

Testing Planning Domains (without Model Checkers)

Franco Raimondi, Charles Pecheur, Guillaume Brat

Overview

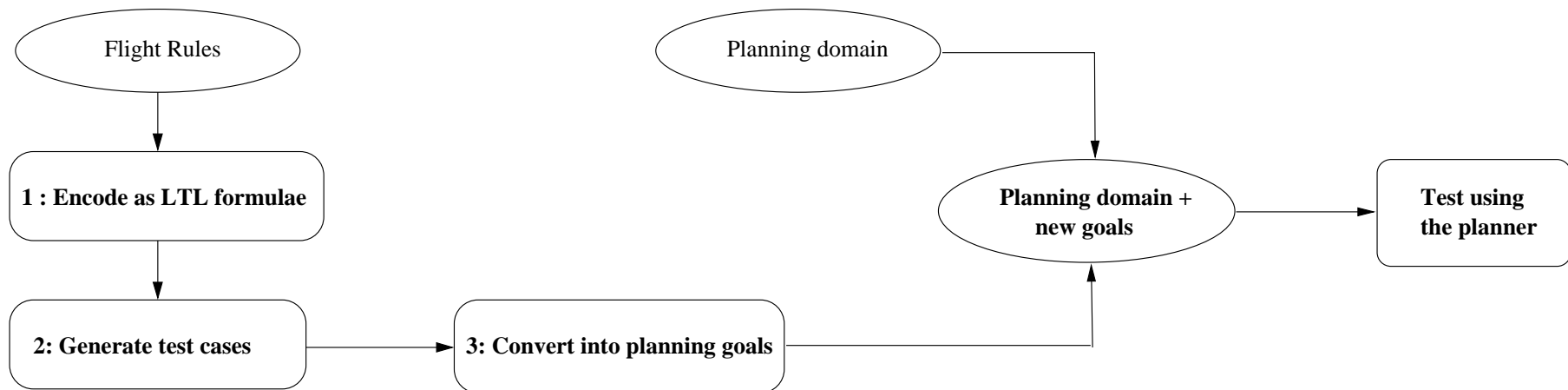
- Motivational introduction
- Flight rules
- Review of MC/DC and definition of UFC with weak/strong coverage.
- Planning: Europa 2 and NDDL
- The Rover Example: NDDL code, flight rules and trap formulae.
- From trap formulae to planning goals.
- Conclusion.

Motivation

- Planner are used extensively in a number of autonomous applications (e.g., the NASA rovers exploring Mars' surface).
- Methodology are needed to verify that certain requirements (the *flight rules*) are not violated when executing plans.
- Verification of planning domain has been investigated in the past by translating planning models into input for model checkers (but problems with state space explosion).

Our contribution

Key idea: translate the verification problem into a planning problem.



Flight rules

- Flight rules are requirements that must be satisfied in every execution (currently written in plain English).
- Typically, flight rules are *temporal patterns* and can be encoded as LTL formulae.
- Example: *all Instruments must be stowed when moving*, which is translated into $\varphi = G(\text{moving} \rightarrow \text{stowed})$

NEXT: How to build a suitable set of test cases to guarantee coverage of the above by extending MC/DC.

Brief overview of MC/DC (required coverage for avionic SW)

- Every basic condition in a decision in the model has taken on all possible outcomes at least once.
- Each basic condition has been shown to independently affect the decision's outcome.

An example: let $\varphi = p \vee q$. φ can be either true or false, and it can be so either because of p , or because of q . Test cases:

p	q	$p \vee q$
\top	\perp	\top
\perp	\perp	\perp
\perp	\top	\top

Each test case can be captured by a (Boolean) formula. Let φ^{pos} be the set of test cases for positive outcomes, and φ^{neg} the set for negative outcomes.

$$\varphi^{pos} = \{p \wedge \neg q, \neg p \wedge q\}; \quad \varphi^{neg} = \{\neg p \wedge \neg q\}.$$

Unique First Cause coverage

[Whalen et al., 2006] UFC extends MC/DC to requirements based testing (i.e., testing flight rules).

