General Game Playing for Managing Autonomous Vehicle Traffic

Michael Thielscher
University of New South Wales, Australia

Dongmo Zhang
Western Sydney University, Australia

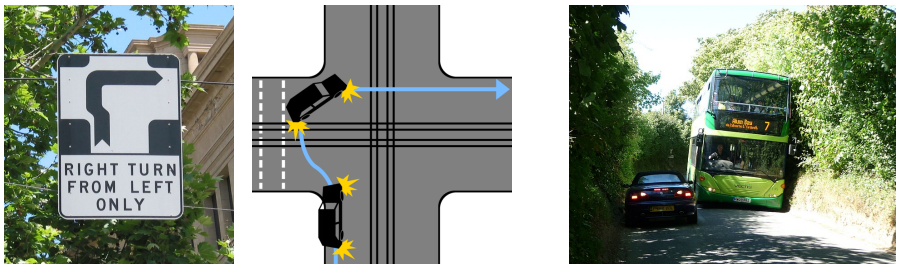
The emergence of self-driving cars has raised a significant research challenge for AI: How can autonomous vehicles, all by themselves, adapt to the dynamics of road regulation and unforeseen traffic situations? This paper explores a new approach to this problem by building upon existing techniques from General Game Playing. We show how the game description language (GDL), extended by concurrent action constraints, can be used to formally describe dynamically changing traffic rules to an autonomous vehicle that enters an automatically regulated road section. We also show how these GDL descriptions can be mapped into an answer set program to allow vehicles, if they are equipped with a solver, to learn to navigate through an automatically regulated road network.

1 Introduction

Over the last decade, research on autonomous vehicles has made revolutionary progress. Recent advancements in Artificial Intelligence (AI), and especially machine learning, allow self-driving cars to learn how to

Both authors are delighted to contribute to this Festschrift in honor of our long-term friend and colleague Andreas Herzig, with whom we have shared – at countless conferences, seminars, workshops and mutual visits in both France and Australia – deep conversations, successful collaborations and, on occasion, volleyballs and table-tennis tables. Andi's fundamental work on logics for actions, change, knowledge and belief always had, and continues to have, great influence on our own research into agent-based approaches to AI.

handle complex road situations based on data from millions of accumulated driving hours, many more than any human driver can achieve. Autonomous driving gives us hope for safer, more convenient, efficient, and environmentally friendly transportation. However, it also introduces new challenges, the most important of which is how can a self-driving car adapt, all by itself, to the dynamics of road regulation and traffic situations it has not been trained for [1, 3]? This can happen anytime, anywhere; e.g. rules may vary with countries, states, regions and even segments of a road; a vehicle may not be aware of newly introduced traffic rules; and exceptional emergency/unforeseeable traffic situations may occur. Some examples are depicted below.[1]



This raises the question: *Can a vehicle travel safely on roads without prior knowledge of the situations and rules that may apply?* Humans have a remarkable ability to adapt on the spot to unforeseen situations; for a self-driving car, however, this still poses a major challenge.

In this paper, we address this problem using a formal language for describing dynamic systems. This is motivated by research in General Game Playing, which is concerned with the very related question of how a computer can learn to play a new game simply from the rules. The general *Game Description Language (GDL)* has been developed to communicate rules of arbitrary games to a general game-playing system, which can then, all by itself, develop a strategy to play [5]. With extensions of GDL [9], this has been successfully applied to various domains, incl. financial markets [14], auctions [10], and automated negotiations [2].

This paper shows how GDL can be used to provide formal descriptions of dynamically changing traffic rules and regulations. The moti-

[1] The right-most image is a cropped version of "*Southern Vectis 1145 HW09 BBZ and Pixley Hill 5.JPG*" by Arriva436, via Wikimedia Commons, CC BY 3.0.

vation is multifacted: First, a formal description provides a clear, unambiguous account of the rules that govern a road segment, as opposed to using computer vision especially under adverse conditions that make traffic signs harder to identify and interpret. Second, autonomous vehicles can use a formal specification to negotiate with each other or with a central infrastructure [15]. Third, a GDL description can be used to formally verify desired properties of a dynamic system [6], e.g. deadlock-freeness or optimal traffic flow. Fourth, a dynamic description can be automatically adapted to unforeseen situations, e.g. when parts of a road network are flooded or emergency vehicles need to be given priority.

