

Formal Checklists  
for  
Remote Agent Dependability

Grit Denker & Carolyn Talcott  
SRI International

WRLA 2004

# Problem

Software for deep space missions needs to be

- Autonomous -- must operate remotely over extended periods
- Robust -- must operate under conditions that can not be predicted
- Dependable -- mission failure is costly

# Mission Data System Framework (MDS)

Developed to support design and deployment of software for space missions

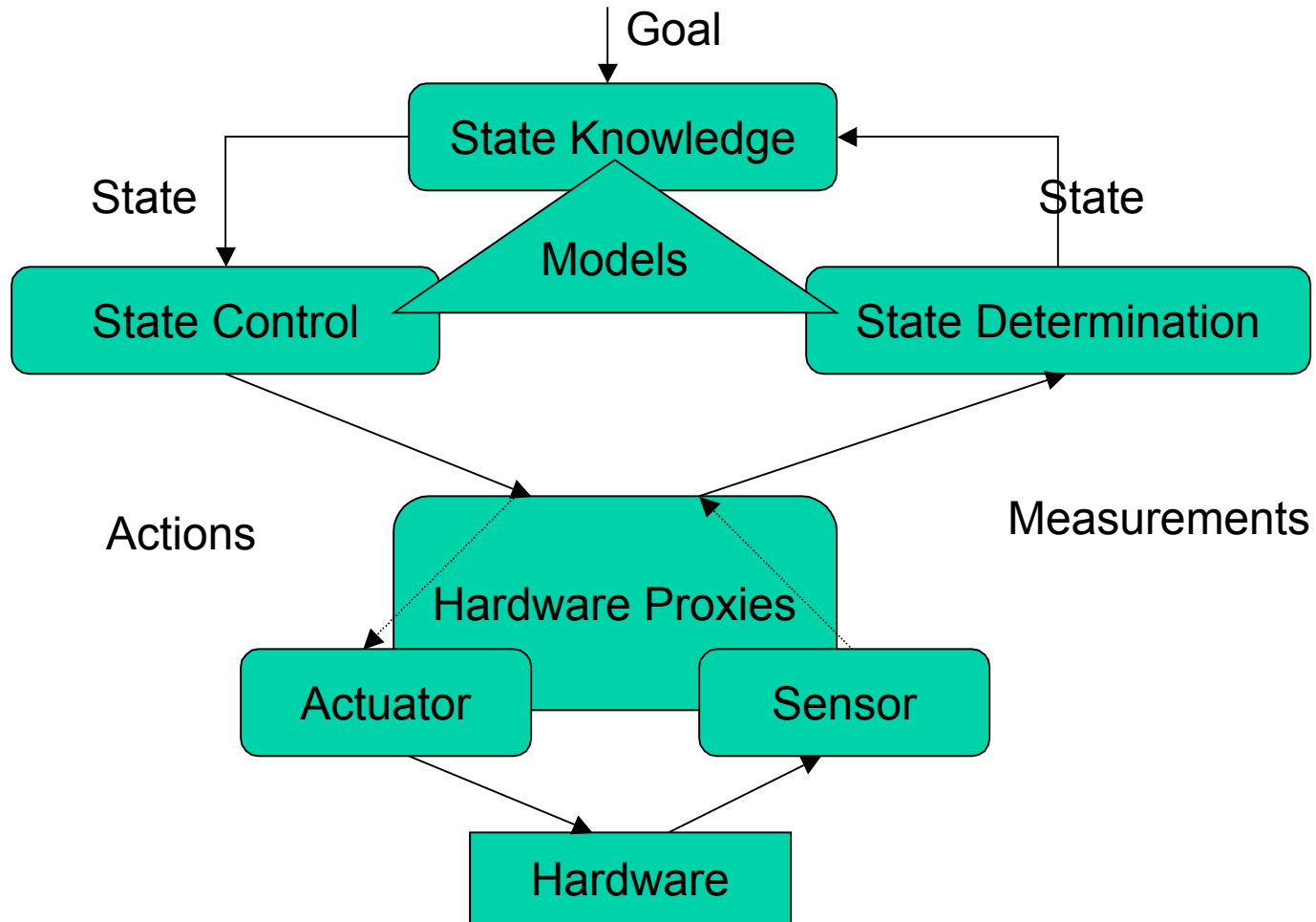
- Reusability of components
- State and model based design
- Goal-oriented operation

