Reflecting on Planning Models: A Challenge for Self-Modeling Systems

Jeremy Frank
NASA Ames Research Center
Talk Outline

• Motivating Example
  – Planning for cyber-physical systems (a Spacecraft)
  – Command and Telemetry representation
  – Model-based Planning representation

• Declarative Abstractions and Refinements

• Detecting Model Errors
  – Data Driven / Learning
  – Fault Management / ‘Oracle’

• The Challenge: Correcting Model Errors
Example

• Suppose we are developing a mission planning system for a spacecraft.
  – This could be for a ground system or as part of an autonomous spacecraft.

• How does a spacecraft plan a change the direction (attitude) it is pointing?
  – A change of pointing is called a slew.
Example

- Attitude is angles \(<x, y, z>\) in some absolute coordinate frame (e.g. Earth-centric).
- Slews are constrained by solar panel power generation, thermal, communications to Earth, sensor and instrument performance and safety (among other things).
Example (Commands)

//Startracker
s({o,f})

//CPU
c({o,f})
\(u(p)\)
//Power dist.
pd({s,b})

//Reaction
//Wheels
r(wi,{c,o},v)

//Battery
pd({s,b})

---

3/28/18
AAAI Spring Symposium
Example (Data)

//Startracker
x,y,z
dx,dy,dz
s
//CPU
on,off
u
//Power dist.
s1,s2
P
//Reaction
//Wheels
w1,w2,w3,w4
dw1,dw2,dw3,dw4
//Battery
b,db

Earth
Slewing
CPU
Sun
Moon

>45
>2
3/28/18
AAAI Spring Symposium
The Planning Model

• A planning *model* consists of:
  – Objects – things in the world.
  – Predicates – properties of things. (True/False)
  – Functions – properties of things. (Numbers)
  – Actions – ways of changing the properties of things.

• A planning problem consists of:
  – A model.
  – An initial state description.
  – A set of goal states.

• The planner reads the model, initial states, and goals, and produces a plan.
Example

pointing
cpu-on
generating
slewing
discharging

battery
4.0
2.0
0.0

CPU

>45

Sun

>2

Moon

Earth

Slewing
Model-Based Planning

(:durative-action slew
 (:parameters (?from - attitude
               ?to - attitude)
 :duration (= slew-time ?from ?to)
 :condition (and
             (at start (pointing ?from))
             (at start (cpu on))
             (over all (cpu on))
             (at start (generating))
             (at start (>= (battery ?sc) 3.0)))
 :effect (and
          (at start (decrease(battery ?sc) 2.0)
          (at start (discharging ?sc))
          (over all (discharging ?sc))
          (at start (slewing))
          (over all (slewing))
          (at end (not slewing))
          (at start (not generating))
          (at start (not pointing ?from))
          (at end (pointing ?to)))

pointing
cpu-on
generating
slewing
discharging
battery
4.0
2.0
0.0

Action
Object
Condition
Predicates
Temporal Qualifier
Effect
Functions
Predicates
Model-Based Planning

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pointing</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>pointing(moon)</strong></td>
<td></td>
<td></td>
<td><strong>pointing(moon)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>cpu-on</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>cpu-on</strong></td>
<td></td>
<td></td>
<td><strong>cpu-on</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>generating</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>generating</strong></td>
<td></td>
<td></td>
<td><strong>generating</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>slewing</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>discharging</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>battery</strong></td>
<td>4.0</td>
<td>2.0</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Known True
Possibly True
We got nothin'
Function value

3/28/18
AAAI Spring Symposium
10
Model-Based Planning

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pointing</strong></td>
<td>pointing(moon)</td>
<td>pointing(earth)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>cpu-on</strong></td>
<td>cpu-on</td>
<td>cpu-on</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>generating</strong></td>
<td>generating</td>
<td>generating</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>slewing</strong></td>
<td>slewing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>discharging</strong></td>
<td>discharging</td>
<td>discharging</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>battery</strong></td>
<td>4.0</td>
<td>2.0</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Action:
- slew (moon, earth)
Model-Based Planning

