

A Formal Library of Set Relations and Its Application to Synchronous Languages

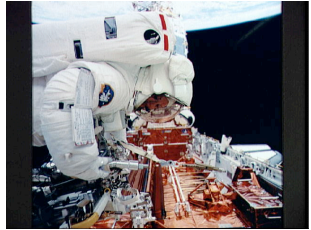
César Muñoz
NASA Langley Research Center

joint work with
Gilles Dowek, École Polytechnique, and
Camilo Rocha, University of Illinois at Urbana-Champaign

University of Paris 6, March 8, 2010



NASA's Autonomy for Operations (A4O)



*NASA's A4O project develops trusted adjustable automation technology to advance the state of the art in **mission operations**, **crew self-scheduling**, **robotic operations**, and **systems operations** for lunar outpost and surface infrastructure operations.*

Current Spacecraft Operations

Highly Simplified View

- ▶ Spacecraft operations involve ground and space crew, tasks, resources, and constraints on these elements.
- ▶ Constraints are solved by experts who write plans.
- ▶ Plans become procedures (detailed instructions for operating equipment).
- ▶ Procedure are manually executed and monitored by astronauts.

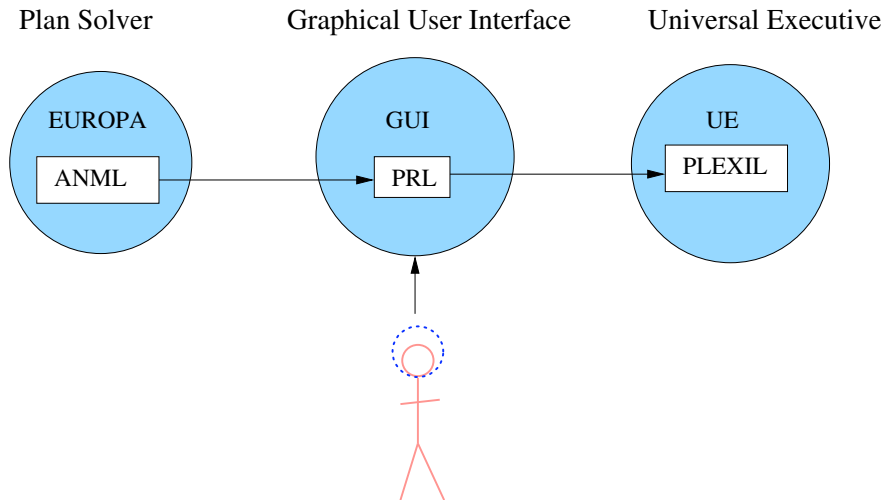
- ▶ Plans and procedures usually have static representations.
- ▶ Software is used to display and simulate plans.

