Augmenting Scenario Description Languages for Intelligence Testing of Automated Driving Systems

Yun Tang, Antonio A. Bruto da Costa*, Patrick Irvine, Tudor Dodoiu, Yi Zhang, Xingyu Zhao, Siddartha Khastgir, and Paul Jennings

***Abstract*— Scenario-based verification and validation (V&V) has emerged as the predominant approach for the performance evaluation of automated driving systems (ADSs). Many scenario-generation methods have been proposed to search for critical scenarios, i.e. disengagement or traffic rule violations. However, the widely adopted binary (pass/fail) criterion suffers from two main limitations, i.e., the difficulty of locating root causes and the lack of statistical guarantee of testing sufficiency. Recently, new scenario engineering approaches focusing on the intelligence of ADSs enlightened a promising pathway via dynamic driving task decomposition and function atom constraints. However, none of the state-of-the-art scenario description languages support such approaches. To fill this gap and facilitate further research into this promising direction, in this work, we propose a generic architecture to extend the existing scenario description languages for the intelligence testing of ADSs. The case study with WMG SDL demonstrates the capability and flexibility of the proposed extension design in defining intelligence function constraints.**

I. INTRODUCTION

Background Scenario-based verification and validation (V&V) has emerged as the predominant approach for the performance evaluation of automated driving systems (ADSs) [1]–[3]. Compared to the distance-based approach where the ADSs are required to drive millions of miles [4], [5] to cover sufficient diversity of driving situations due to the long-tail effect, scenario-based methods aim to eliminate the redundancy and distil critical scenarios of interest directly from various data sources, e.g., domain expert knowledge [6]–[12] or naturalist driving data [12]–[24] by diverse types of scenario generation (aka, parameter sampling) algorithms. The generated scenarios are commonly evaluated against two metrics, i.e., criticality (e.g., distance-to-collision [22], [23] and time-to-collision [10], [12], [21], [25]), and coverage (e.g., parameter value combination [11], [12]).

However, existing metrics suffer from two main limitations, i.e., the difficulty of locating root causes and the need for explicit correspondence to the AV-under-test's underlying capabilities (aka intelligence). First, upon the occurrence of critical events (e.g., disengagements or traffic rule violations such as collisions), the existing binary-based (i.e., pass/fail) or robustness-based (e.g., time-to-collision) criticality metrics do not clue how the critical events happened. Manual fault localization in modularized AV systems (e.g., Apollo [26], and Autoware [27])) is difficult primarily due to the ripple effect (e.g., a frame of perception error may propagate through the system and cause a collision half a minute later). Meanwhile, it is even more challenging for end-to-end [28] AV systems due to the lack of explainability of AI models. Second, covering a complete set of scenario parameter value combinations does not provide statistical correspondence on the inner capabilities of the system-under-test, which should be the actual purpose of scenario-based testing.

*Corresponding author

All authors are with WMG, University of Warwick, Coventry, United Kingdom. Emails: {yun.tang, antonio.bruto-da-costa, patrick.irvine, tudor.dodoiu, yi.zhang.16, xingyu.zhao, s.khastgir.1, paul.jennings}@warwick.ac.uk

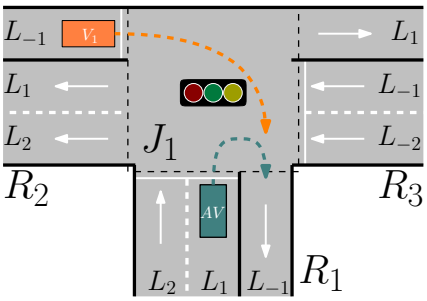


Fig. 1: An example U-turn scenario at signalized junction. R_1, R_2 and R_3 are three roads connected by the T-junction J_1 and L_i are road lanes.

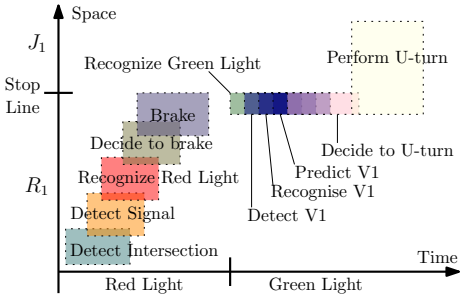


Fig. 2: Constrained function atoms implied by the scenario in Fig. 1.

Recent discussions on intelligence-guided scenario-based testing [29], [30] enlightened a promising pathway for addressing the abovementioned limitations by decomposing the dynamic driving tasks into constrained function atoms. Intuitively, to navigate a vehicle across public road networks with other traffic participants, the AV systems, regardless of the implemented architecture and Operational Design Domain (ODD) [31], must share a common set of intelligence [30], [32], including but not limited to, self-localization and navigation through different types of roads and intersections, detection and response to diverse objects and events,

identification and compliance with relevant traffic rules and regulations. For example, from the U-turn scenario in Fig. 1, we can derive the list of function atoms and constraints as shown in Fig. 2. Hence, by validating the AV's scenario execution logs against the constrained function atoms, we can easily track the root cause of critical events and collect statistical evidence on the intelligence coverage [30].

Motivation Intuitively, when a concrete [33] scenario is defined, the static road networks, the environmental conditions, the dynamics of traffic objects and the list of possible events are fixed. As a result, the scenario definition implies the set of driving intelligence to pass the scenario, regardless of the AV system. More importantly, the progression of scenario dynamics implies the spatial and temporal constraints of the demanded intelligence function atoms. For example, the AV-under-test needs to complete the detection and recognition of the traffic signal status before driving too close to the intersection so that the vehicle has sufficient time to brake and stop before the stop line if necessary, as shown in Fig. 2. When such constraints on the function atoms are explicitly defined, the root cause for any critical event can be easily tracked down, and the statistical coverage of the tested intelligence can be easily estimated.

