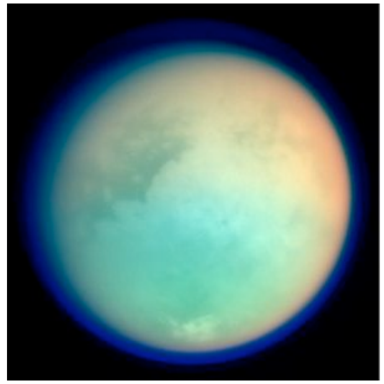
# State-Based Hybrid Control and Symbolic Model Checking

Julia M. B. Braman
NASA-Johnson Space Center
September 16, 2009

# Motivation

Aerobot, lower atmosphere probe of Titan, a moon of Saturn (Courtesy NASA/JPL-Caltech)



NASA-JSC's Lunar Electric Rover (LER) and Chariot Chassis


Alice, Caltech's entry to the DARPA Urban Challenge (http://team.caltech.edu)

**Control Plan**
- Reconfigurable
- Fault Tolerant
- Complex

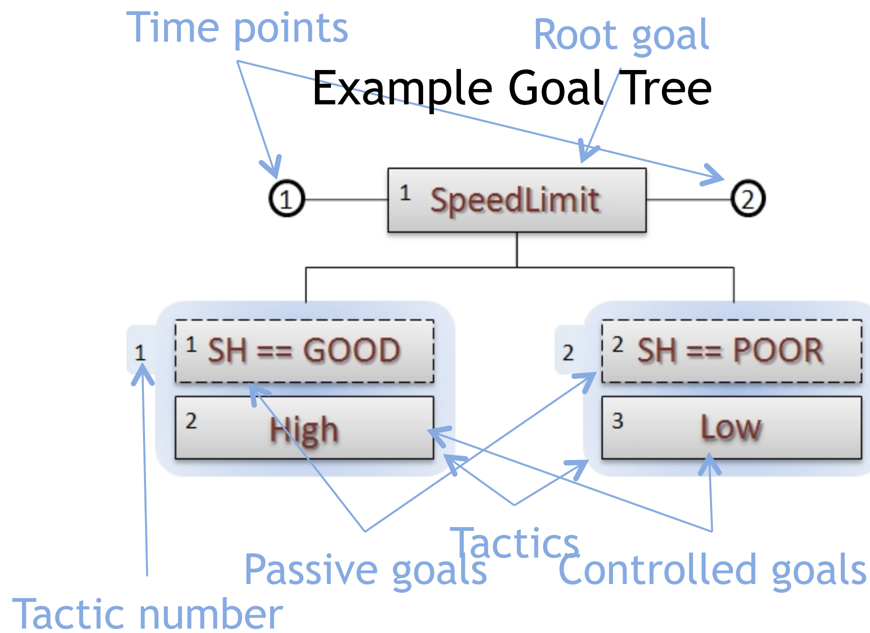
Testing vs.
Formal Methods

# Outline

- Introduce control architectures & symbolic model checking
  - *Goal networks = Hybrid systems*
- Design for Verification software, **SBT Checker**
  - *Titan Aerobot goal network example*
- State-based symbolic model checker, **InVeriant**
  - *Titan Aerobot goal network example*
- Hybrid Systems
  - *Lunar Electric Rover hybrid system example*

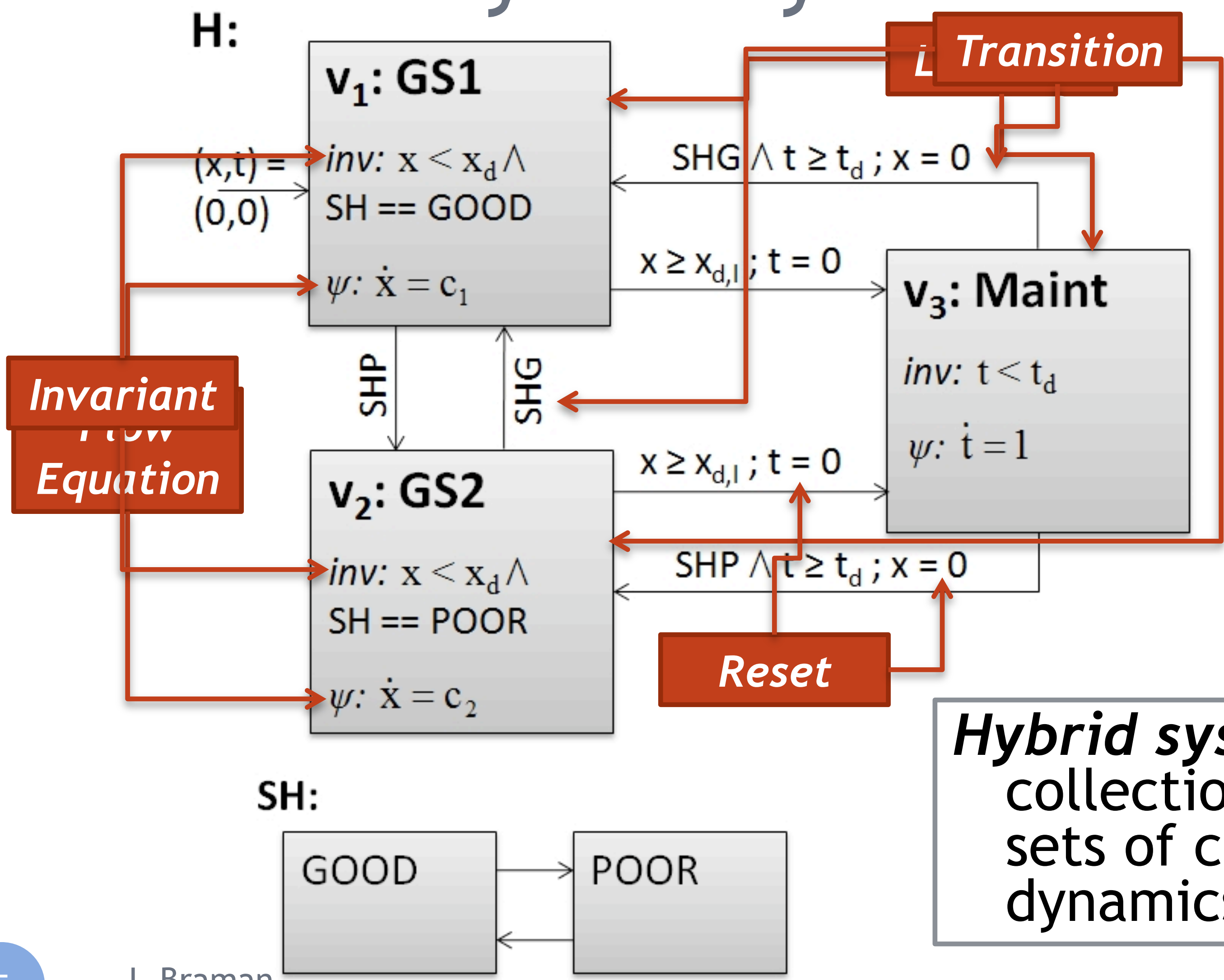
# Goal Network Control Programs

- Goal networks
  - Based on Mission Data System (MDS), a control architecture developed at the Jet Propulsion Laboratory
  - Collections of goal trees
- Goals
  - Constraint on state variable
  - Controlled (associated with commands) or passive




Each goal is denoted by
1. Index
2. Parent goal's index
3. Tactic number

# Linear Hybrid Systems




***Hybrid systems*** are collections of discrete sets of continuous dynamics.

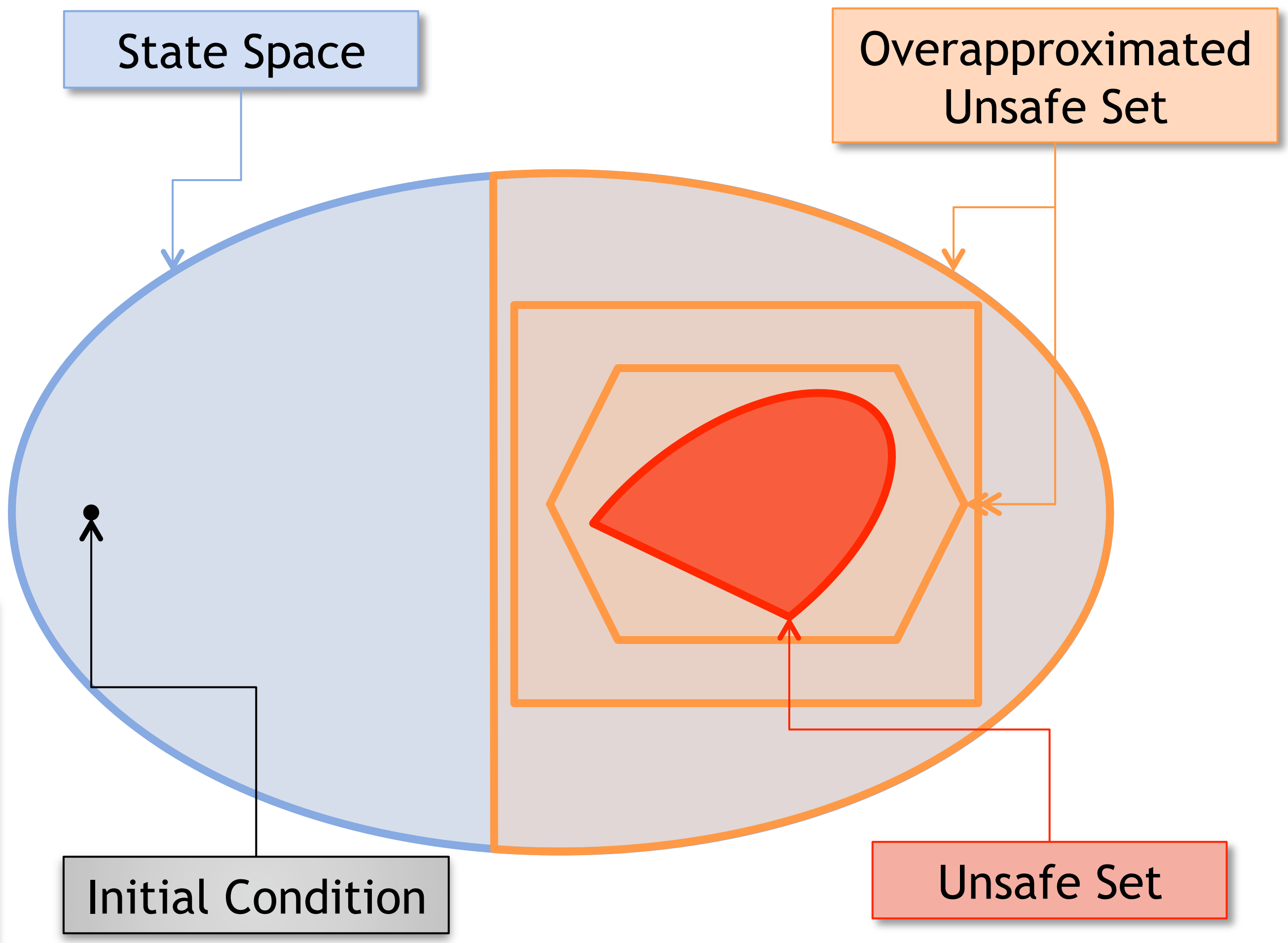
# Symbolic Model Checking

***Model Checking***
(Spin, NuSMV):

1. Searches over entire state space
2. No continuous states

***Symbolic Model Checking***
(PHAVer, HyTech):

1. Overapproximation
2. Significant complexity issues

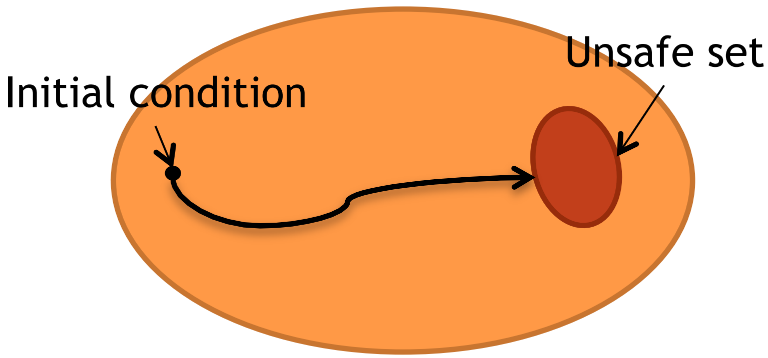


# Safety vs. Liveness

***Safety*** tests to see if a specified unsafe set is reachable w/o regards to specific paths

**This method deals with safety analysis only.**

***Liveness*** tests whether conditions are reachable with regards to how and how often




$\Diamond \Box s$
(stability)

