Analysis and Testing of PLEXIL Plans

Jason Biatek, Michael W. Whalen, Sanjai Rayadurgam, Mats P.E. Heimdahl, Michael Lowry

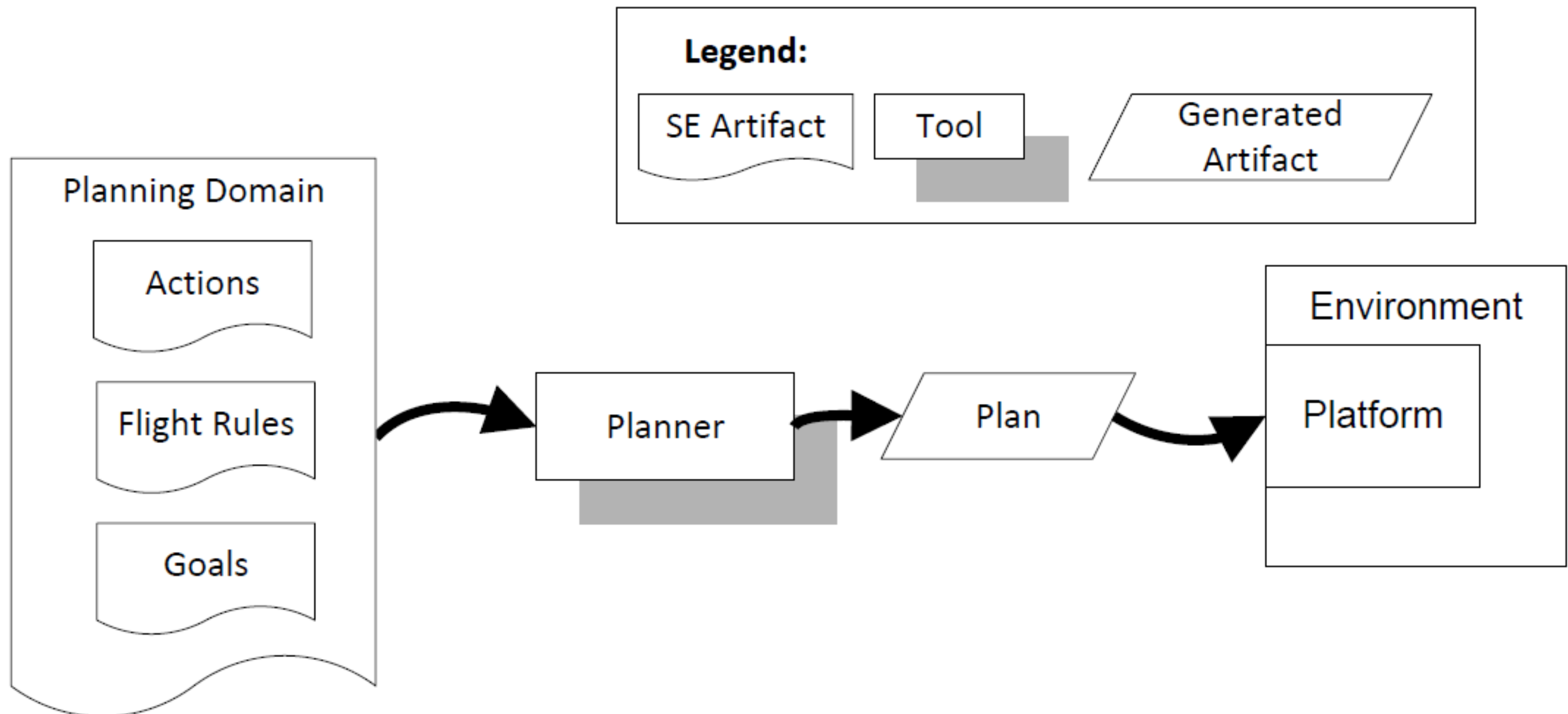