Contributions Despite the above benefits of intelligence-guided testing, we notice that none of the existing state-of-the-art scenario description languages (SDLs), e.g., WMG SDL [34], Scenic [35], and ASAM OpenSCENARIO/OpenDRIVE [36], [37], support the specification of such function atom constraints. To fill the gap, in this work, we i) design a generic extension architecture to existing SDLs to enable intelligence testing; ii) implement the extension in our state-of-the-art SDL, i.e., WMG-SDL [34] that is powering a quarter million scenarios on the Safety Pool^{TM} [38] scenario database and discuss a case study in detail.

II. BACKGROUND

A. Scenario, Intelligence and Function Atoms

Authors in [30] provide a theoretical foundation for intelligence testing, including the relationship between intelligence and function atoms, as illustrated in Fig. 3. Specifically, each scenario represents a dynamic driving task (DDT) consisting of the start and destination of the AV-under-test. The AV is expected to perform a series of Object and Event Detection and Response (OEDR) sub-tasks (e.g., avoid collision with another vehicle and stop before the stop line when approaching the intersection under red light), which can be further broken down into function atoms (e.g., detect and recognize traffic signals, detect, recognize and predict the other vehicle and decelerate). Intelligence-guided scenario-based testing [30] denotes a scenario engineering approach where scenarios are generated for evaluating intelligent tasks and function atoms.

B. Signal Temporal Logic

We draw inspiration from the Signal Temporal Logic (STL) language in designing the function atom constraints. STL has been widely adopted in critical scenario generation

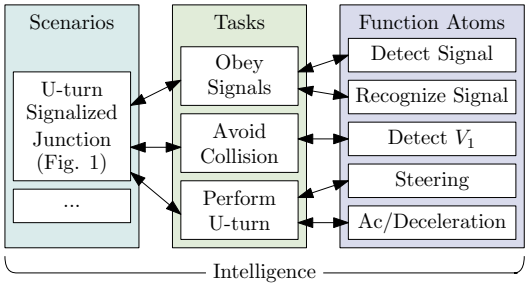


Fig. 3: Relationship illustration among scenarios, tasks, function atoms and intelligence adapted from [30].

```
SCENERY ELEMENTS:
Junctions : None Roads : R1 : speed limit of [ 10 ] Road Sign [ Speed Limit ] as [S1]...
Number of lanes [ 2 ] as [ R1.L1 , R1.L2 ] ...

DYNAMIC ELEMENTS : INITIAL :
Vehicle [ Ego ] in [ R1.L1 ] at speed [ 15.0 to 15.0 ] AND
Vehicle [ V1 ] in [ R1.L2 ] with a [ Longitudinal ] offset of [ 50 to 50 ] to [ Ego ]  AND
Pedestrian [ V2 ] in [ R1 . L1 ] with a [ Longitudinal ] offset of [ 90 to 90 ] AND with
a [ Lateral ] offset of [ -2 to -2 ] to [ Ego ] AND Global timer [ T1 ] = [ 0 ]
WHEN : [ Ego ] is [ Going_Ahead ] DO :
[ V1 ]
   PHASE 1 : [ Drive_Away ] [ - , 8 to 8 , 0 to 0 ]
   WHILE : [ V1 ] [ Longitudinal ] offset to [ Ego ] >= [ 30 ]
   PHASE 2 : [ LaneChangeRight_CutIn ] [ - , 8 to 8 , 0 to 0 ]
   WHILE : [ T1 ] <= [ 60 ] AND :
[ V2 ]
   PHASE 1 : [ Stopped ] [ - , 0 to 0 , 0 to 0 ]
   WHILE : [ V1 ] [ Longitudinal ] offset to [ Ego ] >= [ 30 ]
   PHASE 2 : [ Walk_Cross ] [ - , 5 to 5 , 0 to 0 ]
   WHILE : [ T1 ] <= [ 60 ] AND :

ENVIRONMENT ELEMENTS : Rainfall [ None : N/A ] ...
```

Fig. 4: An example of a (simplified) WMG SDL scenario consisting of a cut-in vehicle (id: $V1$) and a jaywalker (id: $V2$). The manoeuvre phases of the two actors are synchronized, i.e., the transitions from *Phase 1* to *Phase 2* of both actors occur simultaneously.

methods as the robustness value provides valuable guidance on the scenario parameter sampling directions towards desired critical events [39]. STLs are commonly specified in the following format:

$$\begin{aligned}\phi &::= \text{true} \mid \pi \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \\ &\quad \mid \phi\, \mathbf{U}_{[a,b]}\, \phi \mid \phi\, \mathbf{R}_{[a,b]}\, \phi \mid \mathbf{F}_{[a,b]}\, \phi \mid \mathbf{G}_{[a,b]}\, \phi \\ \pi &::= f(x) > c \mid f(x) \geq c \mid f(x) < c \mid f(x) \leq c\end{aligned}$$

where ϕ is an STL formula; true is a boolean constant; π represents a predicate, defined over the signal (e.g., position or velocity trace); $\neg$, $\wedge$, and $\vee$ are the standard logical negation, conjunction, and disjunction operators, respectively; $\phi_1 \mathbf{U}_{[a,b]} \phi_2$ ($\phi_1 \mathbf{R}_{[a,b]} \phi_2$) is the *Until* (*Release*) operator representing that ϕ_1 (ϕ_2) must hold until ϕ_2 (ϕ_1) becomes true within time bounds a and b; $\mathbf{F}_{[a,b]}$ and $\mathbf{G}_{[a,b]}$ are the *Finally* and *Globally* operators, representing that a formula ϕ must hold at some point or at all points, respectively, within the time interval.

