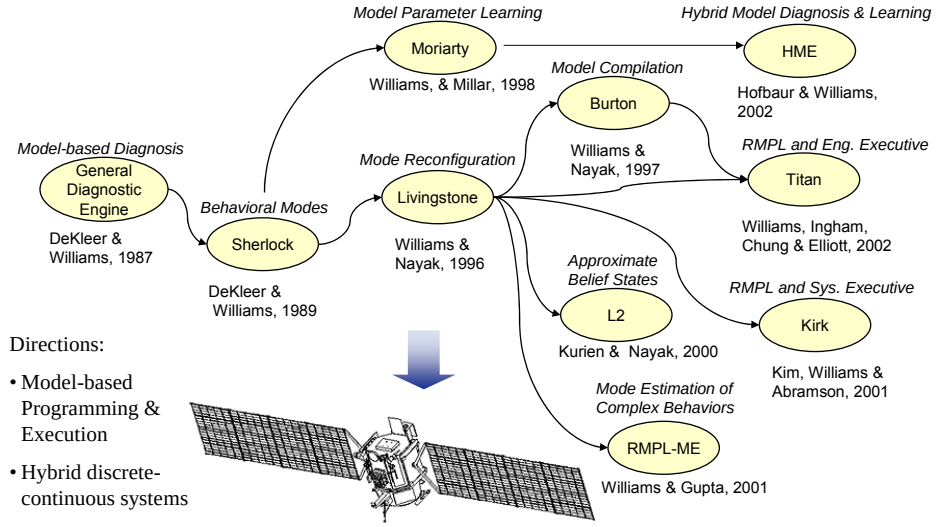
Advanced Methods in Model-based Autonomy

Outline

- Introduction & Overview
- Model-based Programming
- Execution of Model-based Programs
- Fundamentals of Model-based Reasoning
- Modeling via State Analysis
- Advanced Methods
  - Timed Model-based Programming
  - Hybrid Model-based Programming
  - Model-based Temporal Planning
  - Integration of Activity Planning and Path Planning
  - Verification of Model-based Programs
- Conclusion

Model-based Autonomy Heritage



Directions:

- Model-based Programming & Execution
- Hybrid discrete-continuous systems
- Complex & collaborative behaviors