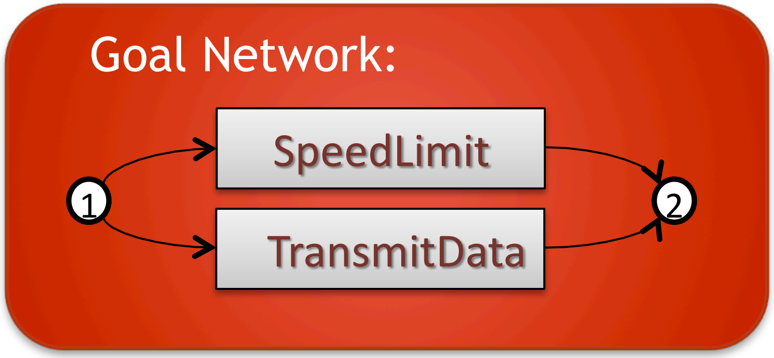
*Can we reach this state during execution?*

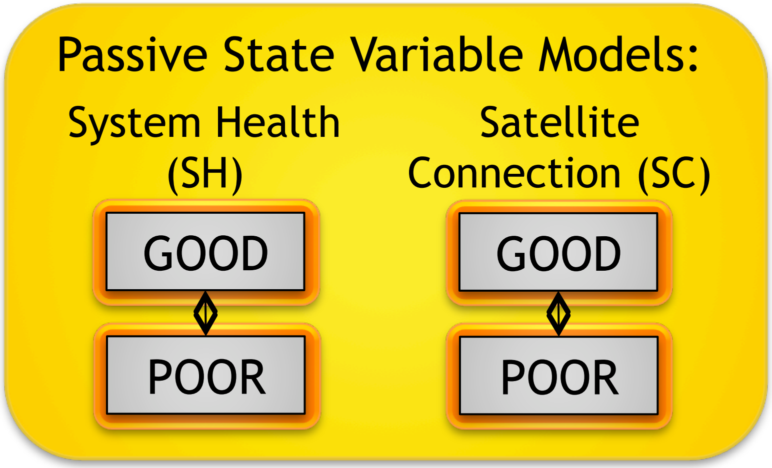
*Are we eventually always in this state?*

# Previous Work

- **Conversion for Verification**
  - AgentSpeak -> Promela, JPF2 (*Bordini et al., 2004*)
  - MPL -> Livingstone (*Simmons et al., 2000*)
- **Hybrid Systems Verification**
  - HyTech (*Henzinger et al., 1997*)
  - PHAVer (*Frehse, 2005*)
  - Uppaal (*Larsen et al., 1997*)
- **Design for Verification/Correct-by-Design**
  - D4V (*Mehlitz & Penix, 2003*)
  - Control from LTL (*Kress-Gazit et al., 2007*)

# Goal Network Execution



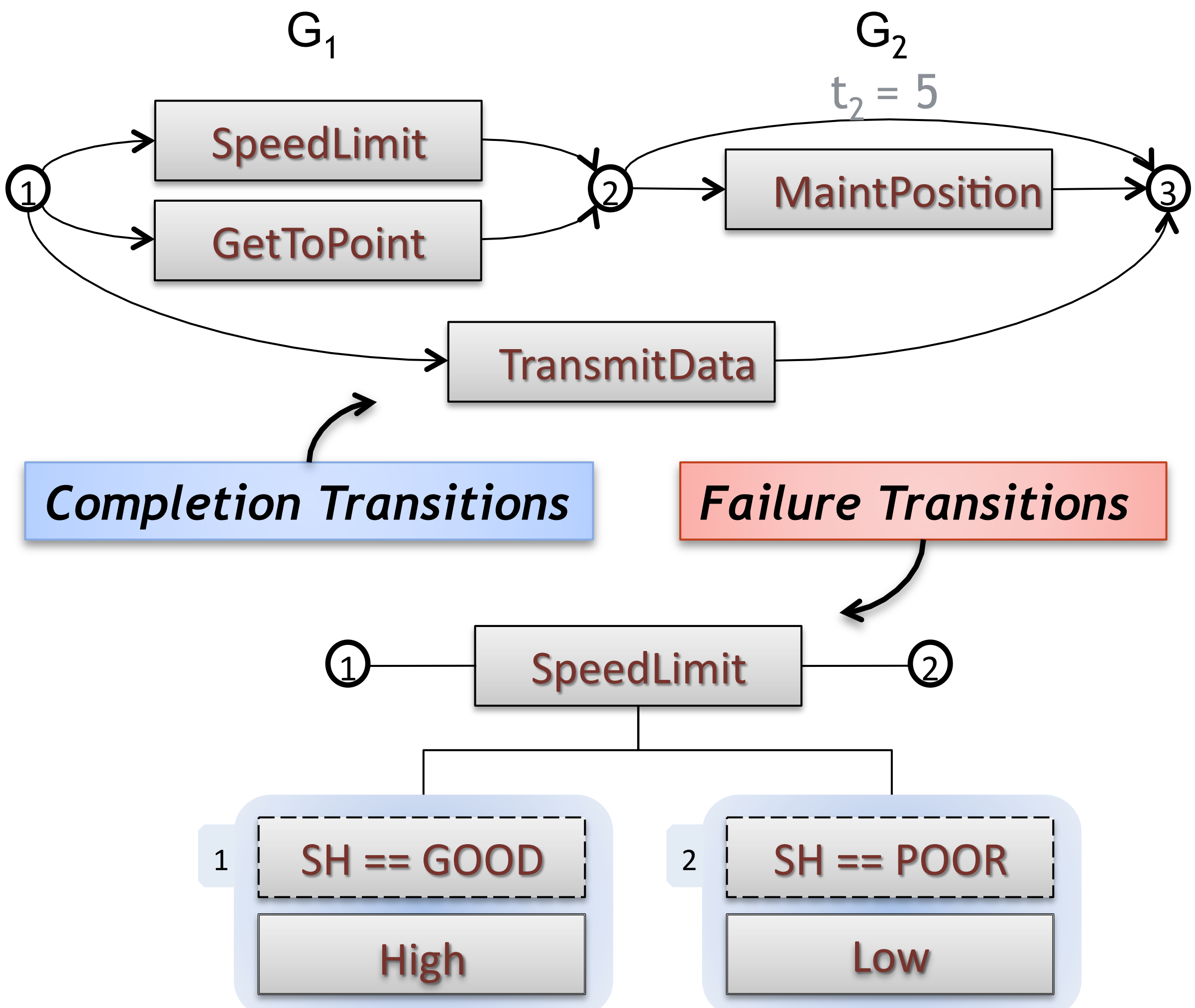




J. Braman

September 16, 2009

# Groups and Transitions




***Group***:
Set of goals active between consecutive time points

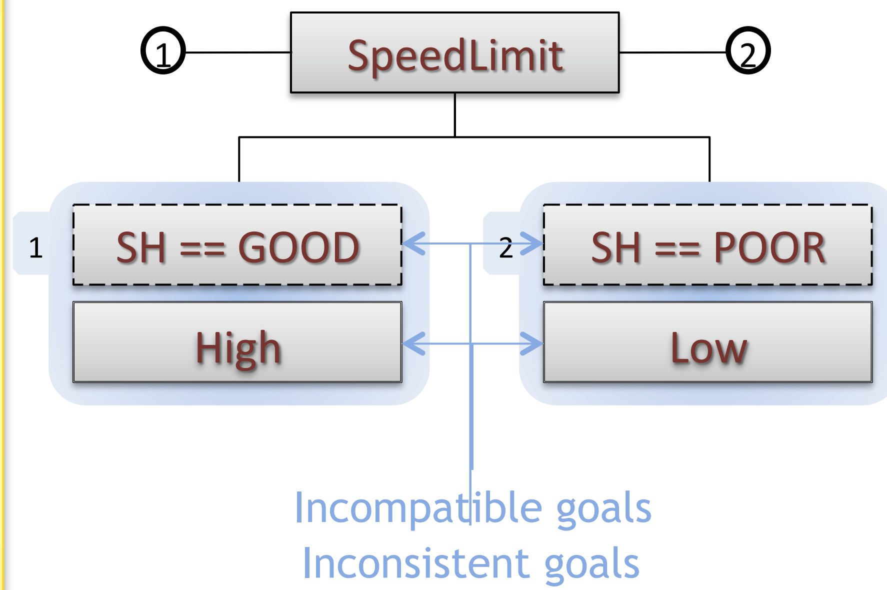
***Completion Goal***:
Controlled goal with transition constraint (GetToPoint)

***Time Constraint***:
Bounds on the amount of time spent executing a group

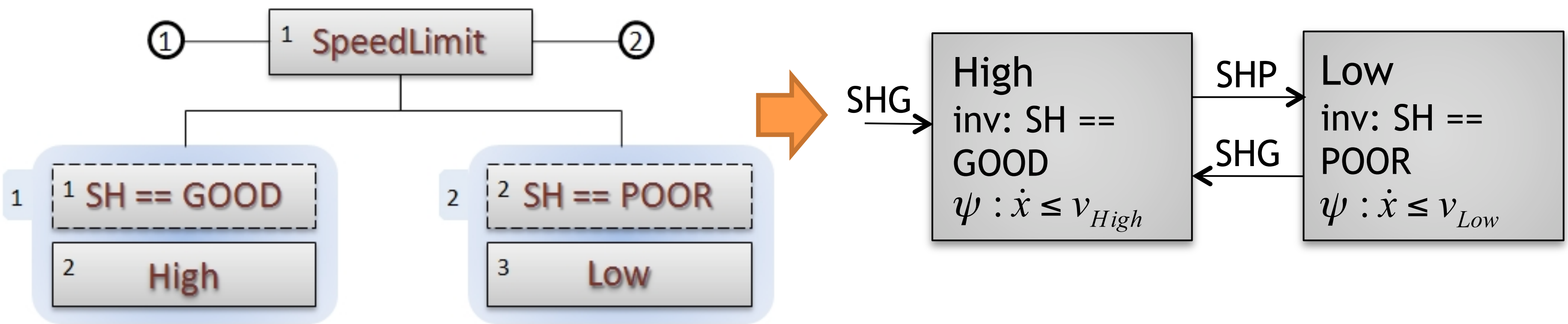
# Executable Sets of Goals

*Properties:*

1. All goals in same group
2. All root goals in group
3. Relational goal tree restrictions
4. Compatible
5. Consistent



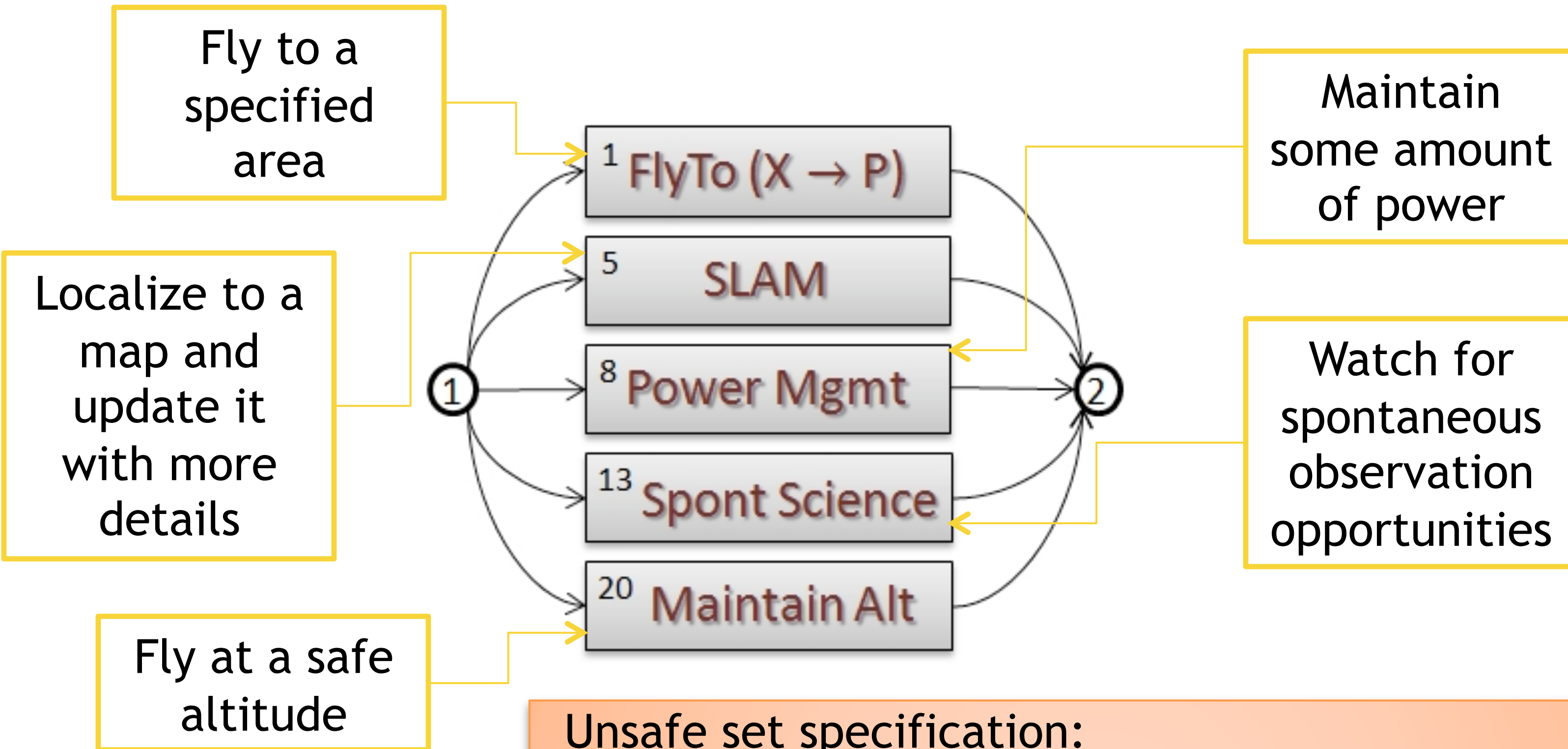
# Conversion Bisimulation




| Goal Networks | Hybrid Systems |
|---|---|
| Executable set | Location |
| Failure (upon goal re-elaboration)<br>Completion (upon goal achievement) | Transitions |
| Passive goal constraints | Invariants |
| Controlled goal constraints | Flow equations<br>Resets |