Flight Control Room for ISS



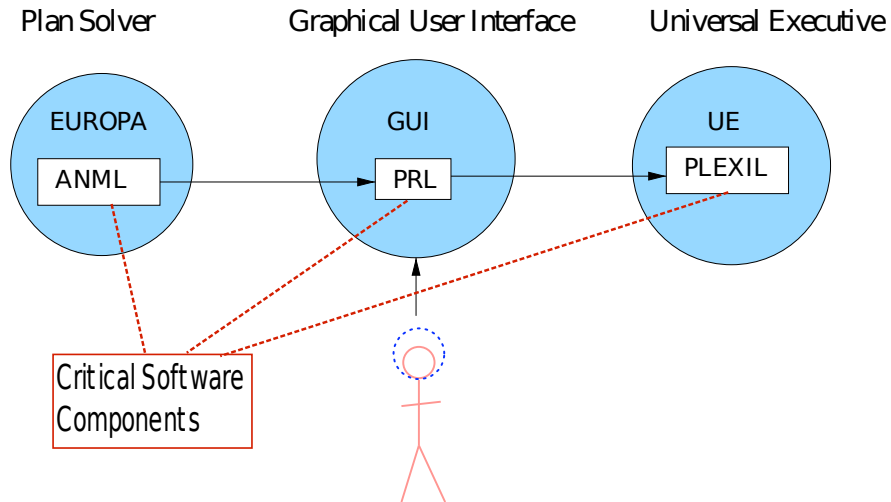
Future Spacecraft Operations

Highly Idealized View



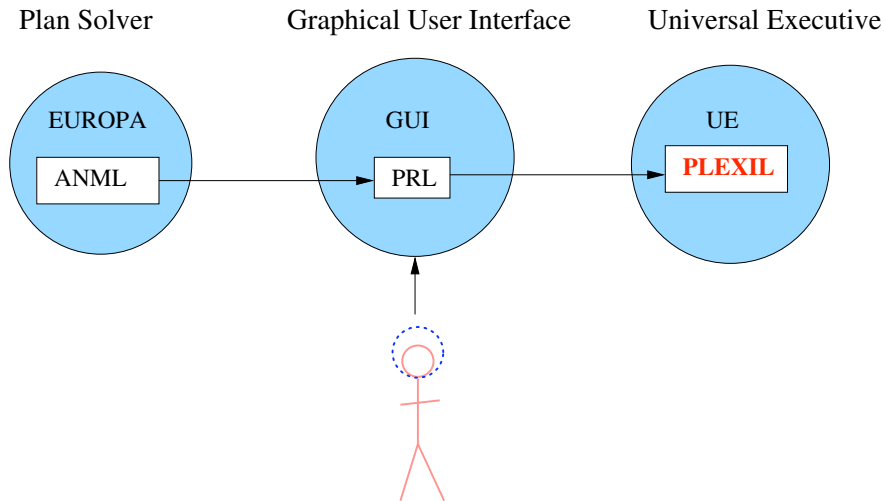
Future Spacecraft Operations

Highly Idealized View



Future Spacecraft Operations

Highly Idealized View



Plan Execution Tasks

- ▶ Track completed and uncompleted tasks
- ▶ Determine if tasks are ready to start
- ▶ Know who is performing a task
- ▶ Determine if a task can be skipped, and why
- ▶ Determine if a task has constraints
- ▶ Track when tasks complete
- ▶ Understand and handle task failure
- ▶ Evaluate state of procedure and plan

PLan EXecution Interchange Language (PLEXIL)

PLEXIL is ...

- ▶ a reactive, synchronous, and deterministic language for ...
- ▶ commanding and monitoring space systems.

PLEXIL is open source: <http://plexil.sourceforge.net>

PLEXIL

- ▶ A PLEXIL program (**plan**) is a tree of **nodes** that communicate via shared variables and interact with the external environment via lookups and commands.
- ▶ Each node consists of
 - ▶ A list of **variables**: The scope of these variables is the node's descendants.
 - ▶ A set of **conditions**: Start, End, Repeat, Skip, Pre, Post, and Invariant.
 - ▶ An **execution state**: Inactive, Waiting, Executing, Finishing, or Finished.
 - ▶ An **outcome**: Success or Failure.
 - ▶ A **priority** for solving simultaneous writing access to variables.
 - ▶ A **body**: List, Assignment, or Command.

Execution

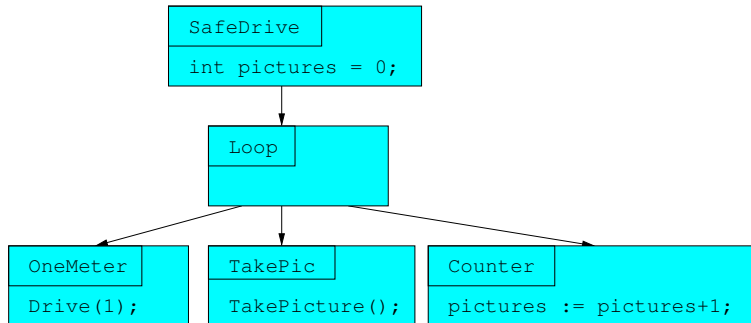
PLEXIL execution is event-driven.

