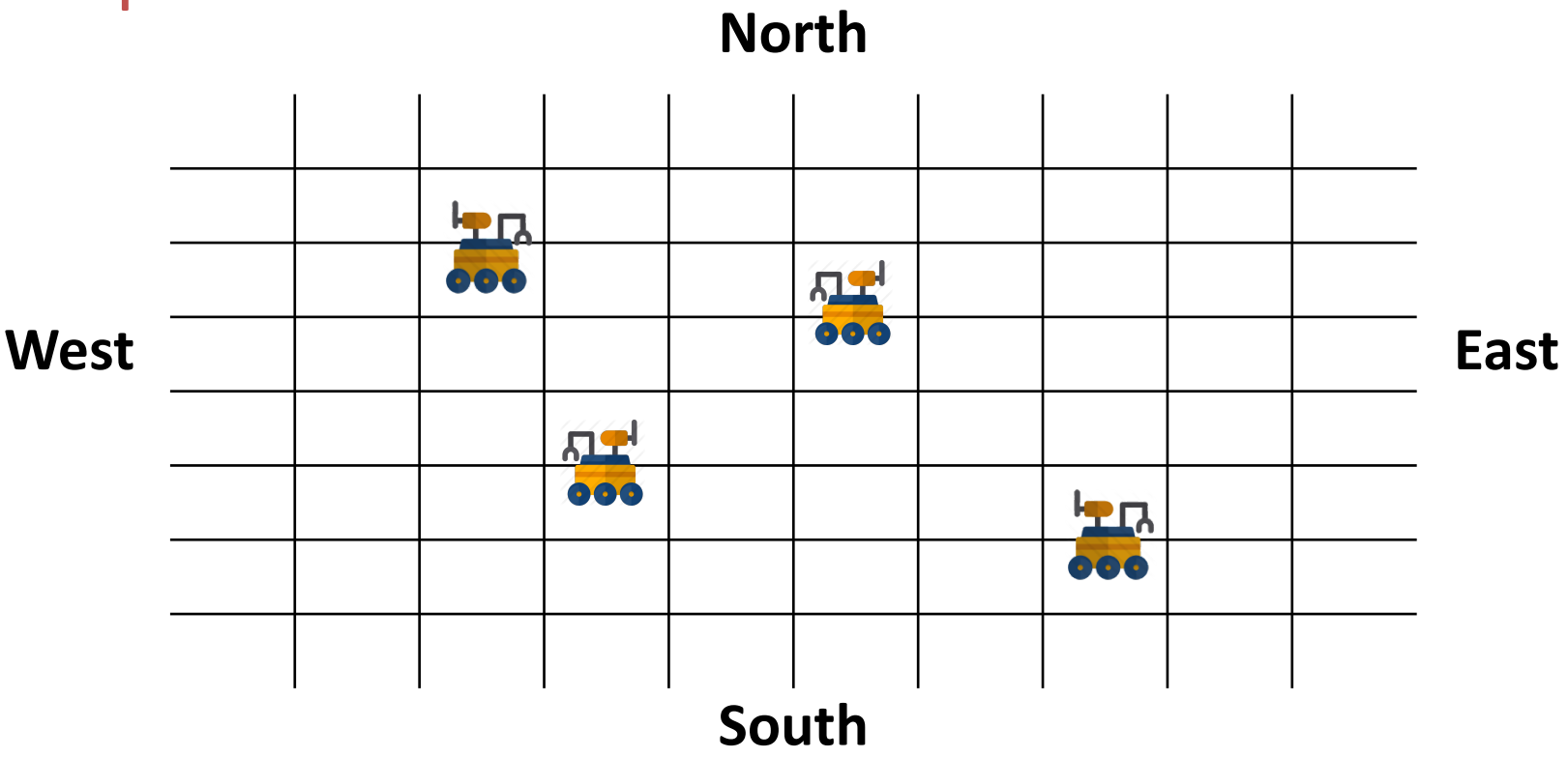
CS:5810
Formal Methods in Software Engineering

Case Study: Autonomous Rovers

Copyright 2017 Paul Meng and Cesare Tinelli.
These notes are copyrighted materials and may not be used in other course settings outside of the University of Iowa in their current form or modified form without the express written permission of one of the copyright holders. During this course, students are prohibited from selling notes to or being paid for taking notes by any person or commercial firm without the express written permission of one of the copyright holder.

