Right-of-Way Rules as Use Case for Integrating GOLOG and Qualitative Reasoning[⋆]

Florian Pommerening, Stefan Wölfl, and Matthias Westphal

Department of Computer Science, University of Freiburg,
Georges-Köhler-Allee, 79110 Freiburg, Germany
{pommeren,woelfl,westpham}@informatik.uni-freiburg.de

Abstract. Agents interacting in a dynamically changing spatial environment often need to access the same spatial resources. A typical example is given by moving vehicles that meet at an intersection in a street network. In such situations right-of-way rules regulate the actions the vehicles involved may perform. For this application scenario we show how the Golog framework for reasoning about action and change can be enhanced by external reasoning services that implement techniques known from the domain of Qualitative Spatial Reasoning.

1 Introduction

Agents interacting in a dynamically changing spatial environment often need access to the same spatial resources. A typical example is given by vehicles meeting at an intersection in a street network. In such situations right-of-way rules regulate the actions the vehicles involved may perform. For instance, the German right-of-way regulations (StVO § 8 and § 9) present a detailed set of rules on how the agents controlling these vehicles should proceed if the intersection is not regulated by special traffic signs (stop sign, traffic lights, etc.).

In this paper we consider this particular application scenario and present an approach that allows for determining rule-compliant actions for the involved vehicles in a real-time environment. Action choices are represented within the Golog framework for reasoning about action and change [1], which is based on the Situation Calculus [2]. To detect actions that may lead to collisions, we use methods known from the field of Qualitative Spatial Reasoning (QSR), that is, we use constraint-based and neighborhood-based reasoning to detect dangerous actions[1]. Qualitative spatial reasoning is integrated in the IndiGolog-framework [4] by calling an external reasoner.

Qualitative reasoning and neighborhood graphs have been previously used to determine rule-compliant actions in the SailAway demonstrator (see, e.g., [5]

[⋆] This work was partially supported by Deutsche Forschungsgemeinschaft as part of the Transregional Collaborative Research Center *SFB/TR 8 Spatial Cognition*, project R4-[LogoSpace].

[1] A quite different approach to reasoning about moving objects in a traffic scenario is to directly represent trajectories of vehicles [3].

and [6]). The main difference to the approach used there is that we use a generic formalism to represent actions and change, namely the Golog language. A Golog representation of actions and space has also been discussed in [7]. However, the reasoning process was not externalized in the method discussed there.

In the next section we present the general idea as well as the main components of our spatial agent framework, which is based on the Golog interpreter IndiGolog and uses qualitative reasoning methods as an external method in order to trigger rule-compliant actions. More details regarding our implementation are then presented in section 3. We show how traffic rules can be represented within Golog, how spatial neighborhood graphs can be used to formulate reasoning tasks for qualitative reasoning, and how individual actions are mapped to these reasoning results. In particular, we present a reusable method that allows for integrating external reasoning in IndiGolog. In section 4 we provide a short evaluation of the system. Finally, in section 5 we give a short summary and an outlook on research problems to be addressed in future work.

2 Integrating Qualitative Reasoning in Golog

For our problem we consider a number of cars meeting at an intersection. Each car travels at a given speed on a planned route through the intersection. Further, each car (driver) aims at crossing the intersection as fast as possible. Since this is not always possible, each car has the ability to accelerate until it reaches its full speed or to decelerate until it stops. The right-of-way rules state which car has to stop in order to avoid a collision. We assume that the intersection has no specific signs regulating the right of way. Further, we also assume right-hand traffic, i.e., left-hand driving, and the right-of-way rule common in continental Europe, i.e., priority is given to the right vehicle.

To represent the essential aspects of this problem, we consider a limited number of cars (at most four) that try to cross an intersection, which consists of four streets entering from the four cardinal directions. For each car, we model the planned route, the current position on this route, maximal speed, and whether it is currently accelerating or decelerating. The route specifies the complete continuous path from the current position of the car to the final destination after crossing the intersection. We try to find crash-free behaviors for all cars. The term *behavior* just refers to a sequence of actions with timestamps, where possible actions are *accelerate*, *hold*, or *decelerate*. In particular, we assume that the given routes are never altered.