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>pointing</td>
<td></td>
<td></td>
<td>pointing(moon)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cpu-on</td>
<td></td>
<td></td>
<td>cpu-on</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>generating</td>
<td></td>
<td></td>
<td>generating</td>
<td></td>
<td></td>
<td>generating</td>
<td></td>
<td></td>
</tr>
<tr>
<td>slewing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>discharging</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>battery</td>
<td>4.0</td>
<td>2.0</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- CPU off!
- CPU not on
- slew (moon, earth)
Declarative Abstractions and Refinements

• Unlike other applications, *abstraction* is a key element of modeling in this type of problem.
  – Planners approximate or abstract the actual spacecraft behavior.
  – These abstractions are typically not *documented*.
Declarative Abstractions and Refinements

• Predicate and Function Abstractions:
  – Functions that map the spacecraft data to planning model predicates or functions.
  – One abstraction per predicate or function.

• Command Refinements:
  – Function mapping a planning action to a command sequence.
  – One refinement per plan action.
Pointing = 
(x_e - \epsilon < x < x_e + \epsilon) \land 
(y_e - \epsilon < y < y_e + \epsilon) \land 
(z_e - \epsilon < z < z_e + \epsilon)

Generating = 
(p = s) \land 
(s_1 + s_2 > \epsilon_1) \land 
(db > \epsilon_2)

2 = db_1 + db_2 + db_3
Slew:
[0]:
s(f);    //startracker
pd(b);  //power dist
u(pi,o); //pointing mode
u(ps,f)  //pointing mode
r(w1,c,500); //x rotate
r(w2,c,500);
[10]
r(w1,c,0);
r(w2,c,0);
[11]
r(w1,c,500); //y rotate
...
[14]
u(pi,f); //pointing mode
u(ps,o)  //pointing mode
s(o);    //startracker
Detecting Modeling Errors and Model Drift

• Modeling is error prone; abstraction compounds the errors that can be introduced.
• Models may also become wrong over time
  – Due to changes in the system
  – Due to changes in the operating environment
  – Due to changing mission goals and objectives
Detecting Modeling Errors and Model Drift

• What manner of modeling errors can occur?
  – Action Failure: Conditions satisfied but action fails
  – Missing Condition: Action condition not in plan so action fails
  – Unexpected Condition: Condition unexpectedly influences action outcome
  – Missing effect: Action succeeds but effect missing
  – Unrealized Effect: Action has unmodeled effect
  – Timing discrepancy: Action length differs or effects occur at different times than expected
Detecting Modeling Errors and Model Drift

• What are the root causes of modeling errors?
  – Missing Command
  – Bad command order
  – Incorrect command input
  – Mis-timed command
  – Missing Abstraction
  – Abstraction error
## Detecting Modeling Errors

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pointing</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>cpu-on</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>generating</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>slewing</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>discharging</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>battery</strong></td>
<td>4.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
</tr>
<tr>
<td><strong>Action</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **pointing (moon)**
- **pointing (earth)**
- **cpu-on**
- **generating**
- **slewing**
- **discharging**
- **slew (moon, earth)**

Effect starts late!
Detecting Modeling Errors

Pointing:
\[(x_e - \varepsilon < x < x_e + \varepsilon) \land (y_e - \varepsilon < y < y_e + \varepsilon) \land (z_e - \varepsilon < z < z_e + \varepsilon)\]

Generating:
\[(p=s) \land (s1 + s2 > \varepsilon_1) \land (db > \varepsilon_2)\]

Pointing error wrong!

Generating = (p=s) \land (s1+s2>\varepsilon_1) \land (db>\varepsilon_2)

2=\text{db}_1+\text{db}_2+\text{db}_3

Earth \(x_e, y_e, z_e\)

Slewing

Sun

Moon

>45

>2
Detecting Modeling Errors

- **pointing** (moon)
- **CPU-on**
- **generating**
- **slewing**
- **discharging**
- **battery**: 4.0, 2.0, 0.0
- **slew** (moon, earth)