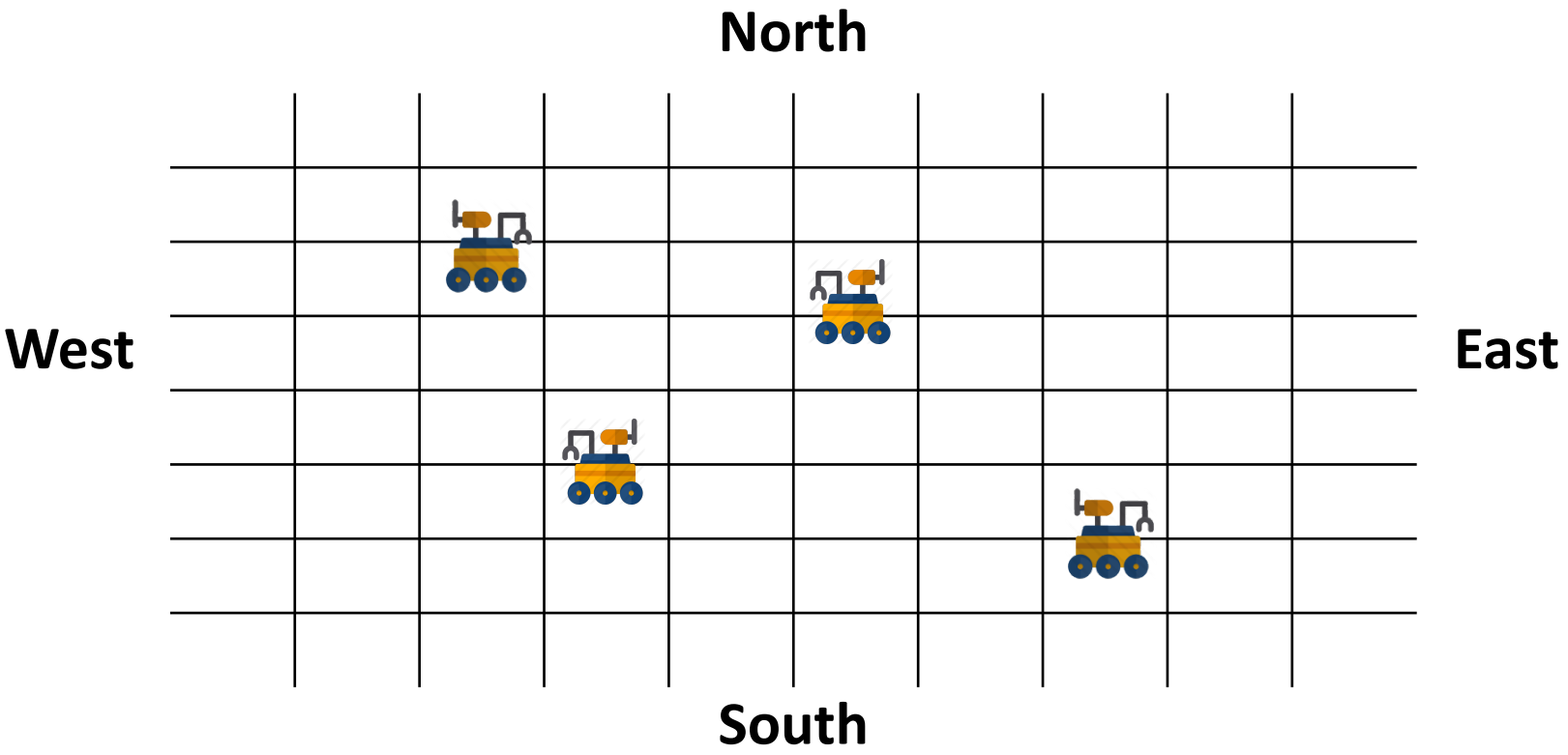
The Task

- Model in Alloy a dynamic domain involving several rovers moving on a two-dimensional space



