Preliminary Investigations on Intelligent Modeling of UML Scenarios

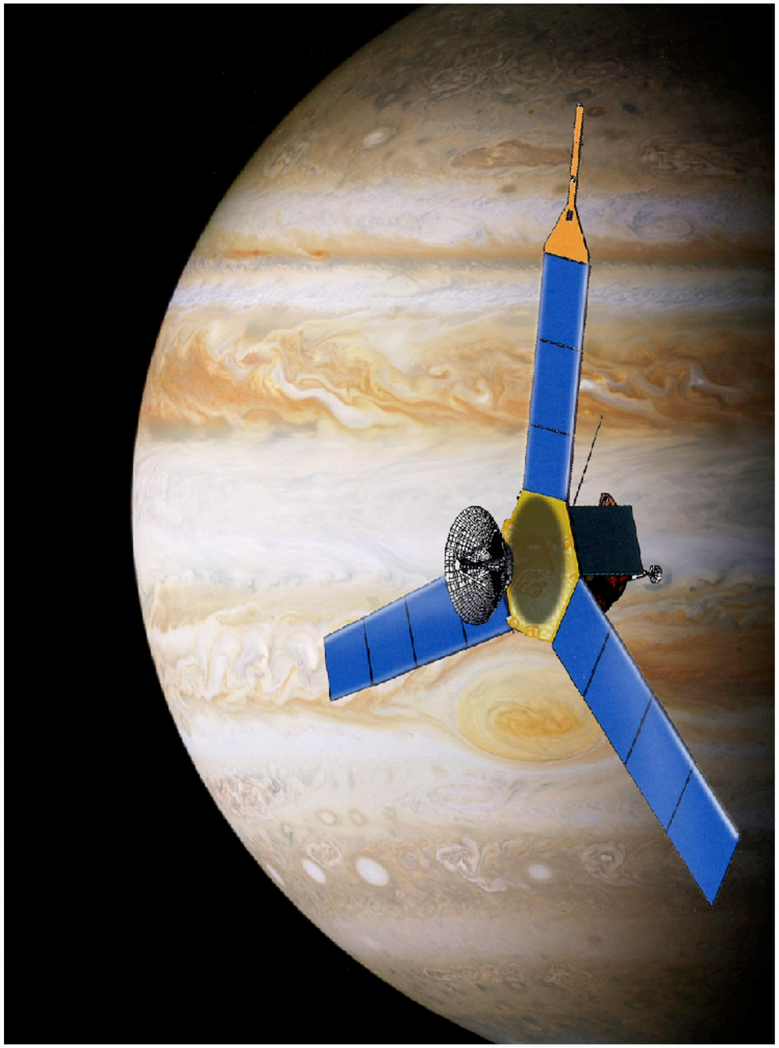
Dr. Tim Menzies, WVU, tim@menzies.us
Chet Tobrey, WVU, ctobrey@mix.wvu.edu
Lee Blake, WVU, lblake2@mix.wvu.edu

Space is a big place

- NASA explores space
- Software engineers explores the space within:
  - programs (late lifecycle)
  - or requirements (early lifecycle)
- Question:
  - how to best explore all that space?

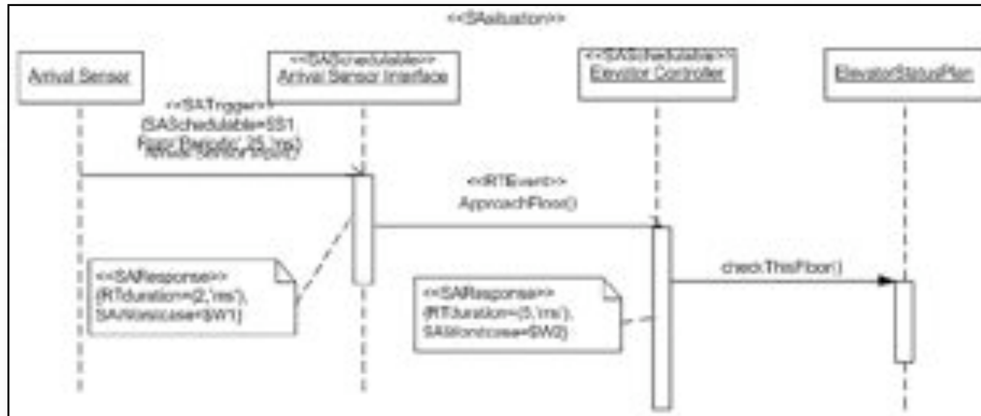
Goal: Better Support for UML modeling in Juno

- New generation IV&V
  - UML-based
  - Create a UML reference model
  - Conduct model-based IV&V
- Can we improve that work?
  - Simulate software activity.
  - Identify key decision points.
  - Quick feedback.
  - Scalable for large systems.
  - Possible reuse with other models.
  - Minimize work required to assess the mode