The remainder of the paper is organised as follows. Section 2 specifies the road and traffic rules using GDL. Section 3 discusses how traffic can be managed based on the GDL representation. Section 4 shows how Answer Set Programming (ASP) can be used by autonomous vehicles to interpret GDL-based traffic specifications and generate travel plans that fulfill their goals while obeying the given traffic rules. Section 5 concludes the paper and outlines several directions for future research.

2 Formalising Roads and Traffic Rules

We begin by showing how the Game Description Language (GDL) can be used as a formal representation framework to specify road configurations and dynamic traffic regulations.

2.1 Specification of roads

We divide a road into blocks called *waypoints*. Each block allows one car to travel on it at any given time. Waypoints are linked by directed edges, representing allowed connections and travel directions between road blocks. Figure 1 presents an irregular intersection as an example, which will be used throughout this paper. The intersection features an unconventional layout including a single-lane shift on the vertical (north-south) road. As a result, traffic flow can take on various forms. For example, a vehicle traveling from b_{22} to b_7 may follow the path $b_{22} \rightarrow b_{20} \rightarrow b_{15} \rightarrow b_9 \rightarrow b_8 \rightarrow b_7$. Similarly, a vehicle originating from b_4 can travel through $b_4 \rightarrow b_5 \rightarrow b_6 \rightarrow b_{16} \rightarrow b_{17} \rightarrow b_{18}$. These travel paths follow from the *road graph*, which encodes the connectivity

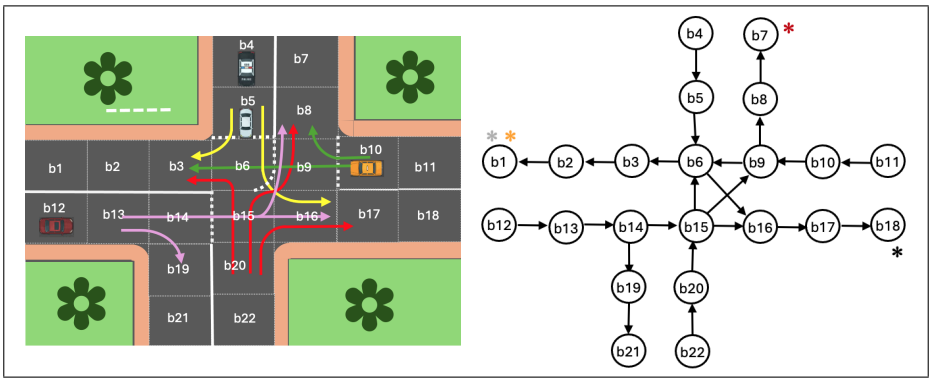


Figure 1: Graph representation of an irregular intersection, including an example state with four vehicles, v_2, v_1, v_3, v_4 (from the top). Asterisks on the right-hand side indicate the intended goal position for each car.

and structure of a traffic network [12] and provides a simple, clear and unambiguous basis for communicating (possibly dynamic) traffic rules to autonomous vehicles.

Following the standard syntax of GDL [2], the rules according to the road graph in Figure 1 can be represented by the following game rules:

```
;;; Specification of waypoints ;;;
(waypoint b1) (waypoint b2) ... (waypoint b22)

;;; Specification of allowed connections between waypoints ;;;
(init (arc b2 b1)) (init (arc b3 b2)) (init (arc b4 b5)) (init (arc b5 b6))
(init (arc b6 b1)) (init (arc b6 b16)) ... (init (arc b22 b20))
```

Note that the allowed traffic flows (arcs) are treated as dynamic state features because, although the configuration defines the baseline connectivity, we want to account for the possibility that at any time traffic is managed dynamically by automatically modifying which transitions are allowed. One example will be traffic-light-based control, which periodically enables or blocks traversal on specific edges leading into an intersection. In Section 3, we will illustrate this form of control using a

[2] Syntax and semantics of GDL are quite intuitive, and we refer the reader to a textbook for the formal details [5]. Predefined keywords, like **init** for the specification of properties of the initial state, are highlighted.

designated agent called the RTA (Roads & Traffic Authority). To provide the RTA with maximal flexibility, including emergency situations, we include in the GDL specification facts that encode all *physically* possible transitions between waypoints; e.g., for our example from Figure 1,

```
(edge b2 b1) (edge b1 b2) (edge b1 b12) (edge b1 b13)
(edge b2 b3) (edge b2 b12) (edge b2 b13) (edge b2 b14)
...
(edge b22 b19) (edge b22 b20) (edge b22 b21)
```

A current state (arcs) will always be a subset of the set of possible edges.