- ▶ At the initial state the root node goes from state `Inactive` to `Waiting` and it's eligible for execution.
- ▶ At state `Waiting`, a change in `Start`, `End`, `Repeat` conditions triggers a series of **atomic** changes in the execution states of the nodes that eventually lead to the execution of assignments and commands.
- ▶ These atomic changes are executed **synchronously** until **quiescence**.
- ▶ When quiescence has been achieved the executive checks for external events a new **execution** cycle starts.

Simple Rover Plan

```
List SafeDrive {
  int pictures = 0;
  End: LookupOnChange(WheelStuck) == true OR pictures == 10;
  List Loop {
    Repeat-while: LookupOnChange(WheelStuck) == false;
    Command OneMeter {
      Drive(1);
    }
    Command TakePic {
      Start: OneMeter.status == FINISHED AND pictures < 10;
      TakePicture();
    }
    Assignment Counter {
      Start: TakePic.status == FINISHED;
      Pre: pictures < 10;
      pictures := pictures + 1;
    }
  }
}
```

SafeDrive



What does this program do ?

```
List Exchange {  
  int x = 0;  
  int y = 1;  
  Assignment X { x:= y; }  
  Assignment Y { y:= x; }  
}
```

...and this one ?

```
Assignment SimplePost {  
  int x = 0;  
  x:= 10;  
  Post: x == 10;  
}
```

...and this ?

```
Assignment Loop {  
  int x = 0;  
  Repeat-while: x < 10;  
  x := x+1;  
}
```

Universal Executive Implementation

- ▶ List Exchange {

```
int x = 0;
```

```
int y = 1;
```

```
Assignment X { x:= y; }
```

```
Assignment Y { y:= x; }
```

```
}
```

Values of x and y are exchanged.

- ▶ Assignment SimplePost {

```
}
```

Post-condition failure.

- ▶ Assignment Loop {

```
int x = 0;
```

```
Repeat-while: x < 10;
```

```
x := x+1;
```

```
}
```

Infinite loop.

Universal Executive Implementation

▶ `List Exchange {`

`...`

`}`

Values of `x` and `y` are exchanged.

▶ `Assignment SimplePost {`

`int x = 0;`

`x := 10;`

`Post: x == 10;`

`}`

Post-condition failure.

▶ `Assignment Loop {`

`int x = 0;`

`Repeat-while: x < 10;`

`x := x+1;`

`}`

Infinite loop.

Universal Executive Implementation

▶ `List Exchange` {

...

}

Values of `x` and `y` are exchanged.

▶ `Assignment SimplePost` {

...

}

Post-condition failure.

▶ `Assignment Loop` {

`int x = 0;`

`Repeat-while: x < 10;`

`x := x+1;`

}

Infinite loop.

PLEXIL Semantics

- ▶ Defined using a stack of 5 different relations: *execution*, *macro*, *quiescence*, *micro*, and *atomic*.
- ▶ The atomic relation consists of 42 (non-trivial) rules.
- ▶ PLEXIL is a *live* language.

Objective

Provide an operational semantics of PLEXIL that is:

- ▶ **formal**, i.e., suitable for machine verifiable proofs,
- ▶ **modular**, i.e., that supports different semantic variants of the language, and
- ▶ **executable**, i.e., that allows for the validation of the executive, in order to study meta-theoretical properties of the language.

Language Properties

- ▶ **Determinism**: The only non-determinism that is allowed originates from external events, e.g., under known external conditions the behavior of a plan can be predicted.
- ▶ **Compositionality**: Under known external conditions, the behavior of plans executed in parallel can be inferred from the independent execution of each one of them.

In Mathematics, Proofs Are Long But Definitions Are Short

- ▶ In computer science definitions are long and proofs are
- ▶ However, formal semantics are built with a limited number of **mathematical constructions** and the **properties of interest** are often the same.

In Mathematics, Proofs Are Long But Definitions Are Short

- ▶ In computer science definitions are long and proofs are longer.
- ▶ However, formal semantics are built with a limited number of mathematical constructions and the properties of interest are often the same.

A Formal Library of Set Relations . . .

