

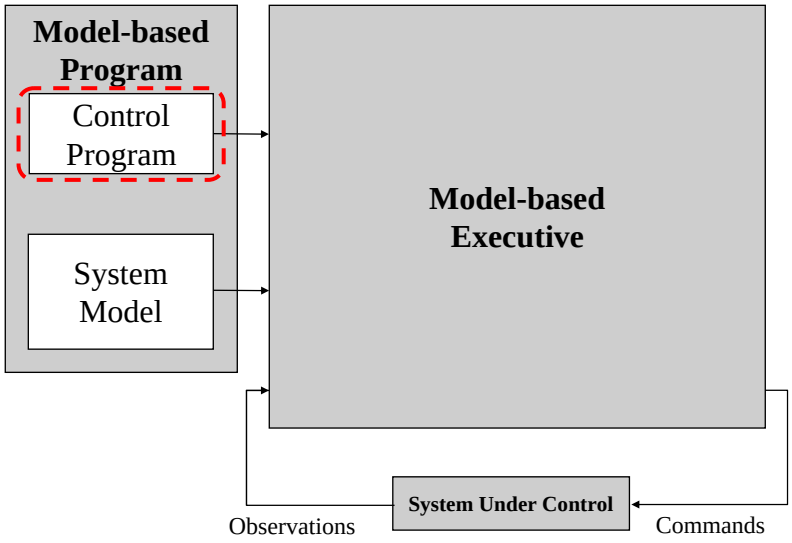

# Model-based Programming




## *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

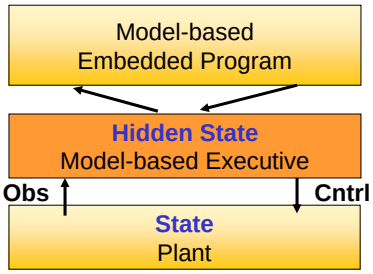



3

## 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…)

4

## 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**

5

## 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…**

6

## Reactive Model-based Programming Language

| | RMPL Constructs (Reactive Combinators) | |
|---|---|---|
| Basic Constructs | constraint (goal) assertion | g |
| | maintenance constraint | A maintaining c |
| | conditional execution | if c thennext A |
| | guarded transition | unless c thennext A |
| | full concurrency | A , B |
| | sequential composition | A ; B |
| | iteration | always A |
| | preemption | do A watching c |
| Derived Constructs | delay | next A |
| | conditional execution with default behavior | if c thennext A elsenext B |
| | temporally extended conditional execution | when c donext A |
| | iterated conditional execution | whenever c donext A |

7

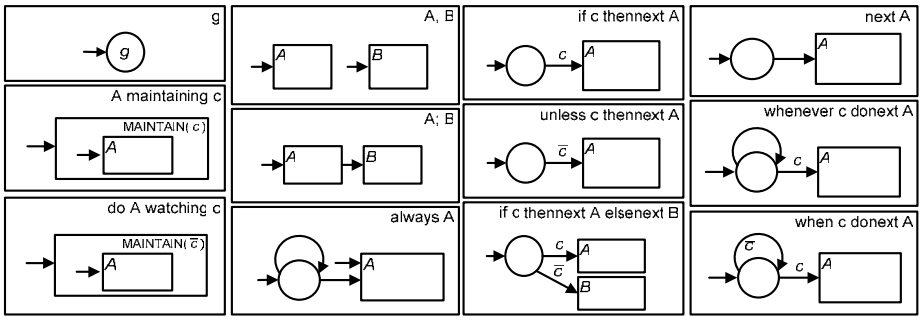
## RMPL – Alternative Syntax

| | RMPL Constructs (Reactive Combinators) | |
|---|---|---|
| Basic Constructs | constraint (goal) assertion | (g) |
| | maintenance constraint | (do-maintaining c A) |
| | conditional execution | (if-thennext c A) |
| | guarded transition | (unless-thennext c A) |
| | full concurrency | (parallel A B) |
| | sequential composition | (sequence A B) |
| | iteration | (always A) |
| | preemption | (do-watching c A) |
| Derived Constructs | delay | (next A) |
| | conditional execution with default behavior | (if-thennext-elsenext c A B) |
| | temporally extended conditional execution | (when-donext c A) |
| | iterated conditional execution | (whenever-donext c A) |