C. Scenario Description Language

SDLs are domain-specific languages for the scenario-based V&V of ADSs. The utility of SDLs lies in their ability to provide a precise and unambiguous description of the operation context for a dynamic driving task, commonly encompassing scenario parameters for static road networks, dynamic behaviours of diverse types of traffic participants, environmental conditions, etc. Currently, three languages are widely adopted and under active maintenance, i.e., the WMG's two-level SDL [34], Scenic [35], and ASAM OpenX (OpenSCENARIO and OpenDRIVE) [36], [37]. The WMG SDL language embeds the scenario parameters in natural languages and aligns closely with industrial standards. On the other hand, Scenic has a grammar analogous to that of the Python programming language and focuses on probabilistic scenario modelling. Lastly, the OpenSCENARIO XML is an XML format and is widely supported by the esmini [40] and Carla [41] simulators.

In this work, we take the WMG SDL as an example and implement the extension following its grammar. Fig. 4 is a simplified WMG SDL scenario consisting of three main components, i.e., *Scenery Elements* representing the static road networks, *Dynamic Elements* representing the dynamic behaviours of the traffic and *Environment Elements* representing the environmental conditions. WMG SDL defines the scenario dynamics at the manoeuvre instead of the trajectory level. It is up to the scenario execution engine (e.g., a simulator) to compute the actual actor trajectories at runtime.

III. The Intelligence Testing Extension

We suggest an object-oriented design for the intelligence testing extension, adaptable to any SDL. Fig. 5 presents the class diagram in Unified Modeling Language (UML).

Function Constraint Logic contains a list of *Function Atoms*, *Constraint Formulas* and *Interval Boundaries*. They are separated into different classes for reusability.

Function Atom is the basic building block for the extension, which contains four main attributes, i.e., the *id* as a unique identifier used in the constraint formulas, the function *name*, which can be assigned to one of *Function Name* enumeration options, the *target* which can be assigned to the id of any static object or dynamic actor in the scenario, and the *detail* is optional and enables additional specification (e.g., a predicate) regarding the function atom instance. For example, a function atom instance can be defined as *(id: FA1, name: Recognize, target: V2, detail: as class Pedestrian)*. Note that we do not specify the grammar for the *detail* attribute since it should be customized per individual scenario execution (or evaluation) engines.

The **Constraint Formula** defined constraints with an optional operator and interval. It allows the assignment of at most two function atoms, i.e., *antecedentFunction* and *consequentFunction*. To enable nested constraints, each constraint formula also allows the assignment of at most two other constraint formulas, i.e., *antecedentFormula* and *consequentFormula*. Note that the attribute *antecedentFunction* (resp. *consequentFunction*) and *antecedentFormula* (resp. *consequentFormula*) are mutually exclusive. The *operator* attribute takes one of the *Logic Operator* enumeration options. The constraint formula is evaluated only when the *operator* is not *None*; otherwise, the constraint formula instance is treated as a reference by another nested formula. The *intervalStart* and *intervalEnd* specify the constraint interval boundaries. Since the scenario progression is usually unpredictable due to the asynchronous communication between the scenario execution engine and the AV-under-test, it is often impractical to bound the constraints using scenario execution time only. For example, the constraint "*Detect the speed sign within 5 to 10 seconds after the start of the scenario*" may not be valid given a varying (and often unpredictable) initialization time by the Ego vehicle before it starts to move. Hence, we propose an abstract *IntervalBoundary* class consisting of five flexible boundary types.

Interval Boundary The five boundary types are temporal, function, spatial, phase, and event and are mutually exclusive for individual *IntervalBoundary* instances.

Temporal Boundary supports absolute or relative constraints in the temporal domain. The *value* attribute specifies the time offset in seconds, where a negative value represents a time before, and a positive value indicates a time afterwards. To specify a time w.r.t. a scenario engine timer (which can be, e.g., a global timer throughout the scenario or a local timer which resets with each manoeuvre phase as defined in the WMG SDL language [34]), the corresponding timer's ID can be assigned to the *timer* attribute. For example, assume *G_TIMER_1* is a global timer ID, then *(timer: G_TIMER_1, value: 10)* means 10 seconds after the scenario starts. Often, scenario engineers are interested in the relative time constraint, e.g., "*5 seconds before the vehicle reaches the stop line*" or "*0.5 seconds after the frontal vehicle brakes*". To achieve this, the *boundary* attribute can be utilized to specify the time offsets w.r.t. another interval boundary.

Function Boundary marks the time when there is a change in the evaluation of function atom predicates. For example, the positive edge of the function "Detect the human." indicates the moment the human is detected; the negative edge of the function "Recognize the human as a police officer" indicates the moment when the human is no longer recognized as a police officer. The function boundaries are especially helpful in constraining function atom dependencies, e.g., "recognize the human as a police officer 0.1 seconds after the human is detected".

Spatial Boundary pinpoints the moment when a scenario object reaches a specific cartesian location on the map. The position can be absolute, e.g., with x, y coordinates, or relative position w.r.t. another object or road network element of the scenario. We omit the detailed class definition for the *position* attribute as it depends on the specific SDL.