- ▶ A set of well-known **relations**.
- ▶ A set of **properties** on these relations, e.g., **determinism**, **compositionality**.
- ▶ A set **of theorems** that specifies under which conditions these properties hold.

Formalized in PVS !

... For Synchronous Languages

- ▶ A **serialization procedure** for the **execution** of synchronous relations using a sequential machine.
- ▶ The procedure is **correct** and, for deterministic relations, **complete**.

Implemented in Maude !

Relations

Let \longrightarrow be a binary relation defined on a set U .

- ▶ An element $a \in U$ is **reducible** if $a \longrightarrow a'$. It is a **normal form** if $a \not\longrightarrow a'$.
- ▶ Relations \longrightarrow^n and \longrightarrow^* are defined as usual.
- ▶ The **normalized extension** of \longrightarrow is the relation $\longrightarrow_\downarrow$ defined such that $a \longrightarrow_\downarrow a'$ iff $a \longrightarrow^* a'$ and a' is normal.

Asynchronous Relation

Let \longrightarrow be a binary relation defined on $U \subseteq \mathcal{P}(T)$.

- ▶ The **asynchronous extension** of \longrightarrow is the relation $\overset{\square}{\longrightarrow}$ defined such that $a \overset{\square}{\longrightarrow} a'$ if there exists sets r and r' such that $r \subseteq a$, $r \neq \emptyset$, $r \longrightarrow r'$ and $a' = (a \setminus r) \cup r'$.

- ▶ Example: Assume that

1. $A, B \longrightarrow B, D$.
2. $C \longrightarrow D$.

Therefore,

- ▶ $A, B, C, E \overset{\square}{\longrightarrow} B, C, D, E$, reducing (1).
- ▶ $A, B, C, E \overset{\square}{\longrightarrow} A, B, D, E$, reducing (2).

Parallel Relation

Let \longrightarrow be a binary relation defined on $U \subseteq \mathcal{P}(T)$.

- ▶ The **parallel extension** of \longrightarrow is the relation $\overset{\parallel}{\longrightarrow}$ defined such that $a \overset{\parallel}{\longrightarrow} a'$ iff there exists r_1, \dots, r_n and r'_1, \dots, r'_n such that $r_i \subseteq a$, $r_i \neq \emptyset$, $i \neq j \Rightarrow r_i \cap r_j = \emptyset$, $r_i \longrightarrow r'_i$ and $a' = (a \setminus \bigcup_i r_i) \cup \bigcup_i r'_i$.

- ▶ Example: Assume that

1. $A, B \longrightarrow B, D$.
2. $C \longrightarrow D$.
3. $A, E \longrightarrow D$.

Therefore,

- ▶ $A, B, C, E \overset{\parallel}{\longrightarrow} B, D, E$, reducing (1) and (2).
- ▶ $A, B, C, E \overset{\parallel}{\longrightarrow} B, D$, reducing (2) and (3).

Strategies

Let \longrightarrow be a binary relation defined on $U \subseteq \mathcal{P}(T)$.

- ▶ A **strategy** for \longrightarrow is a function $s(a) = \{r_1, \dots, r_n\}$ such that $r_i \subseteq a$, $r_i \neq \emptyset$, $i \neq j \Rightarrow r_i \cap r_j = \emptyset$ and r_i is reducible for \longrightarrow .
- ▶ Example: Assume that
 1. $A, B \longrightarrow B, D$.
 2. $C \longrightarrow D$.
 3. $A, E \longrightarrow D$.

The following functions are strategies for \longrightarrow .

- ▶ $s1(\{A, B, C, E\}) = \{\{A, B\}, \{C\}\}$.
- ▶ $s2(\{A, B, C, E\}) = \{\{A, E\}, \{C\}\}$.

Synchronous Relation

Let \longrightarrow be a binary relation defined on $U \subseteq \mathcal{P}(T)$ and s be a strategy for \longrightarrow .