NMP Deep Space 1 Remote Agent

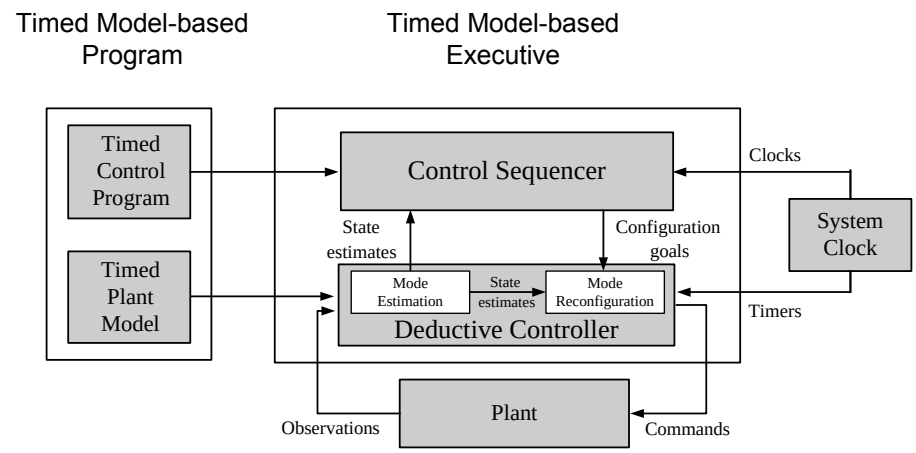
3

Related Work

- State-based Specifications
  - StateCharts (Harel, '87)
  - Timed StateCharts (Kesten & Pnueli, '92)
- Synchronous Programming
  - Esterel (Berry & Gonthier, '92)
  - Lustre (Halbwachs, '93)
- Constraint Programming
  - TCC (Saraswat, Jagadeesan & Gupta, '94)
- Robotic Execution
  - RAPs (Firby, '89)
  - ESL (Gat, '96)
  - TDL (Simmons, '98)
- Timed Formal Modeling
  - Timed Transition Systems (Henzinger, Manna, & Pnueli, '92)
  - Timed Automata (Alur & Dill, '94)
- Model-based Execution
  - GDE, Sherlock (deKleer & Williams, '87-'89)
  - Livingstone (Williams & Nayak, '96-'97)
  - Livingstone2 (Kurien & Nayak, '00)
- Model-based Programming
  - RBurton (Williams & Gupta, '99)
  - Titan (Williams, Ingham, Chung & Elliott, '03)
- Mission Data System
  - MDS (Dvorak, Rasmussen, et al., '00)

4

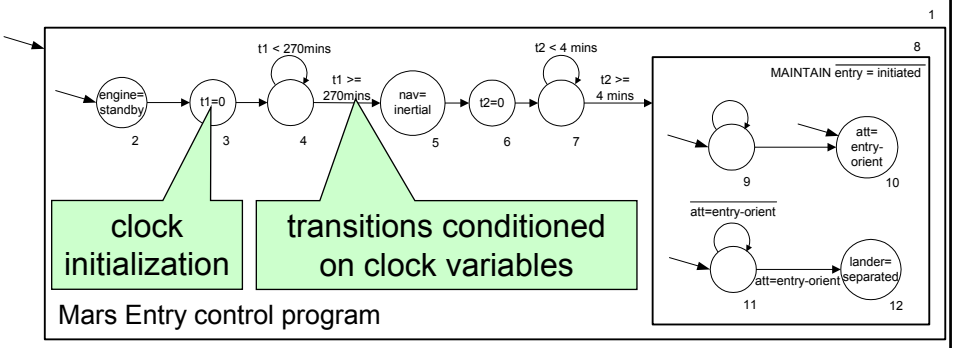
Timed Model-based Execution Architecture



5

Timed Hierarchical Constraint Automata

- Graphical specification language for control programs, in spirit of Timed StateCharts
- Extend Hierarchical Constraint Automata to timed behavior