Phase Boundary as used in manoeuvre-driven SDLs, such as WMG SDL [34] or OpenSCENARIO [36] (as compared to trajectory-type scenario descriptions), to describe multi-stage actor behaviours using manoeuvre phases. For example, an overtake scenario typically includes *lane change*, *accelerate*,

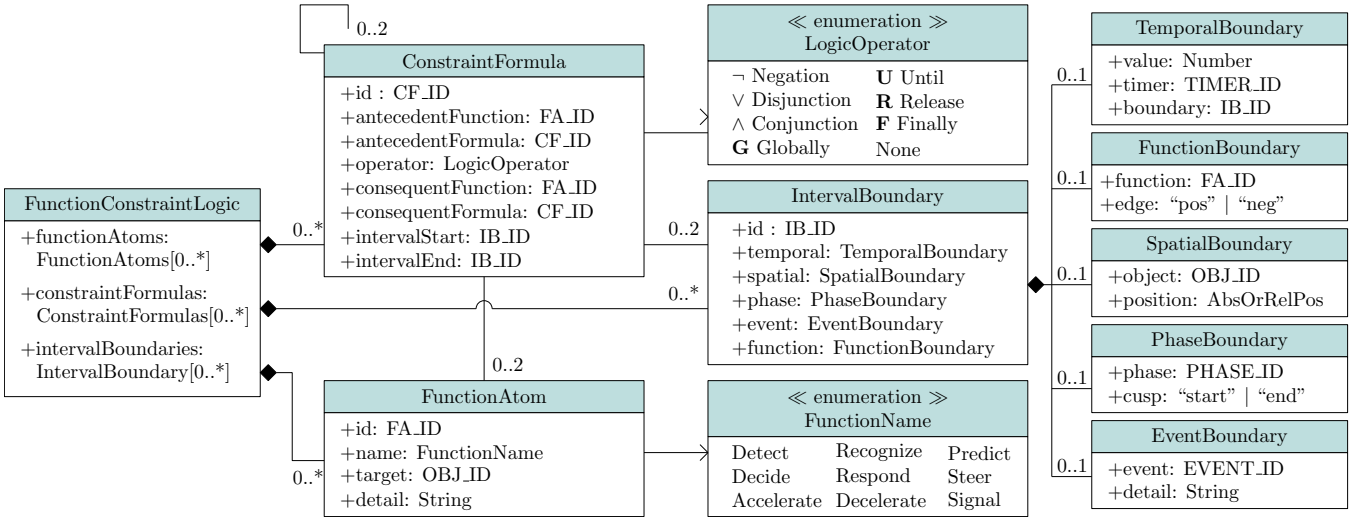


Fig. 5: Class diagram of the proposed intelligence testing extension design for SDLs. The “XX_ID” types are strings commonly with a class prefix, e.g., the id of a function atom instance can be “FA1” where “FA” is the *Function Atom* class prefix.

and *cut-in* phases for the Principal Other Actor. The *PhaseBoundary* class is defined to pinpoint the moment relative to the *start* or *end* (the *cusp* attribute) of a specific manoeuvre phase (the *phase* attribute) of any scenario actor.

Event Boundary captures unscripted scenario events, serving as a general constraint boundary for moments unsuitable for the other three boundary types, e.g., when the ego vehicle fails to plan a drivable trajectory or exits its ODD. Similarly, we leave the implementation of the *detail* attribute to the specific SDLs and execution/evaluation engines.

Logic Operator is an enumeration class containing a list of logic operators defined similarly to their counterparts of the Signal Temporal Logic formulas. Hence, the interval boundary definitions are optional for *Negation*, *Disjunction* and *Conjunction* while they are compulsory for *Globally*, *Until*, *Release* and *Finally*.

Function Name is an enumeration class listing intelligence function names, focusing mainly on OEDR capabilities as summarized in the report [42]. It includes common functions such as *Detect*, *Recognize*, *Predict*, *Accelerate*, and *Signal*. These functions may be interpreted or overridden by implementations of the AV system, and the enumeration class permits flexible extension.

IV. CASE STUDY

We implement the proposed intelligence testing extension in the WMG SDL [34] language. WMG SDL is written using the Eclipse Xtext framework [43]. The extension syntax adheres to the same language design principles of WMG SDL (Fig. 6). We discuss one case study in detail using the example scenario in Fig. 4.

A. Intelligence Function Analysis

Usually, the type of intelligence under test should be planned before generating scenarios, e.g., pairwise collision avoidance, traffic light detection, etc. When only the scenario description is available, it is also straightforward to derive the relevant OEDR capabilities with the following empirical procedures. First, we gather the list of objects in the scenario, i.e., the cut-in vehicle *V1*, the jaywalking pedestrian *V2* and a speed sign *S1* (10 meters per second). Second, for each object, we define the list of possible events that the ego vehicle may encounter throughout the scenario, e.g., *V1* starts to cut in, and *V2* starts to walk across. Next, based on the identified objects and events, we define the logical intelligence functions that must be performed one after another to respond to the detected object or event. For example, in order to avoid collision with *V1*, the ego vehicle should logically i) detect the existence of *V1*, ii) recognize that *V1* is a vehicle, iii) keep track of and predict *V1*'s trajectory and iv) decelerate to yield to *V1*, with the latter function depending on the former. Eventually, we can derive the list of intelligence functions in Table. I.

TABLE I: OEDR behaviours derived from Fig. 4 Scenario, where $A \rightarrow B$ means A enables B (or B depends on A).