Nominal & Off-Nominal Scenarios

- UML scenarios:
  - “a story” about one sequence of actions
  - Undistracted by multiple options
  - To explore options, write another scenaio



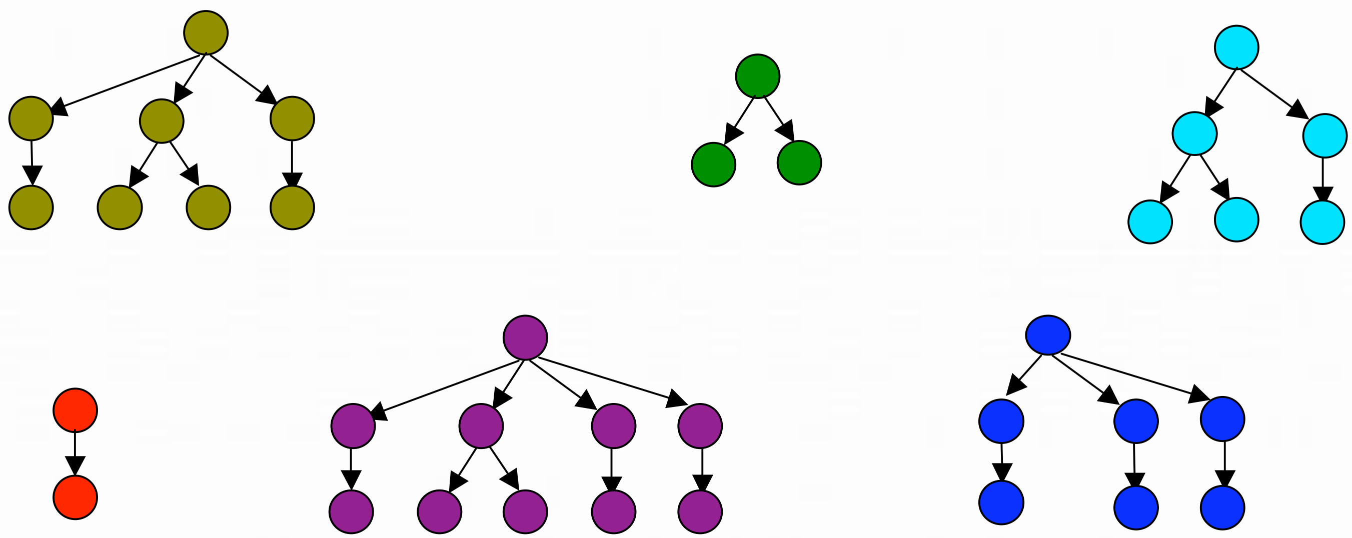
- Nominal scenario: sunny data
- Off-nominal scenario: a rainy-day variant of a nominal scenario
  - Typically, a branch from a nominal scenario
- Nominal : Off-nominal = 1 : 10
- Q: how may scenarios?
  - A: depends on the structure of the design

Roadmap

- Preliminaries: parsing the XML
- Pairwise sample
- Rank by usage
- Combining pair-wise with usage
- Divide and conqueor
- Goal-based analysis

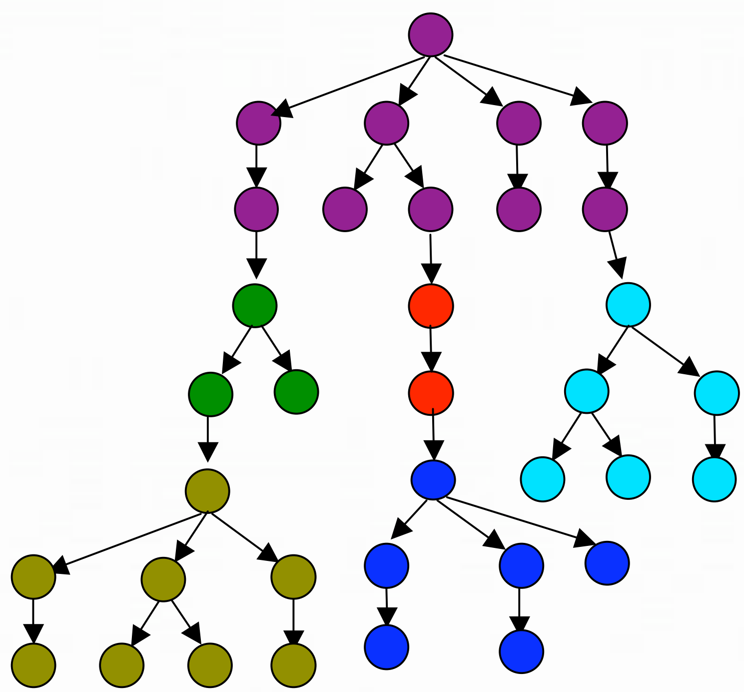
Parsing the XML

- Split XML document into small sub-files.
- Convert each sub-file into an object utilizing SimpleXML and X-Path in php 5
- Convert each object into a small tree representing the XML in the sub-file