8

## Compiling RMPL to HCA

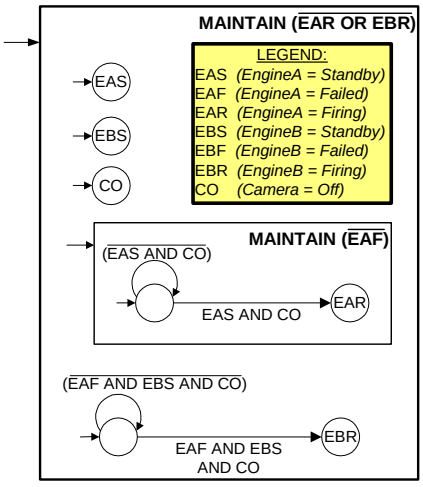
- Hierarchical Constraint Automata (HCA): graphical specification language for control programs, in the spirit of StateCharts
- Writable, inspectable by systems engineers
- Directly executable by Control Sequencer




9

## Compiling RMPL to HCA

```
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))))
```

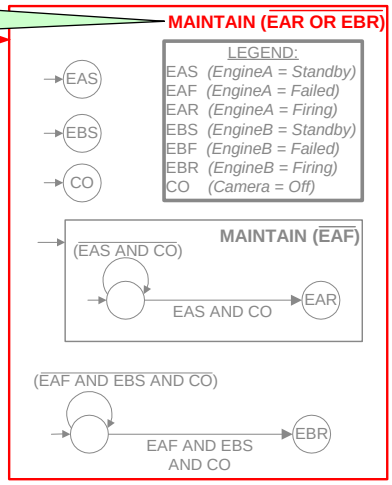



10

## Compiling RMPL to HCA

maintenance constraint

```
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))))
```

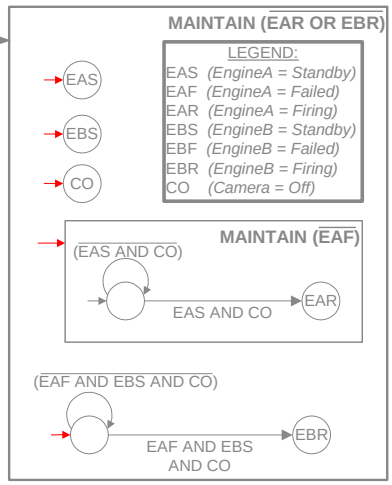



- *conditioned on state constraints*

11

## Compiling RMPL to HCA

```
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))))
```




- *compact encoding: multiple locations can be simultaneously marked*

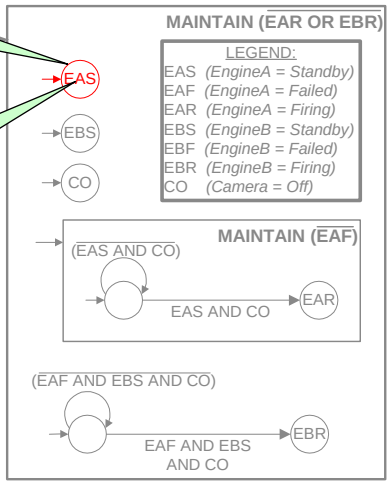
12

## Compiling RMPL to HCA

primitive location

goal constraint (hidden state)

```
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))))
```

