Formalisation and Implementation of Road Junction Rules on an Autonomous Vehicle Modelled as an Agent

Gleifer Vaz Alves[1] [0000−0002−5937−8193], Louise Dennis[2] [0000−0003−1426−1896], and Michael Fisher[2] [0000−0002−0875−3862]

[1] UTFPR - Universidade Tecnológica Federal do Paraná - Brazil
gleifer@utfpr.edu.br
[2] Department of Computer Science, Univ. Liverpool, UK
{L.A.Dennis, MFisher}@liverpool.ac.uk

Abstract. The design of autonomous vehicles includes obstacle detection and avoidance, route planning, speed control, etc. However, there is a lack of an explicitely representation of the rules of the road on an autonomous vehicle. Additionally, it is necessary to understand the behaviour of an autonomous vehicle in order to check whether or not it works according to the rules of the road. Here, we propose an agent-based architecture to embed the rules of the road into an agent representing the behaviour of an autonomous vehicle. We use temporal logic to formally represent the rules of the road in a way it should be possible to capture when and how a given rule of the road can be applied. Our contributions include: i. suggestion of changes in the rules of the road; ii. representation of rules in a suitable way for an autonomous vehicle agent; iii. dealing with indeterminate terms in the Highway Code.

Keywords: Agent · Autonomous Vehicles · Temporal Logic · Rules of the Road.

1 Introduction

Usually, the design of current control software in autonomous vehicle does not explicitely implement the rules of the road. Here, we propose an architecture, where an agent represents the behaviour of an autonomous vehicle and temporal logic is used to formally specify a subset from the rules of the road. With this, we aim to formally verify that an agent endowed with the rules of the road actually respects the flow of traffic without any sort of conflicts, inconsistency or redundancies in the use of the rules.

Autonomous Vehicles and the Rules of the Road: One can easily enumerate possible advantages the deployment of autonomous vehicles may bring to cities, e.g. reduce indices of traffic congestion, driver inactivity, and also the number of accidents [11]. With this in mind, several companies have been working towards the goal of launching (fully) autonomous vehicles on our roads on a daily basis.

Predictions as to when (fully) autonomous vehicles will appear on our roads vary (e.g., [2] and [16]), but it expected to be within the next 5 years. However, there are plenty of questions which should be addressed in order to have these vehicles driving safely on the roads. The design of an autonomous vehicle must consider obstacle detection and avoidance, route planning, safety, speed control, etc. But, how about the road rules: is an autonomous vehicle behaviour adapted to the road rules? Prakken mentions in [15], that his work seems one of the first towards the comprehension of how an autonomous vehicle design should be established in accordance with the road rules. Nevertheless, Prakken's approach is a conceptual approach without addressing either implementation or formal verification. Previously [1], we have presented the first steps towards the formalisation of the rules of the road. In [17], Vellinga also discusses the necessity to understand how the road rules should be adapted into an autonomous vehicle. The author presents road rules from California (USA), the UK and the Netherlands.

In the UK, the government has also shown concern about the regulation of autonomous vehicles. That is why the Law Commission in the UK has released (Nov. 2018) a consultation paper in order to review the regulatory framework for the safe deployment of autonomous vehicles [12]. In this review, different topics are addressed, including the Highway code which is responsible for determining the so-called Rules of the Road in the UK. This set of rules establishes how one should use the road for overtaking, road junctions, pedestrian crossing, and so on [8]. Moreover, in June 2019, the Law Commission made available a summary of the responses concerning the aforementioned consultation paper [12]. Among the presented topics we highlight the adaption of the road rules, which according to the document should address the following issues (among others):

1. Apply analogue driving rules into a digital highway code (section 6.1 in [12]).
2. Struggle to determine a digital highway code that sets precise rules for every instance. In the document, it is mentioned that: *"is impossible to predict all future scenarios in advance ... it is not desirable nor realistic to ask developers to deterministically prescribe the behaviour of automated driving systems in advance for every scenario"* (section 6.5 in [12]).
3. Establish a forum on the application of road rules to autonomous vehicles, some possible scenarios which should be considered are (section 6.7 in [12]):
   (a) interpretation of indeterminate terms in legislation and in the Highway code, e.g., *road users should take extra care*, or *there is a safe gap large enough.*
   (b) identify possible additions to the Highway code to resolve conflicts involving autonomous vehicles. As mentioned in [12] (footnote 7 on page 12), usually conflicts are resolved through human communication. As an example, a human driver may use hand and arm gestures to give way for another human driver in a road junction.