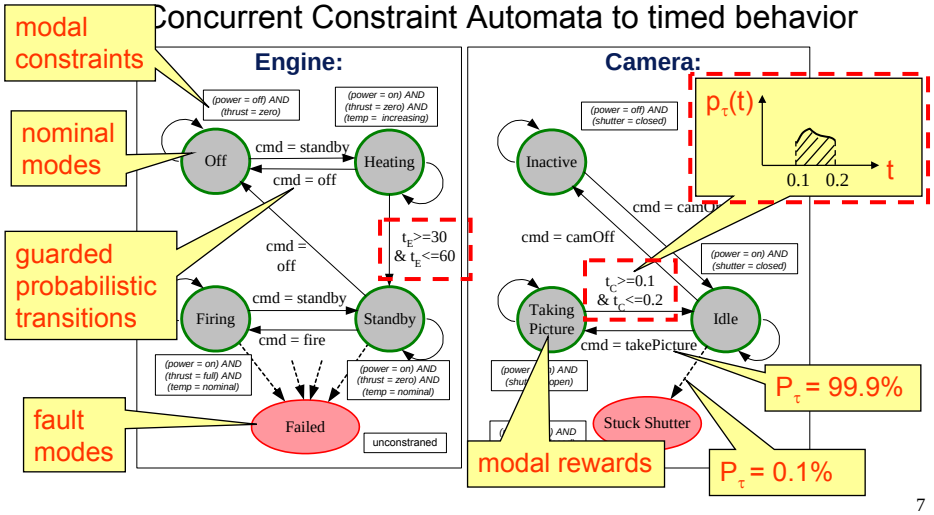


- *clocks provide timing mechanism*
- *conditioned on time & state constraints*

6

Timed Concurrent Constraint Automata

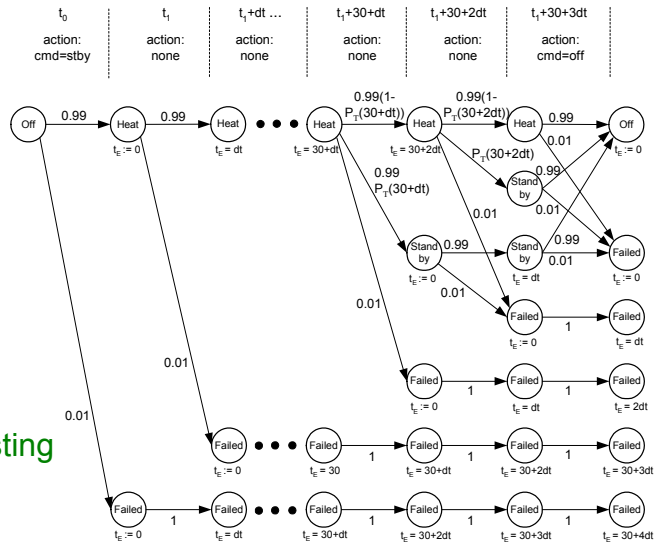
- Variant of *Factored POSMDP* (state not directly observable, next state depends on current state *& time spent in state*)
- Concurrent Constraint Automata to timed behavior



7

Timed Mode Estimation

- For physical plants modeled as TCCA (POSMDP):



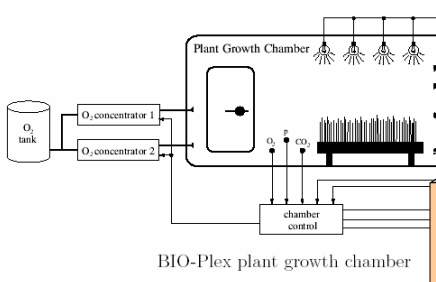
Bad news:
state space gets much larger…

Good news:
can leverage existing OPSAT engine!

8

Hybrid Modeling and Estimation

- Model the system as a network of Probabilistic Hybrid Automata
- Frame fault diagnosis as state estimation in this model

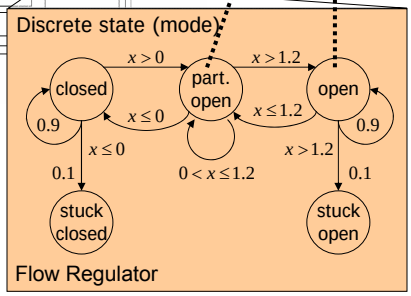


BIO-Plex plant growth chamber

Continuous state: flow x

$$\begin{cases} x_{k+1} = q_{cmd} + \mathcal{N}(0,10^{-4}) \\ q_{out} = x \end{cases}$$

$$q_{out} = 8.0$$



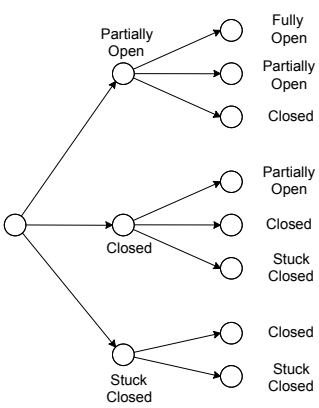
Innovative features:

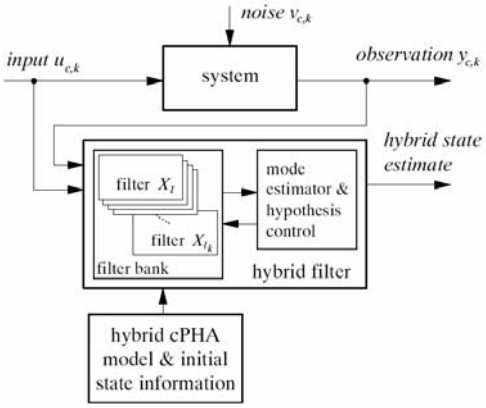
- non-linear dynamics
- mode transitions dependent on continuous state
- Detect failures or mode changes from **subtle symptoms**