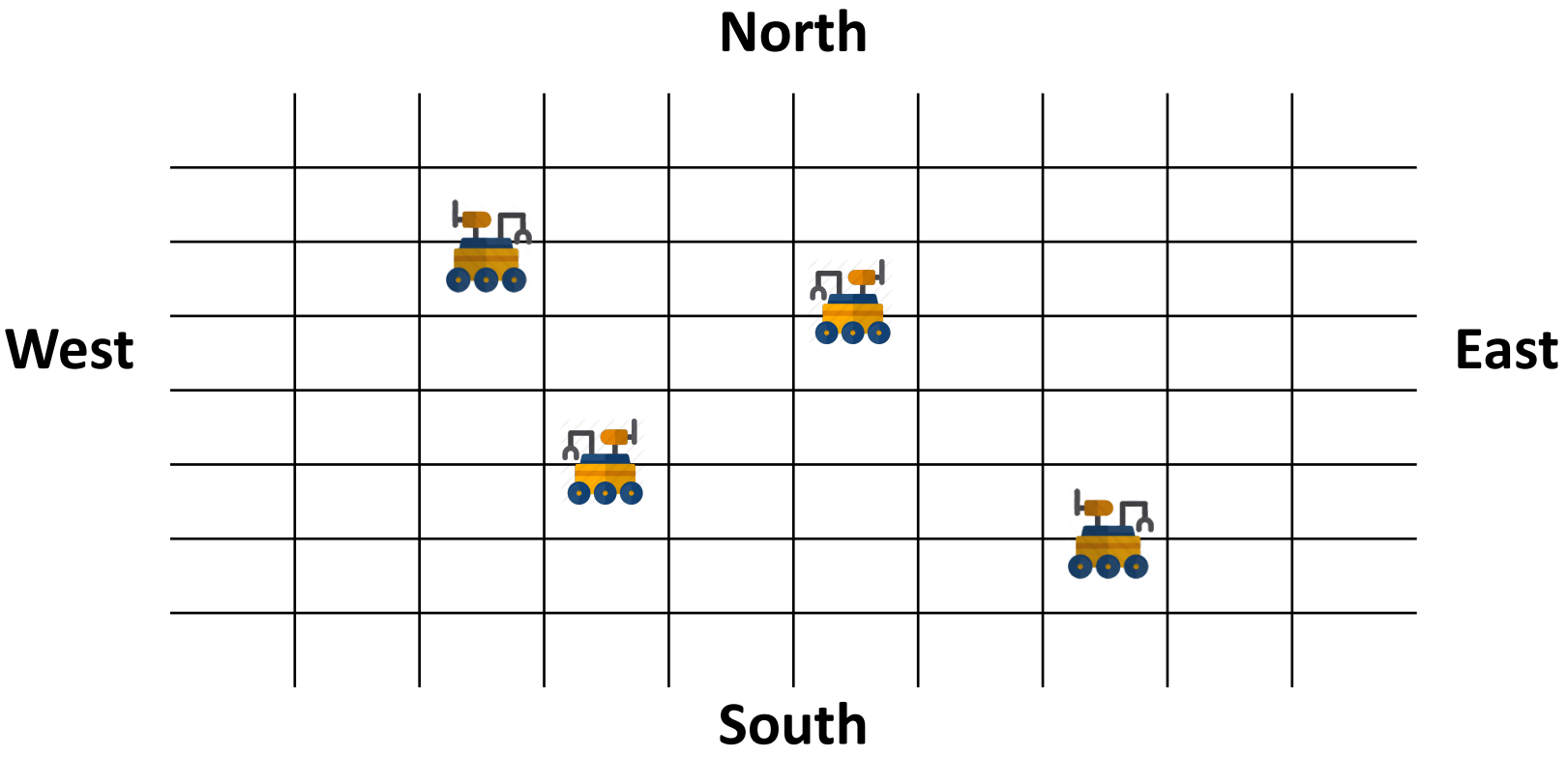
Facts about the System

- There are one or more identical rovers
- Each rover can be turned on and off



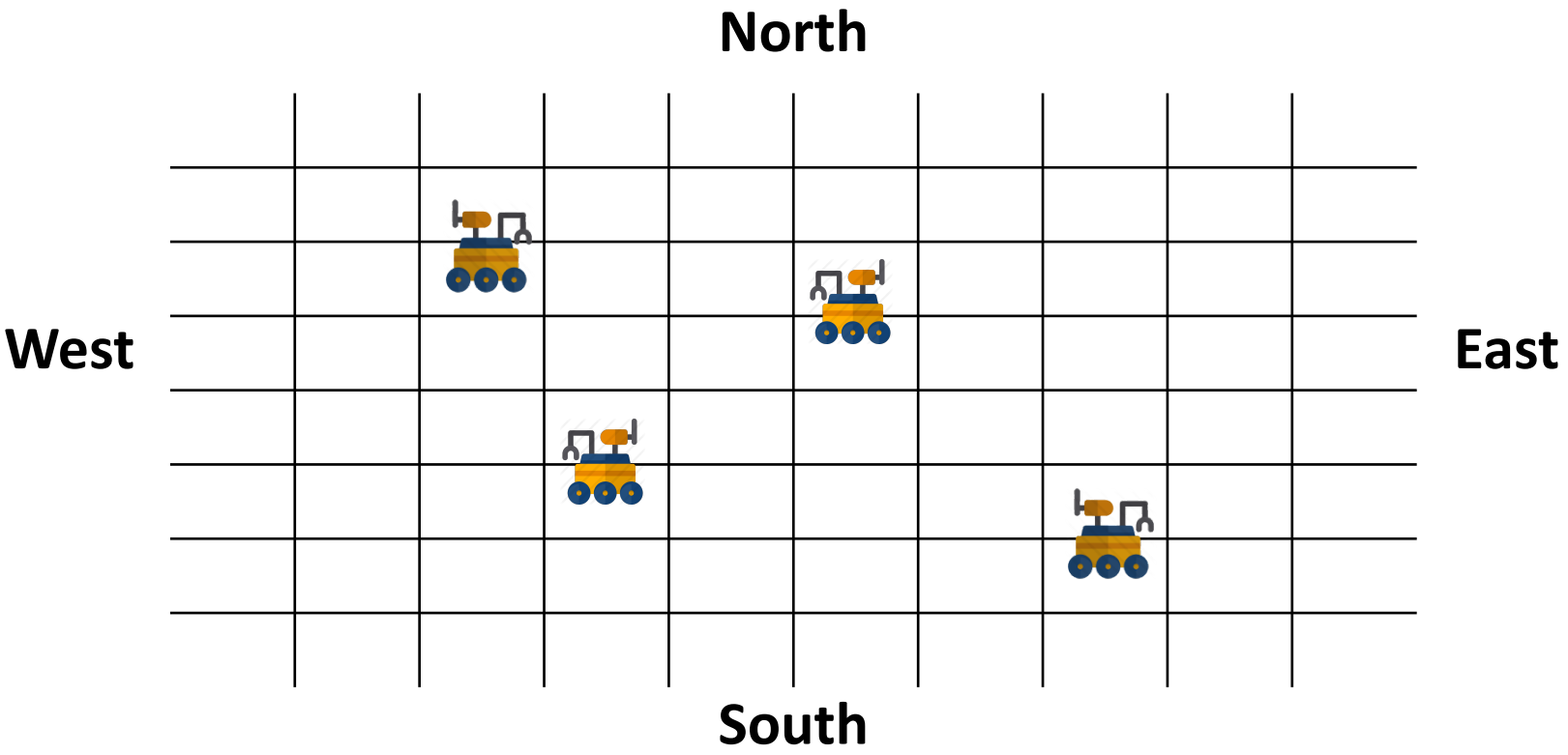
Facts about the System

- Each rover can only move forward, or turn in place to the left or to the right