- ▶ The **synchronous extension** or $\xrightarrow{|s|}$ is the relation $\xrightarrow{|s|}$ defined such that $a \xrightarrow{|s|} a'$ iff there exists r'_1, \dots, r'_n such that $s(a) = \{r_1, \dots, r_n\}$, $r_i \longrightarrow r'_i$ and $a' = (a \setminus \bigcup_i r_i) \cup \bigcup_i r'_i$.
- ▶ Example: Assume that
 1. $A, B \longrightarrow B, D$.
 2. $C \longrightarrow D$.
 3. $A, E \longrightarrow D$.

Therefore,

- ▶ $A, B, C, E \xrightarrow{|s1|} B, D, E$, where $s1(\{A, B, C, E\}) = \{\{A, B\}, \{C\}\}$.
- ▶ $A, B, C, E \xrightarrow{|s2|} B, D$, where $s2(\{A, B, C, E\}) = \{\{A, E\}, \{C\}\}$.

Determinism

A binary relation \longrightarrow is **deterministic**, iff

▶ $a \longrightarrow b'$ and

▶ $a \longrightarrow b''$,

implies $b' = b''$.

Determinism

Let \longrightarrow be a binary relation defined on $U \subseteq \mathcal{P}(T)$ such that \longrightarrow is deterministic.

Theorem:

- ▶ \longrightarrow^n is deterministic.
- ▶ $\longrightarrow_\downarrow$ is deterministic.
- ▶ $\xrightarrow{|s|}$ is deterministic.

Remark: \longrightarrow^* , $\xrightarrow{\square}$, and $\xrightarrow{\parallel}$ are not, in general, deterministic.

Compositionality

Let \longrightarrow be a binary relation defined on $U \subseteq \mathcal{P}(T)$ and A a set of pairs of elements of $U \subseteq \mathcal{P}(T)$.

- ▶ \longrightarrow is **compositional on A** iff for all $\langle a, b \rangle \in A$, $a \longrightarrow a'$, and $b \longrightarrow b'$, implies $a \cup b \longrightarrow a' \cup b'$.
- ▶ \longrightarrow is **strongly compositional on A** iff for all $\langle a, b \rangle \in A$,
 - ▶ $a \longrightarrow a'$, and $b \longrightarrow b'$, implies $a \cup b \longrightarrow a' \cup b'$,
 - ▶ $a \longrightarrow a'$, and b normal form, implies $a \cup b \longrightarrow a' \cup b$, and
 - ▶ a normal form, and $b \longrightarrow b'$, implies $a \cup b \longrightarrow a \cup b'$.

Compositionality

Let \longrightarrow be a binary relation defined on $U \subseteq \mathcal{P}(T)$ and A a set of pairs of elements of $U \subseteq \mathcal{P}(T)$.

- ▶ \longrightarrow is **compositional on A** iff for all $\langle a, b \rangle \in A$, $a \longrightarrow a'$, and $b \longrightarrow b'$, implies $a \cup b \longrightarrow a' \cup b'$.
- ▶ \longrightarrow is **strongly compositional on A** iff for all $\langle a, b \rangle \in A$,
 - ▶ $a \longrightarrow a'$, and $b \longrightarrow b'$, implies $a \cup b \longrightarrow a' \cup b'$,
 - ▶ $a \longrightarrow a'$, and b normal form, implies $a \cup b \longrightarrow a' \cup b$, and
 - ▶ a normal form, and $b \longrightarrow b'$, implies $a \cup b \longrightarrow a \cup b'$.

Compositionality: Synchronous Relation

Let \longrightarrow be a binary relation defined on $U \subseteq \mathcal{P}(T)$ and A a set of pairs of elements of $U \subseteq \mathcal{P}(T)$ such that \longrightarrow is compositional on A .

- ▶ **Theorem:** If A is a *set of disjoint pairs* and s is a strategy for \longrightarrow that *commutes with union on A* , then $\xrightarrow{|s|}$ is **strongly compositional on A** .
- ▶ A is a set of **disjoint pairs** iff for all $\langle a, b \rangle \in A$, $a \cap b = \emptyset$.
- ▶ s **commutes with union** on A iff for all $\langle a, b \rangle \in A$, $s(a \cup b) = s(a) \cup s(b)$.