2.2 Specification of vehicles

Next, we specify actual traffic on the road as a set of vehicles that are currently on the road. Each vehicle can be represented with a tuple,

$$(vehicleId, currentPosition, destination)$$

An example is the scenario depicted in Figure 1 with four vehicles,

$$\{(v_1, b_5, b_1), (v_2, b_4, b_{18}), (v_3, b_{10}, b_1), (v_4, b_{12}, b_7)\} \tag{1}$$

The information can be expressed in GDL by the following facts:

```
(role v1) (role v2) (role v3) (role v4) (role rta)
(destination v1 b1) (destination v2 b18)
(destination v3 b1) (destination v4 b7)
(init (at v1 b5)) (init (at v2 b4)) (init (at v3 b10)) (init (at v4 b12))
```

Vehicles are represented as agents (so-called roles in GDL). The special role RTA can control traffic by enabling and disabling edges as traversable arcs. For simplicity, while the current location is modeled as a state-dependent fluent, each vehicle's destination is a static fact.

2.3 Specification of right-of-way rules

The right of way determines who has the legal priority to proceed on the road. Specific laws governing the right of way can vary; for example, in some countries with right-hand traffic, drivers must yield to traffic from the right at unsigned intersections and from the left at roundabouts.

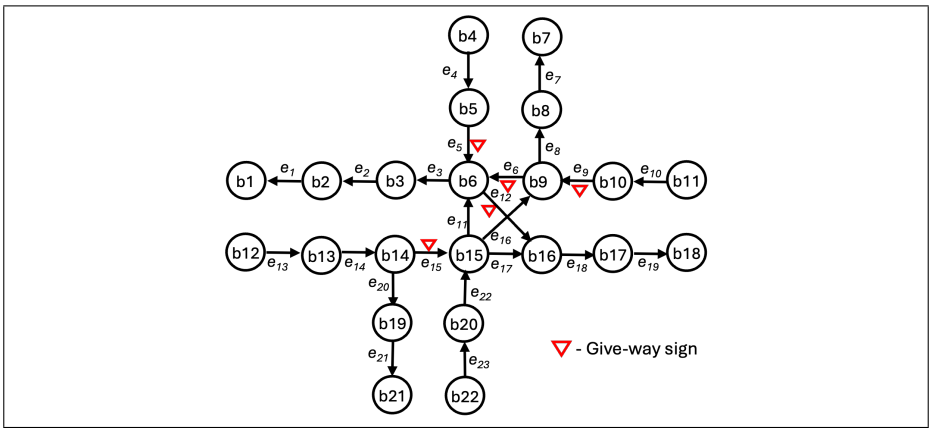


Figure 2: Give-way sign as an example of priority-based protocol.

We previously introduced a way to represent the priority of vehicles on different road segments as a protocol for traffic control [11]. A similar approach can be used to represent the right of way by a binary relation.

A relation $\beta \subseteq \mathcal{E} \times \mathcal{E}$ over a set of edges $\mathcal{E}$ is a *priority relation* if

1. $(e, e') \in \beta$ implies $(e', e) \notin \beta$;
2. for $(e, e') \in \beta$, if $e = (b_1, b_2)$ and $e' = (b'_1, b'_2)$, then $b_1 \neq b'_1$.

Intuitively, $(e, e') \in \beta$ means that a vehicle traveling on arc e has higher priority than, i.e. must give way to, a vehicle traveling on e'.