Autonomy, Agents and Formal Verification: According to Herrmann et al. [11] an automated vehicle includes several stages of automation, where there is a person in the loop, at least in order to handle specific traffic scenarios, e.g. an emergency situation. On the other hand, in an autonomous vehicle, a person is out of the loop and the system is responsible for all driving tasks everywhere and at all times. For the sake of clarity, in our paper we use the term *autonomous vehicle* to represent the vehicle modelled in our system, which indeed is described by means of an intelligent agent [18]. An intelligent agent can be easily used to represent the behaviour of an autonomous vehicle, as we have previously presented in [9], where an agent is endowed with strategies to avoid obstacles in a simulated environment.

Agent-based modelling is a suitable approach to represent high-level decisions of an autonomous system. As illustrated by Marks, in [13], there are several layers in an autonomous vehicle stack. Here we are mainly concerned with the Reasoning and Decision Layer. As a result, the low-level layers, like Sensor, Localization and Control layers are out of our scope. Moreover, an agent programming approach is indeed a reasonable technique when we take into account the code complexity for vehicles. In 2010, some vehicles had ten million software lines of code (SLOC). In 2016, the SLOC number has increased to around 150 million [4]. Thus, agent-oriented programming could be seen as a suitable approach for the high-level decisions of an autonomous vehicle. Usually, a program written in an agent programming language has fewer lines than (the same program written) in other general-purpose languages and also an agent language is a good choice for prototyping.

In our work we use the GWENDOLEN agent programming language [6] in order to implement a BDI (*Belief-Desire-Intention*) agent [3] to capture the behaviour of an autonomous vehicle. By using GWENDOLEN, we can also take advantage of the *Model Checking Agent Programming Language* (MCAPL) framework [7], where the *Agent Java Path Finder* (AJPF) model checker can be used to formally verify the behaviour of an intelligent agent. When comparing to other techniques, like machine learning, by using a model checking agent-based architecture, we intend to avoid the so-called *black box* problem [5], i.e. the lack of transparency to control and understand the decision-making process. Notice that by doing the formal verification of a GWENDOLEN agent is possible to give the explicit reasons that the agent has used to select a given decision.

BDI Agent: An agent program language which implements a BDI agent usually has the following structure for an agent plan:

```
trigger_event : guard <- body
```

Where a given agent may have different plans in order to achieve a certain goal. Using our translation we may establish the following mapping:

- Goal is determined by the specific road junction rule.
- The `trigger_event` is given by a new belief or a goal.
- The `guard` is defined by a set of beliefs.
- The `body` is represented as a set of actions.

Contributions: Our major goal is the proposal of an agent-based architecture in order to embed the rules of the road in an agent representing the high-level decisions of an autonomous vehicle. With this in mind, we point out the following questions: **i.** How can we handle the use of ambiguous terms in the road rules, when embedding these rules into an autonomous vehicle? **ii.** How can we formally verify the behaviour of an autonomous system endowed with the rules of the road? **iii.** By having a simple and direct mapping of the Highway code into a digital highway code, can we say it is enough to have an autonomous vehicle driving safely on the roads? Notice that here we only address the first question.

In this paper, we extend the formalisation proposed in [1] by setting up a language and a grammar for the road junction rules together with an agent-based architecture capable of capturing the behaviour of an autonomous vehicle in an urban traffic environment. Moreover, we establish a translation of road junction rules written in Temporal logic into a BDI agent plan. As an instance of our architecture, we present the formalisation and implementation of a given road rule from the UK Highway Code.

2 Road Junction Rules: language and grammar

In this section we present the so-called RoR language and grammar created to represent the Rules of the Road. The RoR language is used in the next section to formalise the Rules of the Road. We intend that our language should be expressive enough to represent the rules of the road (specifically the road junction rules), but also as simple as possible. As it follows we present the operators, terms, and actions from the RoR language, which is based on Linear Temporal Logic (LTL) [10].

2.1 Operators and Constants

- Operators from LTL:
  $\wedge$, $\vee$, $\rightarrow$, $\neg$.
  $\Box$, $\Diamond$, $\circ$, $\cup$.
- where:
  - $\Box$: *always*
  - $\Diamond$: *eventually*
  - $\circ$: *next*
  - $\cup$: *until*
- Constants: `True`, `False`.