Object	Event	Intelligence Function
Cut-in Vehicle V1	Cut-in	Detect → Recognize → Predict → Decelerate to yield
Jaywalker V2	Jaywalk	Detect → Recognize → Predict → Decelerate to yield
Speed Sign S1	-	Detect → Recognize → Decelerate to obey

B. Intelligence Constraint Specification

Intelligence function constraints can be derived from the AV system's functional design requirements for those scenarios generated particularly for the AV system, or domain knowledge (e.g., traffic rules and common practices) for generic scenarios. Assume we have the following constraints:

1) Detect *V1* within 0.2 seconds after the scenario starts;

```
INTELLIGENCE CONSTRAINTS :
Functions:
[FA_ID] : [FA_NAME_ENUM] (object [OBJ_ID])? ([DETAIL_STR])?
(AND FA_ID] : ... )* END

Constraints:
[CF_ID] : ensure: [ (function FA_ID | formula CF_ID) (, OPERATOR_ENUM)?
                    (, (function FA_ID | formula CF_ID))? ] (from [IB_ID] to [IB_ID])?
(AND [CF_ID] : ... )*  END

Interval Boundaries:
[IB_ID] when
  # temporal boundary
  ([TIME_VAL] sec relative to (timer [TIMER_ID] | boundary [IB_ID]) ) |
  # function boundary
  (at (positive | negative) edge of function [FA_ID]) |
  # spatial boundary
  (object [OBJ_ID] at (
    # absolute position in coordinates
    (coordinate [ABS_X_VAL, ABS_Y_VAL, ABS_Z_VAL]) |
    # absolute position in map
    ((road [ROAD_ID] | junction [JUNCTION_ID]) lane [LANE_ID]
    (longitudinal offset of [LONG_OFFSET_VAL])?
    (lateral offset of [LAT_OFFSET_VAL])?) |
    # relative position
    ((longitudinal offset of [LONG_OFFSET_VAL])?
     (lateral offset of [LAT_OFFSET_VAL])? relative to [OBJ_ID]) ) ) |
  # phase boundary
  (phase [PHASE_ID] (starts | ends) ) |
  # event boundary
  (event [EVENT_ID] : (DETAIL_STR) occurs)
(AND [IB_ID] ...)* END
```

Fig. 6: The extension syntax implemented for WMG SDL Level 2 uses parenthesis *( )*, question mark *?* and asterisk * for enclosing, optional and zero or more instances, respectively. The terms XX_ID, XX_ENUM, XX_STR and XX_VAL represent the ID of a scenario element category, an enumeration option, a string and a numeric value, respectively. # comments are for ease of reading, not part of the syntax.

2) Recognize *V1* as a vehicle within 0.1 seconds after it is detected;
3) Predict the lane following trajectory of *V1* within 2 seconds after it is recognized;
4) Predict the lane changing trajectory of *V1* within 0.5 seconds after manoeuvre phase 2 starts;
5) Detect *S1* within 0.2 seconds when the relative distance between the ego and the sign is less than 50 meters;
6) Recognize the speed limit value as 10 meters per second within 0.1 seconds after *S1* is detected;
7) Decide to decelerate down to the speed limit within 0.1 seconds after *S1* is recognized;
8) Start to decelerate within 0.2 seconds after the deceleration decision is made;
9) Decide to (further) decelerate within 0.1 seconds after the lane-changing trajectory of *V1* is predicted;
10) Start to decelerate within 0.2 seconds after the (further) deceleration decision is made;

For brevity, the constraints for the pedestrian are omitted. Based on the above specifications, the function atoms, interval boundaries and constraints can be expressed in WMG SDL as presented in Fig. 7, Fig. 8, and Fig. 9, respectively.

```
Functions:
[FA_1]  : [Detect] object [V1] AND
[FA_2]  : [Recognize] object [V1] "as vehicle" AND
[FA_3]  : [Predict] object [V1] "lane following trajectory" AND
[FA_4]  : [Predict] object [V1] "lane changing trajectory" AND
[FA_5]  : [Detect] object [S1] AND
[FA_6]  : [Recognize] object [S1] "speed limit at 10" AND
[FA_7]  : [Decide] "deceleration for speed limit" AND
[FA_8]  : [Decelerate] "for speed limit" AND
[FA_9]  : [Decide] "deceleration for object V1" AND
[FA_10] : [Decelerate] "for V1" ... END
```

Fig. 7: Function specifications for Fig. 4 scenario.

```
Interval Boundaries:
# Global Timer T1-related boundary
[IB_T1_a] when [0.0] sec relative to timer [T1] AND
[IB_T1_b] when [0.2] sec relative to timer [T1] AND
# FA_1: Detect V1
[IB_V1_a] when [0.0] sec relative to positive edge of function [FA_1] AND
[IB_V1_b] when [0.1] sec relative to positive edge of function [FA_1] AND
[IB_V1_c] when [0.0] sec relative to negative edge of function [FA_1] AND
# FA_2: Recognize V1
[IB_V1_d] when [0.0] sec relative to positive edge of function [FA_2] AND
[IB_V1_e] when [2.0] sec relative to positive edge of function [FA_2] AND
[IB_V1_f] when [0.0] sec relative to negative edge of function [FA_2] AND
# FA_4: Predict V1 "lane changing trajectory"
[IB_V1_g] when [0.0] sec relative to positive edge of function [FA_4] AND
[IB_V1_h] when [0.1] sec relative to positive edge of function [FA_4] AND
# FA_9: Decide to yield to V1
[IB_V1_i] when [0.0] sec relative to positive edge of function [FA_9] AND
[IB_V1_j] when [0.2] sec relative to positive edge of function [FA_9] AND
# FA_3: Predict V1 "lane following trajectory"
[IB_V1_k] when [0.0] sec relative to positive edge of function [FA_3] AND
# Manoeuvre phase 2-related boundary
[IB_PH_2_a] when phase [2] starts AND
[IB_PH_2_b] when [0.5] sec relative to boundary [IB_PH_2_a] AND
[IB_PH_2_c] when phase [2] ends AND
# Speed limit sign S1-related boundary
[IB_S1_a] when object [Ego] at longitudinal offset of [-50] relative to [S1] AND
[IB_S1_b] when [0.2] sec relative to boundary [IB_S1_a] AND
[IB_S1_c] when [0.0] sec relative to positive edge of function [FA_5] AND
[IB_S1_d] when [0.1] sec relative to positive edge of function [FA_5] AND
[IB_S1_e] when [0.0] sec relative to positive edge of function [FA_6] AND
[IB_S1_f] when [0.1] sec relative to positive edge of function [FA_6] AND
[IB_S1_g] when [0.0] sec relative to positive edge of function [FA_7] AND
[IB_S1_h] when [0.2] sec relative to positive edge of function [FA_7] AND
... END
```