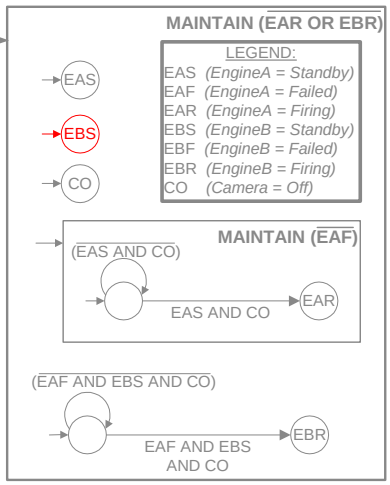



- *act on hidden state*

13

## Compiling RMPL to HCA

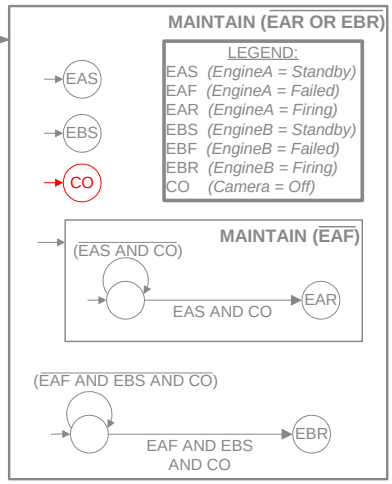
```
OrbitInsert()::
(do-watching ((EngineA = Firing) OR
              (EngineB = Firing))
    (parallel
      (EngineA = Standby)
      (EngineB = Standby)
      (Camera = Off)
      (do-watching (EngineA = Failed)
          (when-donext ( (EngineA = Standby) AND
                         (Camera = Off)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




14

## Compiling RMPL to HCA

```
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))))
```

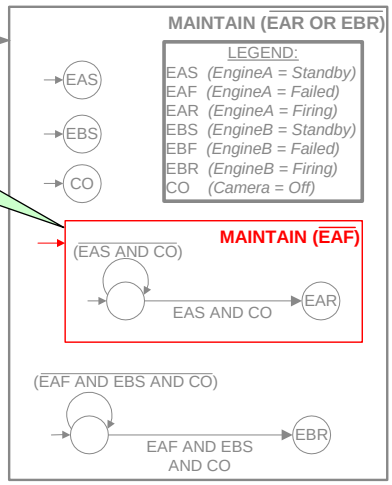



15

## Compiling RMPL to HCA

```
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))))
```

composite location



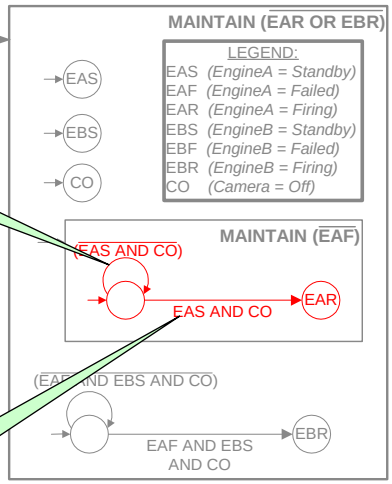

16

## Compiling RMPL to HCA

```
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))))
```

transition

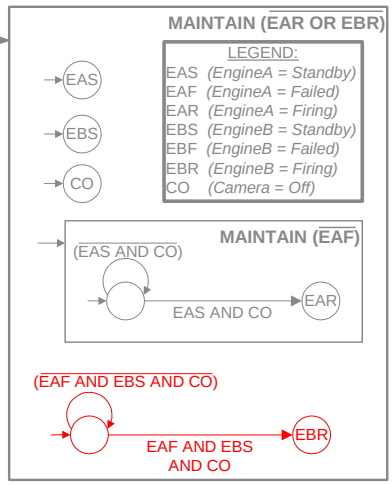
transition guard




17

## Compiling RMPL to HCA