2.2 Terms: agent and objects

Terms in the RoR language are used together with actions. We have two kinds of terms: Agent and Objects.

- Agent (Ag) defines the agent who has an active role in a given action (and road junction rule).

– Objects (Obj) represents the different objects that can be used in an action to represent the elements of a given road junction rule. There are four different kinds of objects. The first three concrete and the last is an abstract object:

  Space: establishes the environment where a given road rule occurs.
  Dynamic: determines the dynamic objects used in a rule that are situated in an environment.
  Static: defines the static objects used in a rule that are situated in an environment.
  Abstract: represents the abstract notions which are related to a road rule.

As it follows, we present the Agent and Objects in the RoR language.

Agent

– Autonomous Vehicle: represents an intelligent agent conducting the vehicle. Abbreviation: `AV`.

Objects

– Concrete Space Objects

  Junction: a junction between two or more roads. Abbreviation: `JC`.
  Road: a road that usually has a single traffic direction. Abbreviation: `RO`.
  Main Road: the main road has both traffic directions. Abbreviation: `MR`.
  Lane: a road or the main road may be divided by two or more lanes. Abbreviation: `LA`.
  Filter Lane: a filter lane is a special lane used to guide the driver to turn in a road. Abbreviation: `FL`.
  Central Reservation: a central reservation on a dual carriageway is used by a car to wait for the safe moment to cross a road. Abbreviation: `CR`.
  Box Junction: a box junction has criss-cross yellow lines painted on the road. Abbreviation: `BJ`.
  Box Junction at Signalled Roundabouts: Similar to Box Junctions, but with signalled roundabouts. Abbreviation: `BJS`.

– Concrete Dynamic Objects

  Road User: a road user can be any of the following, another vehicle, pedestrians, cyclists, motorcyclists, powered wheelchairs, mobility scooters or horse rider. Abbreviation: `RU`.
  Long Vehicles: a long vehicle can be a bus, a lorry or a truck. Abbreviation: `LV`.

– Concrete Static Objects

  Stop sign or Solid white line across the road: both are signs which means that you should stop at a junction. Abbreviation: `ST`.
  Give way sign or Triangle marked on the road: both are signs which means you should give way to traffic. Abbreviation: `GW`.

Broken white lines across the road: also means to give way traffic, but when you are emerging from a junction on the main road. Abbreviation: `BWL`.
Traffic light: a Traffic light which may have a Green, Red or Amber light. Abbreviation: `TL`.
Green Light: on a traffic light. Abbreviation: `GL`.
Amber Light: on a traffic light. Abbreviation: `AL`.
Red Light: on a traffic light. Abbreviation: `RL`.
Advanced stop line: some signal-controlled junctions have advanced stop lines to allow cycles to be positioned ahead of other traffic. Additionally, an Advanced stop line has two lines marking its area, the so-called: First White Line and Second White Line. Abbreviation: `AD`.
First White Line. Abbreviation: `FWL`.
Second White Line. Abbreviation: `SWL`.
Mirrors: a driver is supposed to use the mirrors of his vehicle to observe the traffic. Abbreviation: `MI`.

– Abstract Objects

Safe Gap: usually when turning on a junction a driver is supposed to verify if there is a Safe Gap for the vehicle on the road. Abbreviation: `SG`.

  * NB: for the sake of simplicity of RoR language, it is used `SG` to represent not only the previous description, but also any situation where the `AV` needs to take extra care when turning on any kind of junction, crossing roads, crossing a box junction, waiting in a lane for turning right or left, among others. That is why `SG` is used in several road junction rules.

Blind Spot: when waiting to cross the main road it may be necessary to check for blind spots. Abbreviation: `BS`.
Possible Collision: in some very specific rules some exceptions are allowed but if and only if a collision may occur in a given road junction environment. Abbreviation: `PC`.
Oncoming Traffic: in some scenarios, it might be necessary to look for oncoming traffic in a corresponding environment. Abbreviation: `OT`.
Behind: when preparing to turn into a junction it is necessary to look behind (possibly using the mirrors) for oncoming traffic. Or the driver can also be turning right behind another vehicle which is also turning right in the same junction. Abbreviation: `BH`.
Front: the driver can turn right in front of another vehicle which is also turning right at the same junction. Abbreviation: `FR`.
Both Directions: when waiting to cross the main road it may be necessary to watch out for traffic in both directions on the main road. Abbreviation: `BD`.