# Titan Aerobot Mission Example

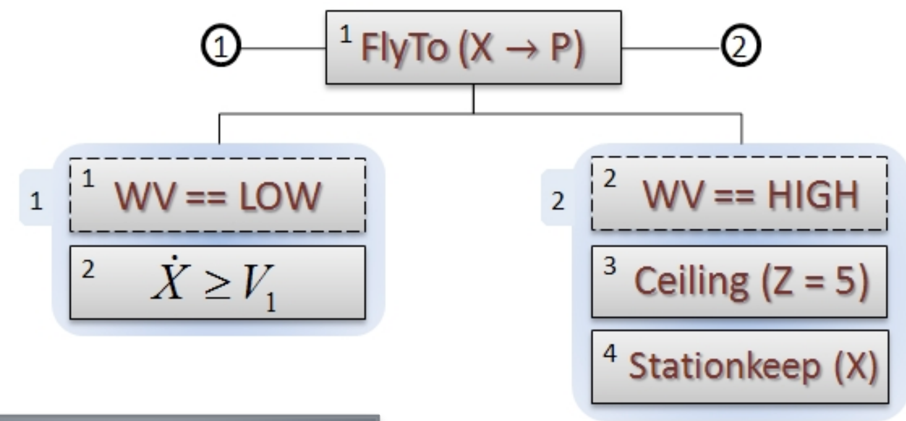
The Titan Aerobot must:




Unsafe set specification:

1. Power < 10%
2. Position uncertainty is high, ground visibility is low and the altitude is less than the maximum terrain clearance altitude.

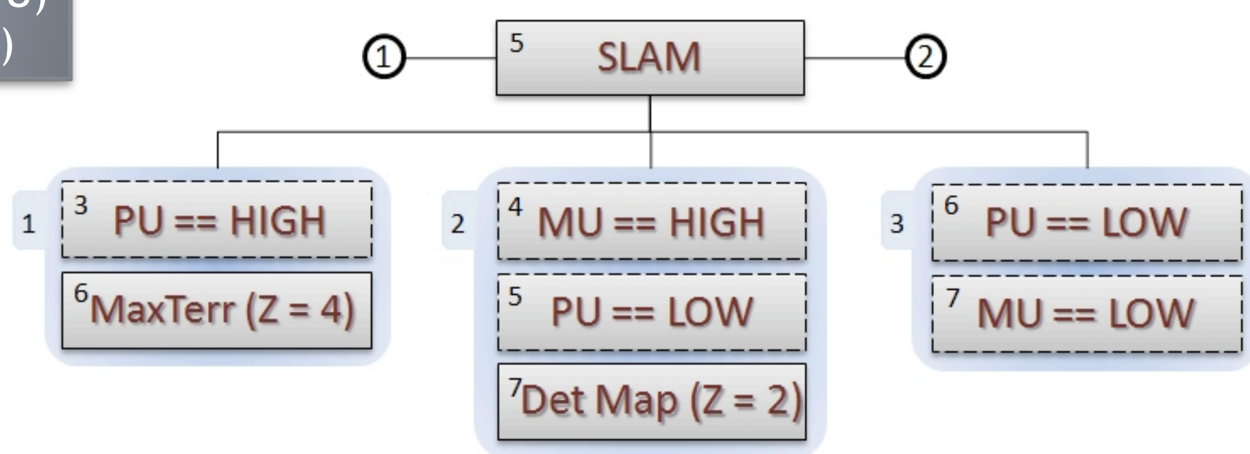
# Titan Aerobot Goal Trees



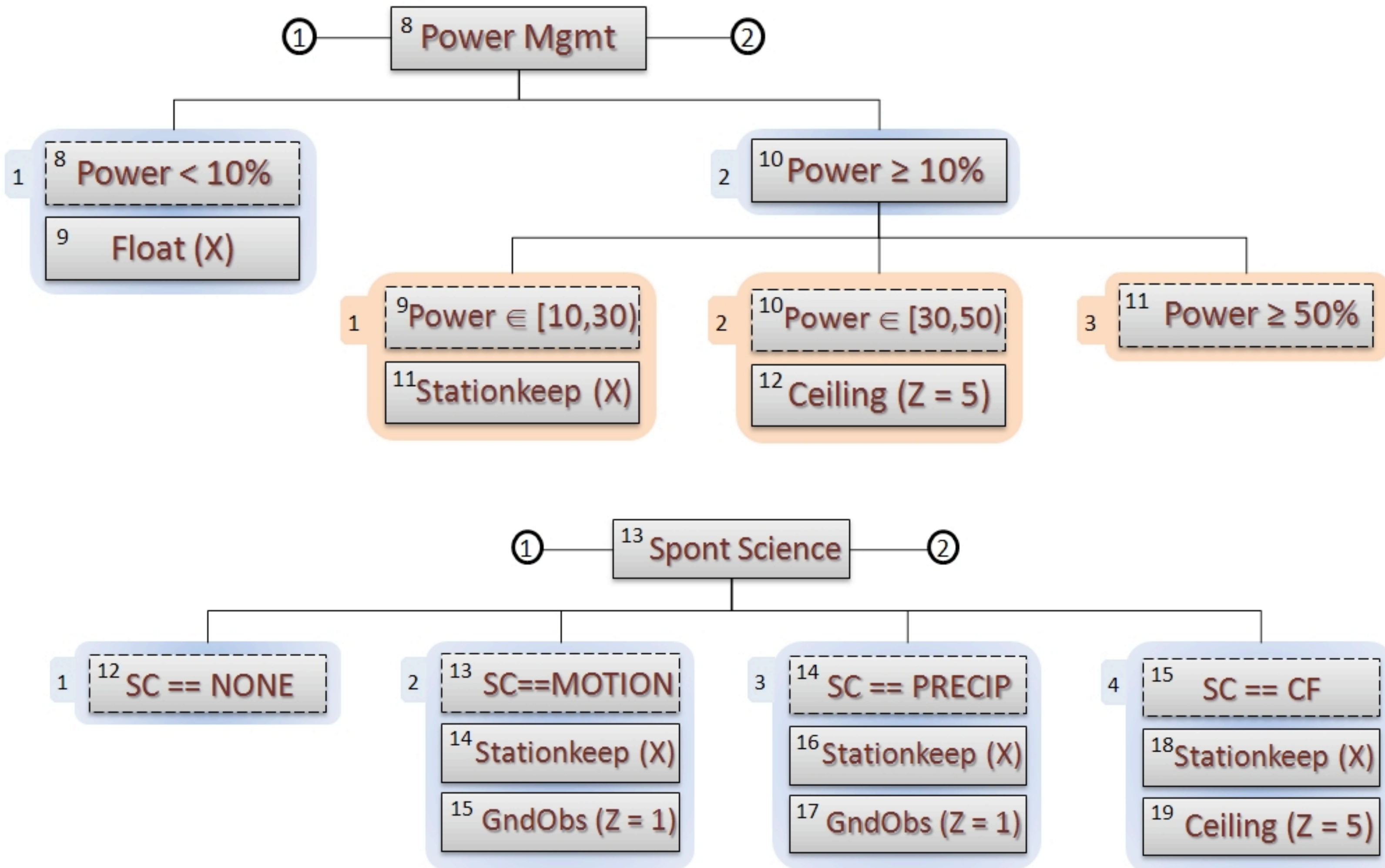

Altitude Levels (Low to High)

| | |
|---|---|
| 1 | Ground Observation |
| 2 | Detailed Map |
| 3 | Minimum En Route |
| 4 | Maximum Terrain Clearance |
| 5 | Service Ceiling |

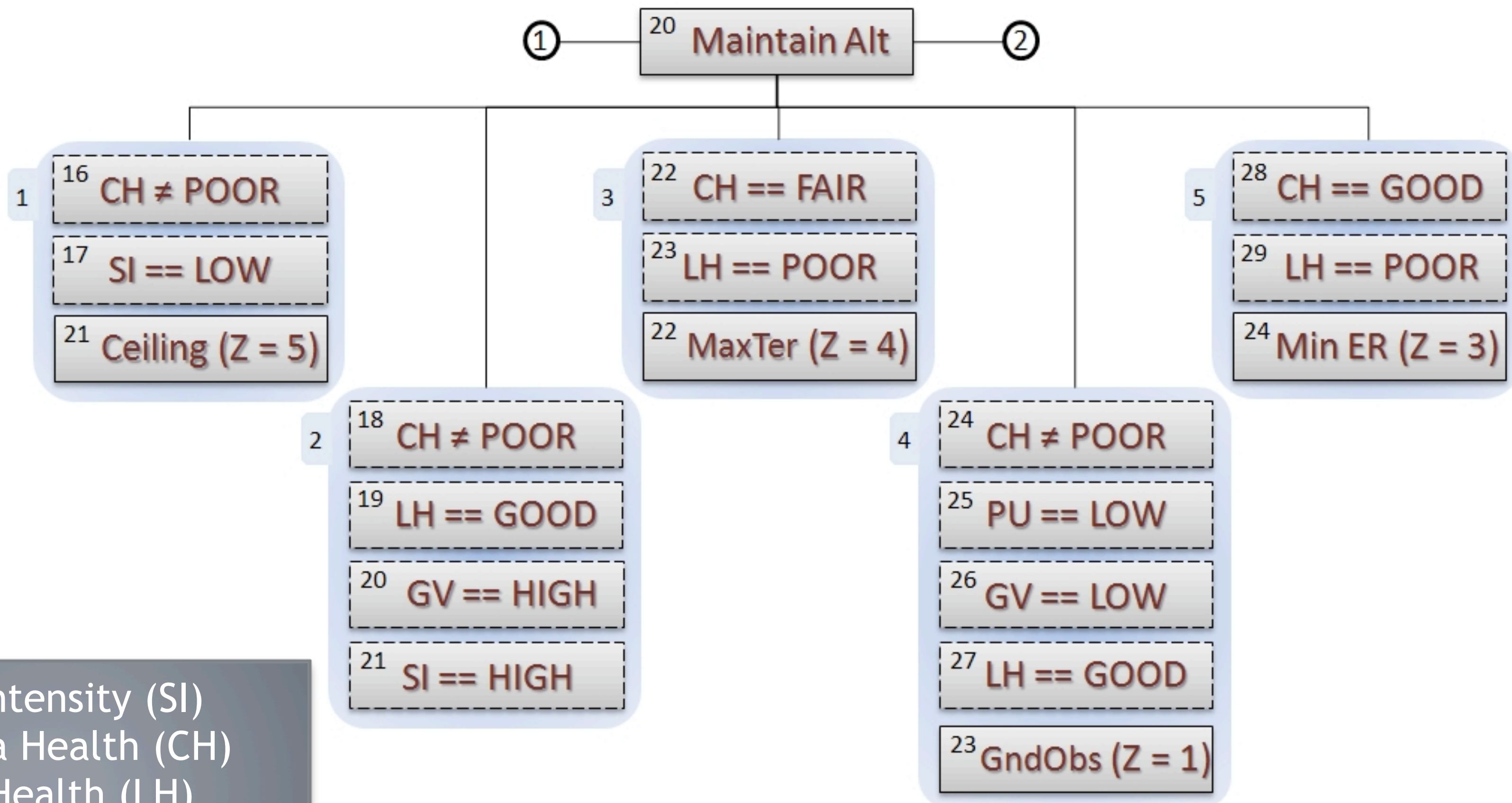
Wind Vector (WV)
Position Uncertainty (PU)
Map Uncertainty (MU)