# MDS: State-based design

All domain knowledge is explicit and shared

- Globally shared state variables
  - hold all system state
- Domain models specify
  - Range of values for state variables
  - Operations for controlling values of state variables
- Each state variable has associated
  - A controller
  - An estimator

# MDS Architecture



## MDS: Goal-oriented operation

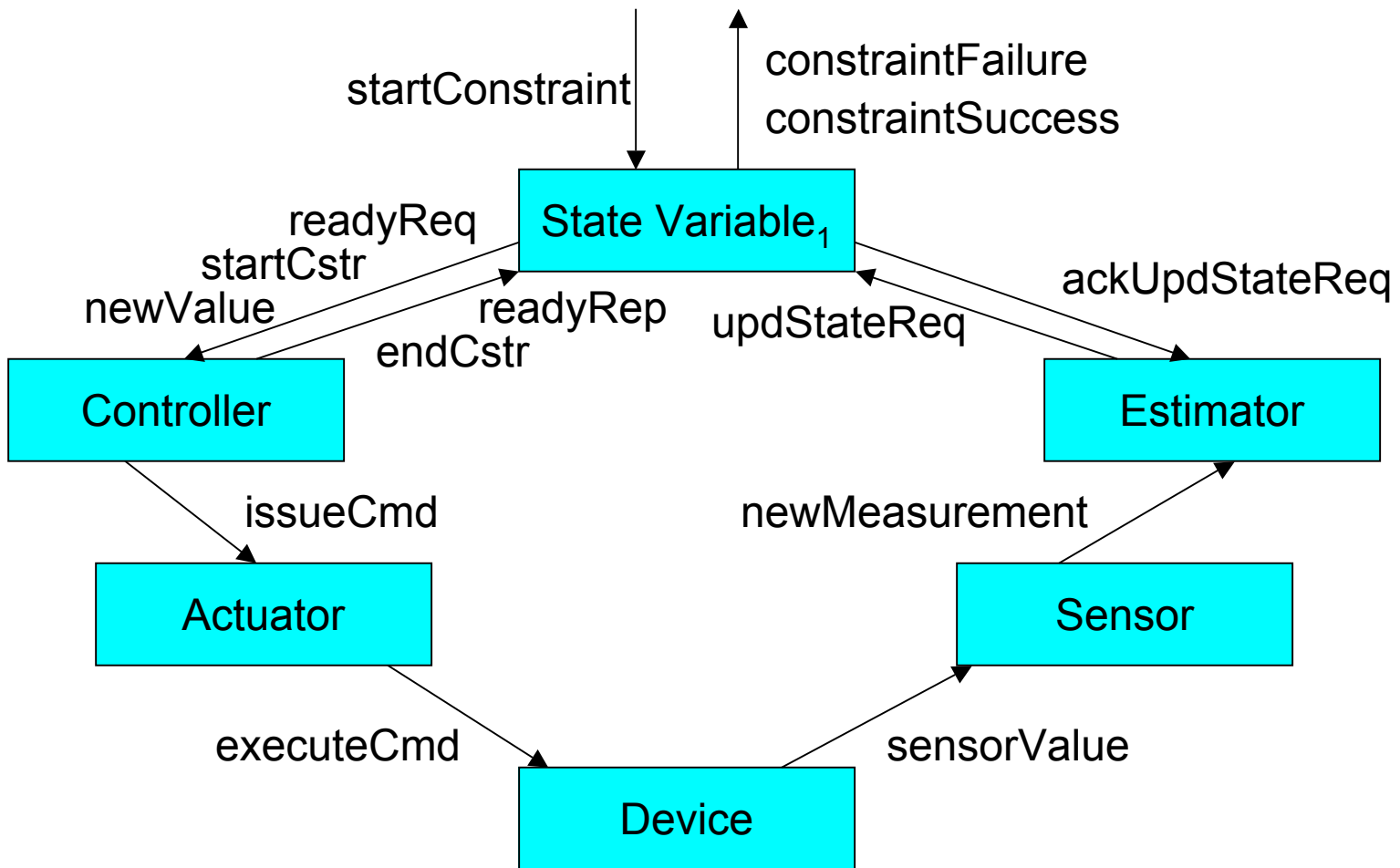
The operator interacts with a remote autonomous system in terms of *what* rather than *how*.