Figure 2 illustrates an example of a priority relation defined using give-way signs, based on the road layout shown in Figure 1. The intersection contains five give-way signs located on arcs e_5, e_6, e_9, e_{12}, and e_{15}, which establish the priorities for vehicles navigating the intersection. For example, a vehicle intending to travel along arc e_5 has lower priority than vehicles traveling along arcs e_6 and e_{11}. Formally, these priorities can be represented by the following priority relation:

$$\beta = \{(e_6, e_5), (e_{11}, e_5), (e_{11}, e_6), (e_{16}, e_9), (e_{16}, e_{12}), (e_{17}, e_{12}), (e_{22}, e_{15})\}$$

The formal representation of the priority relation provides a much more precise specification than give-way signs or road markings. For instance, vehicles on arc e_{12} must yield to those on arcs e_{16} and e_{17} – a detail that is difficult to convey or interpret from road markings alone.

A priority relation can be literally translated into GDL thus:

```
(init (prio b9 b6 b5 b6)) (init (prio b15 b6 b5 b6))(init (prio b15 b6 b9 b6))
(init (prio b15 b9 b10 b9)) (init (prio b15 b9 b6 b16))
(init (prio b15 b16 b6 b16)) (init (prio b20 b15 b14 b15))
```

Note that priority rules can be dynamically changed under certain circumstances as well, for example, for emergency vehicles.

2.4 Specification of actions

We now specify, in GDL, the actions available to each role, that is, for all vehicles and the traffic controller RTA, using the pre-defined keywords `input` (to define the domain of actions), `true` and `legal` [5]:

```
;;; Stay ;;;
(<= (input ?v stay) (role ?v) (distinct ?v rta))
(<= (legal ?v stay) (role ?v) (distinct ?v rta))
;;; Exit ;;;
(<= (input ?v exit) (role ?v) (distinct ?v rta))
(<= (legal ?v exit) (true (at ?v ?b)) (destination ?v ?b))
;;; Go ;;;
(<= (input ?v (go ?d)) (role ?v) (distinct ?v rta) (waypoint ?d))
(<= (legal ?v (go ?d)) (true (at ?v ?s)) (true (arc ?s ?d))
    (not (must_yield ?s ?d)))
(<= (must_yield ?s ?d) (true (prio ?os ?od ?s ?d)) (true (at ?v ?os ?od)))
```

The last two rules imply that a vehicle is permitted to go along an arc only if no other vehicle is waiting to go along a higher-priority arc.

The traffic controller has the authority to change arcs and priorities:

```
;;; No operation ;;;
(input rta noop)
(legal rta noop)
;;; Enable arc ;;;
(<= (input rta (addarc ?b1 ?b2)) (edge ?b1 ?b2))
(<= (legal rta (addarc ?b1 ?b2)) (edge ?b1 ?b2)(not (true (arc ?b1 ?b2))))
;;; Disable arc ;;;
(<= (input rta (delarc ?b1 ?b2)) (edge ?b1 ?b2))
(<= (legal rta (delarc ?b1 ?b2)) (true (arc ?b1 ?b2)))
```

```
;;; Add priority ;;;
(<= (input rta (addprio ?s1 ?d1 ?s2 ?d2)) (edge ?s1 ?d1) (edge ?s2 ?d2))
(<= (legal rta (addprio ?s1 ?d1 ?s2 ?d2)) (distict ?s1 ?s2)
    (true (arc ?s1 ?d1)) (true (arc ?s2 ?d2))
  (not (true (prio ?s1 ?d1 ?s2 ?d2))) (not (true (prio ?s2 ?d2 ?s1 ?d1 ))))
;;; Delete priority ;;;
(<= (input rta (delprio ?s1 ?d1 ?s2 ?d2)) (edge ?s1 ?d1) (edge ?s2 ?d2))
(<= (legal rta (delprio ?s1 ?d1 ?s2 ?d2)) (true (prio ?s1 ?d1 ?s2 ?d2)))
```

Finally, we specify the effect of all actions with the following state update rules using the pre-defined GDL keywords `does` and `next`:

```
(<= (next (at ?v ?d)) (does ?v (go ?d)))
(<= (next (at ?v ?d)) (true (at ?v ?d)) (does ?v stay))
(<= (next (exited ?v)) (does ?v exit))
(<= (next (exited ?v)) (true (exited ?v)))
(<= (next (arc ?b1 ?b2)) (does rta (addarc ?b1 ?b2)))
(<= (next (arc ?b1 ?b2))
    (true (arc ?b1 ?b2)) (not (does rta (delarc ?b1 ?b2))))
(<= (next (prio ?s1 ?d1 ?s2 ?d2)) (does rta (addprio ?s1 ?d1 ?s2 ?d2)))
(<= (next (prio ?s1 ?d1 ?s2 ?d2)) (true (prio ?s1 ?d1 ?s2 ?d2))
    (not (does rta (delprio ?s1 ?d1 ?s2 ?d2))))
```

3 Automated Traffic Management With GDL

Similar to General Game Playing (GGP), the road and traffic rules described in GDL serve to formalise and communicate rules to participating agents. In this context, the roadside infrastructure functions as the so-called Game Master [5] while vehicles entering, or already within the controlled space, act as players. Additionally, the RTA plays the role of a special agent responsible for dynamically managing traffic rules.