9

Hybrid Estimation

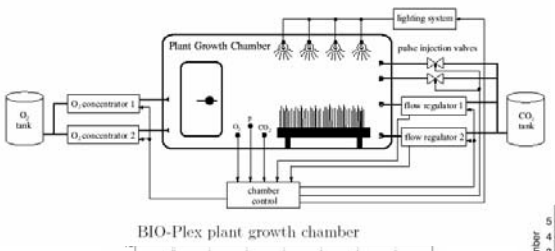
- Track a set of mode sequences with a bank of Kalman Filters



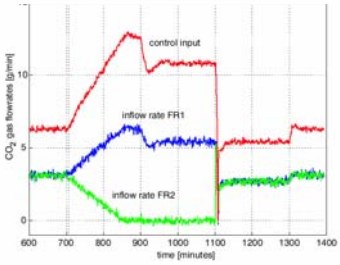


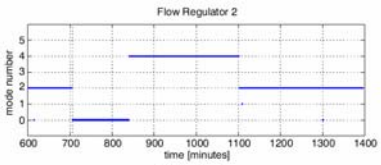
10

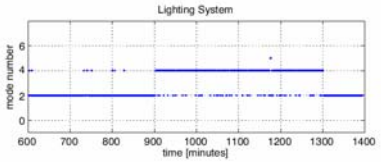
Experimental Results: K-best filtering



BIO-Plex plant growth chamber

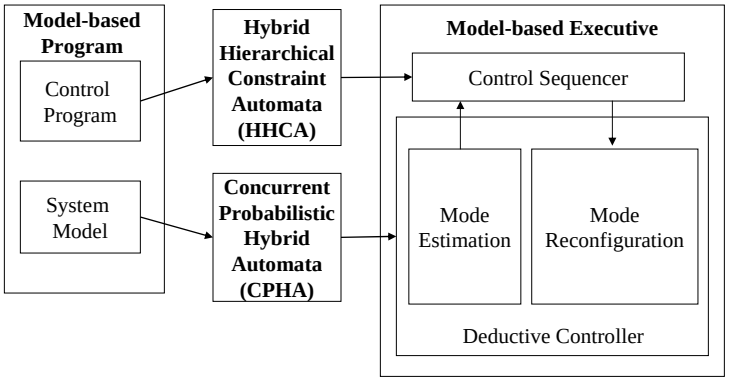






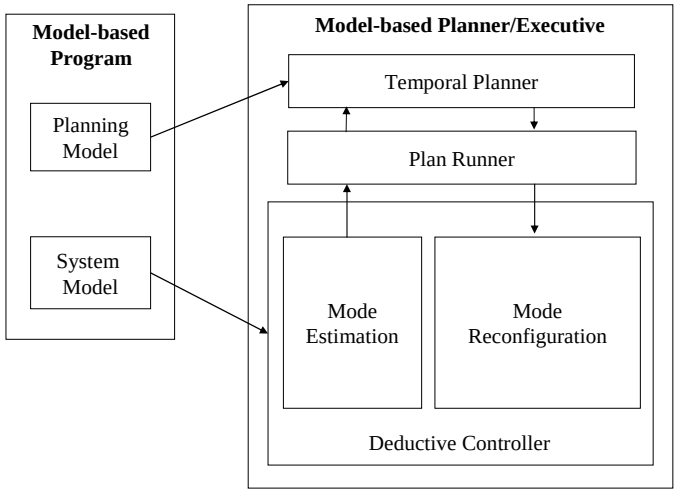
11

Hybrid Model-based Execution



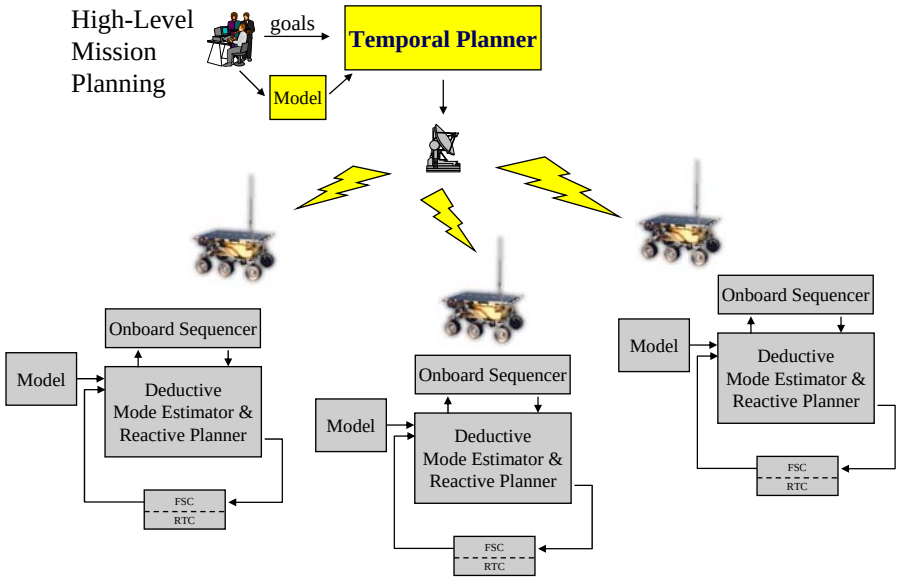
12

Model-based Temporal Planning & Execution



13

Model-based Temporal Planning



14

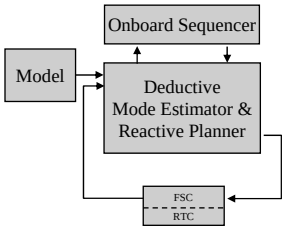
Model-based Temporal Planning

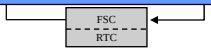
High-Level Mission Planning

goals → **Temporal Planner**

Reactive, Temporal Planner:

- Fast planning performed as graph search
- Encoding of non-deterministic choice
- Conditional planning via encoding of pre/post conditions and maintenance conditions.





15

Model-based Temporal Planning

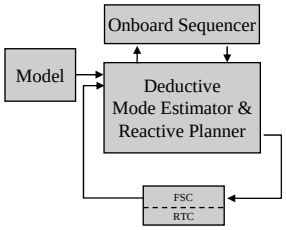
High-Level Mission Planning