# Titan Aerobot Goal Trees



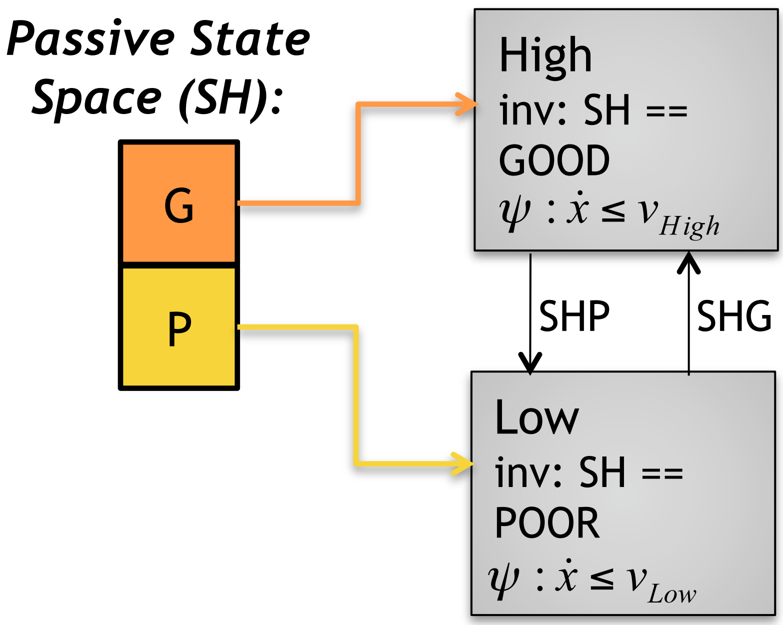
# Titan Aerobot Goal Trees




Sun Intensity (SI)
Camera Health (CH)
LRF Health (LH)
Ground Visibility (GV)
Position Uncertainty (PU)

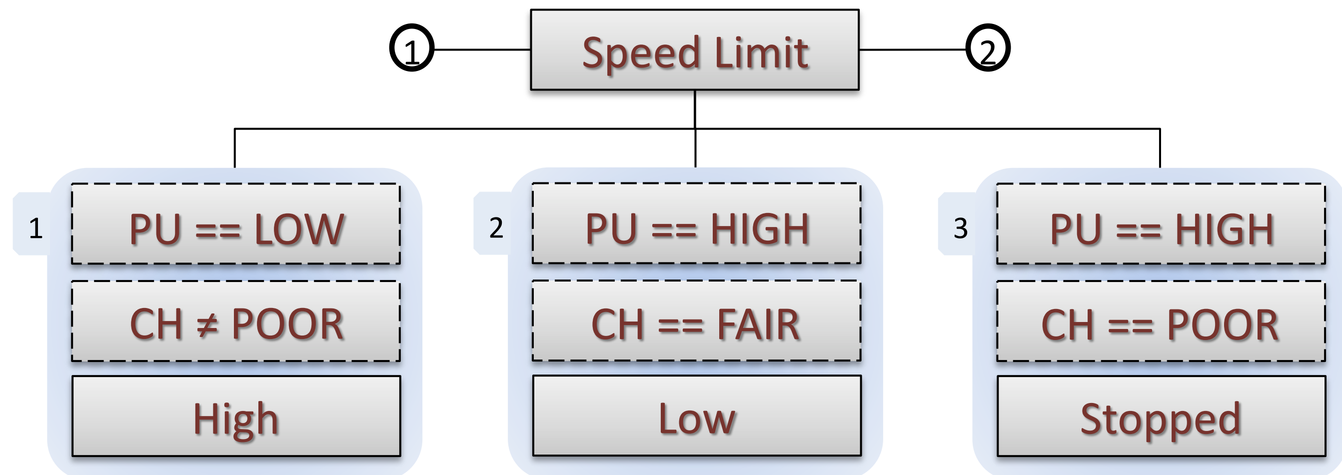
# State-Based Transitions

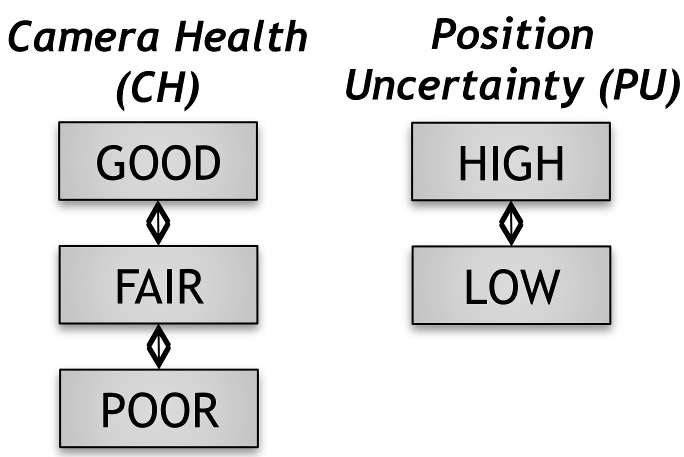
**Definition:** A goal network (hybrid system) has ***state-based transitions*** if each state in the passive state space satisfies the passive constraints (invariant) of some executable set (location).




***SBT Checker:***
Design for verification tool that checks for unconstrained passive states

# State-Based Transitions Example



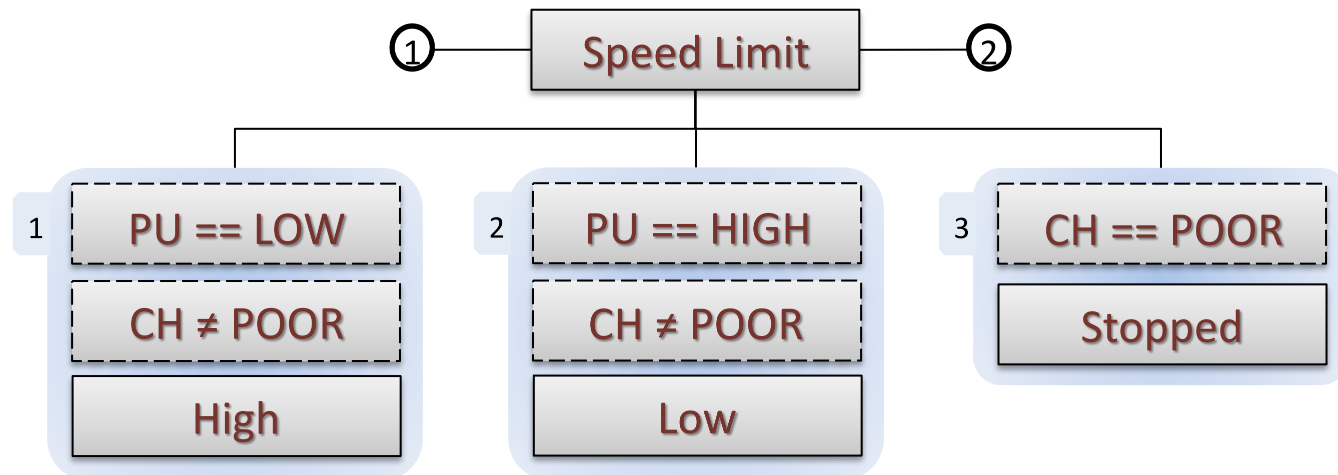

| **PU** | | POOR | FAIR | GOOD |
|---|---|---|---|---|
| | LOW | | 1 | 1 |
| | HIGH | 3 | 2 | |

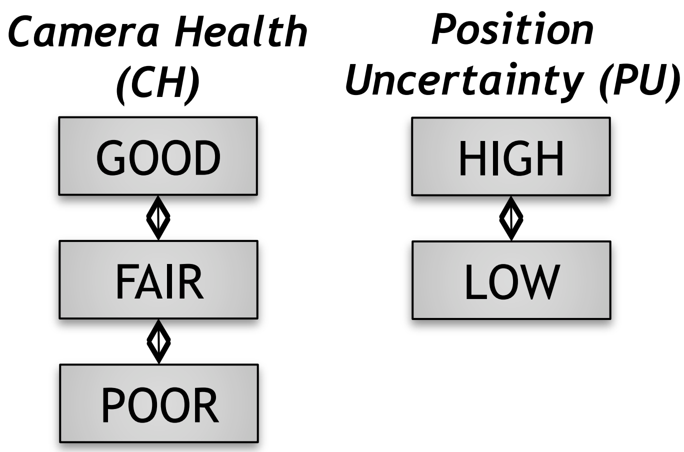
**CH**

***SBT Checker Output:***
(CH == GOOD [illegible] PU == HIGH) [illegible]
(CH == POOR [illegible] PU == LOW)


J. Braman

# State-Based Transitions Example




| PU \ CH | POOR | FAIR | GOOD |
|---|---|---|---|
| LOW | 3 | 1 | 1 |
| HIGH | 3 | 2 | 2 |

***SBT Checker Output:***
False




J. Braman

# SBT Checker

**Theorem:** If

1. Each goal tree in a goal network has state-based transitions
2. All controlled constraints are consistent

The goal network has state-based transitions.

***SBT Checker:***

Design for verification software tool

- Leverages modularity of state-based transition requirement
- Checks individual goal trees/hybrid automata for state-based transitions
- Returns missing passive constraints



J. Braman

# SBT Checker Usage

*Command Line:*

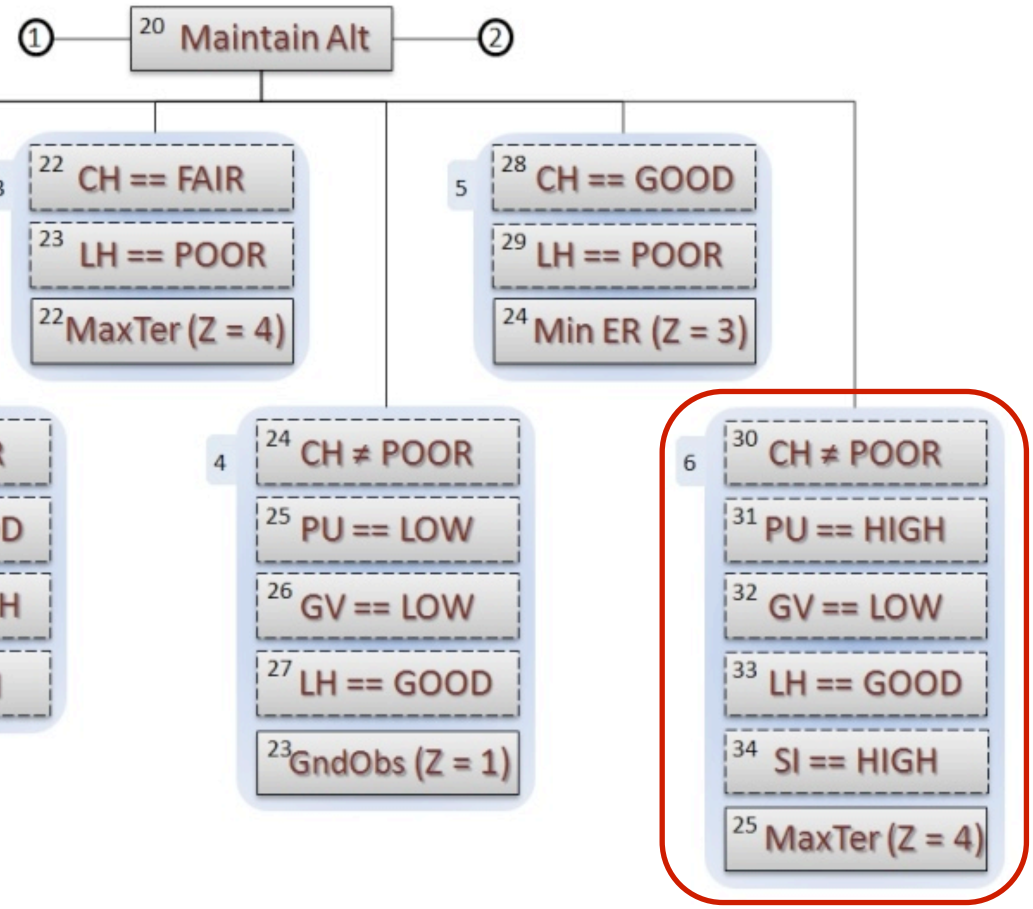
1. Create XML file, including
   a) Models of passive state variables used
   b) Invariants of all executable sets of the goal tree
2. Load "SBTChecker" package into Mathematica using following command:
   >> Needs["SBTChecker`"];
3. Store path of XML file in 'pathfile' variable
4. Run the SBT Checker using the following command:
   >> SBTCheck[pathfile,CSType->GN]

*GUI:*

1. Load "SBTChecker" package into Mathematica using following command:
   >> Needs["SBTChecker`"];
2. Launch SBT Checker using the following command:
   >> SBTCheckGUI
3. Input models of passive state variables and invariants of executable sets in goal tree
4. Press "Run Check" button