However, unlike traditional GGP, the traffic rules do not define explicit goals for each player. Instead, vehicles propose their desired plans of actions, which the infrastructure evaluates and accepts or rejects. A typical interaction between vehicles and controller proceeds as follows:

1. When a new vehicle v enters the infrastructure-controlled space,

the infrastructure sends v the relevant rules and the current state, including positions and plans of all cars in the system.

2. Vehicle v suggests a desired action plan to the infrastructure.

3. The infrastructure accepts the plan and updates the internal system state (including removing from its internal representation any vehicles that have exited the control space).

We refer to this procedure as vehicle-to-infrastructure (V2I) negotiation. A similar interaction protocol can be defined for the RTA agent.

Traffic control by the Road&Traffic Authority RTA can manage traffic in various ways, by altering arcs or adjusting priorities. The following is an example of how to control traffic in a manner similar to traffic lights, by having RTA periodically perform the following actions in sequence:

```
(delarc b5 b6); (delarc b20 b15); noop; ... noop; (delarc b10 b9);
(delarc b14 b15); (addarc b5 b6); (addarc b20 b15); ...
```

In the same way, RTA can also manage traffic by altering the priority settings, e.g. by virtually moving the give-way sign on arc e_{15} to e_{22}:

```
(delprio b20 b15 b14 b15); (addprio b14 b15 b20 b15); noop; noop; ...
```

Constraints on simultaneous actions In standard GDL, a joint action by all players is legal whenever each individual action is [5]. Our setting requires to generalise from this principle and allow game specifications with *constraint rules* of the following general form, by which combinations of actions can be defined as inexecutable:

$$(<= \bot \ rule\_body)$$

This extends Herzig and Varzinczak's concept of *inexecutability laws* [8] from single to joint actions. In our setting, we impose the following constraints on the simultaneous execution of actions of vehicles to prevent collisions: (1) A vehicle is not allowed to enter an occupied waypoint unless the car currently there moves away at the same time; (2) two

vehicles are not allowed to move to the same waypoint simultaneously; and (3) a car is not permitted to travel along an arc if, at the same time, another vehicle is traveling along a higher-priority arc. Formally:

```
(<= FALSE (does ?v1 (go ?d)) (true (at ?v2 ?d)) (does ?v2 stay))
(<= FALSE (does ?v1 (go ?d)) (does ?v2 (go ?d)) (distinct ?v1 ?v2))
(<= FALSE (true (prio ?s1 ?d1 ?s2 ?d2)) (true (at ?v1 ?s1))
          (does ?v1 (go ?d1)) (true (at ?v2 ?s2)) (does ?v2 (go ?d2)))
```

The addition of constraints on simultaneous actions to GDL requires an extension of the semantics, which defines a formal state transition system induced by a set G of GDL rules [5]. The only necessary modification is to disable any state transitions from a state S and joint action A for which $G \cup S^{\texttt{true}} \cup A^{\texttt{does}} \models \bot$ holds, where $S^{\texttt{true}}$ is a set of facts that describe the current state using the keyword `true` and $A^{\texttt{does}}$ is a set of facts that describe the individual actions in A using the keyword `does`.

The possibility that all agents make legal moves which, in combination, are not legal also necessitates a modification of the standard execution protocol for GDL, according to which players choose and submit their moves simultaneously to the game controller. There are different ways in which this can be achieved, for example, using a first-come, first-serve approach. In our setting, we assume that vehicles arrive asynchronously in the controlled area and automatically negotiate with the infrastructure a course of actions that must be compatible with the plans of all the other vehicles that have entered the environment previously.