Fig. 8: Interval boundaries for Fig. 4 scenario.

```
Constraints:
# these constraints ensures the first execution of the intelligence functions
[CF_1] : ensure [FA_1, Finally] from [IB_T1_a] to [IB_T1_b] AND
[CF_2] : ensure [FA_2, Finally] from [IB_V1_a] to [IB_V1_b] AND
[CF_3] : ensure [FA_3, Finally] from [IB_V1_d] to [IB_V1_e] AND
[CF_4] : ensure [FA_4, Finally] from [IB_PH_2_a] to [IB_PH_2_b] AND
[CF_5] : ensure [FA_5, Finally] from [IB_S1_a] to [IB_S1_b] AND
[CF_6] : ensure [FA_6, Finally] from [IB_S1_c] to [IB_S1_d] AND
[CF_7] : ensure [FA_7, Finally] from [IB_S1_e] to [IB_S1_f] AND
[CF_8] : ensure [FA_8, Finally] from [IB_S1_g] to [IB_S1_h] AND
[CF_9] : ensure [FA_9, Finally] from [IB_V1_g] to [IB_V1_h] AND
[CF_10] : ensure [FA_10, Finally] from [IB_V1_i] to [IB_V1_j] AND
# these constraints ensures continuous execution of the intelligence functions
[CF_11] : ensure [FA_1, Globally] from [IB_V1_a] to [IB_PH_2_c] AND
[CF_12] : ensure [FA_2, Globally] from [IB_V1_d] to [IB_V1_c] AND
[CF_13] : ensure [FA_3, Until, FA_4] from [IB_V1_k] to [IB_V1_g] AND
[CF_14] : ensure [FA_4, Globally] from [IB_PH_2_a] to [IB_PH_2_b] AND
... END
```

Fig. 9: Constraint formulas for Fig. 4 scenario.

C. Key Observations

Observations on Functions Function atoms, like in Fig. 7, are predicates evaluated at each scenario execution time frame (assuming discrete time). For example, if vehicle *V1* is detected at every time frame; then FA_1 is true for that frame. In addition, function atoms should be distinguished by the "name" and "detail" attributes, e.g., FA_3 and FA_4 are different.

Observations on Intervals Assume all scenarios, progressing as per scripted manoeuvre phases, can terminate within a limited time. Then, each interval's start and end boundary can always be explicitly defined. Consequently, the interval boundaries often come in pairs, as shown in Fig. 8, especially when used to define the constraints w.r.t. function dependencies, e.g., CF_2, CF_3, and CF_(6-10). Note that there may exist multiple ways to define the same interval boundary, e.g., boundary IB_V1_b is equivalent to "*when [0.1] sec relative to boundary [IB_V1_a]*".

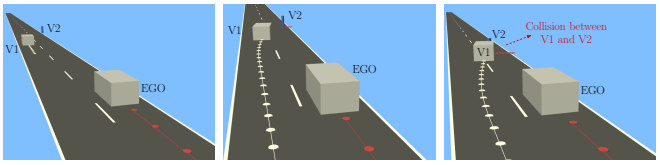
Observations on Constraints Constraints CF_(1-10) correspond to those defined in Section IV-B. The operator *Finally* is used to "ensure" the intelligence function succeeds as specified. However, this is insufficient as, e.g., CF_1 could be true if the vehicle *V1* is detected only for a single time frame. Hence, we define complementary constraints, i.e., CF_(11-14), to "ensure" continuous success of the intelligence functions from the moment they become active to the appropriate end boundaries, e.g., end of the scenario. For example, CF_1, CF_2, CF_11 and CF_12 together "ensure" the vehicle *V1* is detected and recognized correctly throughout the scenario.

In summary, the case study showcases the feasibility and flexibility of the proposed extension architecture. Note that the extension design remains extensible and embraces customization as new requirements arise.

V. DISCUSSION

Automatic Intelligence Constraint Specification We performed manual analysis in the case study. However, as the scenario description gets complex with more multi-phased actors, it becomes impractical, if not infeasible, to perform the manual analysis solely relying on the scenario description (without scenario execution). For example, regardless of why the scenario is generated, a scenario may contain 100 scripted actors, among which only 5 are relevant to the ego's DDT. In addition, unscripted events may develop from the scripted actor manoeuvres. For example, an unscripted collision event between the two background actors occurred in the second phase of the Fig. 4 scenario, as shown in Fig. 10. To handle such cases and foster scalability, scenario execution with the actual system-under-test or any surrogate models is inevitable and based on which result, automatic constraint specification techniques can be explored.