- A goal is a constraint on the value of a state variable over some time interval
- Goals are achieved by successive elaboration into subgoals
  - The base case is a goal that can be achieved by commands, issued by a controller to execute an operation
  - Subgoals are organized in a goal net -- a timed constraint net
- Goal elaboration uses the domain models

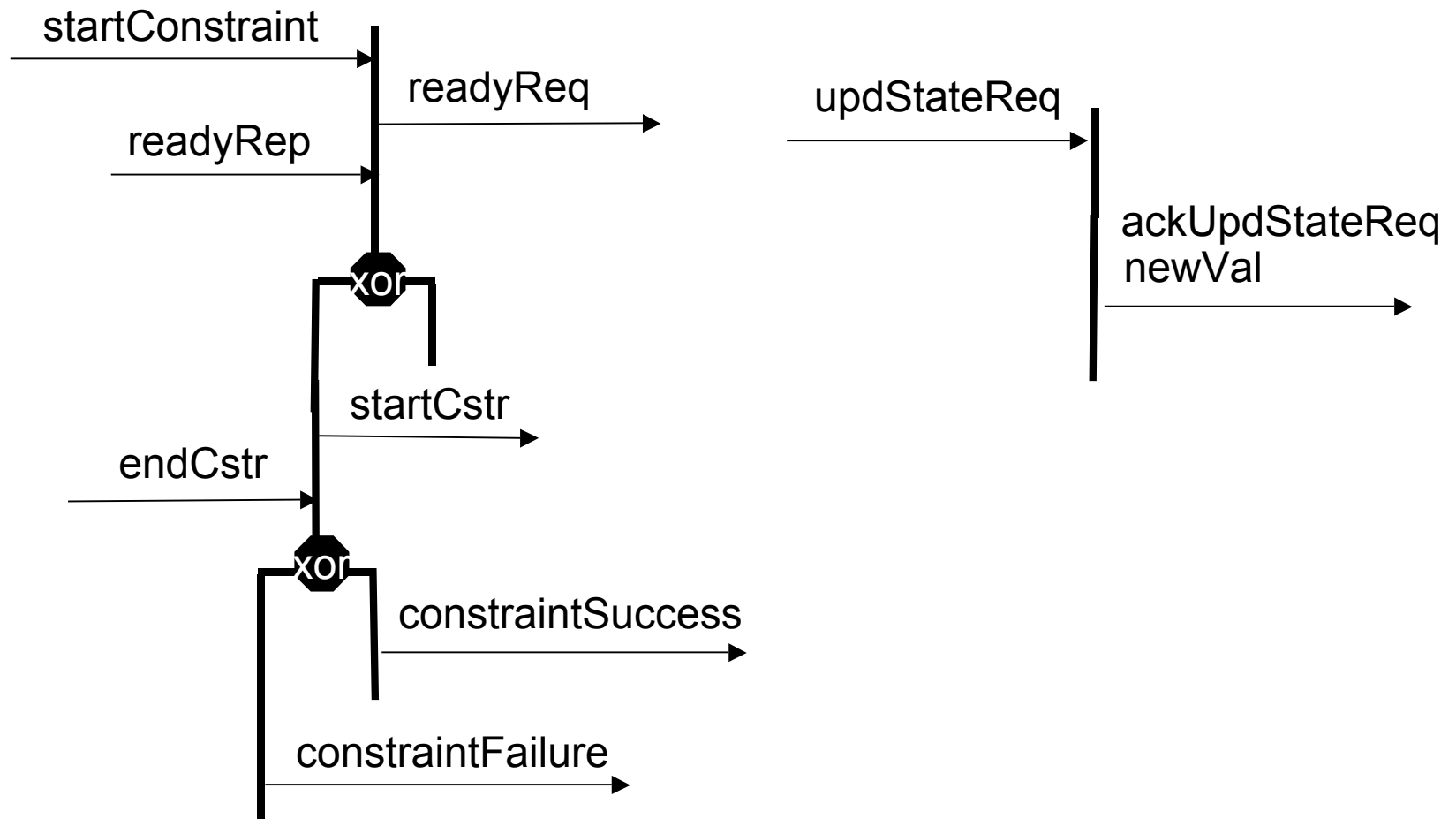
# Formal Specification of MDS in Maude

- MDS Architecture as *abstract classes*
  - Data types
  - Classes and attributes
  - Interfaces -- messages accepted and sent
  - Generic behavior -- control flow
- Case study -- the SCROver (shadow of a space rover)
- Checklists
- Goal Nets
- Goal elaboration rules