An axiomatisation in GDL can help with engineering traffic control. Automated theorem proving techniques in general game playing [6] can be used to formally check a set of rules against desired properties.

4 ASP Game Playing for Traffic Management

The syntax and semantics of GDL are closely related to that of Answer Set Programming (ASP), which has been used in general game playing for a variety of purposes, including for solving single- and two-player games [7, 13] as well as to automatically prove properties of games [6]. In this section, we show how ASP can also be used by vehicles for V2I negotiation or by the RTA to control traffic when rules have been for-

malised in GDL.

ASP Encoding of traffic management game rules Using ASP for reasoning and planning with GDL rules requires to incorporate time-points [13]. Formally, the *temporal extension* of a set of GDL rules is obtained by adding time as an extra argument to all the predicates except those that are static (such as the predefined keyword `role` or, in our example formalisation, the predicate `destination`). The temporal extension also links the keywords `init`/`next` to the keyword `true` via

- replacing (`init` f) by `true(f,0)`
- replacing (`next` f) by `true(f,T+1)`

The use of ASP requires a maximal time horizon, which needs to be chosen sufficiently long for the reasoning/planning problem at hand. Finally, the following general clauses complement the ASP-encoding of the game rules by requiring that all roles make one move at a time and that there must never be a move that is not legal at the time it is taken:

```
time(0..maxTime).

1 { does(R,M,T) : input(R,M) } 1 :- role(R), time(T).
:- does(R,M,T), not legal(R,M,T), time(T).
```

Decentralised strategy generation Assume that a new vehicle appears at an entry waypoint. According to our V2I protocol, the traffic management system informs the car about (1) the traffic rules, (2) the RTA traffic management schedule, and (3) the positions and accepted plans of all vehicles already in the environment. Based on these rules, the newly arrived vehicle can then negotiate its own timed path with the infrastructure.

To do so, the car can use ASP to find an optimal path to its intended destination that takes into account the scheduled actions of the RTA and of all other cars. Recall, as an example, the vehicle state (1) and suppose that v_4 newly arrives while v_1, v_2 and v_3 entered the system previously and already have successfully negotiated with the infrastructure individual timed paths to their intended destination:

$$v_1: \quad b_5 \xrightarrow{0} b_6 \xrightarrow{1} b_3 \xrightarrow{2} b_2 \xrightarrow{3} b_1$$

$$v_2: \quad b_4 \xrightarrow{0} b_5 \xrightarrow{1} b_6 \xrightarrow{2} b_{16} \xrightarrow{3} b_{17} \xrightarrow{4} b_{18}$$
$$v_3: \quad b_{10} \xrightarrow{5} b_9 \xrightarrow{6} b_6 \xrightarrow{7} b_3 \xrightarrow{8} b_2 \xrightarrow{9} b_1$$

This information can be encoded in ASP as follows:

```
does(v1,go(b6),0). does(v1,go(b3),1). ... does(v1,exit,4).
does(v2,go(b5),0). does(v2,go(b6),1). ... does(v2,exit,5).
does(v3,noop,0). does(v3,noop,1). ... does(v3,exit,10).
```

For this example, we also assume time-based traffic control in the form of simulated traffic lights, with the same arcs as depicted in Figure 1 except $b_5 \to b_6$ and $b_{20} \to b_{15}$ being disabled initially. A time-based traffic management schedule given to the newly arrived vehicle can then be encoded in ASP as follows:

```
does(rta,noop,0).
does(rta,delarc(b10,b9),1). does(rta,delarc(b14,b15),2).
does(rta,addarc(b5,b6),3). does(rta,addarc(b20,b15),4).
does(rta,noop,5). does(rta,noop,6).
does(rta,delarc(b5,b6),7). does(rta,delarc(b20,b15),8).
does(rta,addarc(b10,b9),9). does(rta,addarc(b14,b15),10).
does(rta,noop,11).
```

Based on an encoding of the GDL rules along with the current schedule, the vehicle can use ASP to calculate an optimal plan for itself, with the aim of proposing that solution to the automated traffic manager. To this end, vehicle v_4 defines its own goal as

```
goal(T) :- time(T), true(exited(v4),T).
:- 0 { goal(T) : time(T) } 0.
```

This rules out any potential answer set in which the goal of v_4 to exit the controlled environment (at its intended destination) is never reached.