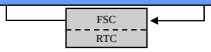
goals → **Temporal Planner**

Input: Planning models specified as RMPL models

Processing: Transforms RMPL models into intermediate HCA representation, then from HCA to Temporal Plan Network (TPN) representation.

Output: Temporally-constrained network of events





16

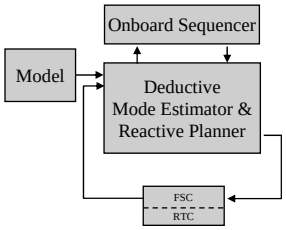
Model-based Temporal Planning

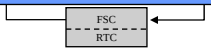
High-Level Mission Planning

goals → **Temporal Planner**

Code for describing:

- Concurrent Task Management
- Resource Management
- Temporal Constraints
- Multiple Method Selection





17

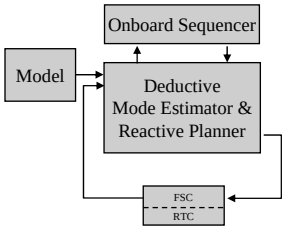
Model-based Temporal Planning

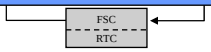
High-Level Mission Planning

goals → **Temporal Planner**

RMPL Constructs:

• constraints	c
• conditional execution	if c thennext A
• guarded transition	unless c thennext A
• full concurrency	A , B
• iteration	always A
• **non-deterministic choice**	**choose A, B, …**
• **activity timing**	**A within time [t^-,t^+]**