J. Braman

# Titan Aerobot Mission Verification



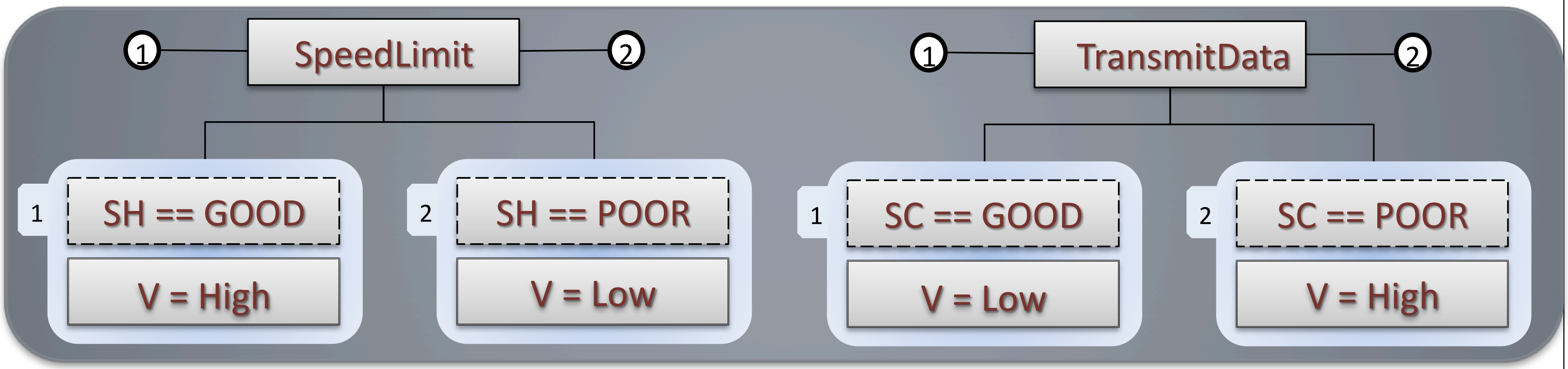

SBT Checker found missing tactic!



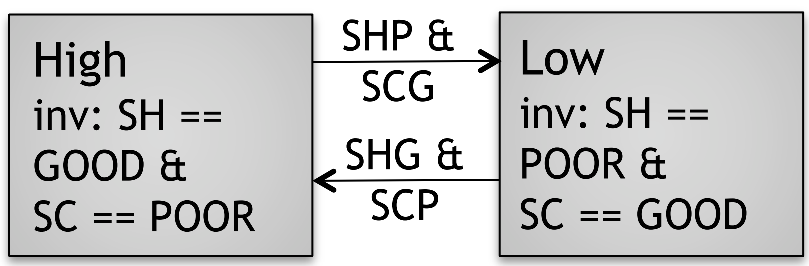
J. Braman

September 16, 2009

# Consistent Controlled Constraints




**Rule:** Can only merge goals on V if $c_1 == c_2$

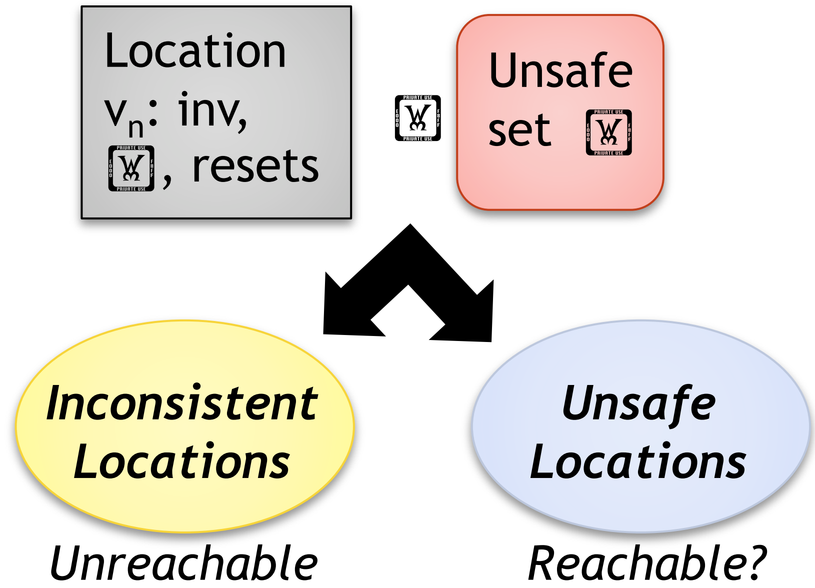



Notice that two states are missing:

1. SHG & SCP
2. SHP & SCG

**Point:** Transitions are not state-based!

# InVeriant




**Method:**

1. Find locations, invariants, dynamical constraints, and resets.
2. Compare unsafe set constraints with each location.
3. Find path for rate-driven, continuous dependent state variables constrained.

**Theorem:**

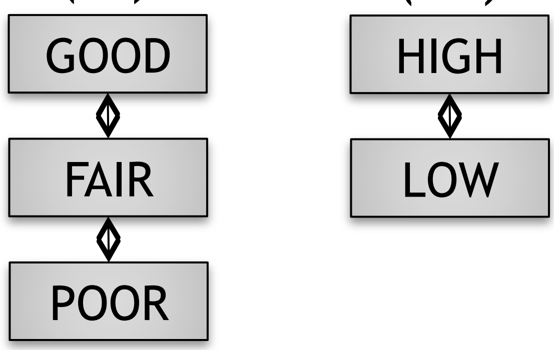
If the hybrid system has state-based transitions, the reachability of locations depends only on the reachability of the states of the passive state variables constrained.

# Reachability of Unsafe Locations

***Discrete Passive State Variables***

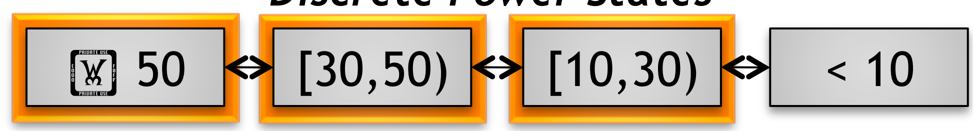
*Camera Health (CH)*

*Wind Vector (WV)*

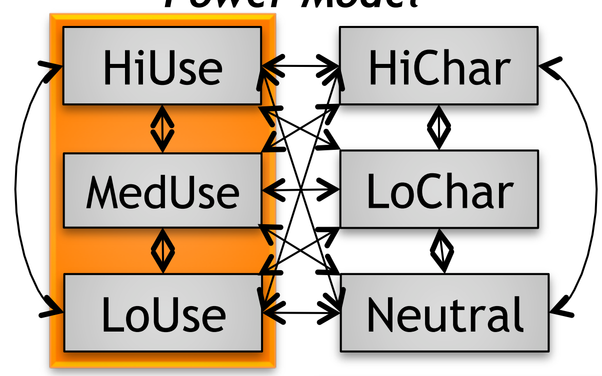



***Continuous, Rate-Driven Passive State Variables***

*Discrete Power States*




*Power Model*




Is Power < 10 reachable?

Find path from Power = 100 (initial condition) to Power < 10.

Transitions depend on:
Position
Altitude
Wind Vector
Sun Intensity

# InVeriant Usage

1. Create XML file that includes the following information
   a) Passive state variable models
   b) Merging rules and constraint types for controlled state variables
   c) Goals in each goal tree
   d) Unsafe set
2. Load InVeriant package using the following command:

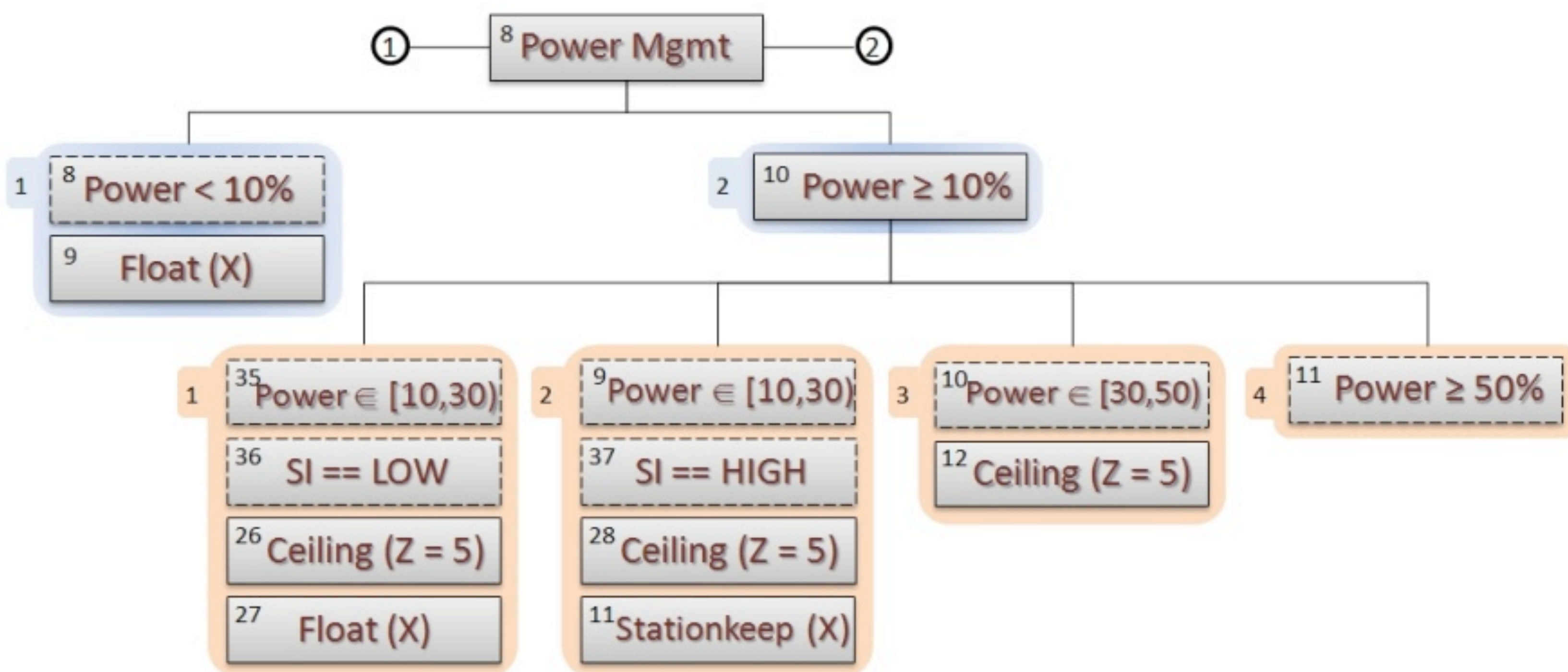
   >> Needs[“InVeriantSMC`”];
3. Store path of XML file in ‘pathfile’ variable
4. Model check the goal network using the following command:

   >> InVeriant[pathfile,CSType->GN]

# Titan Aerobot Mission Verification

InVeriant found

- Locations that satisfy unsafe power condition
- Path from initial condition (Power = 100) to unsafe condition (Power < 10)



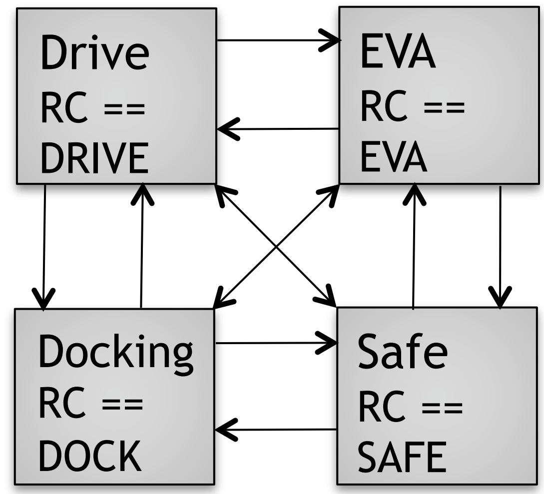
# Benefits of SBT Checker/InVeriant

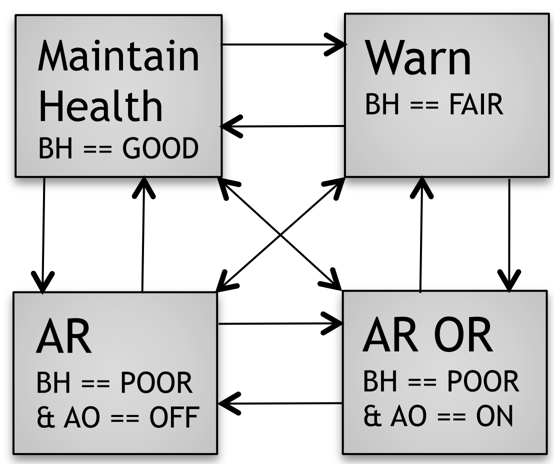
- Titan system stats:
  - About 600 locations
  - Over 300,000 transitions
  - 11 state variables, 9 passive, 2 controlled
    - Discrete passive state space (no power) is nearly 600,000 states
- Conversion/PHAVer method
  - 5 hours to convert (thousands of transitions)
  - PHAVer did not complete
- SBT Checker/InVeriant
  - 15 minutes to convert
  - 2 minutes to verify