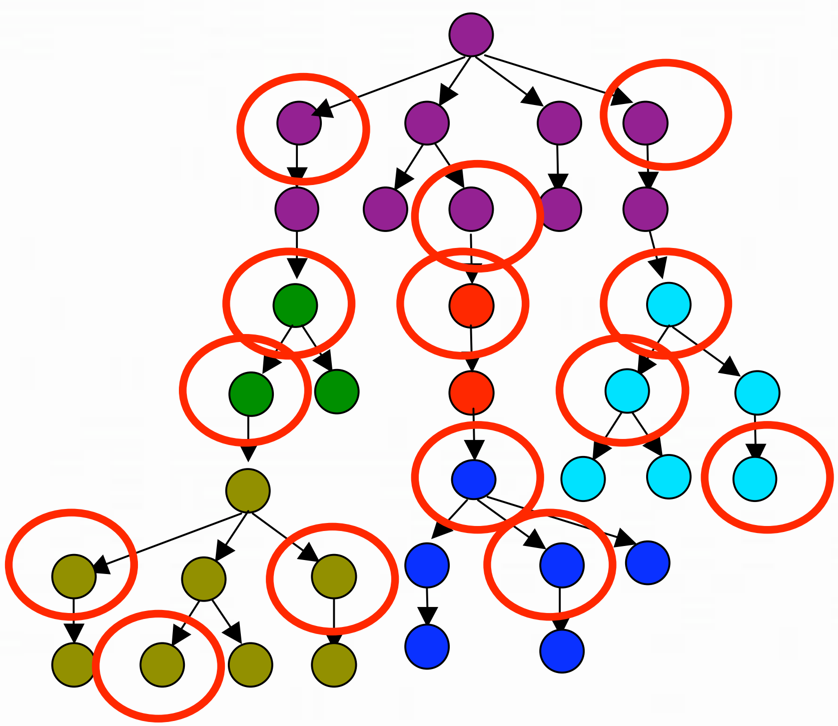
Parsing the XML

- Connect each small tree to create a very large tree (XML Tree) representing the entire XML document



Parsing the XML

- Search XML Tree for required node types (Node, Edge, Guard) and extract “branch” of XML Tree discarding unneeded “branches”.



Grammar Generation

- Extract pertinent attributes from each XML “branch” creating an array.
  - Nodes : Name, ID, Group, Incoming Edges, and Outgoing Edges
  - Edges : ID, Group, Source Nodes, Target Nodes
  - Guards : Name, ID, Group, Edge
- Match Nodes to Edges
  - using Incoming, Outgoing, Source, and Target
- Merge Edges into Nodes.
  - Replace Node's Outgoing with the appropriate Edge's Target
- Search Node array for Nodes with no Incoming (Initial Nodes)
- Trace “tree” from Node to Node extracting entire “diagrams.”
- Convert each Diagram Trace into a Grammar

Grammar Generation

- Each Grammar consists of an entire “diagram” extracted from the Juno XML.
  - Each diagram represents a task or group of related tasks that the Juno software performs.
  - Each diagram is represented by a separate grammar to facilitate easier and faster testing because they can be used independently
- Each Grammar is accompanied by a setup function that relates the Node IDs used in the grammar to the plain English names that describes the node.
- Finally, a Guard Grammar and setup function are created to allow testing of certain initial “conditions” applied to the grammar prior to analysis.

Output: a LISP program

Sample of Grammar

```
(defparameter *alignspinaxis*
  '((XX -> XY) 1
    (XX -> XJ) 1
    (XX -> KL) 1
    (AB -> X) 1
    (AB -> D) 1
   …)
```