18

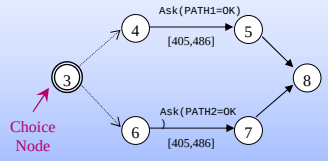
Model-based Temporal Planning

High-Level Mission Planning

goals

Temporal Planner

TPN Representation



1. STN-like temporal representation of upper & lower bounds between events
2. Choice encoded via choice nodes (i.e. only one trajectory branches from it)
3. Conditions encoded via Ask(C) & Tell(C)

Onboard Sequencer

Model

Deductive Mode Estimator & Reactive Planner

FSC RTC

FSC RTC

19

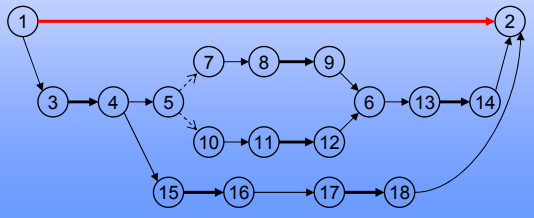
Model-based Temporal Planning

High-Level Mission Planning

goals

Temporal Planner

RMPL Planning Model can be compiled into a Temporal Planning Network



Onboard Sequencer

Model

Deductive Mode Estimator & Reactive Planner

FSC RTC

FSC RTC

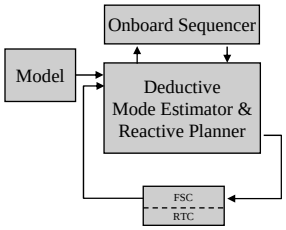
20

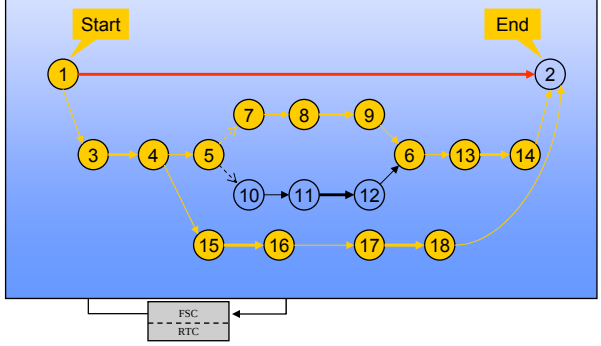
Model-based Temporal Planning

High-Level Mission Planning

goals → **Temporal Planner**

Planner traces set of paths from start to end of TPN that satisfy temporal & state constraints. Paths correspond to temporally consistent execution threads.



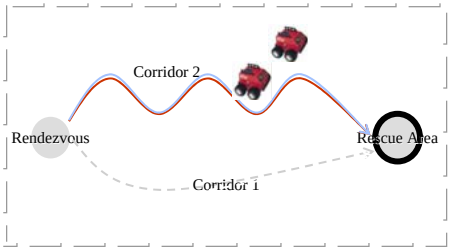


21

Programming Teams in RMPL

RMPL Programs

- Describe concurrent sensing, actuation and movements activities.
- **Choose** specifies redundant strategies and contingencies.
- **[A,B]** Specifies timing constraints.



Enroute

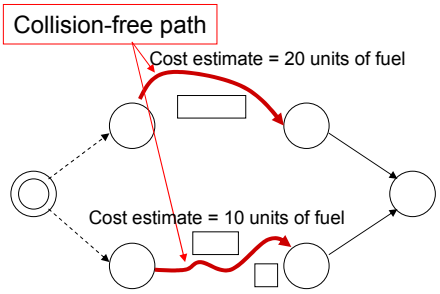
```
(Group-Enroute() [l,u] (
(sequence
   choose (
      (do-watching (PATH1=OK)
         ((Group-Traverse-Path(PATH1_1,PATH1_2,PATH1_3,RE_POS))[l*90%,u*90%])
      )
      (do-watching (PATH2=OK)
         ((Group-Traverse-Path(PATH2_1,PATH2_2,PATH2_3,RE_POS))[l*90%,u*90%])
      ))
   (parallel
      ((Group-Transmit(OPS,ARRIVED))[0,2])
      (do-watching(PROCEED=SIGNALLED)
         ((Group-Wait(HOLD1,HOLD2))[0,u*10%]))
)))
```