- InVeriant is fast and efficient
  - No transitions
  - Passive state space does not contribute to complexity
- SBT Checker is scalable because of modularity
- State-based transitions imposes structure on control system
  - Good design practice
- Can reason about rate constraints
- Can apply to hybrid systems w/o group structure

# Hybrid System Theory

- Classes of hybrid systems can be verified with this approach
  - Automata must have either
    - State-based transitions
    - Completion transitions
  - Automata must run concurrently
  - Automata may be timed (completion transitions)



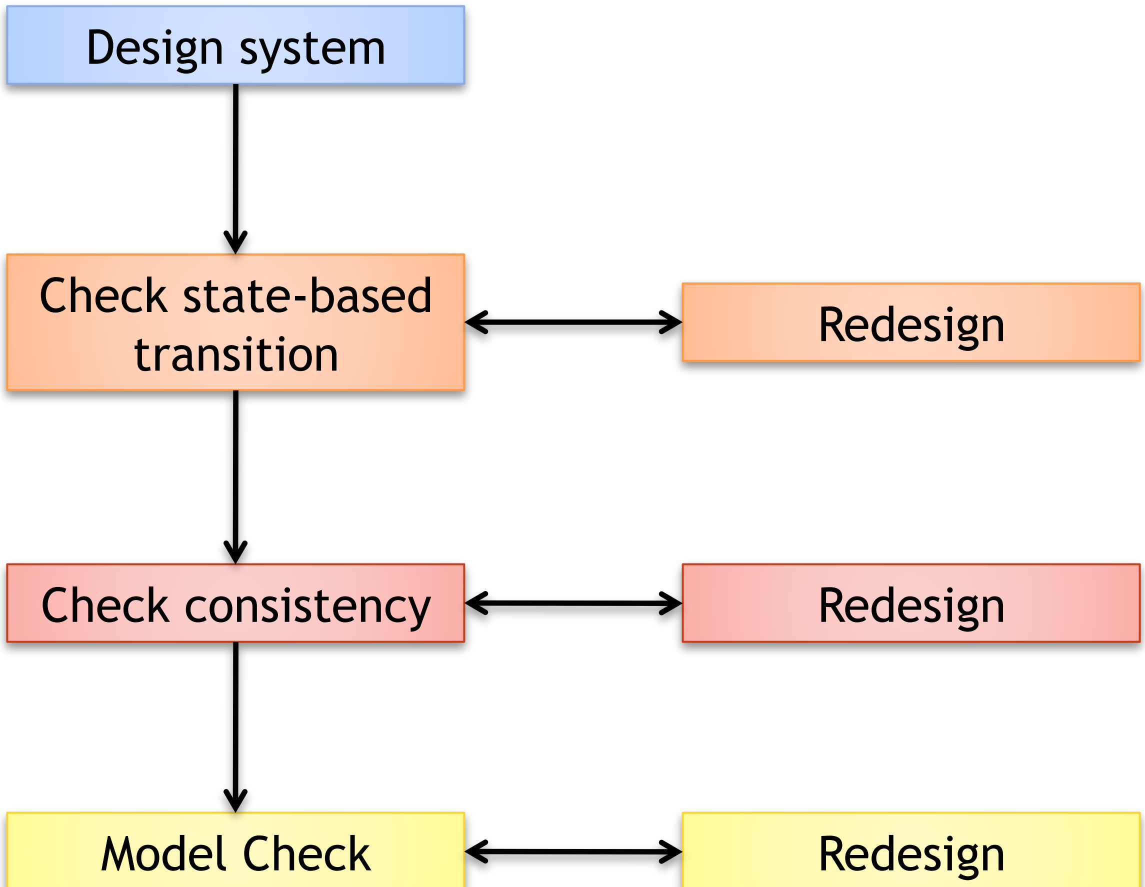


J. Braman

# Lunar Electric Rover

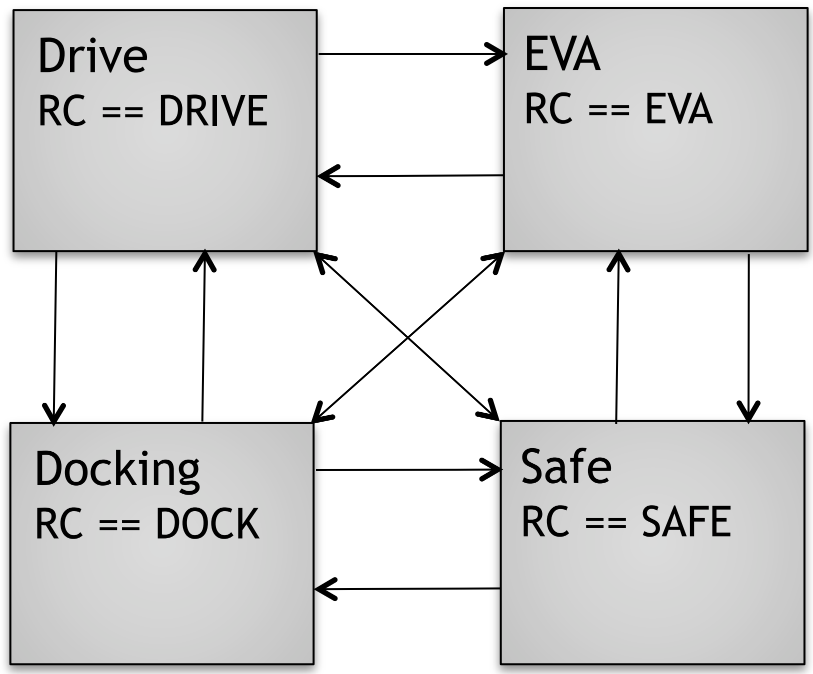
[Chariot B Movie Clip]

# Design Flow

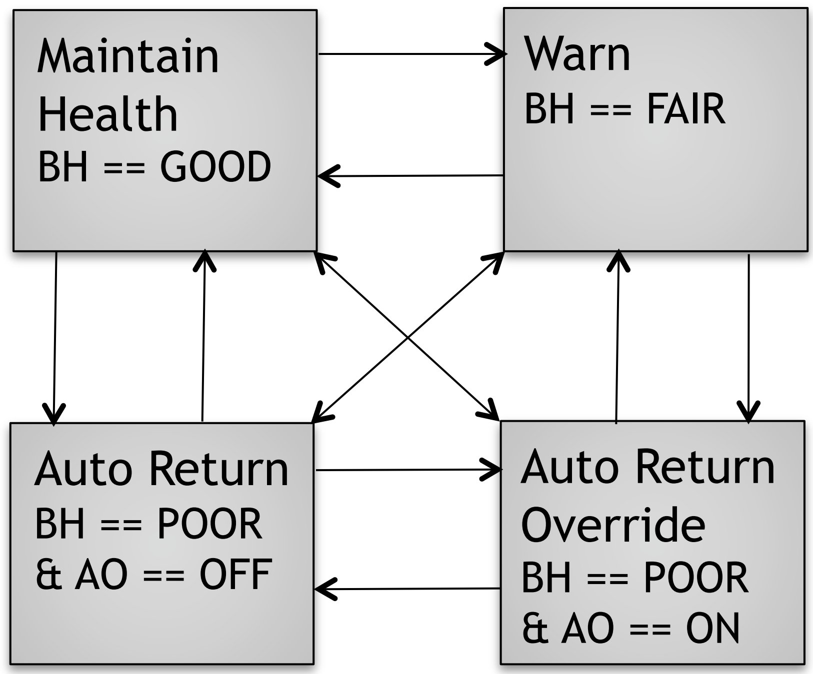


Design system
Check state-based transition
Redesign
Check consistency
Redesign
Model Check
Redesign

