A view on some important extensions to the standard

OpenSCENARIO ASAM workshop – 13-Nov-2018

Yoav Hollander - Founder and CTO, Foretellix
blog.foretellix.com

foretellix
verifying autonomous systems

Introduction

- The OpenSCENARIO work so far has taken us a significant way forward

- The challenge for this standard is huge and important
  - The current list of requirements is really good (and ambitious)
  - *And* I expect the requirements to grow

- Today I'll discuss just the requirements for making scenarios specifications

  Clear, Composable, Measurable and Portable

- For each, I'll connect it to current requirements, and suggest a direction

Clear

- Scenarios should be easy to read, write and *debug*

- How
  - Use a powerful language with clear semantics
  - Allow for abstraction and extensibility
  - Create a library of basic building blocks
  - Merge graphic input *into* the language
  - Use constraints to make descriptions *concise* (next slide)

- Reference (to the list of features and requirements):
  - F008: High-Level Maneuver Descriptions: Suggests a DSL as an option
  - R012: The description format shall be suitable for manual scenario creation in text editors
  - Section 5.4: Test specifications

Clear (making it *concise*)

- Write as little as possible - let the machine do the rest
  - Abstract scenarios should produce *many different, interesting* instances, exposing *unconsidered combinations*
  - Writing everything explicitly is not manageable – it does not *scale*

- How
  - Use constraints to express parameter relations / restrictions
    - Constraints have *multi-directional* semantics
    - Do not enumerate combinations – full values will appear only in *concrete* scenario
  - Use default values when appropriate
    - Allow for both random and deterministic modes (see later)

- Reference
  - F007: Parameter Stochastics: … The variation details shall become part of the scenario description
  - F010: Automatic parameter calculation: … specify mathematical formulas to calculate parameters
  - R002: Define elements as 'mandatory' only when absolutely needed

Clear (example)

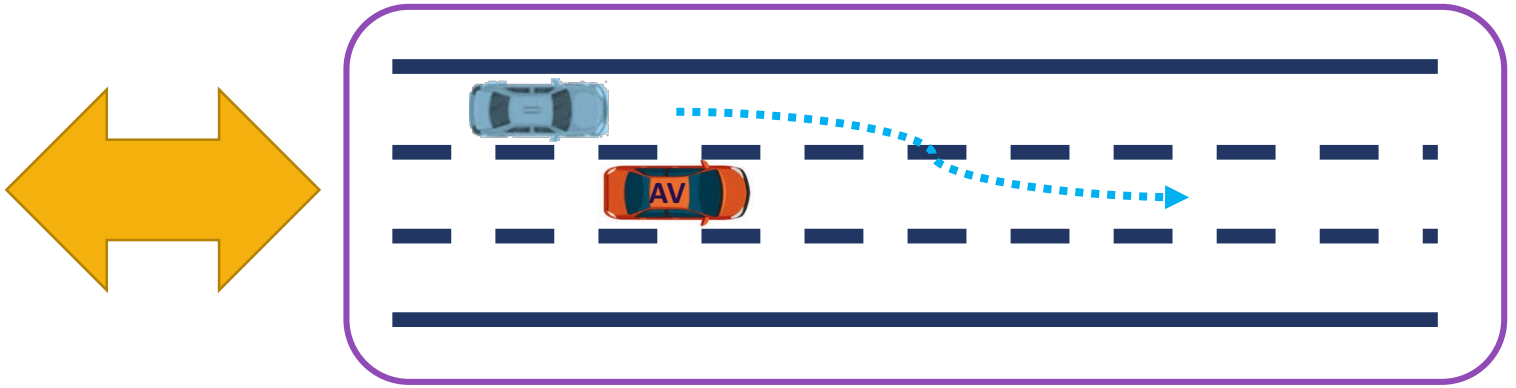
```
scenario ego::cut_in_and_slow is {
   car1: car;     // The other car
   side: av_side;  // Car1 starts on this side of the ego
   stretch: stretch {.lanes >= 2; .length in [120..250] * meter};

   do serial {

      get_ahead: phase(duration: in [1..5] * second) {
         p1_ego: ego_car.drive {+drive_on(stretch)};
         p1_car1: car1.drive {
            +behind(ego_car, at: start);
            +ahead_of(ego_car, [5..10] * meter, at: end);
            +on_side_of(ego_car, side);
            +faster_than(ego_car);  }};

      change_lane: phase(duration: in [1..5] * second) {
         p2_ego: ego_car.drive;
         p2_car1: car1.drive {+change_to_lane_of(ego_car); }};

      slow: phase(duration: in [1..5] * second) {
         p3_ego: ego_car.drive;
         p3_car1: car1.drive {+slow_down([10..15] * kph); }};
};
```