A test suite achieve UFC of a set of requirements (expressed in LTL) if:

1. Every basic condition in any formula has taken on all possible outcomes at least once.
2. Each basic condition has been shown to affect the formula's outcome as the *unique* first cause.

A basic condition a is the UFC for a formula φ along a path π if, in the first state along π in which φ is satisfied, it is satisfied because of a (see paper for a formal definition).

Trap formulae

[Whalen et al., 2006] A *trap formula* $ufc(a, \varphi)$ is a temporal formula characterising adequate test cases for a in φ .

$$ufc(a, a) = a; ufc(a, \neg a) = \neg a$$

$$ufc(a, \varphi_a \vee \varphi_b) = ufc(a, \varphi_a) \wedge \neg \varphi_b$$

$$ufc(a, \mathbf{F} \varphi_a) = (\neg \varphi_a) \mathbf{U} ufc(a, \varphi_a)$$

$$ufc(a, \mathbf{G} \varphi_a) = \varphi_a \mathbf{U} (ufc(a, \varphi_a) \wedge \mathbf{G} \varphi_a)$$

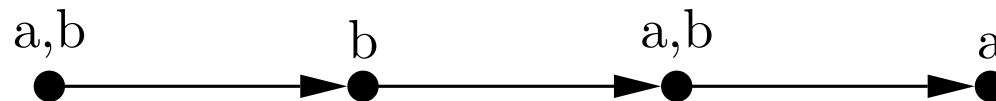
Problem: this only applies to *infinite* paths. If tests (paths) need to be finite, we need to change this definition.

Strong and weak UFC coverage

A finite path (test case) π_f gives:

1. *strong* evidence for φ (written as $\pi_f \models^+ \varphi$) if π_f “carries all necessary evidence for” φ ;
2. *weak* evidence for φ (written as $\pi_f \models^- \varphi$) if π_f “carries no evidence against” φ ;

Example: a strong test case for a in $F(a \wedge \neg b)$ and a weak test case for a is $G(a \vee b)$.



(see paper for the formal definition of ufc^\pm)

Summary of UFC

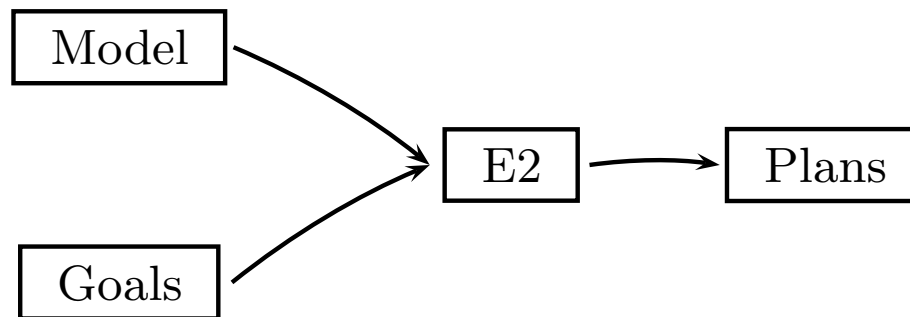
Given an atomic condition a appearing in a formula φ (a requirement) and an execution model M :

- if there is a test case $\pi_f \models ufc^+(a, \varphi)$ in the traces of M , then π_f shows that a can *necessarily* positively affect φ ;
- if $\pi_f \models ufc^-(a, \varphi)$, then π_f only shows that a can *possibly* positively affect φ ;
- if there is no π_f in M for which $\pi_f \models ufc^\pm(a, \varphi)$: if φ is a desired property, then this means that a is a *vacuous* condition in φ w.r.t. M ;
- if φ is a negative (forbidden) property, then it confirms that this particular case of φ cannot happen, which is the desired result;
- A test fails if it is possible to find a path π_f in M such that $\pi_f \models ufc^\pm(a, \varphi)$, where φ is a negative property.