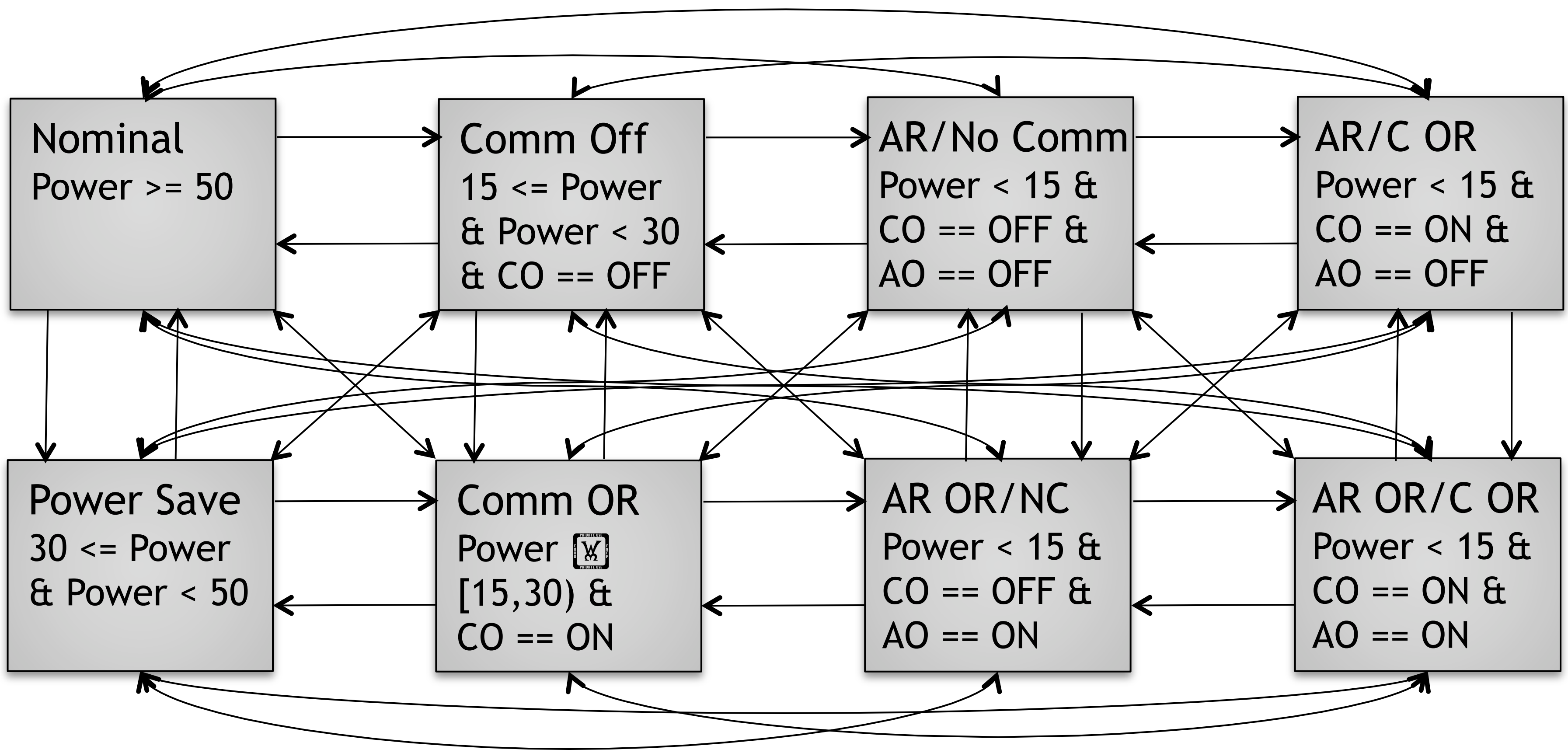
# LER Example Problem



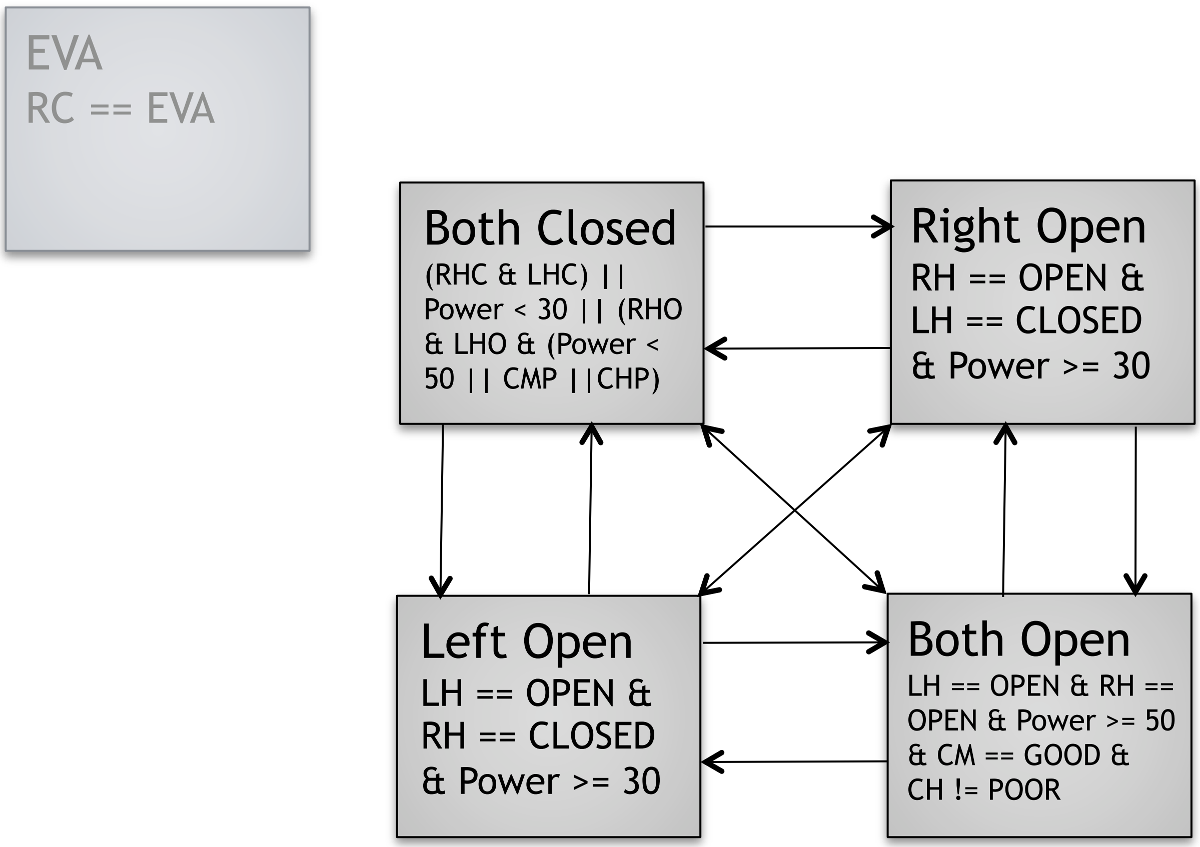
# LER Example Problem – ECLS



# LER Example Problem - Power



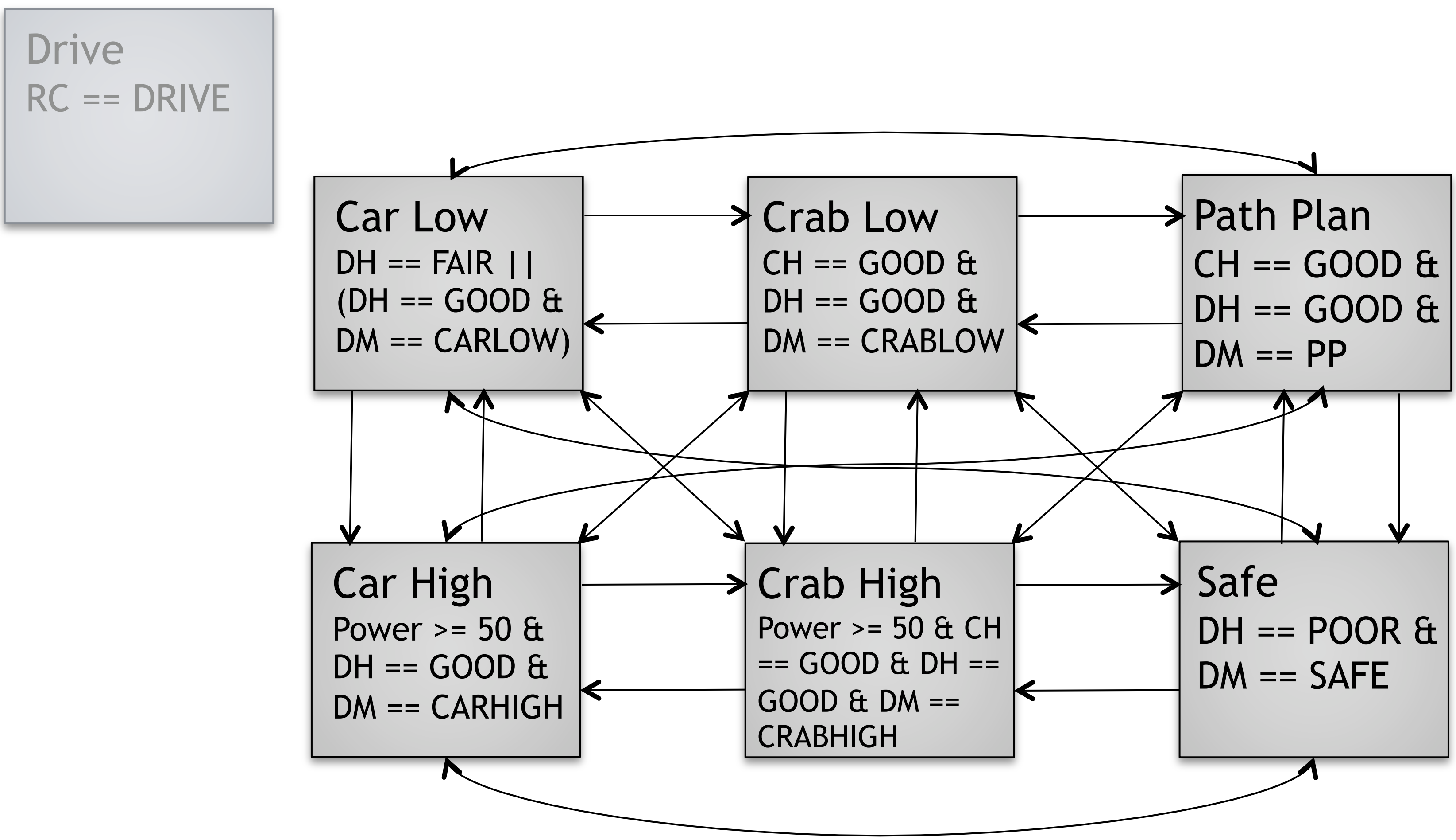
# LER Example Problem - EVA



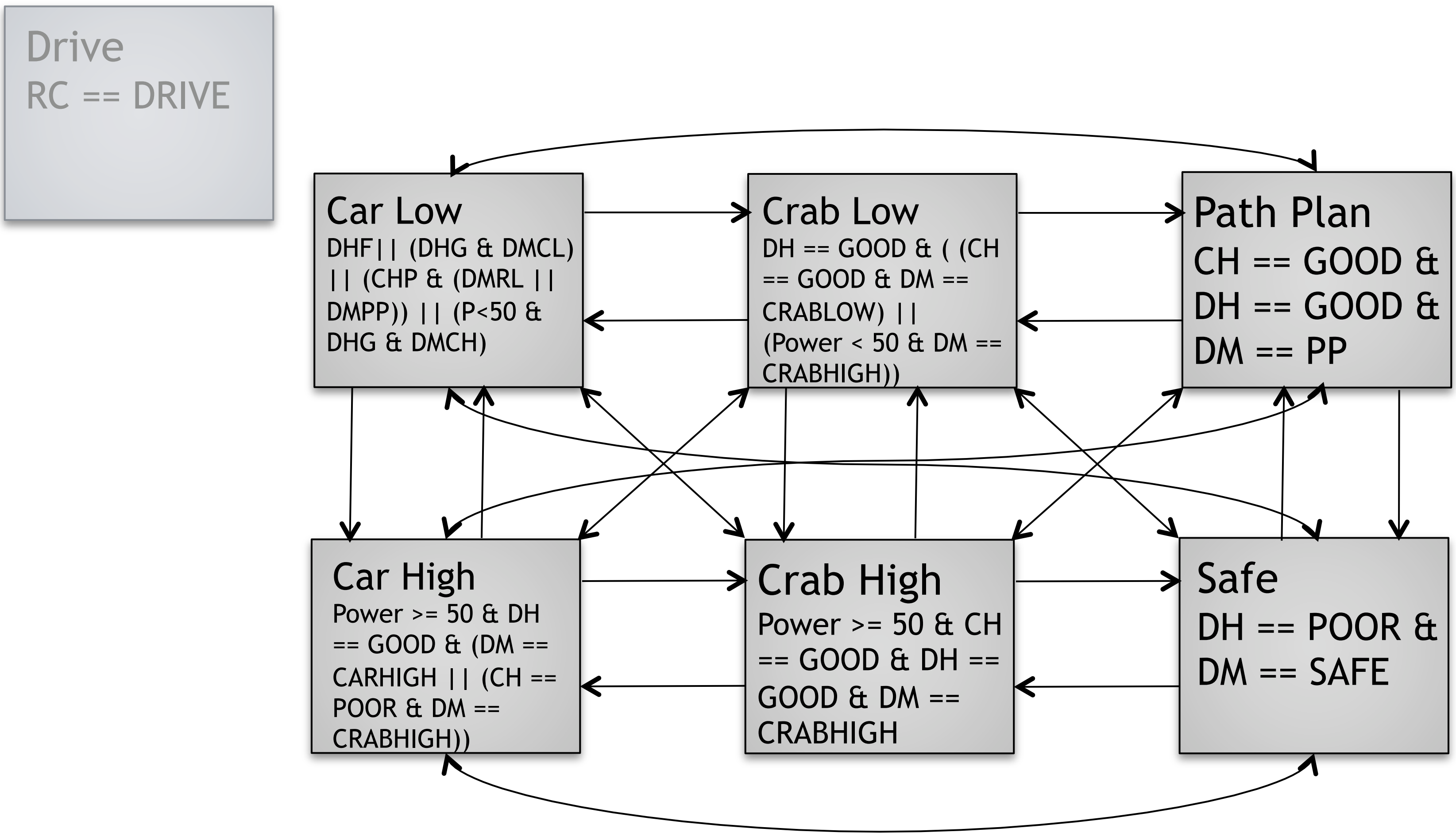



J. Braman

# LER Example Problem - Drive



# LER Example – Drive Redesigned



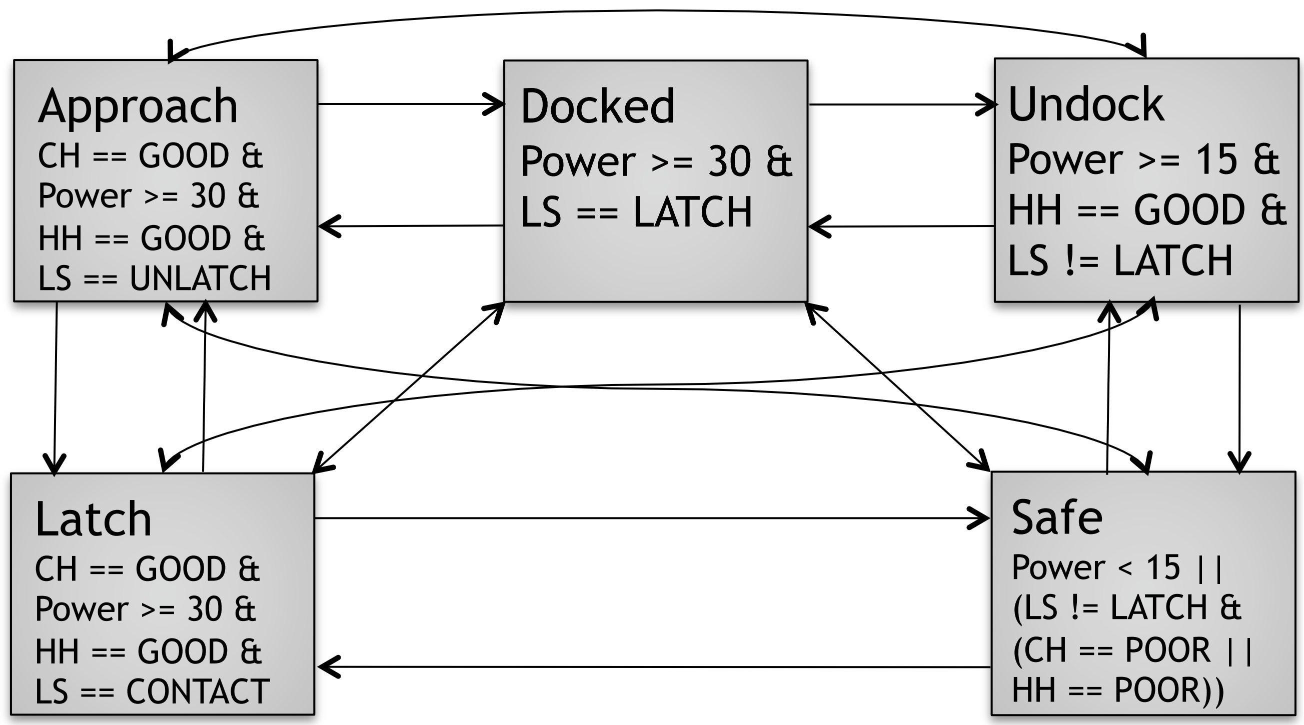



J. Braman

September 16, 2009

# LER Example Problem - Docking

Docking
RC == DOCK

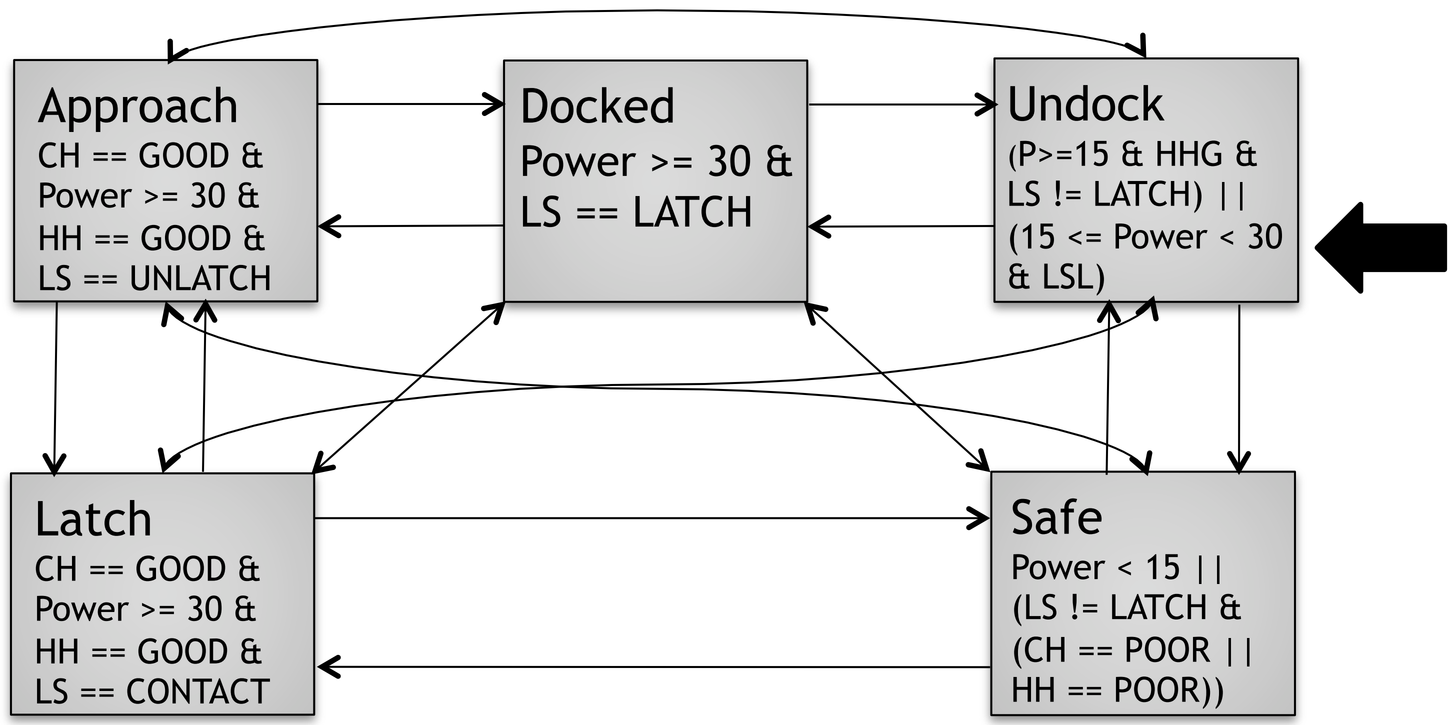




J. Braman

# LER Example – Docking Redesigned

Docking
RC == DOCK



# Inconsistent Controlled Constraints

InVeriant finds locations that have inconsistent controlled constraints

Two culprits:

1. "warn" state variable – changed so that merge between no alarm and alarm would result in an alarm
2. "os" state variable – changed so that auto-return would win the merge

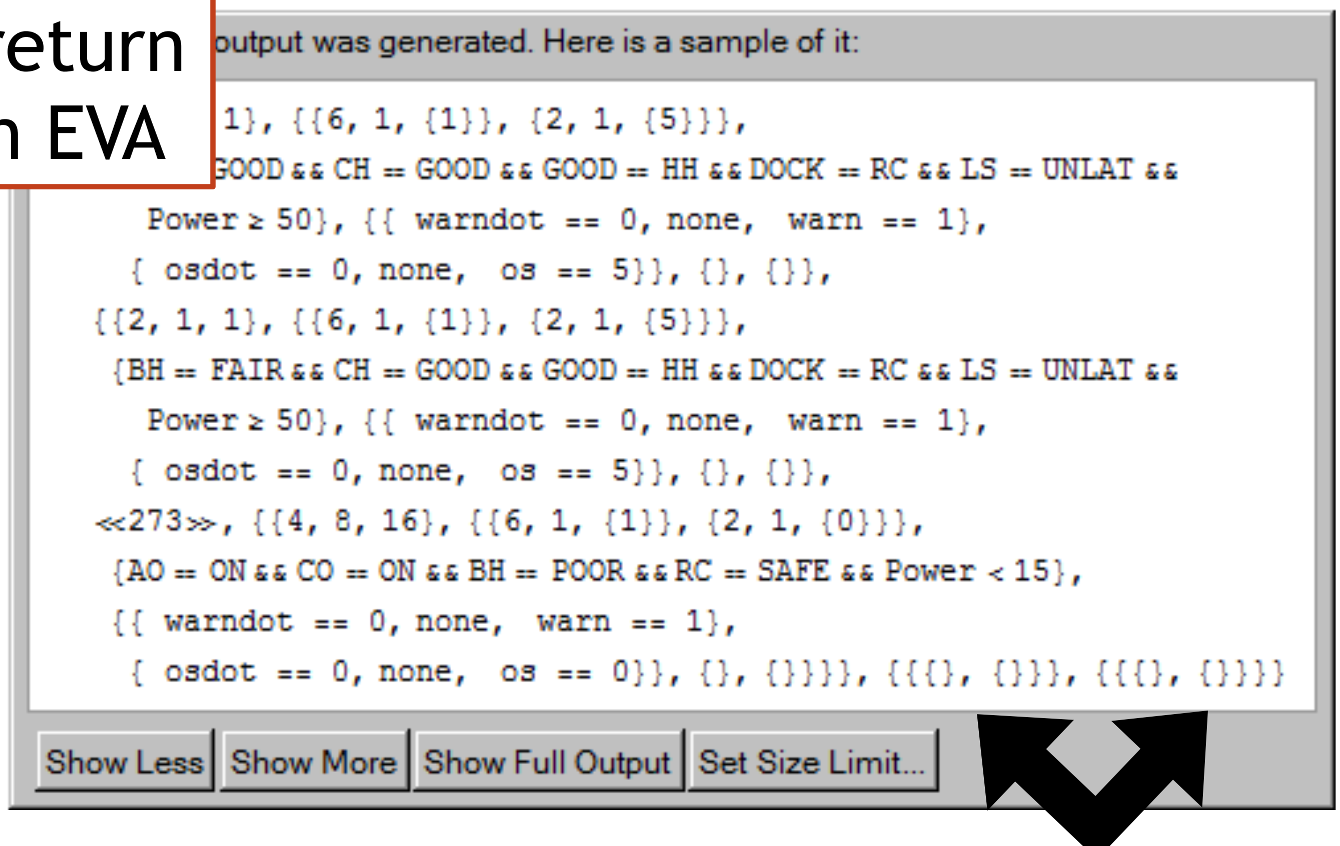
```
In[1]:= Needs["InVeriantSMC`"]

In[2]:= pathfile = NotebookDirectory[] <> "LER_LHA_incon.xml";
        InVeriant[pathfile, CSType → LHA]

        Part::partw : Part 1 of {} does not exist. >>
        Part::partd : Part specification f[1] is longer than depth of object. >>