Sample of Setup Function

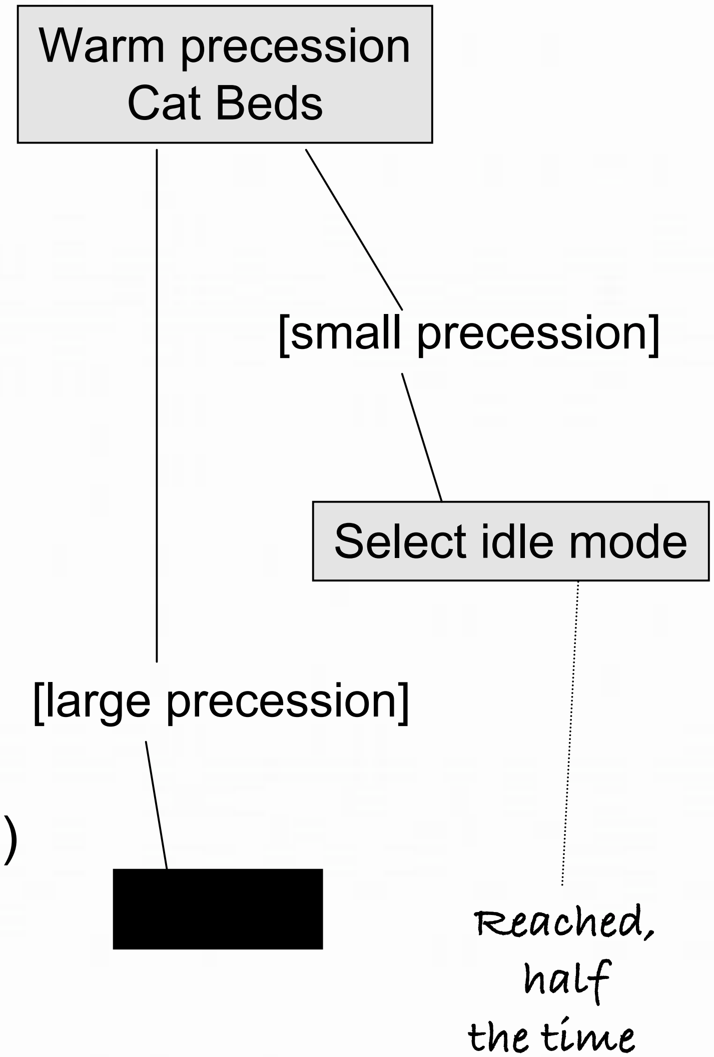
```
(defun setupalignspinaxis()
  (setf *name-list*
   '(A  generate_torque_vector
     B get_imu_data
    C  fire_thrusters
    D generate_pulse_width_commands
    E set_to_idle_mode
    F calculate_spin_rate_error
    G determine_current_spin_rate
    F warm_up_cat_beds
    G select_spin_rate_control_mode
    H send_adjust_spin_rate_commmands
…
```

Roadmap

- Preliminaries: parsing the XML
- Pairwise sample
- Rank by usage
- Combining pair-wise with usage
- Divide and conqueor
- Goal-based analysis

Exploring the gaurds

- View “warm precession Cat Beds” as a gaurded variable with range = 2
  - Model that as (2)
- Given 5 binary choices:
  - (2 2 2 2 2)
  - 2^5 = 32 scenarios
- 268 guarded nodes in the system
  - Usually, range=2 (but sometimes, 10)
  - $2^{268} = 4.7 \times 10^{80}$ scenarios



Sampling guard space

- Assumption:
  - the simplest bugs from a single input parameter.
  - Harder bugs: from pairs
  - Harder harder bugs: from tripple
- N-wise constraints
  - No two tests can have the same values to N variables
- Pair-wise testing

Parameter Sizes	AETG [1)]	IPO [2)]	TConfig [3)]	CTS [4)]	Jenny [5)]	TestCover [6)]	DDA [7)]	AllPairs [McDowell] [5)]	PICT	EXACT [8)]
3^4	9	9	9	9	11	9	?	9	9	9
3^{13}	15	17	15	15	18	15	18	17	18	15
$4^{15}\ 3^{17}\ 2^{29}$	41	34	40	39	38	29	35	34	37	?
$4^1\ 3^{39}\ 2^{35}$	28	26	30	29	28	21	27	26	27	21
2^{100}	10	15	14	10	16	10	15	14	15	10
10^{20}	180	212	231	210	193	181	201	197	210	?

Pair-wise testing on JUNO

```
> (pairwise '(2 2 2 2 2 ))
     ((2 2 1 1 1)
      (2 1 2 2 2)
      (1 2 2 1 2)
      (1 1 1 2 1)
      (0 2 2 2 1)
      (0 1 1 1 2))
```