Unexpected Effect!
Detecting Modeling Errors

(:durative-action slew
  :parameters (?from - attitude
               ?to - attitude)
  :duration (= slew-time ?from ?to)
  :condition (and
              (at start (pointing ?from))
              (at start (cpu on))
              (over all (cpu on))
              (at start (generating))
              (at start (>= (battery ?sc) 3.0)))
  :effect
  (and
   (at start (decrease(battery ?sc) 2.0)
   (at start (discharging ?sc))
   (over all (discharging ?sc))
   (at start (slewing))
   (over all (slewing))
   (at end (not slewing))
   (at start (not generating))
   (at start (not pointing ?from))
   (at end (pointing ?to))
Detecting Modeling Errors

(:durative-action slew
  :parameters (?from - attitude
               ?to - attitude)
  :duration (= slew-time ?from ?to)
  :condition (and
    (at start (pointing ?from))
    (at start (cpu on))
    (over all (cpu on))
    (at start (generating))
    (at start (>= (battery ?sc) 3.0)))
  :effect (and
    (at start (decrease(battery ?sc) 2.0)
    (at start (discharging ?sc))
    (over all (discharging ?sc))
    (at start (slewing))
    (over all (slewing))
    (at end (not slewing))
    (at start (not generating))
    (at start (not pointing ?from))
    (at end (generating))
    (at end (pointing ?to)))

pointing

cpu-on

generating

slewing
discharging

battery

4.0
2.0
0.0

Fixed Model
“Reflection” Architecture

- Planner
- Model
- Abstractions
- Fixer(s)
- Errors
- Validation
- Actual States
- Expected States
- Predicate Abstraction
- Telemetry
- Commands
- Command Refinement

Plan → Actions

Errors → Validation → Actual States → Expected States → Predicate Abstraction → Telemetry → Commands → Command Refinement → Actions → Plan
“Reflection” Architecture: Algorithm Sketch

• All abstractions either
  – Map a domain of a telemetry variable into a domain of smaller cardinality
    • \( f(X) \Rightarrow Y \) s.t. \(|X| > |Y|\)
    • Special cases: eliminate element of a discrete domain, map reals to integers, map integers to positive integers
  – Map \( n \) variables to \( m < n \) variables
    • \( f(x_1...x_n) \Rightarrow \{y_1...y_m\} \)
    • Cardinality must still be reduced:
      \(|X_1|/|X_2|...|X_n|<|Y_1|/|Y_2|...|Y_m|\)
    • Special cases: eliminate variable
“Reflection” Architecture: Algorithm Sketch

• Planner / plans
  – N actions
  – $P_n$ action conditions / effects for action n
  – S predicates in the model (one abstraction per predicate)
  – $M_p$ predicate instances in a plan

• System
  – $C_n$ commands in refinement of action n
  – T telemetry items
  – R values for each item per run
  – $M_s << TR$ predicate instances produced by simulation

• A similar analysis can be done for numerical abstractions
  – A finite number of numerical abstractions are formalized!
“Reflection” Architecture: Algorithm Sketch

• Generate refinement from plan
  – \( \sum C_n \) (N actions, \( C_n \) commands in refinement of action n)
• Execute / simulate refined command sequences
• Generate plan abstraction from telemetry
  – T telemetry items, R values for each item per run, S predicate types.
  – First pass is to generate states: for all R, for all abstractions S, each abstraction uses at most T telemetry items. This gets us runtime TRS.
  – 2nd pass is to determine start / end times of predicates; this is another SR.
  – (There are important assumptions about the form of the abstractions i.e. they only use values at one time tic)
  – Total: TSR + SR
“Reflection” Architecture: Algorithm Sketch

[Diagram of a matrix with annotations]

Abstraction

Abstraction
“Reflection” Architecture: Algorithm Sketch
"Reflection" Architecture: Algorithm Sketch
“Reflection” Architecture: Algorithm Sketch

• Generated warnings
  – Check to see if simulated predicate start / end time match compared to plan predicate start / end times for same predicate types; sufficient, but overkill, to check every pair $m \in M_p$, $o \in M_s$.
    – $O(M_p M_s)$

• Generate action discrepancies
  – N actions, $P_n$ action conditions / effects
  – For each condition/effect of an action, may need to search all $M_s$ predicate to match conditions / effects
    – $O(M_s \Sigma_{n \in N} (P_n))$
Promise of Reflection on Models