        Part::partd : Part specification f[1] is longer than depth of object. >>
        Part::partd : Part specification f[1] is longer than depth of object. >>
        General::stop : Further output of Part::partd will be suppressed during this calculation. >>
        InVeriantLHA::rwn : There exist inconsistent controlled constraints,
            {{1, 1, 1}, {3, 1, 1}, {1, 2, 1}, {3, 2, 1}, {1, 3, 1}, {3, 3, 1}, {1, 4, 1}, {3, 4, 1}, {1, 5, 1}, {2, 5, 1}, <<310
```

# Verification of LER Example

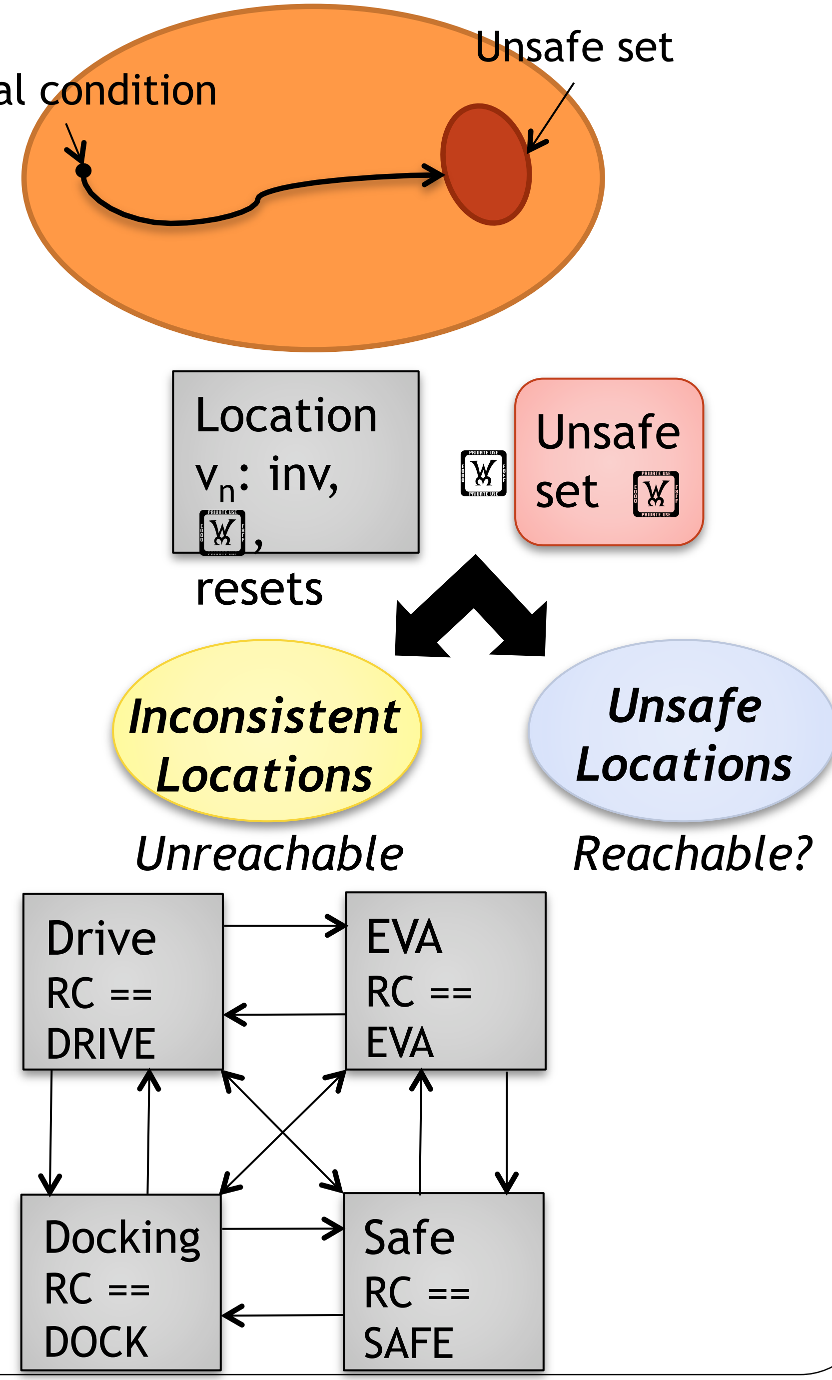
*Unsafe set*:
No auto-return during an EVA




**Safe!**

# Conclusions

- Formal Methods useful for analyzing control of fault tolerant systems
- Design for verification is essential to reduce state space concerns (**SBT Checker**)
- **InVeriant** model checker leverages properties of state-based design to formally verify a class of hybrid systems



# Useful Information

- SBT Checker:

  http://www.cds.caltech.edu/~braman/software/SBTChecker.zip
- InVeriant

  http://www.cds.caltech.edu/~braman/software/InVeriantSMC.zip
- Documentation

  http://www.cds.caltech.edu/~braman
- LER Example

  http://www.cds.caltech.edu/~braman/software/LERExample.zip