Among the possible plans that the ASP may compute are,

$$v_4: \quad b_{12} \xrightarrow{3} b_{13} \xrightarrow{4} b_{14} \xrightarrow{5} b_{15} \xrightarrow{6} b_{16} \xrightarrow{7} b_9 \xrightarrow{8} b_8 \xrightarrow{9} b_7$$
$$v_4: \quad b_{12} \xrightarrow{1} b_{13} \xrightarrow{2} b_{14} \xrightarrow{4} b_{15} \xrightarrow{5} b_6 \xrightarrow{6} b_{16} \xrightarrow{7} b_9 \xrightarrow{8} b_8 \xrightarrow{9} b_7$$

Vehicle v_4 always has to wait for the "green light" to move into b_{15}, but some plans contain unnecessary movements. Using optimisation statements [4], a vehicle may optimise the time of arrival as first priority ("`@2`") and the path length as second priority ("`@1`") as given below.

```
#minimize { T@2 : does(v4,exit,T) }.
#minimize { 1@1,T : does(v4,go(D),T) }.
```

An example of an optimal plan under these requirements is,

$$v_4: \quad b_{12} \xrightarrow{2} b_{13} \xrightarrow{4} b_{14} \xrightarrow{5} b_{15} \xrightarrow{6} b_9 \xrightarrow{7} b_8 \xrightarrow{8} b_7$$

This may still not satisfy the infrastructure, which may in addition require that cars move forward as far as they can get before stopping, in case other cars will enter the intersection later. This requirement could be incorporated into the optimization as follows:

```
#minimize { T@3 : does(v4,exit,T) }.
#minimize { 1@2,T : does(v4,go(D),T) }.
#minimize { T@1 : does(v4,go(D),T) }.
```

The optimal plan is $b_{12} \xrightarrow{0} b_{13} \xrightarrow{1} b_{14} \xrightarrow{4} b_{15} \xrightarrow{6} b_9 \xrightarrow{7} b_8 \xrightarrow{8} b_7$.

RTA planning The infrastructure can allow the RTA to take over full control and use centralised planning under special circumstances such as for emergency vehicles. Suppose, for example, that v_2 (i.e., the police car in Figure 1) identifies itself as an authorised emergency vehicle that needs to take priority over all other cars. ASP can then be used for centralised control to generate a plan for all the cars simultaneously with the goal to get the emergency vehicle to its destination as quickly as possible. This can include the temporary activation of arcs that would not normally be present, such as allowing the emergency vehicle to take a diagonal shortcut or pass through lanes in the opposite direction. In order to ensure that the right plans are found, the ASP should be provided with strict priority orderings. An example is given by the following directives, which, in the order of decreasing preferences, say that (1) the highest priority is to minimise the time it takes the emergency vehicle to reach its destination; (2) the number of arcs added should be minimised; (3) the overall travel time for all other vehicles should be optimised; (4) the movements of all other cars should be minimised:

```
#minimize { T@4 : does(v2,exit,T) }.
#minimize { 1@3,T : does(rta,addarc(B1,B2),T) }.
#minimize { T@2,V : role(V), V!=v2, does(V,exit,T) }.
#minimize { 1@1,V,T : role(V), V!=v2, does(V,go(B),T) }.
```

The resulting global plan is to direct all the vehicles thus:

$$
\begin{array}{ll}
v_1: & b_5 \xrightarrow{0} b_6 \xrightarrow{1} b_3 \xrightarrow{2} b_2 \xrightarrow{3} b_1 \\
\boldsymbol{v_2:} & \boldsymbol{b_4 \xrightarrow{0} b_5 \xrightarrow{1} b_9 \xrightarrow{2} b_{17} \xrightarrow{3} b_{18}} \\
v_3: & b_{10} \xrightarrow{3} b_9 \xrightarrow{4} b_6 \xrightarrow{5} b_3 \xrightarrow{6} b_2 \xrightarrow{7} b_1 \\
v_4: & b_{12} \xrightarrow{1} b_{13} \xrightarrow{3} b_{14} \xrightarrow{4} b_{15} \xrightarrow{5} b_9 \xrightarrow{6} b_8 \xrightarrow{7} b_7
\end{array}
$$

with four new arcs temporarily added: `does(rta,addarc(b5,b9),0)`, `does(rta,addarc(b9,b17),1)`, `does(rta,addarc(b10,b9),2)` as well as `does(rta,addarc(b14,b15),3)`. As with all the preceding examples, this solution is computed instantaneously by the off-the-shelf ASP solver Clingo (`https://potassco.org/clingo/`). Systematic experiments with larger scenarios are left for future work, although we are confident that ASP solver runtimes will not constitute a bottleneck for this approach to automatic traffic management of intersections.