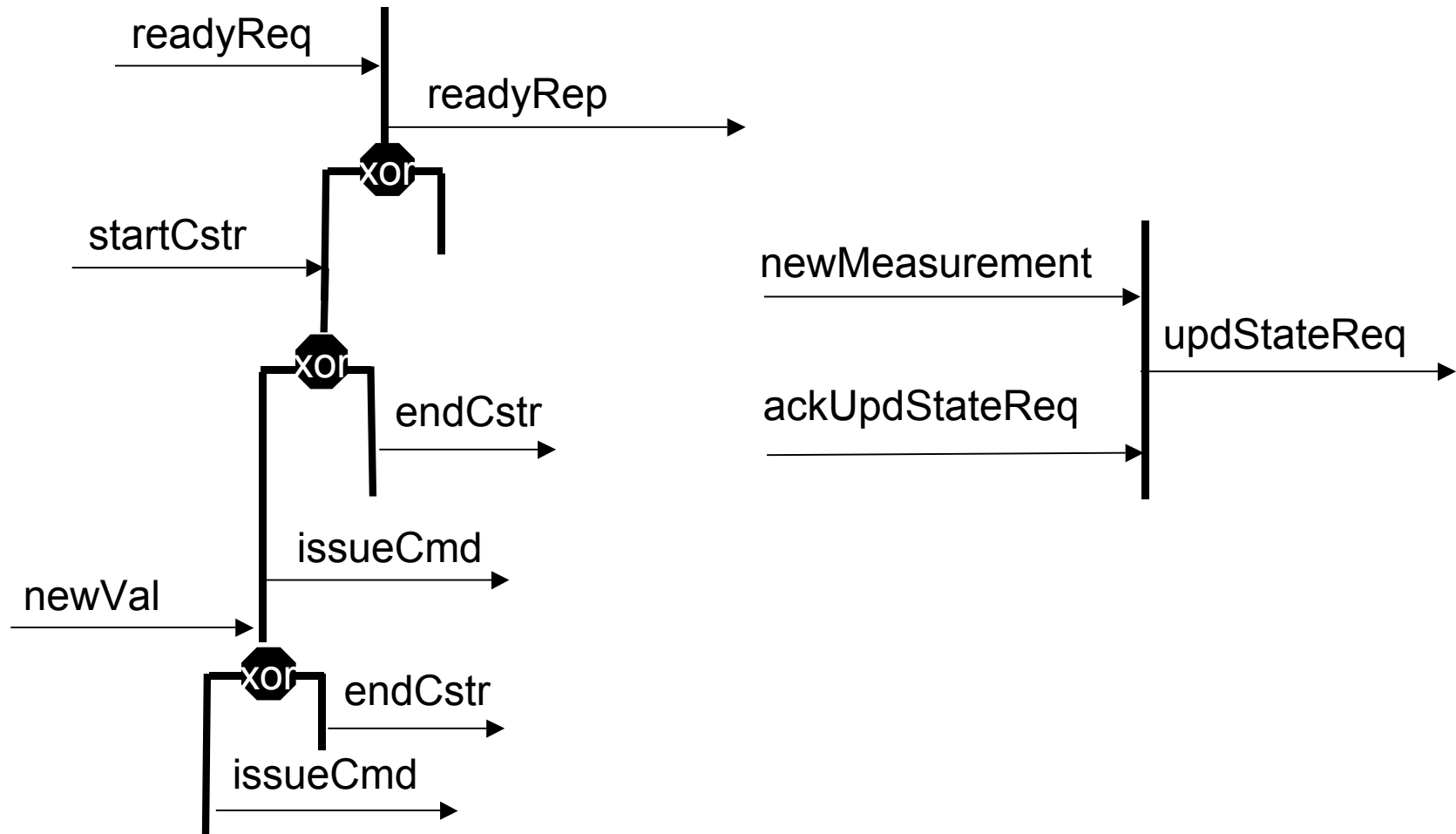
# Maude Modules and Interfaces



# State Variable Interactions



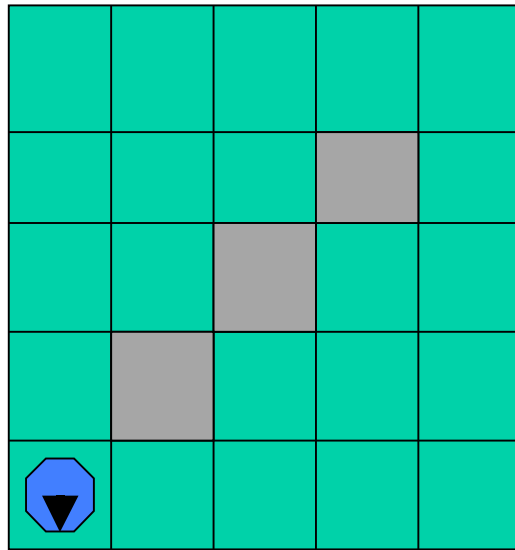
# Controller & Estimator Interactions



# MDS Adaptation -- General Plan

- Device
  - Measurable attributes (sensors)
  - Inputs (actuators)
  - Model effects of inputs on device state
- Intelligent device driver - define
  - High-level goals
  - ★ State variables and value data types
  - Primitive goals (commands)
  - Controller -- interprets commands as actuator inputs
  - Estimator -- compute state value using sensor and other data
  - Goal elaboration rules -- goal -> goal net

# Grid-based Rover Adaptation I



A `bot' constrained to move on an  $n \times m$  grid

The grid may have obstacles

The bot can rotate in 45 degree increments

Typical goal: move to (2,3) and head north

# Grid Based Rover Design

- Device
  - Grid --  $m \times n$  array with obstacles