Autonomy at NASA

National Aeronautics and Space Administration

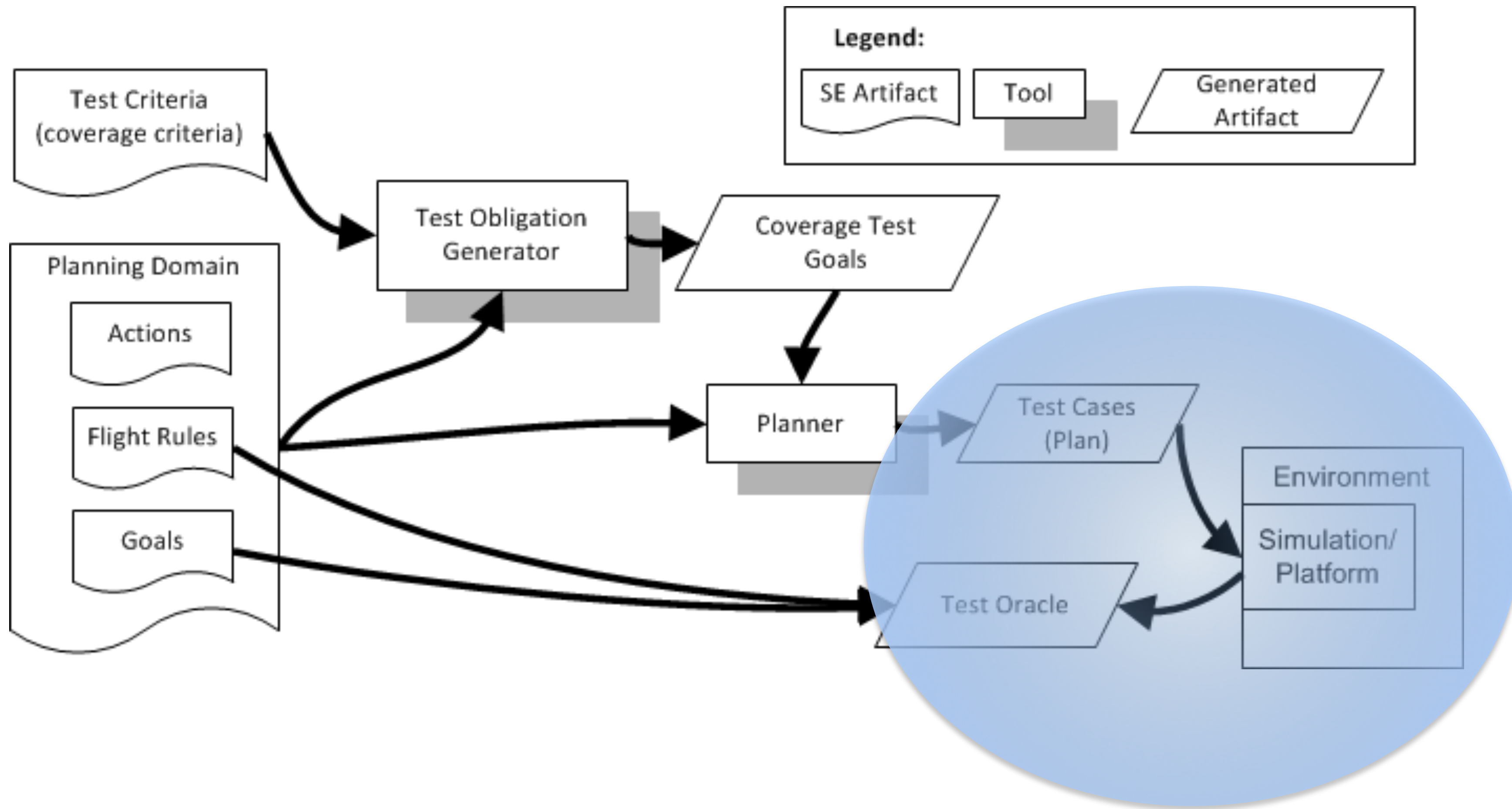
DRAFT Robotics, Tele-Robotics and Autonomous Systems Roadmap