Planning with EUROPA 2.0 (E2)

[Excerpts from C. McGann, *How to Solve It*]

*“Planning can be considered a process of generating descriptions of how to operate some system to accomplish something. The resulting descriptions are called **plans**, and the desired accomplishments are called **goals**. In order to generate plans for a given system a **model** of how the system works must be given.”*



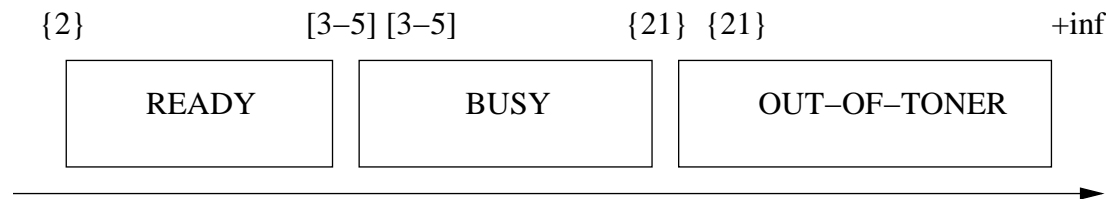
Models and goals are written in NDDL (see below).

Tokens and Timelines

A **token** is “*an instance of a temporally scoped predicate*” (predicates describe true facts). For instance:

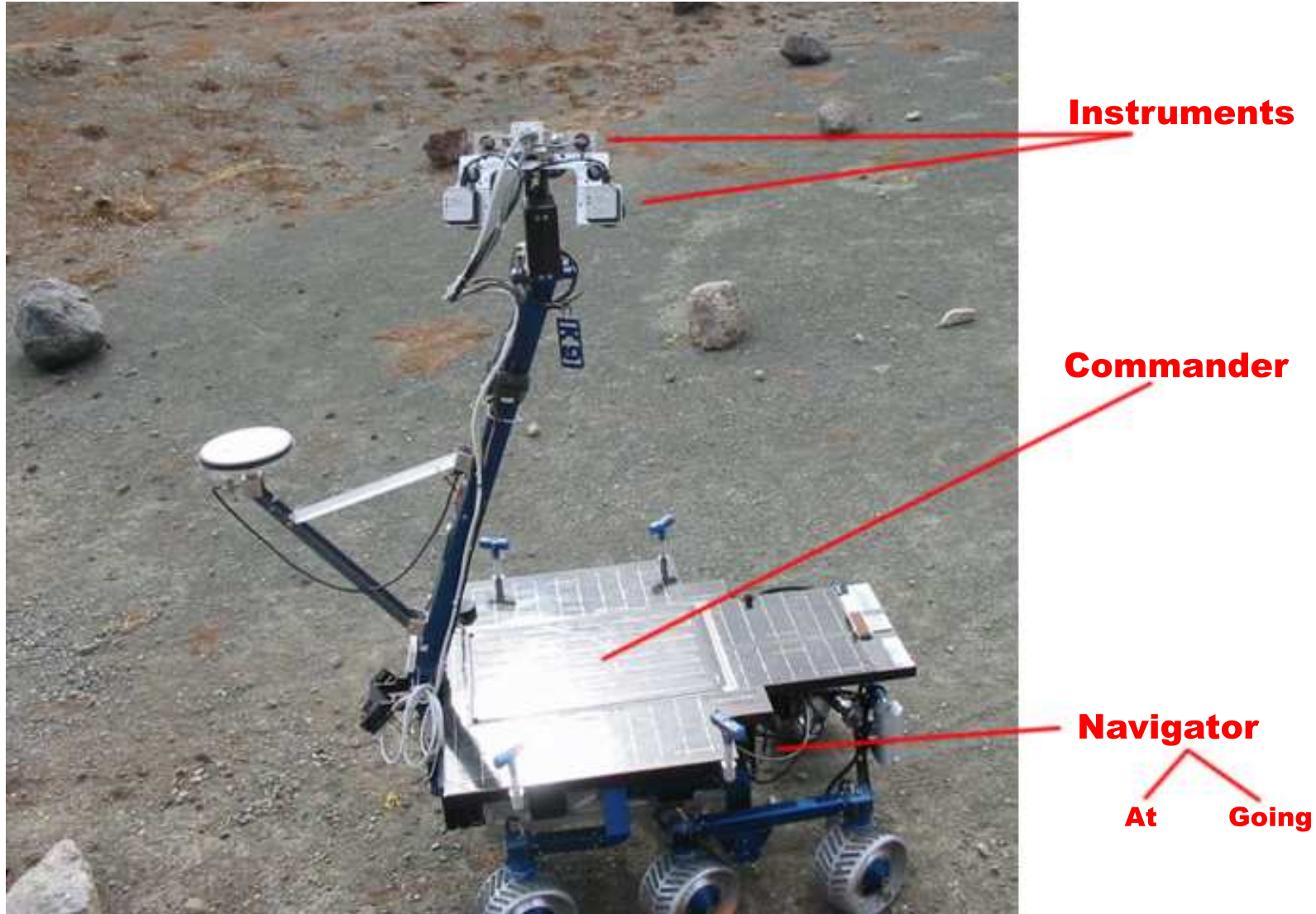


Tokens may represent states of a single object in the system, and are sometimes mutually exclusive. A **Timeline** is a structure where sequences of tokens appear contiguously. For instance, the state of a printer:

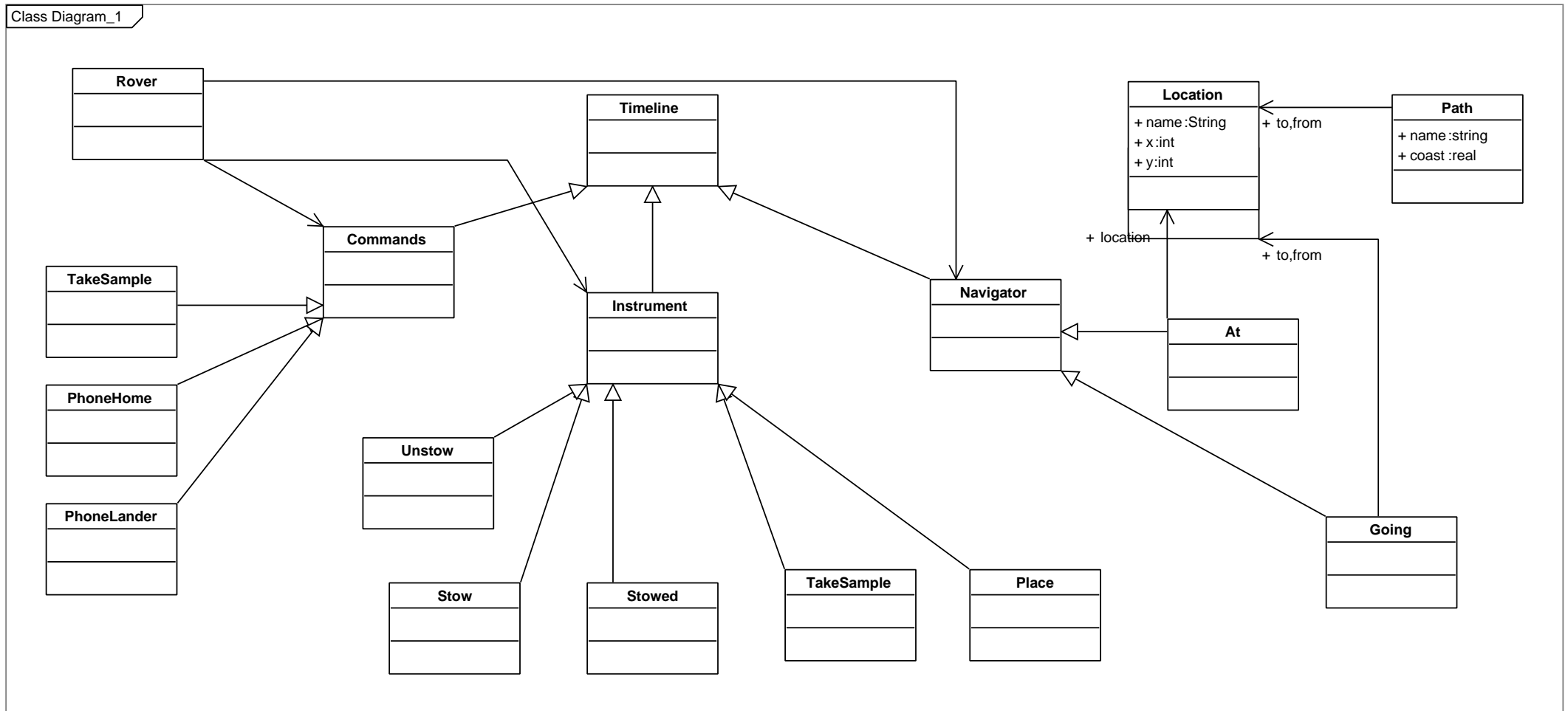


Constraints (rules) are expressed using Allen’s temporal logic constructs, such as “meets”, “met by”, etc. Essentially, these are (in)equalities over tokens’ bounds.

NDDL using an example: simple Rover model



NDDL using an example: simple Rover model



Created with Poseidon for UML Community Edition. Not for Commercial Use.

Some NDDL

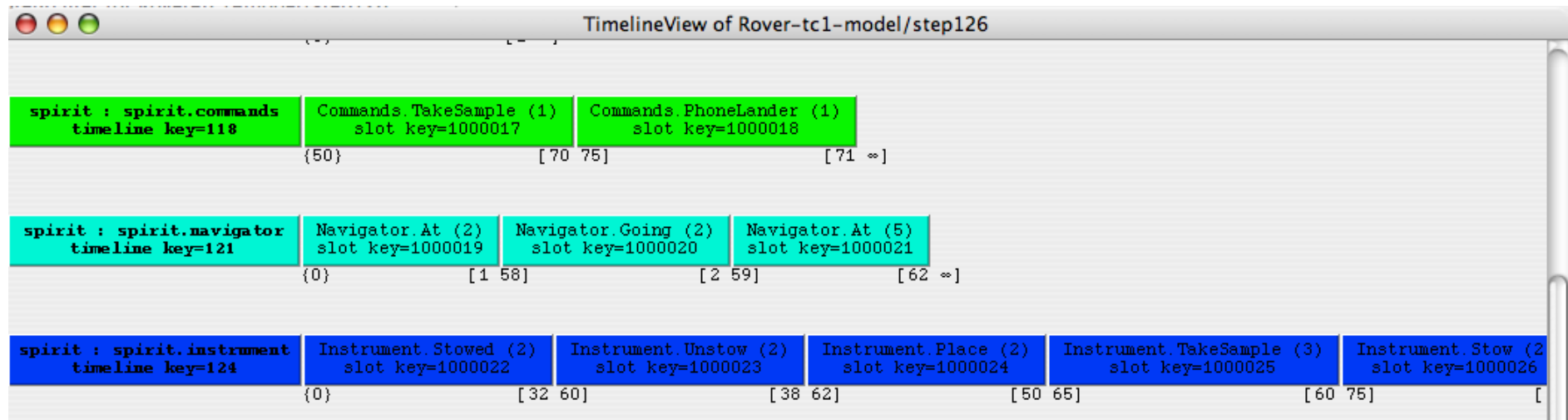
```
class Navigator extends Timeline {
  predicate At{
    Location location;
  }
  predicate Going{
    Location from;
    Location to;
    neq(from, to);
  }
  // prevents rover from going from a location straight back to that location.
}
}
```