• Detect modeling errors prior to launch through testing
• Adapt to changes in spacecraft environment
  – Solar panel power generation due to dust (e.g. during surface operations)
  – Unpredictable gravitational field impact on attitude and orbit determination (e.g. small bodies like asteroids or comets)
  – Unpredictable communications performance
  – Unpredictable lighting conditions
• Adapt to changes in spacecraft performance
  – Solar panel power generation due to age
  – CMG degradation
  – Battery performance (e.g. cell or string failure)
Promise of Reflection on Models

- Advantages of reflection and adaptation onboard:
  - More data than telemetered back to ground
  - Higher rate data
  - Ability to reflect continuously
  - No need to pay costs of communication
Promise of Reflection on Models

• Promising techniques
  – Classification – identify rules distinguishing cases when actions fail vs when they succeed
  – Function approximation – attempt to debug predicate abstractions
Promise of Reflection on Models

- Promising techniques
  - Clustering – identify patterns of behavior in data and map them to predicates
  - Exploratory actions – judicious use of proposed rules in new plans
Promise of Reflection on Models

• What about faults?
  – A special class of degradation / unexpected event
  – Performance changes are detected and reported by fault management algorithms

• Instead of using data to learn and characterize changes in planning model, make use of these fault detection algorithms’ outputs directly.
  – Treat fault management algorithm as ‘oracle’
“Reflection” Architecture

- Planner
- Model
- Abstractions
- Fixer(s)
- Errors
- Validation
- Actual States
- Expected States
- Fault Detection
- Fault Impacts Reasoning
- Commands
- Telemetry
- Predicate Abstraction
- Plan
- Actions
- Command Refinement
Promise of Reflection on Models

Slew:
[0]:
s(f); //startracker
pd(b); //power dist
u(pi,o); //pointing mode
u(ps,f) //pointing mode
r(w1,c,500); //x rotate
r(w2,c,200);
[12]
r(w1,c,0);
r(w2,c,0);
[14]
r(w1,c,500); //y rotate
...
[17]
u(pi,f); //pointing mode
u(ps,o) //pointing mode
s(o); //startracker

Fault on rw2 limits max Velocity!!

Sequence duration updated!

Fault on rw2 limits max Velocity!!

Sequence duration updated!
Promise of Reflection on Models

(:durative-action slew
 :parameters (?from - attitude
   ?to - attitude)
 :duration (= slew-time ?from ?to)
 :condition (and
   (at start (pointing ?from))
   (at start (cpu on))
   (over all (cpu on))
   (at start (generating))
   (at start (>= (battery ?sc) 3.0)))
 :effect
   (and
    (at start (decrease(battery ?sc) 2.0))
    (at start (discharging ?sc))
    (over all (discharging ?sc))
    (at start (slewing))
    (over all (slewing))
    (at end (not slewing))
    (at start (not generating))
    (at start (not pointing ?from))
    (at end (pointing ?to)))

pointing

cpu-on

generating

discharging

battery 4.0

2.0 0.0

3/28/18
AAAI Spring Symposium
Challenges of Reflection

• Algorithms are resource (computationally, memory) intensive
  – “Software has weight”
• Existing algorithms for proposing model fixes may not be sufficient
• Algorithms may not find answers due to insufficient data
• Experimentation in mission critical environments may be dangerous
• Model “configuration management” may be needed if a proposed fix does not perform well
Challenges of Reflection

• Fault management algorithms in general reason at low level of abstraction
  – Output often can’t directly be used to change planning models
  – Requires a similar abstraction between components that can fail and actions in planning model
• Fault management algorithms exist for detecting loss of capability and redundancy, and leaks
• Making connection to action model is hard
  – Example: leak detection. Using leaky system increases resource consumption rate.
  – Operationally, may prefer simply not to use this subsystem (malfunction => don’t use) but sometimes ya gotta do it
A Few Words About a REAL Spacecraft: LADEE

- How would this approach need to scale for LADEE?
  - ~600 Commands
  - ~25000 Telemetry / data
  - 122 Activities
  - 27 States
  - 21 Numerical Resources

- The LADEE planner model has ~ 12000 lines.
- Simulation data produced at 10Hz (cycles / second)
The Takeaway

• Model-based planning:
  – Planning performed at a *high level of abstraction* (compared to system behavior).