Technology Area 04

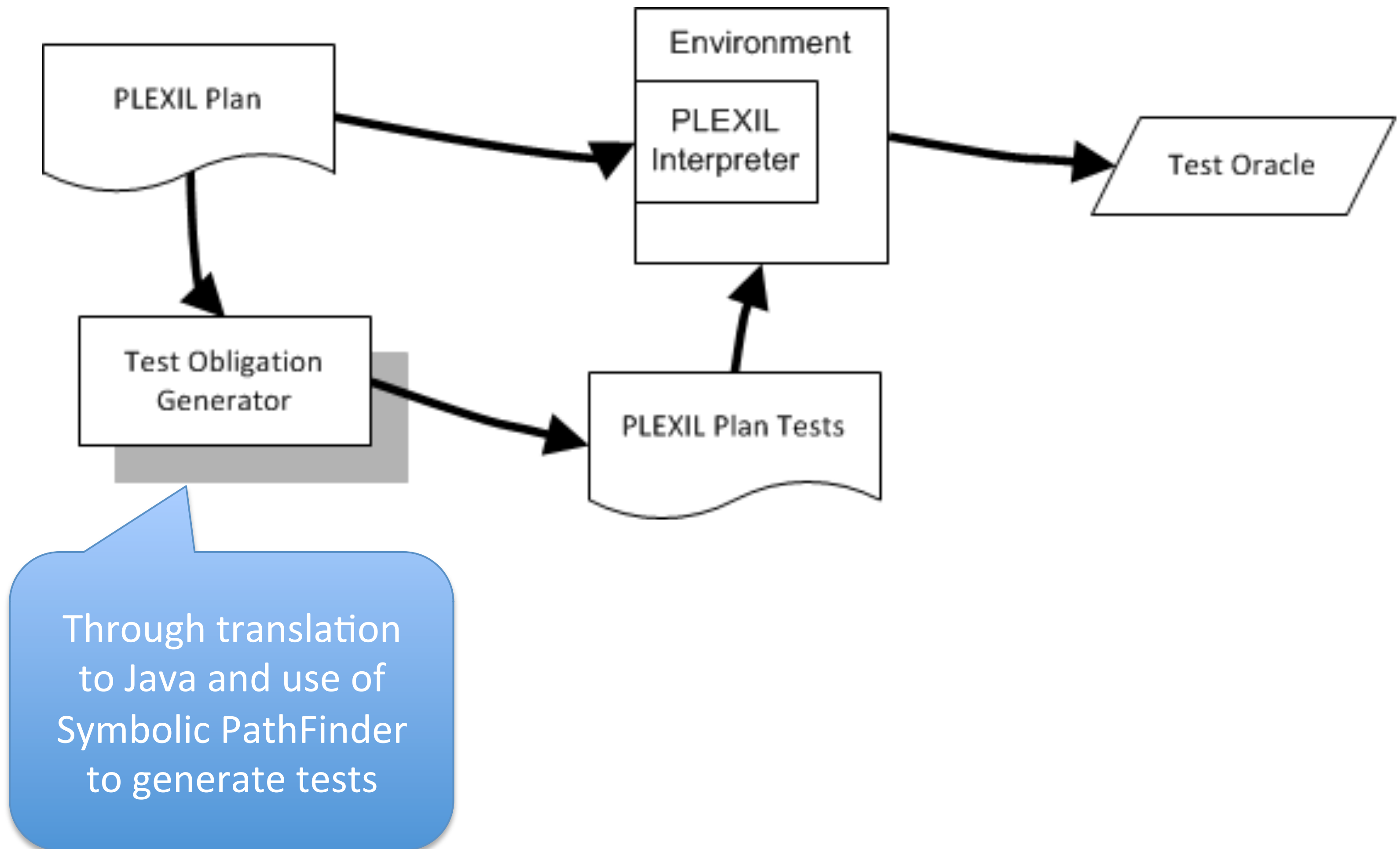
Autonomy Requires Planning



Verification of Planning Systems



Verification of Plan Execution (PLEXIL)