Tokens are constrained using *rules*:

```
Navigator::At{
  met_by(object.Going from);
  eq(from.to, location); // next Going token starts at this location
  meets(object.Going to);
  eq(to.from, location); // previous Going token ends at this location
}
```

NDDL goals and executions

```
Location lander = new Location("LANDER", 0, 0);  
[...]  
goal(Navigator.At initialPosition);  
eq(initialPosition.location, lander);  
goal(Commands.TakeSample sample);  
sample.start.specify(50);  
sample.rock.specify(rock4);
```



A concrete example: flight rules for the rover

1. The Rover Battery State of charge cannot go below X .
2. All Instruments must be stowed when moving.
3. ...

In LTL:

1. $\mathbf{G}(x)$, where x is a proposition true when the charge is $\geq X$.
2. $\mathbf{G}(p \rightarrow q)$, where $p = \text{moving}$ and $q = \text{stowed}$.
3. ... (translation done similarly).

We now apply the machinery presented above to compute test cases for the second requirement, $\mathbf{G}(p \rightarrow q)$.

Test cases for a flight rule

Three test cases for the formula:

1. (true value caused by “stowed”):

$$ufc^-(q, \varphi) = ((\neg p \wedge q)U(\neg p \wedge \neg q));$$

2. (true value caused by “moving”):

$$ufc^-(p, \varphi) = ((\neg p \wedge q)U(p \wedge q \wedge \varphi));$$

3. (this is the negative test case):

$$ufc^{+/-}(p, \neg\varphi) = ufc^{+/-}(q, \neg\varphi) = ((\neg p \vee q)U(p \wedge \neg q))$$

Summary of coverage

- A model for the Rover (NDDL file).
- Flight rules encoded in LTL, for instance $\varphi = G(\textit{moving} \rightarrow \textit{stowed})$.
- Trap formulae for flight rules that guarantee coverage.

Next step: *translation of elements of ufc^\pm into planning goals*, so that E2 can be used *for testing flight rules*.

This is done without modification of the original NDDL model, by adding new timelines, new rules and new goals

Details of the translation into planning goals

- Introduce new timelines for Boolean components of a formula.
- A bottom-up translation, starting from the atoms and building negations, conjunction, disjunctions, etc.
- Temporal operators are translated into goal for these new timelines.

Example: the first positive test case for φ of the previous slide:

$$(\neg p \wedge q)U(\neg p \wedge \neg q)$$

Next slide: the NDDL code corresponding to proposition p (moving) and its relation with the token Going.

```

class Prop_p extends Timeline {
    predicate TRUE { };
    predicate FALSE { };
}

Prop_p::TRUE {
    // Used to populate timeline
    met_by(object.FALSE f1);
    meets(object.FALSE f2);
    Navigator nav;
    equals(Navigator.Going);
}

Prop_p::FALSE {
    met_by(object.TRUE f1);
    meets(object.TRUE f2);
}

Navigator::Going {
    // p is true when Navigator is Going
    Prop_p pp;
    equals(pp.TRUE);
}

```

Negations, conjunctions, etc

- Negations: a new timeline defined using a previous one with `equals` rules.
- Use conjunctions only.
- Encoding a conjunction: this requires some work:
 - First define a timeline with all the combinations of the values of the two conjunct timelines ($\top\top, \top\perp, \dots$).
 - Use `contains` / `contained_by` to define tokens' relationships.
 - For the continuity of the timeline: use enumerations and match each predicate with all the other predicates

Temporal operators as goals

In the example: $(\neg p \wedge q)U(\neg p \wedge \neg q)$, i.e., `timeline1.TRUE` until `timeline2.TRUE`.