  - Rover knows its square and heading
  - Drive input -- one square in direction headed
  - Turn input -- rotate clockwise 45 degrees
- Goal achiever
  - SV: PosAndHeadSV with values  $(x, y \text{ dir } D)$
  - PosAndHeadConstraint having the form  $(x, y \text{ dir } D)$
  - PosAndHeadController: computes COA from current and desired P&H
  - Estimator: simple unit conversion

# Checklists for Specification

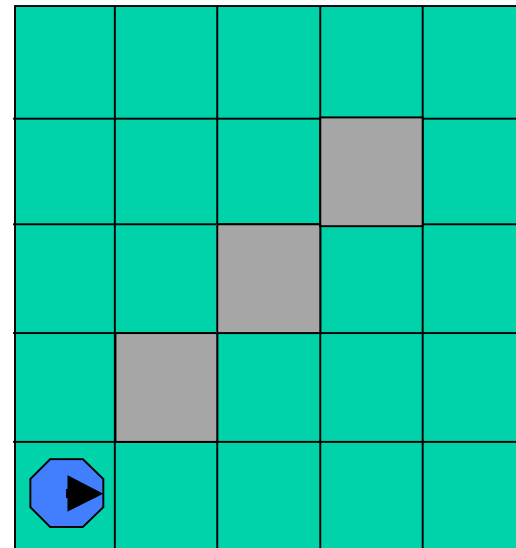
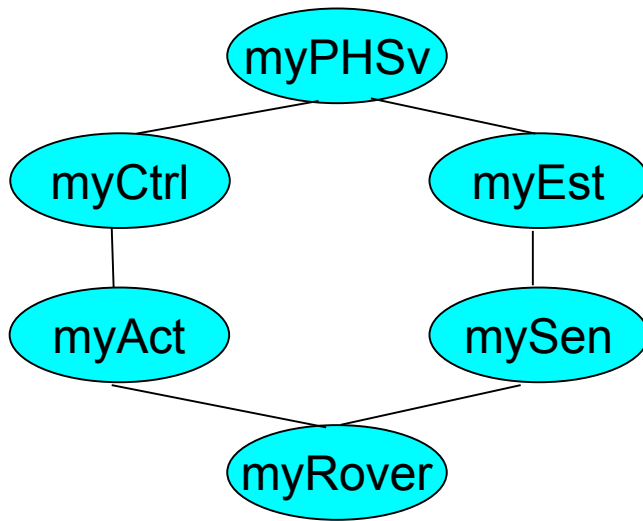
- L0: Syntactic well-formedness
  - Checked by Maude parser
  - Not every specification language has a parser!
- L1: Scenario execution -- per component and composed
  - Specify
    - Initial configuration
    - Goals
    - Expected outcomes
  - Execute (with some rewrite strategy) and check outcomes
  - Using search check all executions (if finite)
- L2: Invariants and temporal properties