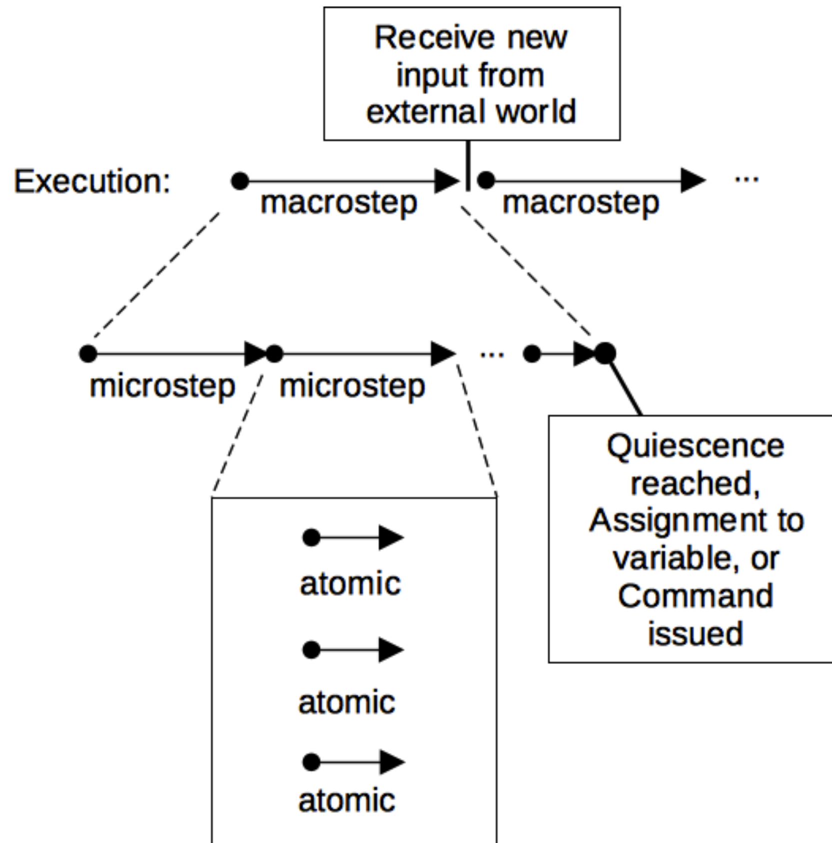


PLEXIL by Example

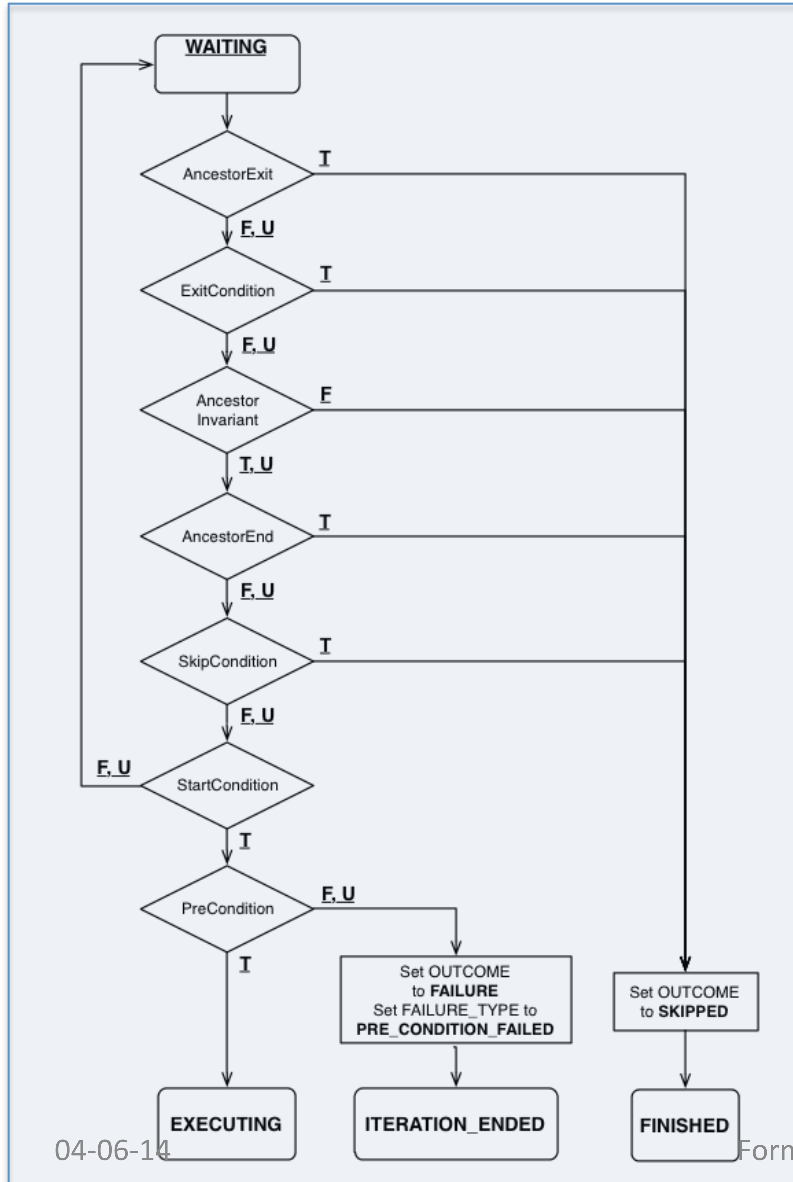
```
SafeDrive: {
  Integer pictures = 0;
  EndCondition
    Lookup(WheelStuck) || pictures
== 10;
  while (! Lookup(WheelStuck))
   Sequence
   {
     OneMeter: { Drive(1); }
     TakePic: {
       StartCondition pictures <
10;
       TakePicture();
     }
     Counter: {
       PreCondition pictures < 10;
       pictures = pictures + 1;
     }
     Print: { print (
     "Pictures taken:",
```