"0"= don't care

```
> (pair-wise '(2 2 2 2 2 2 2 2 2 2 2 2....))
```

58 outputs

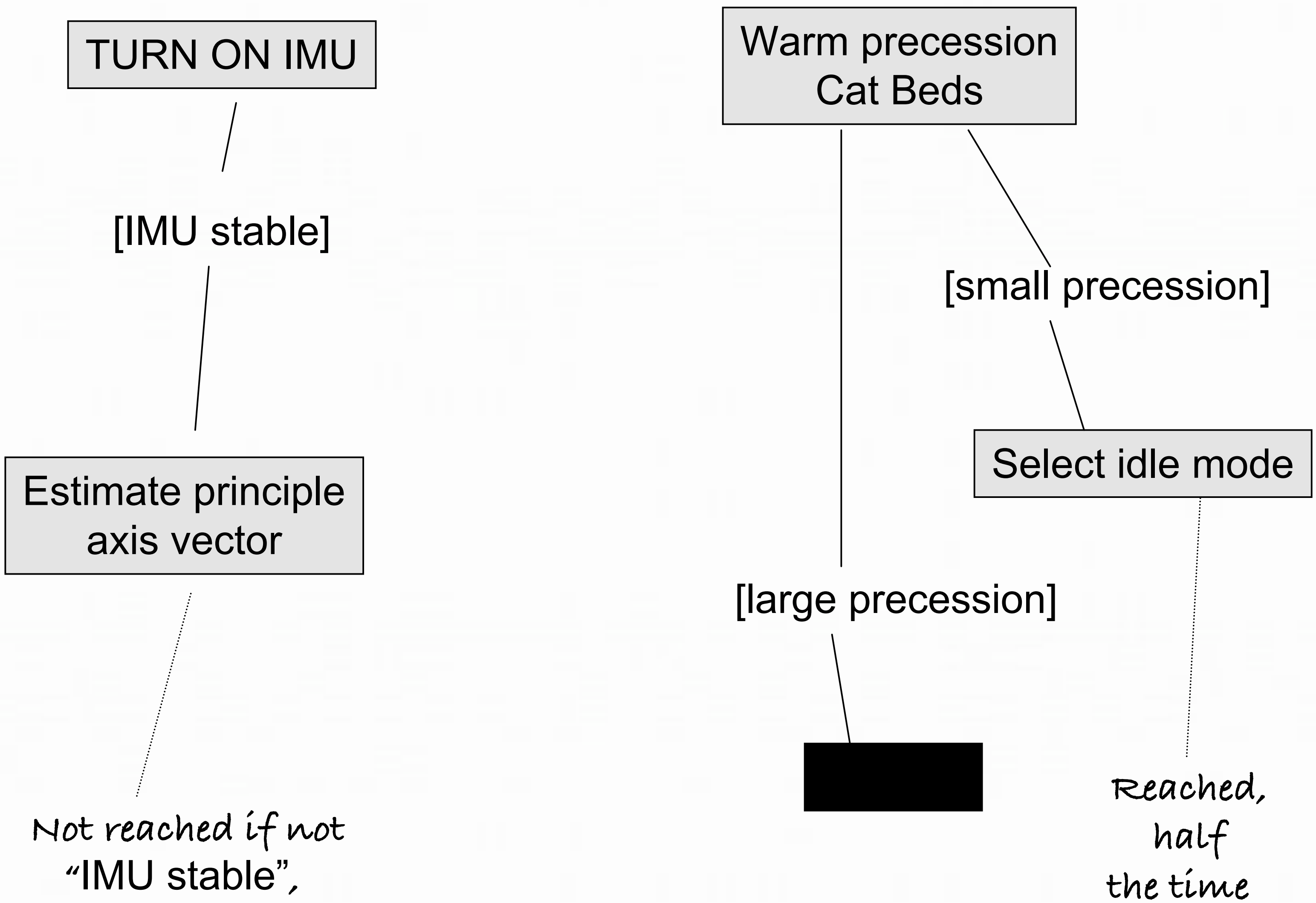
Sort by test effort: Favor "0" tests

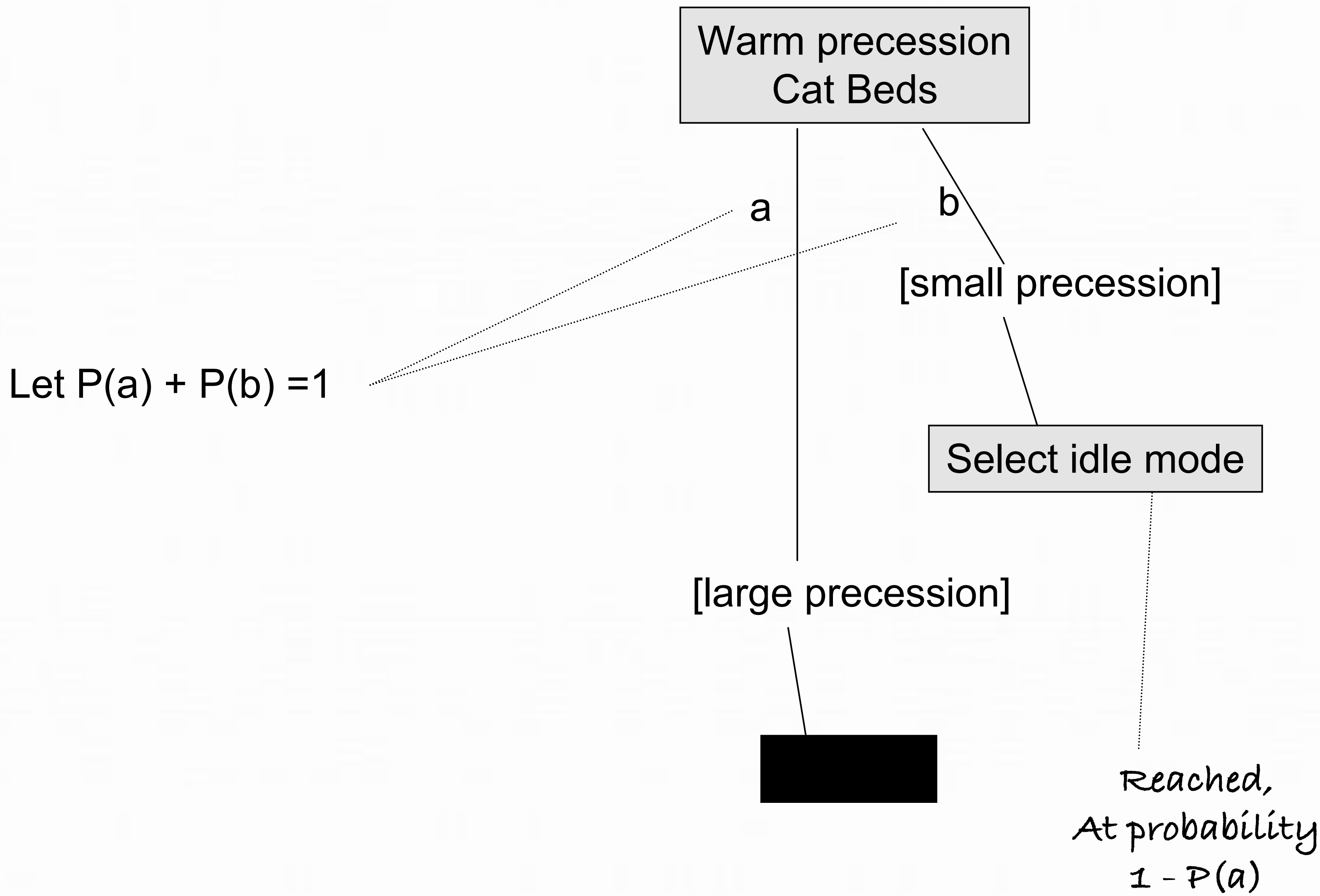
- 58 much less than 10^{80}

- Formally, these tests are equidistant samples across the space of all tests