```
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))))
```



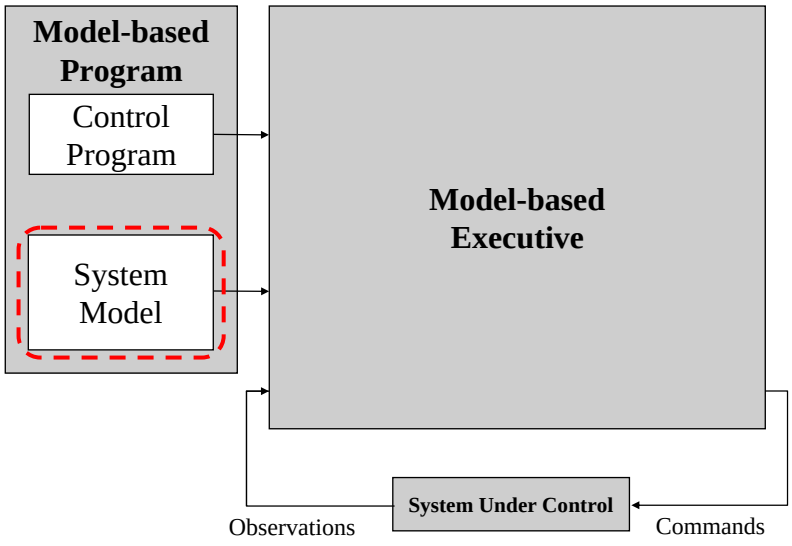

18

## 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.)

19

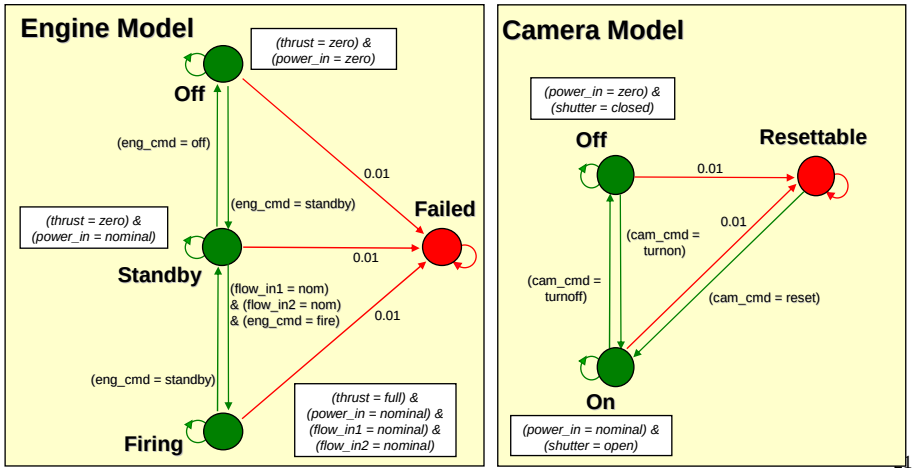
## Model-based Program




20

## Concurrent Constraint Automata

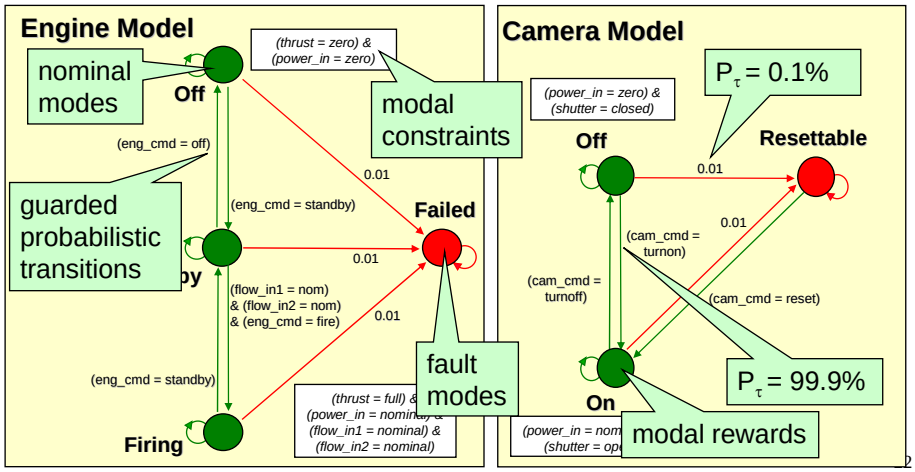
- Variant of *Factored POMDP* (component models, state not directly observable, next state probability distribution depends on current state and control actions)





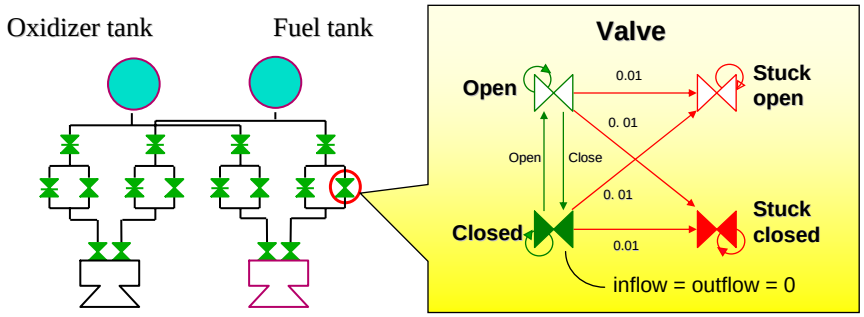

## Concurrent Constraint Automata

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



## Translating CCA to Propositional Logic

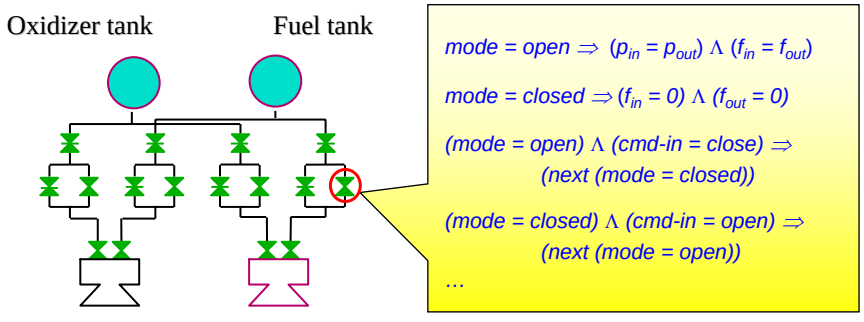
- System Model captured as CCA