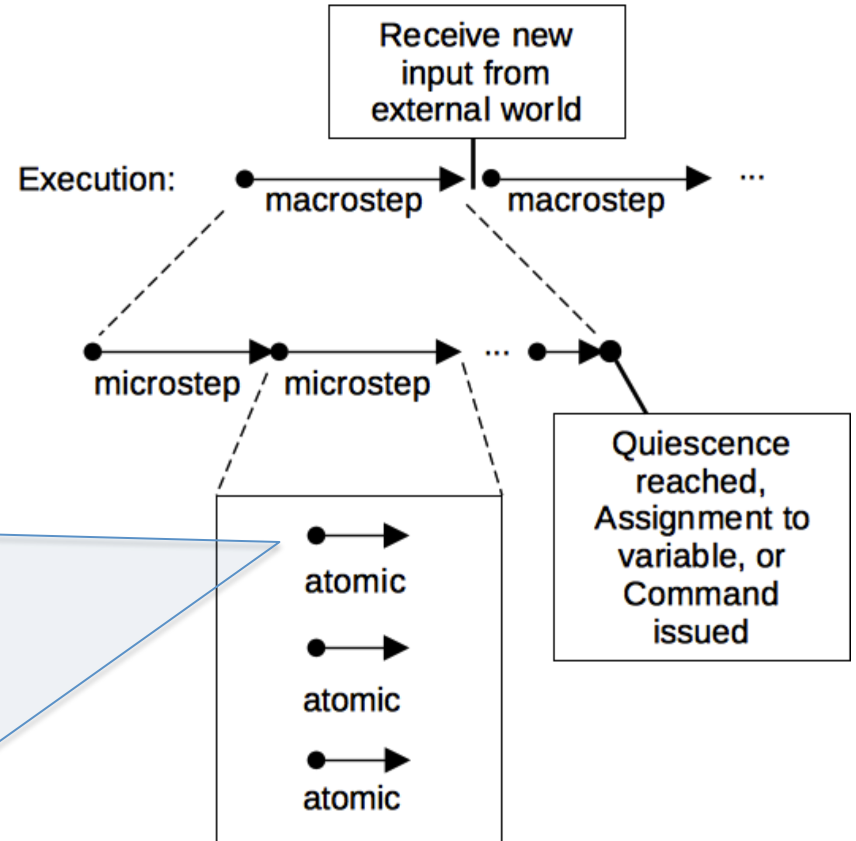
- Plan consists of *nodes*
  - Arranged in a hierarchy
- Node behavior described by a state machine
  - Nodes progress through states like INACTIVE, WAITING, EXECUTING, FAILING, FINISHED
- Transitions between states are guarded by conditions, such as Start, End, Invariant
- *Basic PLEXIL* contains six node types
- *Extended PLEXIL* contains syntactic sugar; 4 additional constructs