Customized Intelligence Constraint Interpretation Regardless of how the constraints are specified, it is up to the scenario engine for interpretation and evaluation. Since there are no regulatory standards on the taxonomy of autonomous driving intelligence, we can only rely on domain knowledge



(a) Phase 1 (b) Phase 2 (c) End

Fig. 10: Simulation of Fig. 4 scenario. The two background actors *V1* and *V2* collide during the second manoeuvre phase.

during the language extension and scenario engine implementation. Hence, the extension architecture shall remain extensible and embrace customization.

General vs Scenario-specific Constraint Selection Specifying the complete set of traffic rules in each scenario with the extension is possible. However, it is undoubtedly preferable to limit only to the scenario-specific OEDR behaviours to foster readability, reusability, and maintainability.

Granularity of Intelligence Function Definition In this work, we follow the same level of granularity as [29] when defining the intelligence functions, e.g., "Detect the vehicle", as it is the lowest abstraction level that's common to all AV systems. However, the function "Detect the vehicle" may be further decomposed into sub-functions, e.g., "*LiDAR point segmentation*" $\rightarrow$ "*LiDAR object detection*" $\rightarrow$ "*Image object detection*" $\rightarrow$ "*Multi-sensor fusion*". The division of intelligence functions into more granular components facilitates the root cause analysis; however, this granularity complicates the task of defining a comprehensive set of constraints for these functions. Moreover, generalizability may also suffer, e.g., not every AV uses the same LiDAR-Camera sensor fusion-based object detection stack. The trade-off needs to be balanced during the extension implementation.

VI. CONCLUSION

In this work, we propose a generic architecture to extend existing SDLs for the intelligence testing of ADSs. We implement the extension to the WMG SDL Level 2 language and discuss a case study in detail with an example scenario. The case study demonstrates the capability of the proposed extension architecture in defining flexible types of intelligence function constraints. Our key observations based on the case study and promising future directions are also discussed. We hope our work contributes as one of the key steps towards the intelligence-based scenario engineering researches.

ACKNOWLEDGEMENT

The work presented in this paper has been supported by UKRI Future Leaders Fellowship (Grant MR/S035176/1). The authors would like to thank the WMG center of HVM Catapult, and WMG, University of Warwick, UK for providing the necessary infrastructure for conducting this study.
For the purpose of open access, the author(s) has applied a Creative Commons Attribution (CC BY) license to any Accepted Manuscript version arising. No new data were created in this study.

REFERENCES

[1] W. Ding, C. Xu, M. Arief, H. Lin, B. Li, and D. Zhao, "A survey on safety-critical driving scenario generation—a methodological perspective," *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[2] Z. Zhong, Y. Tang, Y. Zhou, V. de Oliveira Neves, Y. Liu, and B. Ray, "A survey on scenario-based testing for automated driving systems in high-fidelity simulation," 2021.

[3] X. Zhang, J. Tao, K. Tan, M. Törngren, J. M. G. Sanchez, M. R. Ramli, X. Tao, M. Gyllenhammar, F. Wotawa, N. Mohan *et al.*, "Finding critical scenarios for automated driving systems: A systematic mapping study," *IEEE Transactions on Software Engineering*, vol. 49, no. 3, pp. 991–1026, 2022.

[4] N. Kalra and S. M. Paddock, "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?" *Transportation Research Part A: Policy and Practice*, vol. 94, pp. 182–193, 2016.

[5] X. Zhao, V. Robu, D. Flynn, K. Salako, and L. Strigini, "Assessing the safety and reliability of autonomous vehicles from road testing," in *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*. Berlin, Germany: IEEE, 2019, pp. 13–23.

[6] R. Xu, H. Xiang, X. Han, X. Xia, Z. Meng, C.-J. Chen, C. Correa-Jullian, and J. Ma, "The opencda open-source ecosystem for cooperative driving automation research," *IEEE Transactions on Intelligent Vehicles*, 2023.

[7] M. Kabtoul, M. Prédhumeau, A. Spalanzani, J. Dugdale, and P. Martinet, "How to evaluate the navigation of autonomous vehicles around pedestrians?" *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[8] N. Formosa, M. Quddus, M. K. Singh, C. K. Man, C. Morton, and C. B. Masera, "An experiment and simulation study on developing algorithms for cavs to navigate through roadworks," *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[9] R. Maier, L. Grabinger, D. Urlhart, and J. Mottok, "Causal models to support scenario-based testing of adas," *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[10] B. Zhu, Y. Sun, J. Zhao, J. Han, P. Zhang, and T. Fan, "A critical scenario search method for intelligent vehicle testing based on the social cognitive optimization algorithm," *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[11] Y. Zhu, J. Wang, X. Guo, F. Meng, and T. Liu, "Functional testing scenario library generation framework for connected and automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[12] S. Thal, P. Wallis, R. Henze, R. Hasegawa, H. Nakamura, S. Kitajima, and G. Abe, "Towards realistic, safety-critical and complete test case catalogs for safe automated driving in urban scenarios," in *2023 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2023, pp. 1–8.

[13] R. Song, X. Li, X. Zhao, M. Liu, J. Zhou, and F.-Y. Wang, "Identifying critical test scenarios for lane keeping assistance system using analytic hierarchy process and hierarchical clustering," *IEEE Transactions on Intelligent Vehicles*, 2023.

[14] X. Li, R. Song, J. Fan, M. Liu, and F.-Y. Wang, "Development and testing of advanced driver assistance systems through scenario-based system engineering," *IEEE Transactions on Intelligent Vehicles*, 2023.