## 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

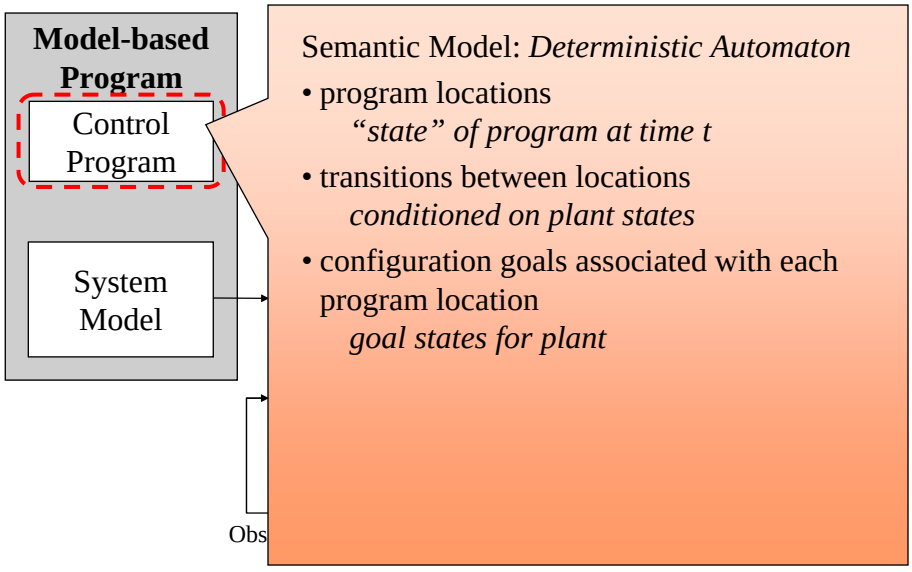


## Semantics of Model-based Programs

- 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

25

## Control Program Semantics




26

## Control Program Semantics

- Control program represented as a *deterministic automaton*:

set of program locations

set of all feasible system states

$$CP = < L_{cp} , \lambda_{cp} , \tau_{cp} , g_{cp} , \Sigma_s >$$

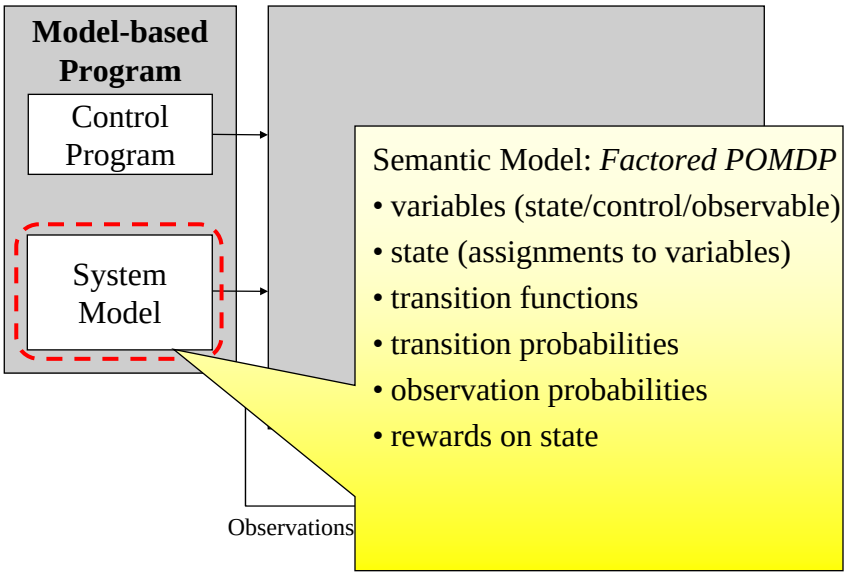
initial program location

transitions between locations, conditioned on state

config. goal $g_{cp}(l) \subset \Sigma_s$

27

## System Model Semantics




28

## System Model Semantics

- Variables:

$$\Pi = \{\Pi^s, \Pi^c, \Pi^o\}$$