foretellix

Composable

- Should be easily to create complex scenarios from simpler ones
  - Without planning all required “knobs”
  - Especially important for finding complex edge cases

- How
  - Have general composition operators: serial, parallel, phase, mix, …
  - Have an easy way to activate and constrain sub-scenarios
  - Take into account explicit and implicit constraints
  - Support extensibility

- Reference
  - F008: High-Level Maneuver Descriptions: … high-level descriptions … contain key-scenarios such as "cut-in“, “"left turn across path"
  - R010: Synchronize maneuvers

foretellix

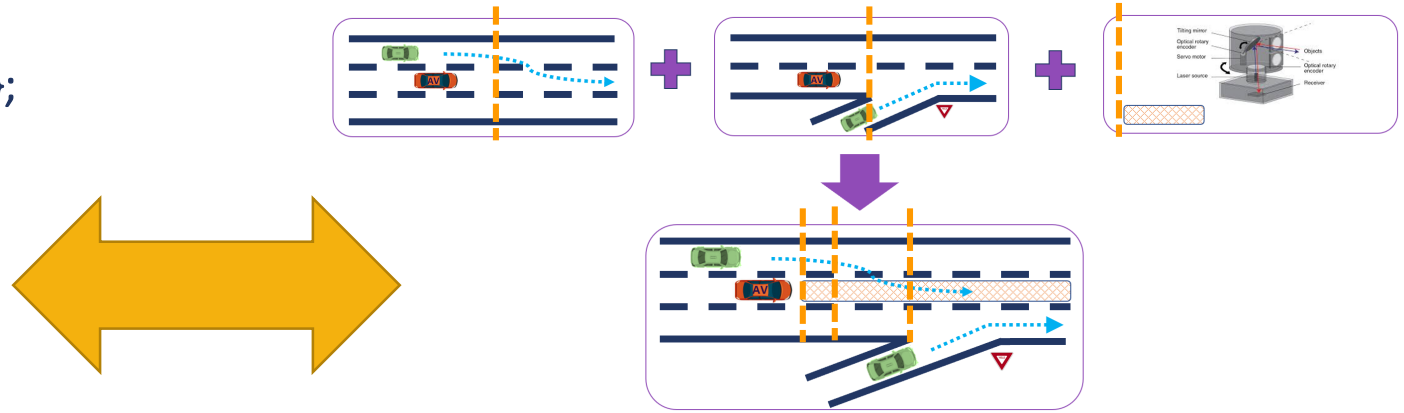
Composable (example)

```
scenario ego::mix_multi is {

    do mix {
        c: cut_in_and_slow {.car1.kind == truck};
        i: interceptor_at_yield {.int_to_ego_offset in [-1..1]};
        f: sensor_failure;
    };

    c_to_i_offset: int {it in [-2..2] * second};
    f_to_i_offset: int {it in [-2..2] * second};

    +synchronize(c.change_lane, i.e_traverse.enter, c_to_i_offset);
    +synchronize(f, i.e_traverse.enter, f_to_i_offset);
};
```



foretellix

Measurable

- Need a common way to measure
  - Which scenarios *happened* and with what parameters
  - Whether the ego vehicle *performed correctly*
  - Consistently across simulators / other execution platforms

- How
  - Scenario descriptions should have a *dual* interpretation
    - Active: How to (try to) cause this scenario
    - Passive: How to monitor that this scenario has happened (and extract parameter values)
  - Use functional coverage to define / analyze what happened