• Availability of a simulation as an ‘oracle’:
  – Abstractions relate planning model to simulation at lower level of abstraction.
  – Abstractions used to identify model errors.

• The detection of model errors can be automated; it is a challenge to automatically propose corrections to the model or the abstractions to eliminate errors.
  – Doing so enables reflection on planning models and therefore self-improvement.
Thank You!
Previous Work (Applications)

• Remote Agent [1], EO-1 [2]
  – Extensive model reviews.
  – Safety reviews to elicit potential hazards.
  – Automated tests stochastically generated by perturbations of nominal scenarios.
  – Executed on simulation platforms of varying fidelity where spacecraft, operations, and safety constraints were checked.
Previous Work
(Academia)

• itSimple [3]
  – Allows some domain behavior modeling using UML object diagrams.
  – Generated plans can be checked against the UML.

• KEEN [4]
  – Similar to itSimple, but uses Timed Game Automata (TGA) instead of UML as domain model.
  – Emphasis on temporal planning domains and temporally flexible plans.

• PDVer [5]
  – Plan domain properties specified in LTL (Linear Temporal Logic).
  – Specification of test cases (goals) automatically from LTL.
Previous Work (Academia)

• VAL [6]
  – Given a plan and a model, determines whether the plan satisfies the constraints in the domain.
  – Limited ability to automatically fix plans.

• Model checking as plan verification [7]
  – Employs Java Pathfinder to check properties of PLEXIL, a language and plan execution system.
  – Requires a system model (or simulator) and associated properties to check.
Previous Work
(Summary)

• There are tools to assist in verification of plans against planning models.
• There are tools to assist in test case generation and model verification.
• *Few to no tools* to assist in validation of models.
• *No tools* to assist in validation against simulations.
Diagnostic Reasoning – Example

- Power Supply
- Power Distribution Unit (RPC)
- Pump
- PDU_BUS_POWER
- PUMP_ROTATION
- PDU_RPC_POWER

<table>
<thead>
<tr>
<th>Condition</th>
<th>PDU_Bus_Power</th>
<th>Pump_Rotation</th>
<th>PDU_RPC_Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pump.Mechanical_Failure</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power_Supply.Fail_to_Gen_Power</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RPC.Fail_Open</td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Diagnosing Power Supply

**Power Supply** → **Power Distribution Unit (RPC)** → **Pump** → **Pump Rotation**

<table>
<thead>
<tr>
<th>Test Case</th>
<th>PDU_Bus_Power</th>
<th>Pump_Rotation</th>
<th>PDU_RPC_Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pump.Mechanical_Failure</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Power_Supply.Fail_to_Gen_Power</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RPC.Fail_Open</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**FAIL Test**

**PASS Test**
Diagnostic Reasoning – Example

Power Supply → Power Distribution Unit (RPC) → Pump → PUMP_ROTATION

PDU_BUS_POWER → PUMP_ROTATION

PDU_RPC_POWER

Test Results:

- **FAIL Test**: Pump.Mechanical_Failure
- **PASS Test**: Power_Supply.Fail_to_Gen_Power, RPC.Fail_Open

<table>
<thead>
<tr>
<th>Condition</th>
<th>PDU_Bus_Power</th>
<th>Pump_Rotation</th>
<th>PDU_RPC_Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pump.Mechanical_Failure</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power_Supply.Fail_to_Gen_Power</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>RPC.Fail_Open</td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Diagnostic Reasoning – Example

- Power Supply
- Power Distribution Unit (RPC)
- Pump

**PDU_BUS_POWER**

- **PUMP_ROTATION**
- **PDU_RPC_POWER**

---

**FAIL Test**

- Pump.Mechanical_Failure
- Power_Supply.Fail_to_Gen_Power
- RPC.Fail_Open

**PASS Test**

- Must be Pump Mechanical Failure!
Future Work

• How can errors be identified and fixed for different modeling language features, such as uncertainty, parameter functions, and decompositions? (e.g. learning)

• How can the architecture be adapted to suggest changes for plan quality?

• How can we take advantage of white box simulators? auto-generate refinements to sim commands? auto-fill model? [13]

• See [14] for tools for authoring abstractions.
References


References