  - Yes, there are gaps in-betweem the samples
  - But at least the space is covered

Roadmap

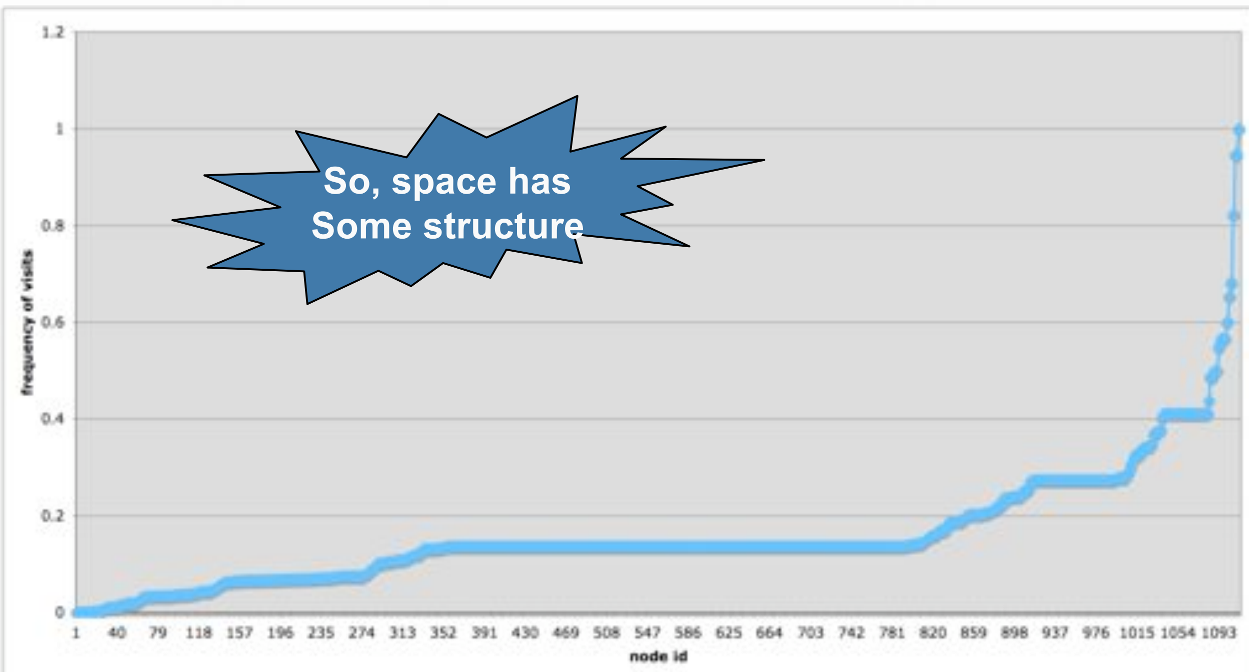
- Preliminaries: parsing the XML
- Pairwise sample
- Rank by usage
- Combining pair-wise with usage
- Divide and conqueor
- Goal-based analysis