Compositionality: Normalized Relation

Let \longrightarrow be a binary relation defined on $U \subseteq \mathcal{P}(T)$ and A a set of pairs of elements of $U \subseteq \mathcal{P}(T)$ such that \longrightarrow is *strongly* compositional on A .

- ▶ **Theorem:** If A is *closed by* \longrightarrow , then $\longrightarrow_{\downarrow}$ is **compositional on A** .
- ▶ A is **closed by** \longrightarrow iff for all $\langle a, b \rangle \in A$,
 - ▶ $a \longrightarrow a'$ implies $\langle a', b \rangle \in A$, and
 - ▶ $b \longrightarrow b'$ implies $\langle a, b' \rangle \in A$.

Compositionality: n -iteration

Let \longrightarrow be a binary relation defined on $U \subseteq \mathcal{P}(T)$ and A a set of pairs of elements of $U \subseteq \mathcal{P}(T)$ such that \longrightarrow is compositional on A .

- ▶ **Theorem:** If A is *weakly-closed* by \longrightarrow , then \longrightarrow^n is *compositional on A* .
- ▶ A is *weakly-closed* by \longrightarrow iff for all $\langle a, b \rangle \in A$,
 $a \longrightarrow a'$ and $b \longrightarrow b'$ implies $\langle a', b' \rangle \in A$.

Serialization Procedure

- ▶ We propose a procedure that transforms \longrightarrow into a set rewriting system \Rightarrow such that if $a \xrightarrow{|s|} b$ then $a \Rightarrow_{\downarrow} b$.
- ▶ The procedure is correct and complete, i.e., $\longrightarrow \equiv \Rightarrow_{\downarrow}$, if \longrightarrow is confluent.

The serialization procedure has been implemented in Maude for an arbitrary set relation system \longrightarrow .

Application to PLEXIL

The Five Semantic Relations

- ▶ The *atomic relation* describes the execution of an individual node in terms of state transitions triggered by changes in the environment.
- ▶ The *micro relation* describes the *synchronous* extension of the atomic relation with respect to the *maximal redexes strategy*.
- ▶ The *quiescence relation* describes the normalized extension of the micro relation.
- ▶ The *macro relation* describes the interaction of a plan with the external environment.
- ▶ The *execution relation* describes the n -iteration of the macro relation corresponding to n time-steps.

PLEXIL Semantics

- ▶ Atomic: \longrightarrow .
- ▶ Micro: $\xrightarrow{|s|}$.
- ▶ Quiescence: $\xrightarrow{|s|}\downarrow$.
- ▶ Macro: $\times \xrightarrow{|s|}\downarrow$.
- ▶ Execution: $\times \xrightarrow{|s|} \downarrow^n$.

PLEXIL Semantics

- ▶ Atomic: \longrightarrow .
- ▶ Micro: $\xrightarrow{|s|}$.
- ▶ Quiescence: $\xrightarrow{|s|}\downarrow$.
- ▶ Macro: $\times \xrightarrow{|s|}\downarrow$.
- ▶ Execution: $\times \xrightarrow{|s|}^n \downarrow$.

PLEXIL Semantics

- ▶ Atomic: \longrightarrow .
- ▶ Micro: $\xrightarrow{|s|}$.
- ▶ Quiescence: $\xrightarrow{|s|}\downarrow$.
- ▶ Macro: $\times \xrightarrow{|s|}\downarrow$.
- ▶ Execution: $\times \xrightarrow{|s|}^n \downarrow$.

PLEXIL Semantics

- ▶ Atomic: \longrightarrow .
- ▶ Micro: $\xrightarrow{|s|}$.
- ▶ Quiescence: $\xrightarrow{|s|}\downarrow$.
- ▶ Macro: $\times \xrightarrow{|s|}\downarrow$.
- ▶ Execution: $\times \xrightarrow{|s|}^n \downarrow$.

PLEXIL Semantics

- ▶ Atomic: \longrightarrow .
- ▶ Micro: $\xrightarrow{|s|}$.
- ▶ Quiescence: $\xrightarrow{|s|}\downarrow$.
- ▶ Macro: $\times \xrightarrow{|s|}\downarrow$.
- ▶ Execution: $\times \xrightarrow{|s|}\downarrow^n$.