2.3 Actions

Definition 1 (Action). *An Action is given by an action name followed by one of two different tuples (with three or two elements) and optional pre and post-conditions:*

`<pre>` *`action_name`* (t_1, t_2, t_3) `<post>`

– *where,*
t_1 is an Agent.
t_2 is a Concrete Space Object.
t_3 can be a Concrete Static or Dynamic Object; or an Abstract Object; or it can be empty.
`<pre>` *some actions require the so-called pre-conditions which should be satisfied for a given action to be applied. We use flags (*`True` *or* `False`*) to represent a given pre-condition from an action,* `True` *means the action can be applied,* `False` *means the action should not be applied.*
`<post>` *similarly there are the so-called post-conditions which represent the application result from a given action.* `True` *means the action has been successfully achieved,* `False` *means the action has not been successfully achieved.*

Notice pre and post-conditions are both optional since not every rule demands this sort of additional context related to the effect of a rule application.

`<pre>` *`action_name`* (t_1, t_2) `<post>`

– *where,*
t_1 can be a Concrete Static or Dynamic Object; or an Abstract Object.
t_2 is a Concrete Space Object.

List of actions We have defined the following list of action names:

```
stop, wait, give-way, cross, enter, exit,
turn-right, turn-left, give-right-signal, give-left-signal,
exists, overtake, turn-keep-left-lane, watch.
```

Example of actions We present four examples of different kinds of actions.

`enter(AV,JC)`

- the action `enter` is given by a tuple with two elements: `AV` represents an Autonomous Vehicle Agent and `JC` represents a Concrete Space Object, the Junction. This action can be read as: "*an AV is supposed to enter when it is at the Junction*".

`watch(AV,JC,RU)`

- the action `watch` is given by a tuple with three elements: `AV` and `JC` represent the same elements from previous action, and `RU` represents a Concrete Dynamic Object, the Road User. This action can be read as: "*an AV is supposed to watch out for Road Users, when it is at the Junction*".

`cross(RU,JC) <False>`

- the action `cross` is given by a tuple with two elements and a post-condition: `RU` represents a Concrete Dynamic Object, the Road User; and `JC` a Concrete Space Object, the Junction. The flag "`False`" is used as a post-condition. This action can be read as: "*A Road User is supposed to cross, when it is at a Junction. According to the post-condition, there is no Road User crossing at the Junction*".

`exists(SG,JC) <True>`

- the action `exists` is also given by a tuple with two elements and a post-condition: `SG` represents an Abstract Object, the Safe Gap; and `JC` represents the Junction. The flag "`True`" is used as a post-condition. This action can be read as: "*A Safe Gap is supposed to exist, and it is at the Junction. According to the post-condition, there is indeed a Safe Gap at the Junction*".

2.4 Grammar

As it follows we present the grammar for RoR language. The grammar is defined to represent the road junction rules and it is presented using Extended Backus-Naur Form (EBNF) style [14].

```
road_junction_rule = context "->" result ;
context = "□" [op_unary] action { op_binary [op_unary] action } ;
result = ["◊"] [op_unary] action |
 ["◊"] [op_unary] action op_binary [op_unary] ["◊"] action ;
op_binary = "∧" | "∨" | "∪" ;
op_unary = "¬" | "○" ;
action =  ["<"pre">"] action_name tuple ["<"post">"] ;
```

– Notice that in this EBNF style grammar the following notation is used:
  `=` represents definition.
  `;` represents termination.
  `[ ... ]` represents optional.
  `{ ... }` represents repetition.
  `" ... "` represents terminal string.

The grammar determines that a road junction rule has a `context`, followed by the "→" operator and terminated by a `result`. A `context` always starts with the operator "□" followed by at least one `action`. While a `result` may have the "◊" operator with a single `action`, or a pair of `actions`.

3 Formalising the Road Junction Rules

The UK Highway code presents the road rules [8]. Here, we address a subset representing the road junction rules. This subset comprises 14 rules which deal with stop signs, traffic lights, turning right and left, crossroads and also watching out for a road user. As an example, we show `rule 170`[3], which establishes the requirements for a driver to enter or wait at a road junction. As it follows, a fragment of `rule 170` is described as seen in [8]. Next, a formal representation is given using LTL.