TURN ON IMU
[IMU stable]
Estimate principle axis vector
Not reached if not "IMU stable",
Warm precession Cat Beds
[small precession]
Select idle mode
[large precession]
[redacted]
Reached, half the time


Warm precession
Cat Beds
a
b
[small precession]
Let P(a) + P(b) =1
Select idle mode
[large precession]
[redacted]
Reached,
At probability
1 - P(a)

Frequency of reaching nodes

- State charts read from NASA-built UML models
  - Dumped to XMI
  - Converted into LISP program
  - Run, making random choices



Roadmap

- Preliminaries: parsing the XML
- Pairwise sample
- Rank by usage
- Combining pair-wise with usage
- Divide and conqueor
- Goal-based analysis

Reduce and sort

- Represent space of all scenarios as combinations of gaurds
  - Reduce that space with pairwise
- Study the frequency of reaching a node
  - Random walks
- Sort the reduced space by frequency of access
  - Expected case testing: sort most-frequent first
  - Rare case testing: sort least-frequent first
- Seconday sort:
  - Least effort (favor tests with more "don't cares")

Roadmap

- Preliminaries: parsing the XML
- Pairwise sample
- Rank by usage
- Combining pair-wise with usage
- Divide and conqueor
- Goal-based analysis

Model Structure: JUNO XMI (July '08)

- Edges:1430
- Nodes: 1229
  - Guards: 267
  - Terminals (no outs): 103
  - Start nodes ("initial node 20"): 67
- Loops:
  - 291 nodes In loops size > 1
  - 35 nodes in loops size = 1 (e.g."record telemetry")
- 75 Clusters (groups of connected nodes: ignoring self-loops)
  - 75% have one initial, one final node
  - Sizes: 4 .. 81
- Note: those clusters will change as the models evolve

Repeat the above on a per-cluster basis

Roadmap

- Preliminaries: parsing the XML
- Pairwise sample
- Rank by usage
- Combining pair-wise with usage
- Divide and conqueor
- Goal-based analysis

Goal-based analysis

- Given some assertions
  - Find the combinations of gaurds that most select for
    - Most assertions satisfied
      - Auto-generate the operations manual
      - E.g. always close the door before re-entry
    - Most assertions violated
      - Disover the worst-case scenario
- Repeated result:
  - Core decisions small subset of all decisions
  - So, another scenario minimization technique:
    - just explore the “key” gaurds
- Different to model-checking:
  - Don’t just explore the space
  - Learn biases that change behaviour in the space
  - Technically: reinforcement learning

Method

Our goal is to automate assertion testing on a model view of the Juno software system (in xmi format).

Ultimately a user will be able to add and test new assertions against the software model on the fly.

XMI -> Grammar Generation -> Simulations-> Score Assertions -> Identify Key Decision Points

Example

- Rule **b** is the key decision point.
- When **b -> bc** the assertion **test1** is met.
- When **b -> ba** the assertion fails.
  - We need to quickly identify these critical junctions.
- Easy to see in small graphs
  - But in larger ones..
- Also, the minimization issue

```
(defparameter *test-graph*
  '((a -> (b c d e f g))
    (b -> ba).5
    (b -> bc).5
    (bc -> test1)1
    (ba -> (bd be))1
    (bd -> fail)1
    (be -> fail 1 )
    (test1 -> t2)1
    (t2 -> t3)1
    (t3 -> t4)1
    (t4 -> t5)1
    (t5 -> t6)1
    (t6 -> t7)1
    (t7 -> t8)1
    (t8 -> t9)1
    (t9 -> testend)1
    (testend -> goal)1
    …
))
```