PLEXIL semantics



PLEXIL semantics

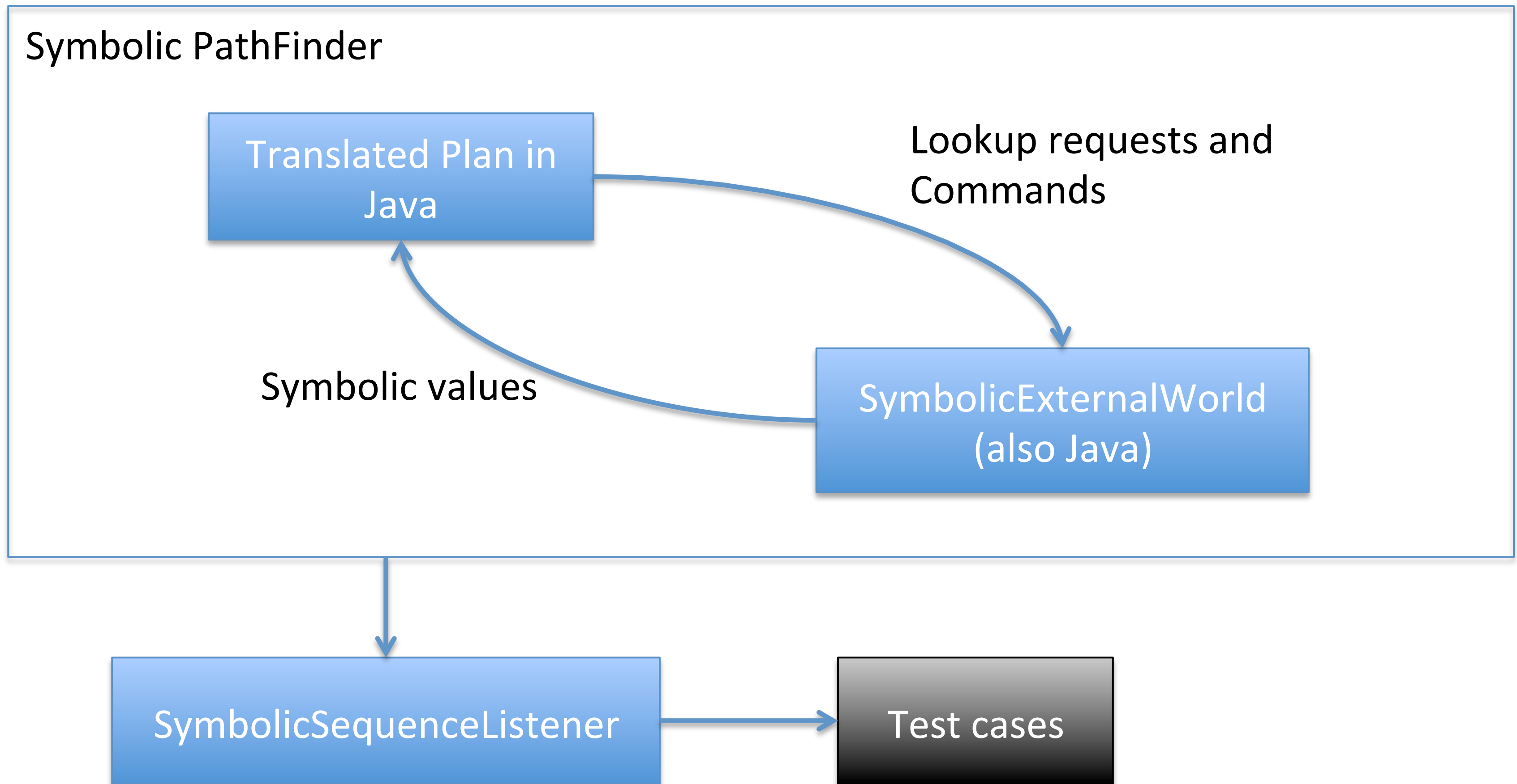




Formalizing PLEXIL Semantics

- PLEXIL has formal semantics
  - Specified in PVS [Munoz and Pasareanu]
  - Specified in Maude [Dowek, Munoz, and Rocha]
- Extensive work on checking formal definitions
  - Completeness and determinism [Munoz]
  - Atomicity and Termination [Dowek]
- Maude can use semantics to generate a model checker
  - Has been used on small plans

PLEXIL, Java, and SPF

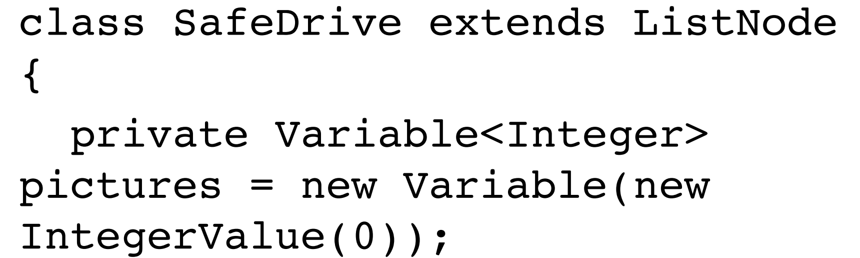


Why Translate to Java?

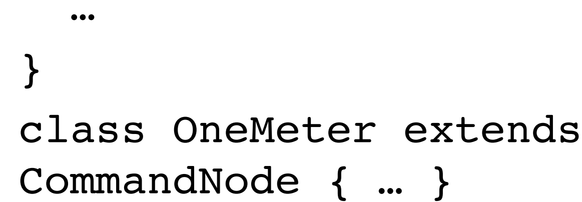
- Variety of Java analysis / TCG tools
  - Java SPF, jCute, EvoSuite, Randoop
- Easy to integrate with environmental models
  - These models are *already* written in Java
  - Can do mixed mode concrete/symbolic execution
- Provides an execution engine
  - Current NASA PLEXIL executive (not our stuff) is an interpreter
  - Could retarget translation to C for fast execution
  - Optimizations for analysis also improve code performance
- Sponsor required it ☺
  - NASA wants to demonstrate capability in JPF/SPF

Translation approach

```
SafeDrive: {
  Integer pictures = 0;
  EndCondition
    Lookup(WheelStuck) ||
pictures == 10;
  while (! Lookup(WheelStuck))
   Sequence
   {
     OneMeter: { Drive(1); }
     TakePic: {
       StartCondition pictures <
10;
       TakePicture();
     }
     Counter: {
       PreCondition pictures
10;
       pictures = pictures + 1;
     }
     Print: { print (
```

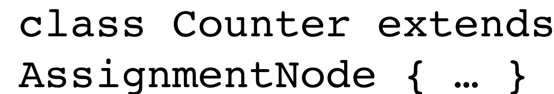


```
class SafeDrive extends ListNode
{
  private Variable<Integer>
pictures = new Variable(new
IntegerValue(0));
  …
}
```



```
class OneMeter extends
CommandNode { … }
```

```
class TakePic extends CommandNode
{ … }
```



```
class Counter extends
AssignmentNode { … }

class Print extends CommandNode
{ …
}
```

Why Generate Tests?