- `Rule 170` from Road Junction Highway Code:
  - You Should watch out for road users (`RU`) (cyclists, motorcyclists, powered wheelchairs/mobility scooters and pedestrians).
  - watch out for pedestrians crossing a road junction (`JC`) into which you are turning.
  - look all around before emerging[4]. Do not cross or join a road until there is a safe gap (`SG`) large enough for you to do so safely.
- `Rule 170`: represented in LTL, when the autonomous vehicle (`AV`) may **enter** the junction (`JC`):

$$\Box\ (\texttt{watch(AV, JC, RU)} \cup \texttt{cross(RU, JC) <False>} \wedge \texttt{exists(SG, JC) <True>)} \rightarrow \Diamond \texttt{enter(AV, JC)}$$

  - `Rule 170 - description`: *it is always the case that the `AV` is supposed to watch for any road users (`RU`) at the junction (`JC`) until there are no road users crossing the junction (`JC`) and also there is a safe gap (`SG`). As a result, at some time the `AV` may enter the junction.*

4 From a Road Junction rule towards a BDI agent plan

In this section, we describe how the formalisation presented in Section 3 can be used in the translation towards BDI agent plans. Through such a translation, we intend to better bridge the gap between the rules of the road and the agent implementation. Additionally, this translation could be used as a first step in the implementation of these rules in a BDI agent programming language different from Gwendolen.

Our translation process is executed through different cases according to the possible actions used to describe the road junction rules.

Here `<pre>` and `<post>` are used as an effect on the given action. In the case of a pre-condition it indicates there is a previous belief that should be satisfied for the given action to be applied. In the case of a post-condition, the application of the action will result in a new belief. Thus, both pre and post-conditions represent beliefs when translated into BDI plans, as shown in the following definitions.

`<pre> action_name` (t_1, t_2, t_3) `<post>`

[3] LTL representation of road junction rules: https://github.com/laca-is/SAE-RoR.

[4] For the sake of clarity of the rules of the road language, we choose to use the term `enter` as an action which represents not only a driver entering a road junction, but also emerging from a road junction to another road.

`<pre> action_name`: the action becomes a belief for the agent.
t_1 as an Agent is translated as an agent.
t_2 as a Space Object is translated as a belief from the environment.
t_3 as an Abstract or Static or Dynamic Object is translated as a belief from the environment.
`<post> action_name`: the action result becomes a belief for the agent.
`action_name`: is the own action which should be part of the body plan in the agent.

`<pre> action_name` (t_1, t_2) `<post>`
`<pre>` and `<post>` will be translated in the same way of the previous case.
t_1 as an Abstract or Static or a Dynamic Object is translated as belief from the environment.
t_2 as a Space Object is translated as a belief from the environment.
`action_name`: is the own action which should be part of the body plan in the agent.

Now, to illustrate the translation, we show an example using the Rule 170, previously seen in Section 3.

– The Goal is given by Rule 170, which is to *enter at the junction*.
– From the four actions written in the formalisation of Rule 170, we extract the following elements, which will be used to create different agent plans with the corresponding trigger events, guards, and body plans.
– We consider that a LTL rules of the road is a flow of actions when translated into agent plans:
  • The flow of action starts with those actions from the "Context" (see Grammar at subsection 2.4), *action-1* will obtain new beliefs (from the agent environment) that will be used as guards for *action-2*; *action-2* will obtain new beliefs used as guards for *action-3*; this goes on until the "Result" action , which can use all beliefs obtained by previous actions.
– Our translation target is the BDI structure (trigger event, guards, body) previously seen in Section 1.

– First action (from the Context):
`watch(AV, JC, RU)`
AV is the Agent.
JC is a belief at Junction from the environment.
RU is a belief there is a road user.
`watch` is the action watch implemented in the environment.
  • Translation target:

```
enter-junction : (empty) <- watch
```

the trigger event is named `enter-junction`, this is the AV-agent goal (this same trigger event is used for all plans).
since this is the first action (in the flow of actions), there is no beliefs obtained from the environment, that is the reason the guards are empty.

  - `watch` is the action used in the body of agent plan (all following actions will be used similarly).

- Second action (from the Context):
  `cross(RU, JC) <False>`

  RU is a belief there is a road user.
  JC is a belief at Junction from the environment.
  `cross(RU, JC) <False>` is a belief related to *RU* and *JC* that there is no road user crossing at the junction.
  `cross` is the action cross implemented in the environment.