[15] L. Balasubramanian, J. Wurst, M. Botsch, and K. Deng, "Open-world learning for traffic scenarios categorisation," *IEEE Transactions on Intelligent Vehicles*, 2023.

[16] Y. Zhang, C. Wang, R. Yu, L. Wang, W. Quan, Y. Gao, and P. Li, "The ad4che dataset and its application in typical congestion scenarios of traffic jam pilot systems," *IEEE Transactions on Intelligent Vehicles*, 2023.

[17] H. Zhang, J. Sun, and Y. Tian, "Accelerated risk assessment for highly automated vehicles: Surrogate-based monte carlo method," *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[18] J. Yang, H. Sun, H. He, Y. Zhang, H. X. Liu, and S. Feng, "Adaptive safety evaluation for connected and automated vehicles with sparse control variates," *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[19] X. Gong, S. Feng, and Y. Pan, "An adaptive multi-fidelity sampling framework for safety analysis of connected and automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[20] J. Zhou, L. Wang, and X. Wang, "Online adaptive generation of critical boundary scenarios for evaluation of autonomous vehicles," *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[21] P. S. Oruganti, E. Deosthale, and C. L. Jimenez, "Assessing safe autonomous vehicle behavior via large scale traffic simulation," in *2023 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2023, pp. 1–7.

[22] F. Scheuer, A. Gambi, and P. Arcaini, "Stretch: Generating challenging scenarios for testing collision avoidance systems," in *2023 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2023, pp. 1–6.

[23] H. Niu, K. Ren, Y. Xu, Z. Yang, Y. Lin, Y. Zhang, and J. Hu, "$(re)^2$h2o: Autonomous driving scenario generation via reversely regularized hybrid offline-and-online reinforcement learning," *arXiv preprint arXiv:2302.13726*, 2023.

[24] Z. Zhu, R. Philipp, Y. Zhao, C. Hungar, J. Pannek, and F. Howar, "Automatic disengagement scenario reconstruction based on urban test drives of automated vehicles," in *2023 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2023, pp. 1–8.

[25] E. Del Re and C. Olaverri-Monreal, "Method for comparison of surrogate safety measures in multi-vehicle scenarios," *arXiv preprint arXiv:2304.08998*, 2023.

[26] "GitHub - ApolloAuto/apollo: An open autonomous driving platform — github.com," https://github.com/ApolloAuto/apollo, [Accessed 11-01-2024].

[27] "GitHub - autowarefoundation/autoware: Autoware - the world's leading open-source software project for autonomous driving — github.com," https://github.com/autowarefoundation/autoware, [Accessed 11-01-2024].

[28] L. Chen, P. Wu, K. Chitta, B. Jaeger, A. Geiger, and H. Li, "End-to-end autonomous driving: Challenges and frontiers," *arXiv preprint arXiv:2306.16927*, 2023.

[29] L. Li, W.-L. Huang, Y. Liu, N.-N. Zheng, and F.-Y. Wang, "Intelligence testing for autonomous vehicles: A new approach," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 2, pp. 158–166, 2016.

[30] L. Li, N. Zheng, and F.-Y. Wang, "A theoretical foundation of intelligence testing and its application for intelligent vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 10, pp. 6297–6306, 2020.

[31] "ISO 34503:2023 — iso.org," https://www.iso.org/standard/78952.html, [Accessed 11-01-2024].

[32] X. Li, P. Ye, J. Li, Z. Liu, L. Cao, and F.-Y. Wang, "From features engineering to scenarios engineering for trustworthy ai: I&i, c&c, and v&v," *IEEE Intelligent Systems*, vol. 37, no. 4, pp. 18–26, 2022.

[33] T. Menzel, G. Bagschik, and M. Maurer, "Scenarios for development, test and validation of automated vehicles," in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 1821–1827.

[34] X. Zhang, S. Khastgir, and P. Jennings, "Scenario description language for automated driving systems: a two level abstraction approach," in *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2020, pp. 973–980.

[35] D. J. Fremont, T. Dreossi, S. Ghosh, X. Yue, A. L. Sangiovanni-Vincentelli, and S. A. Seshia, "Scenic: a language for scenario specification and scene generation," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2019, pp. 63–78.

[36] ASAM, "ASAM OpenSCENARIO® Standard," 2021. [Online]. Available: https://www.asam.net/standards/detail/openscenario/

[37] ——, "ASAM OpenDRIVE® Standard," 2021. [Online]. Available: https://www.asam.net/standards/detail/opendrive/

[38] W. U. of Warwick, "Safety pool - powered by deepen ai and wmg university of warwick," 2023, accessed on Aug 27, 2023. [Online]. Available: https://www.safetypool.ai/

[39] N. Arechiga, "Specifying safety of autonomous vehicles in signal temporal logic," in *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2019, pp. 58–63.

[40] "GitHub - esmini/esmini: a basic OpenSCENARIO player — github.com," https://github.com/esmini/esmini, [Accessed 16-01-2024].

[41] C. Team, "CARLA — carla.org," https://carla.org, [Accessed 16-01-2024].

[42] E. Thorn, S. C. Kimmel, M. Chaka, B. A. Hamilton *et al.*, "A framework for automated driving system testable cases and scenarios," United States. Department of Transportation. National Highway Traffic Safety, Tech. Rep., 2018.

[43] M. Spoenemann, "Xtext - Language Engineering Made Easy! — eclipse.dev," https://eclipse.dev/Xtext/, [Accessed 28-01-2024].