# Checklists: Framework

- Mapping to MDS framework and adaptations
  - architecture specification
  - code structure
- Specify code instrumentation -- generate events
- Specify properties of events to check

# Analysis of Rover System (I)

- Test Scenario:
  - Grid: size 5x5, blocked fields (1,1), (2,2), (3,3)
  - Rover: initial position, e.g., pos(0,0) hd(90)
  - Initial system configuration



# PosHeadSV Initial Configuration

```
mod POSANDHEAD-STATE-VARIABLE-TEST is  
  inc POSANDHEAD-STATE-VARIABLE .
```

```
  op PosAndHeadStateVar : -> PosAndHeadSVCid .  
  op myphsv : -> Object .
```

```
  eq myphsv =  
    < o("MyPosAndHeadStateVar") : PosAndHeadStateVar |  
      myctrl(o("MyPosAndHeadCtrl")),  
      myest(o("MyPosAndHeadEstimator")), req(o("NoOid")),  
      waitAfter(noMsg), cstr(noCstr),  
      val((0,0 dir(E))) > .
```

```
endm
```

# Rover System Initial Configuration

```
mod SYSTEM is
  inc POSANDHEAD-STATE-VARIABLE-TEST .
  inc POSANDHEAD-CONTROLLER-TEST .
  inc POSANDHEAD-ACTUATOR-TEST .
  inc POSANDHEAD-SENSOR-TEST .
  inc POSANDHEAD-ESTIMATOR-TEST .
  inc ROVER-TEST .

  op sys : -> Configuration .
  eq sys = myphsv myphctrl myphactuator myphsensor myphest rov .
  op mkm : Nat Nat Dir -> Msg .
  vars x y : Nat . var d : Dir .
  eq mkm(x,y,d) = msg(o("MyPosAndHeadStateVar"), o("MyRequester"),
                    startConstraint((x y d))) .

  op ic : Nat Nat Dir -> Configuration .
  eq ic(x,y,d) = sys mkm(x,y,d) .
endm
```

# Rover System Checklist L1 I

scenario	expected outcome(s) -- checked using rewrite and search
1. ic(1,0,E)	< o("MyRover") : Rover   atts, pos(1,0),hd(90) > msg(o("MyRequester"), o("MyPosAndHeadStateVar"), constraintSuccess(1 0 E))
2. ic(2,0,E)	< o("MyRover") : Rover   atts, pos(2,0),hd(90) > msg(o("MyRequester"), o("MyPosAndHeadStateVar"), constraintSuccess(2 0 E))
3. ic(1,2,E)	msg(o("MyRequester"), o("MyPosAndHeadStateVar"), constraintFailure(1 2 E, COANoSuccess)))

# Rover System Checklist L1 II

scenario            expected outcome(s) -- checked using rewrite and search

-----  
-----

4. ic(1,0,E)        2 outcomes: for x in {1,2}  
mkm(2,0,E)        < o("MyRover") : Rover | atts, pos(x,0),hd(90) >  
                      msg(o("MyRequester"), o("MyPosAndHeadStateVar"),  
                      constraintSuccess(1 0 E))  
                      msg(o("MyRequester"), o("MyPosAndHeadStateVar"),  
                      constraintSuccess(2 0 E))

In an earlier version

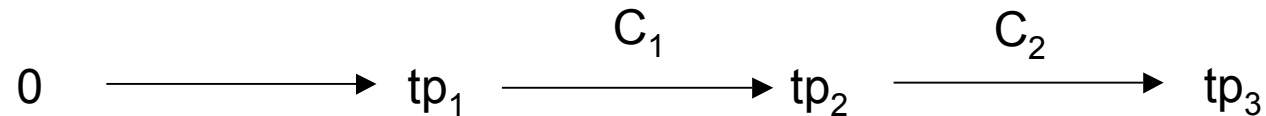
4'. Ic'(1,0,E)    16 solutions

mkm(2,0,E)        Problem: preconditions for startConstraint rule in SV and  
                      readyForNewConstraint in Ctrl are too weak