  - Translation target:

    ```
    enter-junction : JC <- cross
    ```

    now, the second action (`cross`) has obtained a belief (`JC` - Junction) from the environment, which is used as guard in the plan.

- Third action (from the Context):
  `exists(SG, JC) <True>`

  SG is a belief on safe gap from the environment.
  JC is a belief at Junction from the environment.
  `exists(SG, JC) <True>` this flag has effect in the action and in both elements from the tuple, thus we say: "*there is a new belief that exists a Safe Gap at the Junction.*"
  `exists` is the action exists implemented in the environment

  - Translation target:

    ```
    enter-junction : JC, RU <- exists
    ```

    next, the third action (`exists`) still has the same previous guard (`JC`) and obtains a new belief (`RU`, there is no road user), used as a second guard in the agent plan.

- Fourth action (from the Result):
  `enter(AV, JC)`

  AV is the Agent.
  JC is a belief at Junction from the environment.
  `enter` is the action enter implemented in the environment.

  - Translation target:

    ```
    enter-junction : JC, RU, SG <- enter
    ```

    at last, the "Result" action (`enter`) keeps the previous guards and adds a third one, `SG` (Safe Gap), obtained from the third action application.

Notice that when implementing the agent plans in Gwendolen (as next section will present), we have changed some details in the code, but only to have a clear BDI syntax code. The main target objects obtained from the translation are all used in the agent BDI plans.

5 Simulated Automotive Environment for the Rules of the Road

Our proposed architecture (see Fig. 1) is named SAE-RoR (*Simulated Automotive Environment for the Rules of the Road*).

In Fig. 1, the semantics of the arrows represent data. The road junction rules are formally represented using a grammar. With this representation we are able to embed the rules into a GWENDOLEN agent and also formally verify related properties written in LTL using AJPF. Moreover, the agent updates its beliefs from perceptions obtained from the Urban Traffic Environment. This will determine the set of plans that should be executed and sends back to the environment the corresponding actions, e.g. `watch` and `enter` (a road junction).

As an example of a property that could be formally verified is given below (in natural language):

> "*It is always the case that when there is a road user crossing a road junction and/or there is no safe gap at the junction, then the AV-agent will not enter the junction.*"

Notice that our intention is not simply checking this property considering the `rule 170`, but also verify it according to the whole set of road junction rules. In a way we could check if there is any conflict, inconsistency or redundancies among the road junction rules.

Code 1.1 presents a fragment of our AV-agent, which implements a subset of plans from `rule 170`, and is responsible for achieving the goal of entering a road junction (as seen in Section 3).

Listing 1.1. AV-agent plans for `Rule 170`

```
:Initial Goals:

want_enter_junction [achieve]

:Plans:

+!want_enter_junction [achieve] : { B to_watch(X,Y) } <-
watch(X,Y);
+!want_enter_junction [achieve] : { B junction } <-
check_cross(X,Y);
+!want_enter_junction [achieve] : { B junction ,
B road_user(X,Y) } <- wait , give_way , check_cross(X,Y);
+!want_enter_junction [achieve] : { B junction ,
B no_road_user(X,Y) } <- check_safe_gap(X,Y);
+!want_enter_junction [achieve] : { B junction ,
B no_road_user(X,Y) , B safe_gap(X,Y) } <- enter;
```

Notice the AV-agent has a sequence of five plans representing the stages of `rule 170`.

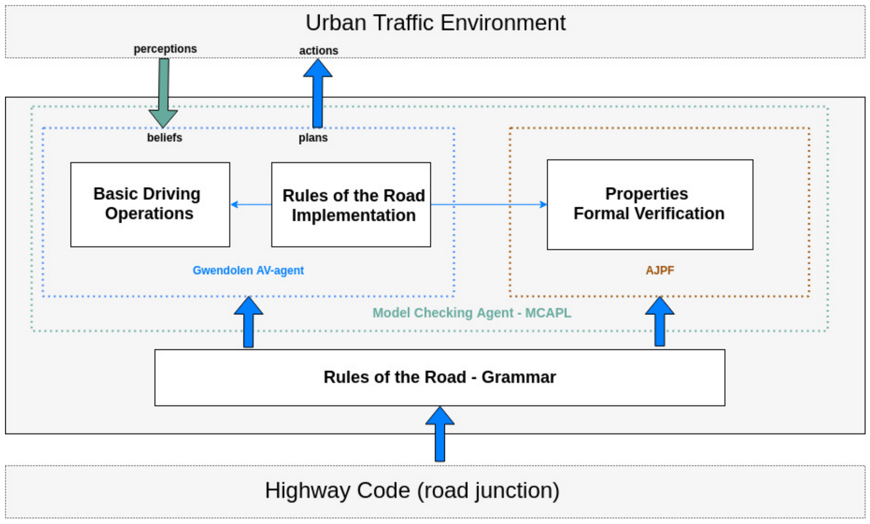
Urban Traffic Environment
perceptions
actions
beliefs
plans
Basic Driving Operations
Rules of the Road Implementation
Properties Formal Verification
Gwendolen AV-agent
AJPF
Model Checking Agent - MCAPL
Rules of the Road - Grammar
Highway Code (road junction)