state vars | control vars | obs vars

$\Sigma$: full assignments $\sigma$ over all vars in $\Pi$
$\Sigma_s$: plant states $s$
$\Sigma_c$: control actions $\mu$
$\Sigma_o$: observations $o$

- Factored POMDP:

$$SM = < \Sigma, T, P_\Theta, P_T, P_O, R >$$

transitions $\tau : \Sigma \rightarrow \Sigma_s$

initial state prob $P_\Theta(s_0)$

transition prob $P_T(s' \mid s, \mu)$

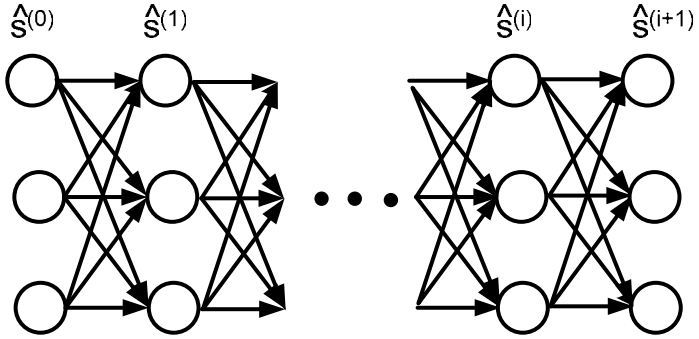
obs prob $P_O(o \mid s)$

state reward $R(s)$

29

## Model-based Program Semantics

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




30

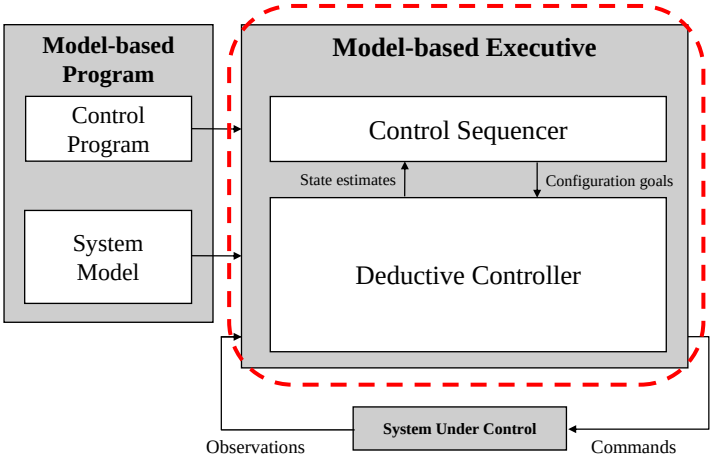
## Summary

- 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

31

## Up Next…

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




32