- We can also do symbolic execution
  - Symbolic exploration of paths
  - Equivalent to proof if all loops are a-priori bounded
- But, in its basic form, it doesn't scale
  - Loops are *not* bounded in real-time systems
  - Many paths generate similar tests.
- Test coverage metrics provide *search pruning*
  - Can provide guidance to "interesting" paths
  - Makes analysis incomplete
- Sponsor required it ☺
  - NASA wants to demonstrate capability in JPF/SPF

PLEXIL Language Analysis

- While analyzing and generating tests, we discovered 2 problems in the PLEXIL language
- In Extended PLEXIL:
  - "If-Then-Else" construct did not handle the UNKNOWN case properly, causing the entire node to become unresponsive
- In Basic PLEXIL:
  - a missing transition caused nodes to execute out of sequence
  - Running a test case in the PLEXIL reference executive put the plan into an unexpected state, causing it to crash
- Both issues were sent to the PLEXIL team, who were able to fix both issues

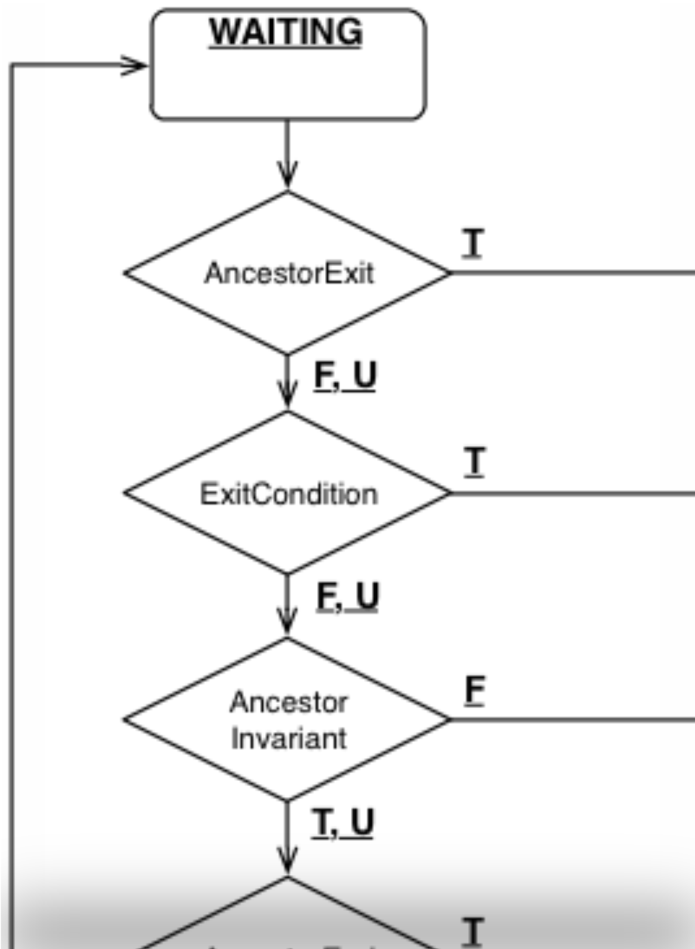
A Note on Formalization

- Plexil has formal semantics
  - Specified in PVS [Munoz and Pasareanu]
  - Specified in Maude [Dowek, Munoz, and Rocha]
- Extensive work on checking formal definitions
  - Completeness and determinism [Munoz]
  - Atomicity and Termination [Dowek]
- **Why were these issues not caught?**
- **Formal ≠ correct**
- Verifying optimizations requires re-examination of definitions
- Testing has benefit of **serendipity**

Optimizations

- Singleton 3-valued logic objects
- Constant propagation and removal of impossible transitions (code specialization)
- Dead variable removal
  - Each node must store the “start” and “end” time for each of the 7 PLEXIL node states, but these can be eliminated if the value is never read
- Lazy evaluation
  - Children of INACTIVE nodes are, by design, also INACTIVE
  - The step function for these children can be skipped entirely because it is guaranteed that the child will simply remain in INACTIVE with no side effects

Optimizations – UNKNOWN biasing



- The first transition depends on the ancestor's exit condition, but only whether it is "true" or "not true"
- Similarly, the ancestor's invariant depends on whether or not it is false

Optimizations – UNKNOWN biasing

- This biasing can be pushed down into the leaves of the expression, allowing native Java Booleans to be used. For example:
  – PLEXIL:
    - alpha && (beta || gamma) && !delta == true
  – Unoptimized Java:
    - alpha.and(beta.or(gamma)).and(delta.not()).isTrue()
  – Optimized Java:
    - alpha.isTrue() && (beta.isTrue() || gamma.isTrue()) && delta.isFalse()