Fig. 1. SAE-RoR Architecture

1. When the agent believes that it should watch for something in the environment (*represented by* (X, Y)), then it should watch out for road users.
   - `want-enter-junction` is the agent goal it should be achieved.
   - `(X,Y)` represents a position in the environment; the agent receives perceptions placed at this position.
   - `B` in GWENDOLEN stands for an agent Belief.
   - `watch(X,Y)` is an action taken by the agent, which has an effect in the environment (*notice that all actions have some effect in the environment*).
2. When the agent believes that it is at the junction, it should check if there is a road user crossing the junction.
   - `check-cross(X,Y)` is an action taken by the agent.
3. When the agent believes that it is at the junction and there is a road user crossing it, then it shoud wait, give way and check the junction again.
   - `wait`, `give-way` are actions taken by the agent.
4. When the agent believes that it is at the junction and there is no road user crossing it, then it needs to check for a safe gap.
   - `check-safe-gap(X,Y)` is an action taken by the agent.
5. When the agent believes that it is at the junction, there is no road user crossing it, and there is a safe gap, then it may enter the junction.
   - `enter` is an action taken by the agent.

6 Final Remarks

We have proposed an agent-based architecture which represents the rules of the road (a subset including the road junction rules) from the UK Highway Code. The rules are formalised using LTL and implemented in a GWENDOLEN agent. Notice that it is an ongoing work and we intend to produce a complete implementation and formal verification of the road junction rules in a forthcoming work, where we will use the MCAPL model checker in order to formally verify the behaviour of the AV-agent.

The Highway Code uses different terms to represent the same sort of concepts. An example can be given with the Safe Gap term (an Abstract Object as seen in subsection 2.2), which indeed is used as a meaning for different terms in the rules of the road. This sort of abstraction is necessary to create a language simple enough and suitable for an AV-agent.

Some rules from the Highway Code overlap each other. An example is the rules 175 and 176, both of which handle traffic lights scenarios. With the LTL formalisation it is possible to analyse such rules and find out that the desired outcome for a digital highway code should include a new rule which combines the main elements from rules 175 and 176. This is suggested because an AV-agent requires consistent and non-ambiguous information in order to build the agent plans (which includes beliefs and actions).

With the SAE-RoR architecture we shall be able to answer the three questions previously presented in the first section: **i**. Use a (formal) grammar and LTL to represent the objects and actions from the Highway Code in order to deal with ambiguity issues. **ii**. Apply model checking with AJPF in order to formally verify the behaviour of the AV-agent, in a way one can check the agent acts according to the expected flow of traffic without conflicts, inconsistency or redundancies when using the rules of the road. **iii**. The direct mapping of the Highway code into a digital version of it does not seem to be enough because some rules may overlap (e.g., rules 175 and 176) and also the AV-agent requires (in some scenarios) additional road context in order to implement a decision-making process. (Notice that we intend to properly answer questions **ii** and **iii** in a forthcoming work.) Indeed, as mentioned in [13], the road context can be used, for example, to determine when a vehicle is nearby a school at a specific time, then it should watch out for children. As future work, we aim to include in our architecture a component responsible for perceiving and extracting the relevant road context in a way that the AV-agent can obtain a new set of beliefs from the road junction rules plus a given road context.

Acknowledgements

Work partially supported through EPSRC research project Verifiable Autonomy [EP/L024845].

The authors kindly thank the anonymous reviewers for the insightful suggestions which have considerably improved the quality of our paper.

