient signal may damage the amplifier.
• The amplified transient signal may damage other devices downstream of the amplifier.

Constraint on the system:
– The amplifier must be turned off before the transmitter can be turned on or off.
– The transmitter must be turned on before the amplifier can be turned on.

Composing Strongly Connected CA

Interdependent Concurrent Transitions

Simultaneous Commanding

Concurrent Automata (CA)

Assuring Proper Execution of Interdependent Transitions