# Composition/Modularity Challenges

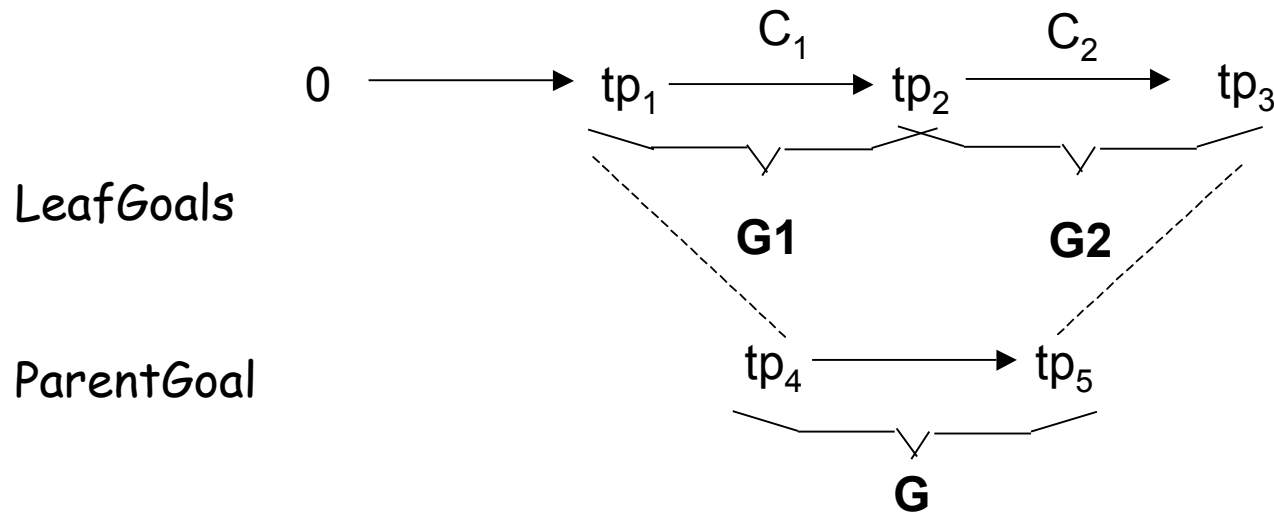
- Assembling adapted MDS components
  - Simple parallel composition (SYSTEM module)
    - Coordination must be embedded in components
    - Reduces modularity, increases potential for errors
  - Explicit coordination mechanism
    - MDS implementation specifies visit order
    - Composition with controller module
    - Strategy language
- Assembling device features
  - Adding arm to rover -- arm position a new state variable
    - Sensed/controlled relative to rover coordinate system
    - Observed / constrained relative to grid coordinate system
  - Adding battery to rover -- battery charge level a new state variable
    - Moving/driving decrease battery charge
    - Global goal to maintain charge level above 20%

# Goal Specification



- Goal -- an active object with
  - beginning and ending timepoints
  - associated state variable and constraint
- Goal can start its constraint when its beginning timepoint is fired
- Goal fires ending timepoint, when it is notified about constraint success or failure

# Goal Net Specification



- Parent goal fires beginning timepoint of subgoal.
- Goal fires ending timepoint when no more subgoals are due.
- Goal notifies parent when no more subgoals are outstanding.

# Accomplishments

- Formal documentation of MDS architecture
- Clarifying composition/synchronization rules
- Gotcha's in developing adaptations
- First checklists

## Future

- Complete MDS architecture with missing details
- Model power and wall following rover capabilities
- Design representation for goal nets
- Develop schemes for goal elaboration
- Build tools for checklists

# StateVariable: Class, Attributes, Instances

- StateVariable is a class with attributes myctrl, myest, val, ...

Sorts:

SVCid < Cid

StateValue

Attributes:

myctrl : Oid -> Attribute

myest : Oid -> Attribute

val : StateValue -> Attribute

Operators:

StateVar : -> SVCid .

uk : -> StateValue . \*\*\* unknown state value

- Instance of class StateVariable

< o("MyStateVar") : StateVar | myctrl(o("MyCtrl")), myest(o("MyEstimator")), val(uk) >

## Example StateVariable Rule

Variables:

sv, ctrl, est : Oid svcid : SVCid

v : StateValue

svatts : AttributeSet \*\*\* placeholder for all other attributes

rl[startConstraintReadyReq]:

< sv : svcid | myctrl(ctrl), req(o'), cstr(cstr'),

waitAfter(noMsg), svatts>

msg(sv,o,startConstraint(cstr))

=>

< sv : svcid | myctrl(ctrl), req(o), cstr(cstr),

waitAfter(msg(sv,o,startConstraint(cstr)), svatts>

msg(ctrl,sv,readyReq) .