Facts about the System

- We will model both static and dynamic aspects of the system



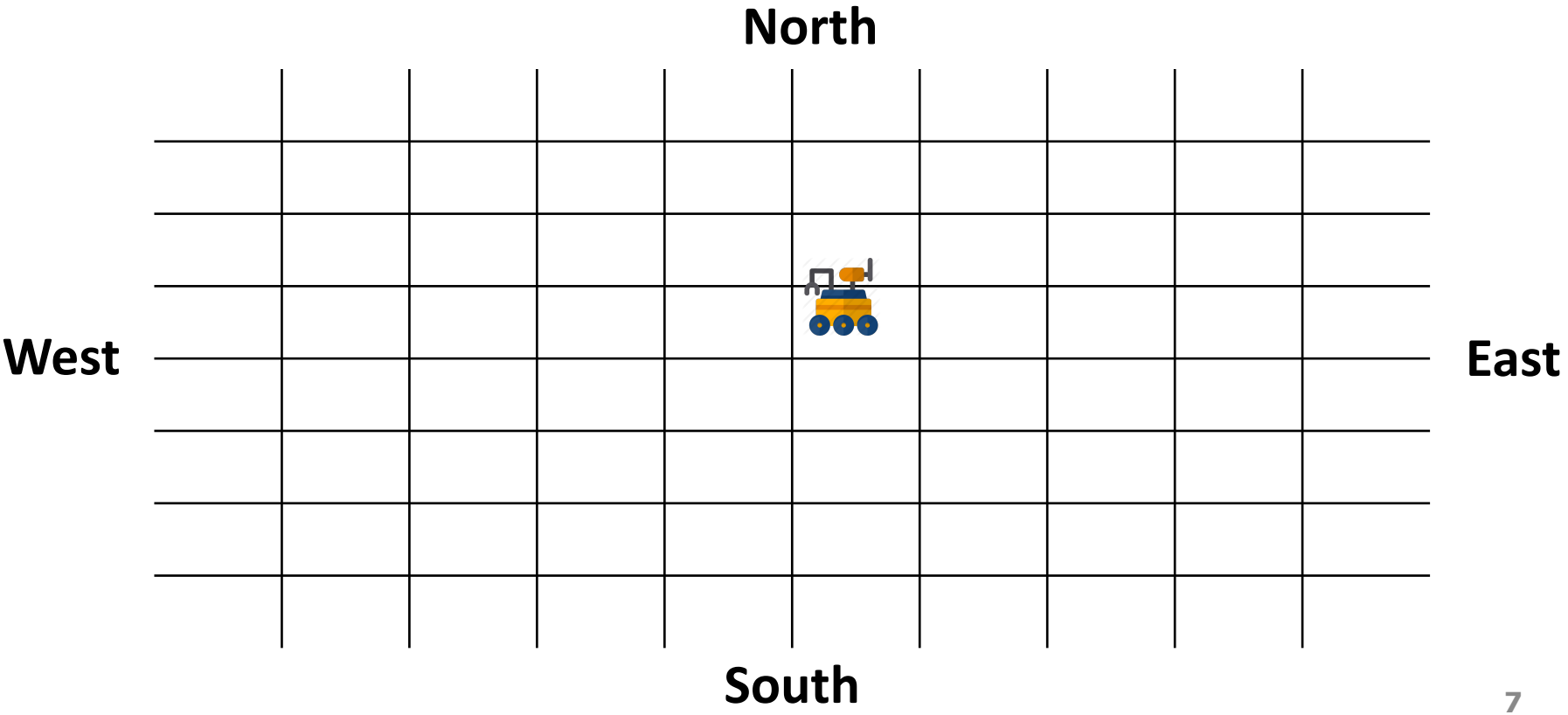
Simplifying Modeling Choices

1) We adopt an interleaving model of time: only one action is performed, by one of the rovers, at a time

2) The two dimensional space is a discrete grid, with
   - the X-coordinate growing indefinitely in the West-East direction and
   - the Y-coordinate growing indefinitely in the South-North

Simplifying Modeling Choices

3) Rovers move only by one position at a time and along the X,Y axes.