Example Output

```
CL-USER> (3demo)
===== Binary Simulation=====
found 1000 egs with median 11.0 [100.%] (min=1.0 spread= 0.0 max=11.0). recommend 29 = 0
found 517 egs with median 11.0 [100.%] (min=11.0 spread= 0.0 max=11.0). recommend 27 = 1
found 237 egs with median 11.0 [100.%] (min=11.0 spread= 0.0 max=11.0). recommend 19 = 0
found 122 egs with median 11.0 [100.%] (min=11.0 spread= 0.0 max=11.0). recommend 23 = 0
found 69 egs with median 11.0 [100.%] (min=11.0 spread= 0.0 max=11.0). recommend 11 = 0
found 37 egs with median 11.0 [100.%] (min=11.0 spread= 0.0 max=11.0). recommend 17 = 0
found 19 egs with median 11.0 [100.%] (min=11.0 spread= 0.0 max=11.0). recommend 28 = 1
found 19 egs with median 11.0 [100.%] (min=11.0 spread= 0.0 max=11.0). recommend 26 = 0
found 19 egs with median 11.0 [100.%] (min=11.0 spread= 0.0 max=11.0). recommend 1 = 0
found 7 egs with median 11.0 [100.%] (min=11.0 spread= 0.0 max=11.0). recommend 15 = 1
```

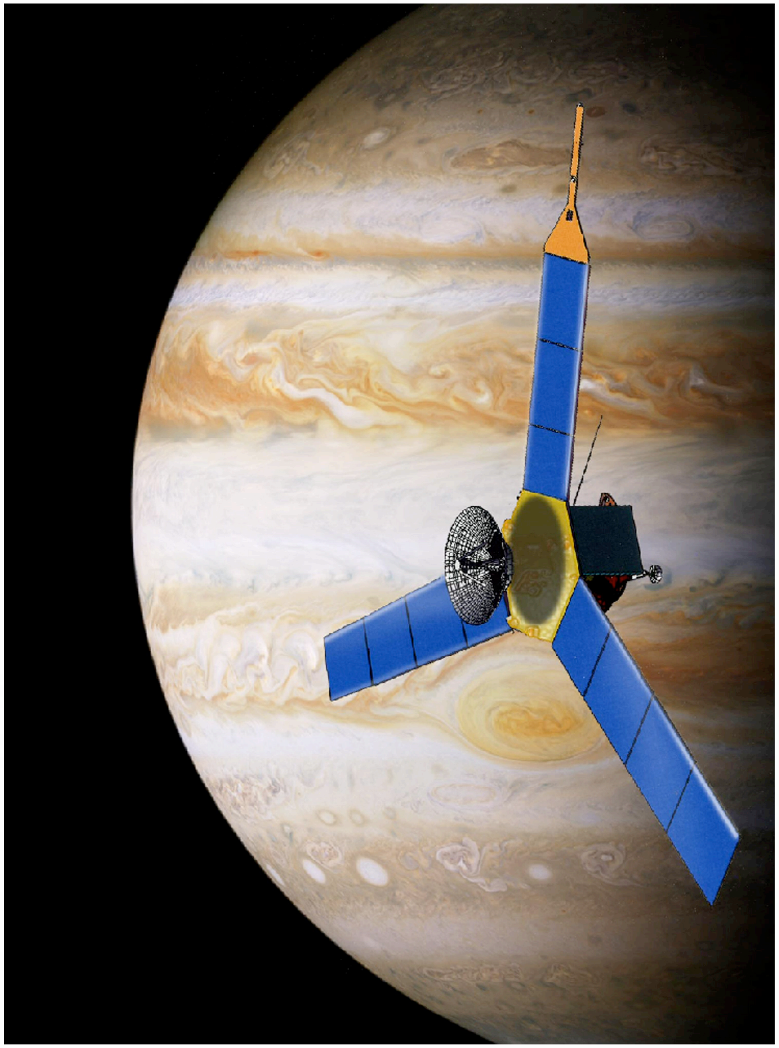
Of 1000 simluations, the test assertions were met when rule 29 (b -> ba) was set to 0.

Praticalities

- Some issues with accessing the right kind of assertions
  - Matching assertions to nodes in the XMI
  - Find assertions that use qualitative time
- Some issues with scale-up
  - Perhaps just implementation trivia
  - Solvable: using the clusters

Goal: Better Support for UML modeling in Juno

- New generation IV&V
  - UML-based
  - Create a UML reference model
  - Conduct model-based IV&V
- Can we improve that work?
  - Simulate software activity.
  - Identify key decision points.
  - Quick feedback.
  - Scalable for large systems.
  - Possible reuse with other models.
  - Minimize work required to assess the mode



Proposed support for UML

- Pairwise sample
- Rank by usage
- Combining pair-wise with usage
- Divide and conqueor
- Goal-based analysis