Modular PLEXIL Semantics

- ▶ $\text{PLEXIL} \equiv \times \xrightarrow{|s|} \downarrow^n$, where $\xrightarrow{|s|}$ consists of 42 rewriting rules.
- ▶ $\text{PLEXIL}_1 \equiv \times \xrightarrow{|s|} \downarrow^n$.
- ▶ $\text{PLEXIL}_2 \equiv \times \xrightarrow{|s|} \downarrow^n$.
- ▶ ...

Modular PLEXIL Semantics

- ▶ $\text{PLEXIL} \equiv \times \xrightarrow{|s|} \downarrow^n$, where $\xrightarrow{|s|}$ consists of 42 rewriting rules.
- ▶ $\text{PLEXIL}_1 \equiv \times \xrightarrow{|s|} \downarrow^n$.
- ▶ $\text{PLEXIL}_2 \equiv \times \xrightarrow{|s|} \downarrow^n$.
- ▶ ...

Modular PLEXIL Semantics

- ▶ $\text{PLEXIL} \equiv \times \xrightarrow{|s|} \downarrow^n$, where $\xrightarrow{|s|}$ consists of 42 rewriting rules.
- ▶ $\text{PLEXIL}_1 \equiv \times \xrightarrow{|s|} \downarrow^n$.
- ▶ $\text{PLEXIL}_2 \equiv \times \xrightarrow{|s|} \downarrow^n$.
- ▶ ...

PLEXIL Semantics in PVS

- ▶ **Determinism**: PLEXIL's atomic relation is deterministic. Hence, PLEXIL's execution relation is deterministic.
- ▶ **Compositionality**: PLEXIL's programs are compositional¹.

¹Under some conditions.

PLEXIL Semantics in Maude

- ▶ Maude is a (sequential) rewriting engine logic.
- ▶ Serialization procedure applied to PLEXIL's micro relation yields an **executable** semantics of PLEXIL that is correct and complete.

What Should The Following Programs Do ?

▶ List Exchange {

```
int x = 0;
```

```
int y = 1;
```

```
Assignment X { x:= y; }
```

```
Assignment Y { y:= x; }
```

```
}
```

Values of x and y are exchanged.

▶ Assignment SimplePost {

```
}
```

Post-condition failure. Error in the design of the atomic relation.

▶ Assignment Loop {

```
int x = 0;
```

```
Repeat-while: x < 10;
```

```
x := x+1;
```

```
}
```

Infinite Loop. Proposed alternative semantics.

What Should The Following Programs Do ?

▶ List Exchange {

...

}

Values of x and y are exchanged.

▶ Assignment SimplePost {

int x = 0;

x := 10;

Post: x == 10;

}

Post-condition failure. Error in the design of the atomic relation.

▶ Assignment Loop {

int x = 0;

Repeat-while: x < 10;

x := x+1;

}

Infinite Loop. Proposed alternative semantics.

What Should The Following Programs Do ?

▶ `List Exchange` {

...

}

Values of x and y are exchanged.

▶ `Assignment SimplePost` {

...

}

Post-condition failure. Error in the design of the atomic relation.

▶ `Assignment Loop` {

`int x = 0;`

`Repeat-while: x < 10;`

`x := x+1;`

}

Infinite Loop. Proposed alternative semantics.

Technical Contributions

- ▶ A PVS library of set relations and formal proofs of properties such as determinism and compositionality.
- ▶ A serialization procedure to simulate synchronous relations in Maude.

Practical Contributions

- ▶ An executable formal semantics of PLEXIL that has been interfaced to a graphical PLEXIL environment.
- ▶ Identified at least 3 design issues in PLEXIL's semantics.

Conclusion

- ▶ Semantics of programming language should leverage from existing libraries and formal theories.
- ▶ Have we identified the fundamental constructions for synchronous languages?
 - ▶ Not all of them, e.g., Cartesian product extension, etc.