22

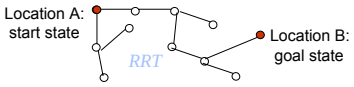
Integrated Activity & Path Planning for Agile Teams

Integrated Activity Planning & Path Planning:

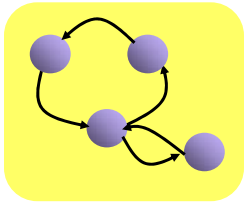
- Search a temporal plan network in best-first order
- Dynamically compute collision-free paths for those plan activities that require moving between locations and the estimated cost of flying along this path
- Continuously interleave activity and path planning to pursue the most promising plan.



Path Planning Method 1:
Explore state space using Rapidly-exploring Random Trees (RRTs)



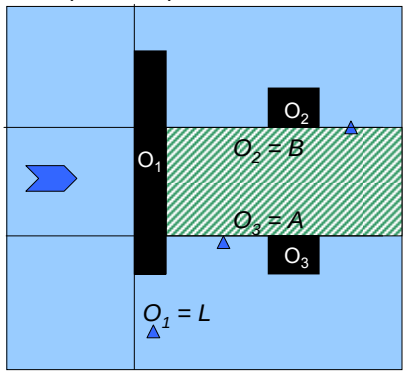
Maneuver Automaton: Describes a set of agile maneuvers with respect to the vehicle's dynamics



Path Planning Method 2:
Clausal Linear Programming

Path Planning through Clausal LP

A simple example:



- Mathematically solving the problem of vehicle control normally involves straightforward Linear Programming
- But the addition of obstacle avoidance introduces an ***Integer Programming*** element
- This makes the problem difficult to solve "online": fast enough for actual vehicles in motion
- To resolve this we transform obstacle and collision avoidance into a ***Constraint Satisfaction Problem***: For each obstacle, the domain is split into four regions (above, below, left, right), one of which is selected
- Integrating the selection of domains with the standard vehicle control leads to a an algorithm that can be used as a ***Hybrid CSP/LP Solver***

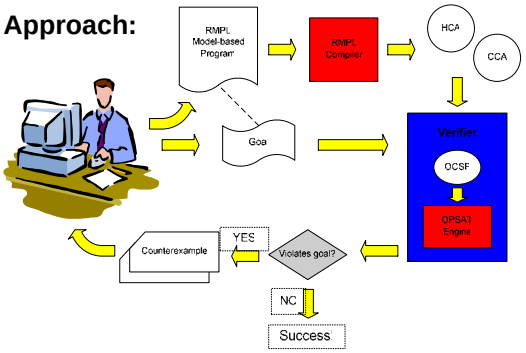
- $\mathbf{s}_i+1 = \mathbf{A}\mathbf{s}_i + \mathbf{B}\mathbf{u}_i$ State Evolution Equation
- $s_{ij} \leq w_{ij}$, etc. State Space Constraints
- $x_i \geq x_{min} \vee x_i \leq x_{max} \vee y_i \geq y_{min} \vee y_i \leq y_{max}$ Obstacle Avoidance (for all time i)
- Similar equation for Collision Avoidance (for all pairs of vehicles)

Verification of Model-based Programs

Motivation:

- Want robust autonomous systems.
- Extend traditional scenario-based testing to verification and validation (V&V).

Approach:



Goals:

- Verify RMPL model-based programs (control program + plant model) against goal specification.
  e.g., ((EngineA = Firing) OR (EngineB = Firing)) for OrbitInsert()
- Extract probabilistic information about program's possible executions.

Up Next…

- Conclusion!