References

1. Alves, G.V., Dennis, L., Fisher, M.: Formalisation of the Rules of the Road for embedding into an Autonomous Vehicle Agent. In: International Workshop on Verification and Validation of Autonomous Systems. pp. 1–2. Oxford, UK (Jul 2018), https://sites.google.com/site/wsvavas2018/home/proceedings
2. BBC News: UK wants fully autonomous cars on road (Feb 2019), https://www.bbc.com/news/technology-47144449
3. Bratman, M.E.: Intentions, Plans, and Practical Reason. Harvard University Press (1987)
4. Burkacky, O., Deichmann, J., Doll, G., Knochenhauer, C.: Rethinking car software and electronics architecture | McKinsey, https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/rethinking-car-software-and-electronics-architecture
5. Davey, T.: Towards a Code of Ethics in Artificial Intelligence with Paula Boddington (Jul 2017), https://futureoflife.org/2017/07/31/towards-a-code-of-ethics-in-artificial-intelligence/
6. Dennis, L.A.: Gwendolen semantics: 2017. Tech. Rep. ULCS-17-001, University of Liverpool, Department of Computer Science (2017)
7. Dennis, L.A., Fisher, M., Webster, M.P., Bordini, R.H.: Model Checking Agent Programming Languages. Automated Software Engineering **19**(1), 5–63 (Mar 2012). https://doi.org/10.1007/s10515-011-0088-x, https://doi.org/10.1007/s10515-011-0088-x
8. Department for Transport: Using the road (159 to 203) - The Highway Code - Guidance - GOV.UK (2017), https://www.gov.uk/guidance/the-highway-code/using-the-road-159-to-203
9. Fernandes, L.E.R., Custodio, V., Alves, G.V., Fisher, M.: A Rational Agent Controlling an Autonomous Vehicle: Implementation and Formal Verification. In: Bulwahn, L., Kamali, M., Linker, S. (eds.) Proceedings First Workshop on Formal Verification of Autonomous Vehicles. Electronic Proceedings in Theoretical Computer Science, vol. 257, pp. 35–42 (2017). https://doi.org/10.4204/EPTCS.257.5
10. Fisher, M.: An Introduction to Practical Formal Methods Using Temporal Logic. Wiley (2011). https://doi.org/10.1002/9781119991472, http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470027886.html
11. Herrmann, A., Brenner, W., Stadler, R.: Autonomous driving: how the driverless revolution will change the world. Emerald Publishing, first edition edn. (2018)
12. Law Commission - UK: Automated Vehicles: Summary of the Analysis of Responses to the Preliminary Consultation Paper (Jun 2019), https://www.lawcom.gov.uk/project/automated-vehicles/
13. Marks, J.: How to ensure the safety of Self-Driving Cars: Part 1/5 (Jun 2018), https://medium.com/@olley_io/how-to-ensure-the-safety-of-self-driving-cars-part-1-5-2fcc891ea90b
14. Pattis, R.E.: Teaching EBNF First in CS 1. In: Proceedings of the Twenty-fifth SIGCSE Symposium on Computer Science Education. pp. 300–303. SIGCSE '94, ACM, New York, NY, USA (1994), http://doi.acm.org/10.1145/191029.191155, event-place: Phoenix, Arizona, USA
15. Prakken, H.: On the problem of making autonomous vehicles conform to traffic law. Artificial Intelligence and Law **25**(3), 341–363 (Sep 2017). https://doi.org/10.1007/s10506-017-9210-0, https://link.springer.com/article/10.1007/s10506-017-9210-0

16. Southworth, P.: Driverless cars to be on Britain's roads by the end of the year, government reveals. The Telegraph (Feb 2019), https://www.telegraph.co.uk/news/2019/02/06/driverless-cars-britains-roads-end-year-government-reveals/
17. Vellinga, N.E.: From the testing to the deployment of self-driving cars: Legal challenges to policymakers on the road ahead. Computer Law & Security Review **33**(6), 847–863 (Dec 2017). https://doi.org/10.1016/j.clsr.2017.05.006, http://www.sciencedirect.com/science/article/pii/S0267364917301334
18. Wooldridge, M., Rao, A.: Foundations of Rational Agency, Applied Logic Series, vol. 14. Springer Netherlands (1999). https://doi.org/10.1007/978-94-015-9204-8, http://www.springer.com/gp/book/9780792356011