- Reference
  - R007: Define simulation results reproducibility
  - Section 5.6: … high probability that different simulators produce different simulation results from the same scenario descriptions … it is proposed to develop reference implementations

foretellix

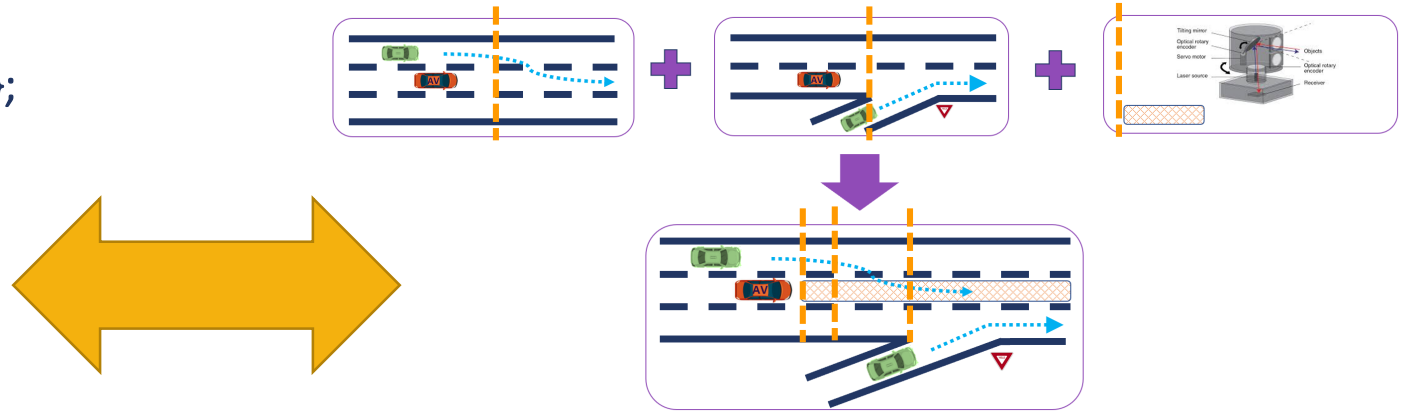
Measurable (example)

```
scenario ego::mix_multi is {

   do mix {
      c: cut_in_and_slow {.car1.kind == truck};
      i: interceptor_at_yield {.int_to_ego_offset in [-1..1]};
      f: sensor_failure;
   };

   c_to_i_offset: int {it in [-2..2] * second};
   f_to_i_offset: int {it in [-2..2] * second};

   +synchronize(c.change_lane, i.e_traverse.enter, c_to_i_offset);
   +synchronize(f, i.e_traverse.enter, f_to_i_offset);
};
```



foretellix

Portable

- The *same* scenario definition should be portable across
  - Execution platforms: Specific simulators, specific test tracks, …
  - Test configurations: Sensor-bypass vs. full sensor simulation, …
  - ODDs: Adapt to operating conditions, country rules and conventions, …
  - Stakeholders: OEMs, subsystem creators, regulators, …
  - Usage modes: Fully-random vs. deterministic, …

- How
  - Enabled by multiple features (constraints, composability, passive/active interpretation and more)

- Example: Portability across random-vs-deterministic usage modes
  - "Infinity minus mode": Parameters get *any random value* unless explicitly *prohibited* (by constraints)
    - For thorough verification and finding edge-case bugs
  - "Deterministic mode": Parameters get *default values* unless explicitly *requested*
    - For test tracks, go-no-go tests, perhaps regulation-requested tests

- Example: Portability across execution platforms
  - Portability for street driving enabled by passive (monitoring) interpretation of scenarios
  - Portability across simulators enabled by default values, adaptation via constraints and more

Summary

- This is a crucial (and very ambitious) standardization effort
  - And there are some inevitable follow-on requirements

- I talked about a subset of the requirements, to make scenarios

  Clear, Composable, Measurable and Portable

- We are looking forward to working with you
  - To create a *high-quality* open scenario standard for the industry
  - In a *reasonable* timeframe

- Thank you

blog.foretellix.com