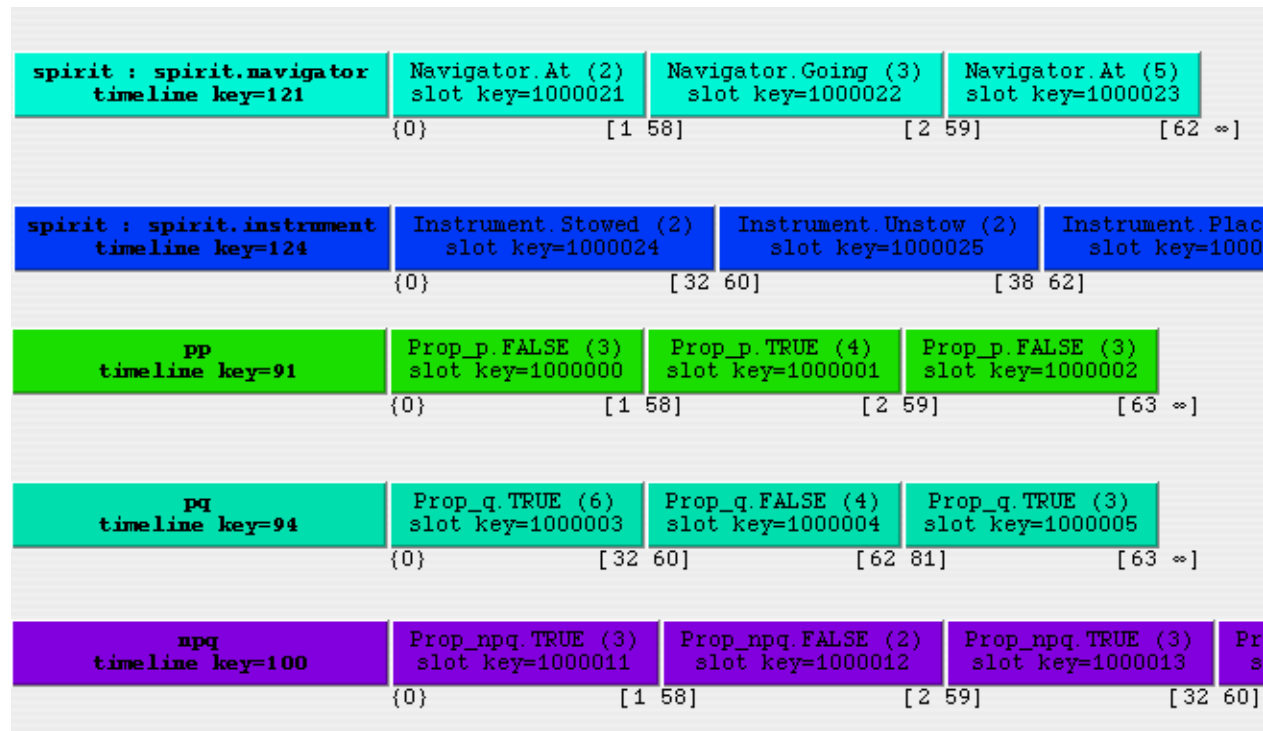
This is translated into:

```
timeline2.TRUE contains_end timeline1.TRUE
```

and other additional constraints (for initial states etc).

This is the actual code `<SWITCH DISPLAY>` to show *.nddl.

Running E2 with this code



(the important point here is: a plan can be obtained with the additional new constraints).

The negative test case:



(the important point here is: **no plan can be obtained**)

Conclusions

- Extension of UFC with definitions of strong/weak acceptance.
- A methodology to test flight rules
 - with coverage guarantees;
 - self-contained (no external tools);
 - using patterns, a tool could be implemented to perform tests automatically;
 - the new timelines+rules are *added* to the original model (i.e., the original model is not modified).
- Not a production scenario yet, but more complex than what can be done with model checkers.

Future work

- Nested temporal operators (flight rules analysed don't contain nested operators, but they could).
- Automatic tools to perform the translation NDDL + LTL into new NDDL code.
- A collection of patterns for flight rules and a graphical interface for plan developers.