Simplifying Modeling Choices

4) A rover turns left or right by exactly 90 degrees
5) A rover can move only in the direction it is facing

Signatures and Fields

```
open util/ordering [Time] as T
open util/ordering [Coor] as C

-- Coordinates, strictly ordered
sig Time {}
sig Coor {}

-- Position models the individual positions
-- in the grid
sig Position {  x: Coor,  y: Coor}
```

Signatures and Fields

```
-- The four cardinal directions
abstract sig Direction {}

one sig North, South, East, West extends
    Direction {}
```

Signatures and Fields

```
some sig Rover {
    -- Direction rover is facing at any one time
    dir: Direction one -> Time,

    -- Rover's position at any one time
    pos: Position one -> Time,

    -- Rover's on/off status at any one time
    on: set Time
}
```

Operators

Turn on
Turn off
Turn left
Turn right
Go

Turn On Operator

```
pred turn_on [rov: Rover, t,t': Time] {
    -- Pre-condition
    Rover is off at time t (!is_on)

    -- Post-condition
    Rover is on at time t’ (is_on)

    -- Frame condition
    All other rovers stay on or off as they were (no_on_changes)
    No rover changes direction (no_direction_changes)
    No rover changes position (no_position_changes)
}
```

Turn Left Operator

```
pred turn_left [rov: Rover, t,t': Time] {
    -- Pre-condition
    Rover is on at time t (is_on)

    -- Post-condition
    Direction Changes (could be North, South, East, or West)

    -- Frame condition
    All rovers stay on or off as they were (no_on_changes)
    No other rover changes direction (no_direction_changes)
    No rover changes position (no_position_changes)
}
```

If-Then-Else in Alloy

$Expr_1$ (=>, implies) $Expr_2$ else $Expr_3$

- $Expr_1$ is a Boolean expression
- $Expr_2$ and $Expr_3$ can be either Boolean or Set expression

E.g. let parents_in_law =
(John.spouse = Mary => Mary.parents
else John.spouse = Lily => Lily.parents
else none)

Go Operator

```
pred go[rov: Rover, d: Direction, t,t': Time] {
    -- Pre-condition
    Rover is on at time t (is_on)
    d is rover's direction at time t

    -- Post-condition
    Position Changes (could move towards North, South, East, or
    West)
    (next_pos[p: Position, d: Direction]: Position)
    -- Frame condition
    All rovers stay on or off as they were (no_on_changes)
    No rover changes direction (no_direction_changes)
    No other rover changes position (no_position_changes)
}
```

The Module Ordering

```
// return the predecessor of e, or empty set if e is
// the first element
fun prev [e: S]: lone S { e.(Ord.Prev) }

// return the successor of e, or empty set of e is
// the last element
fun next [e: S]: lone S { e.(Ord.Next) }
```

Transition System

```
pred System {
        init[T/first]
        all t: Time – T/last | transitions[t, T/next[t]]
}
```

- Facts

```
-- P0 is the origin position of the coordinate system
```

- Init

```
-- Rover R1 is at the origin position, facing East and turned off
-- The other rovers, if any, are at a different position than R1's
```

- Transitions

```
-- Some rover turn on, off, left, right, or go
```

System Goal

```
pred goal[t: Time]{
   -- R1 is not at the origin
   R1.pos.t != P0
   -- R1 is facing north
   R1.dir.t = North
}
pred goalCheck{
        one Rover
        System
        some t : Time | goal[t]
}
```