5 Conclusion

We presented an approach to specifying road configurations and traffic rules using the general Game Description Language (GDL). We extended GDL to allow for additional constraints on the simultaneous executability of actions. We also showed how these extended GDL specifications for traffic control can be translated into Answer Set Programs, enabling both autonomous vehicles to generate travel plans to propose to the infrastructure, and the infrastructure itself to handle exceptional situations by finding optimal plans, e.g. for emergency situations.

In ongoing work, we aim to investigate how rules for dealing with conflicting simultaneous actions can be provided in GDL itself, rather than leaving it to the execution protocol. We also aim to further extend GDL by spatiotemporal representations of traffic management protocols and negotiation mechanisms. This will facilitate the modeling of vehicle and road segment states, as well as reasoning about permissible actions and traffic rules using techniques from General Game Playing and ASP. Beyond vehicle-to-infrastructure negotiation, future work will also investigate vehicle-to-vehicle (V2V) negotiation mechanisms, allowing vehicles greater autonomy in determining their travel plans.

References

[1] N. Chater, J. Misyak, D. Watson, N. Griffiths, and A. Mouzakitis. Negotiating the traffic: Can cognitive science help make autonomous vehicles a reality? *Trends in Cognitive Sciences*, 22(2):93–95, 2018.

[2] D. de Jonge and D. Zhang. GDL as a unifying domain description language for declarative automated negotiation. *Autonomous Agents and Multi-Agent Systems*, 35(1):13, 2021.

[3] K. Dresner and P. Stone. A multiagent approach to autonomous intersection management. *J. of Artif. Intell. Research*, 31(1):591–656, 2008.

[4] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. *Answer Set Solving in Practice*. Morgan & Claypool, 2012.

[5] M. Genesereth and M. Thielscher. *General Game Playing*. Springer, 2022.

[6] S. Haufe, S. Schiffel, and M. Thielscher. Automated verification of state sequence invariants in general game playing. *Artif. Intell.*, 187:1–30, 2012.

[7] Y. He, A. Saffidine, and M. Thielscher. Solving two-player games with QBF solvers in general game playing. In *Proc. of the Intl. Conf. on Autonomous Agents and Multiagent Systems*, 807–815, 2024.

[8] A. Herzig and I. Varzinczak. Metatheory of actions: beyond consistency. *Artif. Intell.*, 171(16-17):951–984, 2007.

[9] G. Jiang, D. Zhang, L. Perrussel, and H. Zhang. Epistemic GDL: A logic for representing and reasoning about imperfect information games. In *Proc. of the Intl. Joint Conf. on Artif. Intell.*, 1138–1144, 2016.

[10] M. Mittelmann and L. Perrussel. Auction description language (ADL): general framework for representing auction-based markets. In *Proc. of the European Conf. on Artif. Intell.*, 825–832, 2020.

[11] J. Qiao, D. Zhang, and D. De Jonge. Priority-based traffic management protocols for autonomous vehicles on road networks. In *Proc. of the Australasian Joint Conf. on Artif. Intell.* Springer, 240-253, 2022.

[12] J. Qiao, D. Zhang, and D. de Jonge. Graph representation of road and traffic for autonomous driving. In *Proc. of the Pacific Rim Intl. Conf. on Artif. Intell.*, 377–384, 2019.

[13] M. Thielscher. Answer set programming for single-player games in general game playing. In *Proc. of the Intl. Conf. on Logic Program.*, 327–341, 2009.

[14] M. Thielscher and D. Zhang. From general game descriptions to a market specification language for general trading agents. In *Agent-Mediated Electronic Commerce*, volume 59 of *LNBIP*. Springer, 259–274, 2009.

[15] D. Zhang. A logic-based axiomatic model of bargaining. *Artif. Intell.*, 174(16-17):1307–1322, 2010.