Experimental Results

- Test case generation performance of:
  - The original, naïve translator
  - The IL-based translator, which also includes:
    - 3-valued logic singletons
    - UNKNOWN biasing
    - Skipping of INACTIVE children
    - Dead variable removal
- The plans used were:
  - Example PLEXIL plans distributed with PLEXIL
  - "Fluid", which describes part of the ISS and is much larger

Experimental Results

	CruiseControl	DriveToSchool	SafeDrive	SimpleDrive
SPF Depth	20	20	25	30
Naïve time	6:41	TO	17:13	19:32
IL time	2:32	13:35	5:12	14:11
Naïve tests	10788	TO	8191	24575
IL tests	10572	2715	8191	24575

TO: timed out (more than 20 minutes)

Experimental Results

Fluid model

SPF Depth	5	10	15
Naïve time	0:55	3:57	TO
IL time	0:47	15:41	TO
IL* time	0:47	5:13	TO
Naïve tests	5	28	TO
IL tests	4	22	TO
IL* tests	5	26	TO

TO: timed out (more than 20 minutes)
IL*: IL without code specialization

Experimental Results

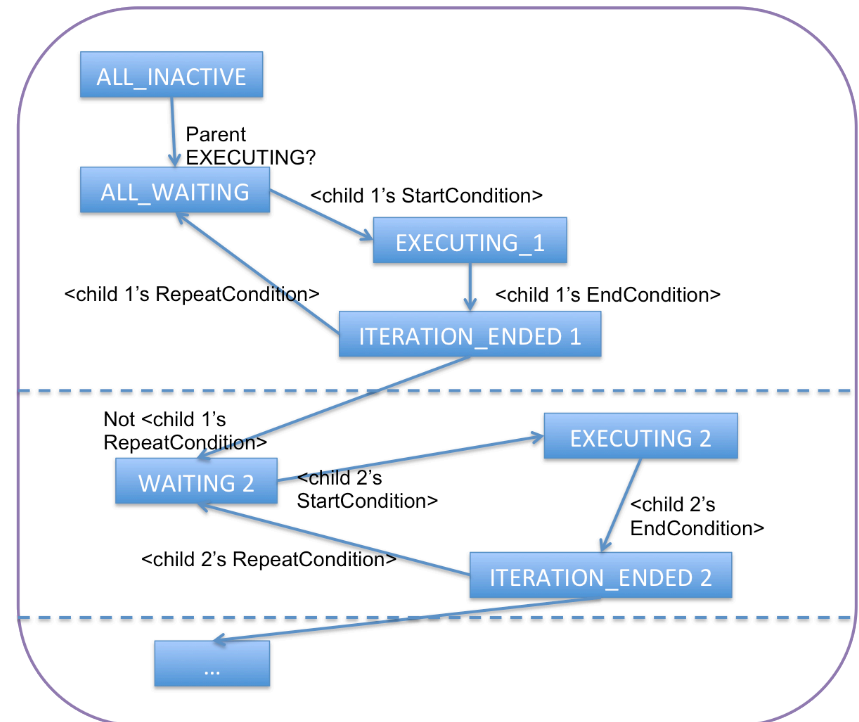
- Unlike the smaller examples, the large Fluid model takes longer to analyze with the new architecture
- We discovered that code specialization (removal of impossible transitions) somehow causes test generation time to increase
- Even with specialization disabled, the naïve version outperforms the IL one
- We are working with the JPF/SPF team to diagnose these issues

Current Work: PLEXIL Intermediate Language

- PLEXIL is a rich language
  - 6 different node types
  - Each have different side effects,
  - Each have slightly different state machines
- For more extensive optimization, such as rearranging and combining nodes, need an intermediate language
- An IL plan consists of:
  - A flat list of all variables
  - A flat list of (universal) state machines, where states and transitions can also include Actions (perform assignment, issue command, etc.)
- States must have a mapping back to PLEXIL's native states (INACTIVE, WAITING, etc.) because this information can be used in expressions

Next steps

- Optimizations (with IL)
  - Sequence merging
    - Combine parallel nodes used in sequence into single state machine.
  - Path compression
    - Combine microsteps when sequence does not matter.
- Analysis of generated tests, including coverage
  - What is a meaningful coverage metric?
- Testing mission code: LADDEE



Thank you!

Ευχαριστώ
Merci
Grazie
Díky
Gracias
Vielen Dank
Obrigado!
Teşekkürler
شكراً
धन्यवाद