The idea, which is presented in more detail in the following, is to represent the action choices of the agents within IndiGolog whereas spatial reasoning tasks are delegated to a specialized reasoner. If two or more vehicles are approaching the intersection, we first check whether the planned routes could result in a collision. If this is not the case, all vehicles proceed as planned; otherwise we determine those cars that need to decelerate in order to avoid the collision.

IndiGolog. Golog describes a family of programming languages based on the situation calculus, a multi-sorted second-order logic. The situation calculus defines three sorts, distinguishing the concepts of *action*, *situation*, and *object.* A situation consists of a sequence of actions concatenated with the distinguished function $do(a, s)$ that returns the situation resulting from executing action a in situation s. These sequences are grounded in some initial situation S_0. Changing aspects of the domain are modeled by functional and relational *fluents.* Fluents are domain-specific functions or predicates with one situation parameter. To define an action and its effects on fluents, three types of axioms are distinguished: *Action precondition axioms* define whether an action is applicable in a situation. *Initial state axioms* describe the value of fluents in the initial situation S_0. Finally, *successor state axioms* are used to express how actions influence fluents.

Golog [1] adds a notion of executability to the situation calculus by introducing an interpreter macro, which transforms a given Golog program into a situation calculus formula. Golog also defines elements typical for programming languages like sequences, conditional executions, loops, and procedures.

In our implementation the Golog variant IndiGolog [4] was used, since it offers two important features. The first one is the `search`-operator that tries to find a possible execution of a non-deterministic Golog program instead of executing it online. The second feature is the environment manager that communicates with multiple device managers and allows for sending *sensing actions* to them. When a sensing action is executed, the execution of the Golog program pauses until the action is finished. The return code of the action can influence (i.e. sense) the values of several fluents. This way of injecting data into a running Golog program allows an easier integration of external programs.

Qualitative Reasoning. To solve problems containing rules like the right of way, it is often helpful to abstract from the absolute numerical values early on and consider only qualitative descriptions of the situations. A variety of different constraint-based qualitative calculi has been used in other studies in order to reduce the size of the state space by abstraction in such cases (e.g. [5]).

In this paper, we qualitatively describe the positions of cars relative to the center of the intersection. For this, a STAR calculus [8] seems appropriate. It describes positional information between points via a partition of the plane into a given number of sectors, which are obtained by m distinct, intersecting lines, thus giving the form of a star (see Fig. 1a). One distinguishes positions on the

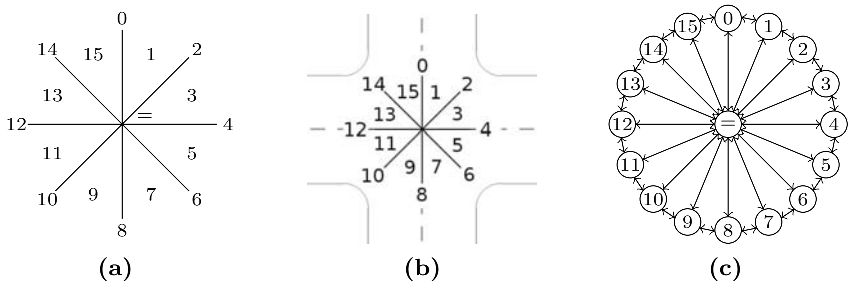


Fig. 1. $\text{STAR}_4(0)$ and its conceptual neighborhood graph

lines and positions in the sectors between them, thereby obtaining $4 \cdot m + 1$ distinct cases including equality of points. $\text{STAR}_4(0)$ is expressive enough for our problem, since we can describe the necessary positions that influence the behavior selection for the cars (see Fig. 1b). To reason about changes in the positions of cars over time, we use the conceptual neighborhood graph (CNG) [9] that makes the possible continuous changes of these relations explicit (Fig. 1c).

3 Implementation Details

The implementation is split up into five different components that communicate via sockets (cf. Fig. 2). First, a Java program contains the full simulation and represents the physical world, in which cars move based on natural laws and crashes can be detected, but not predicted. Cars in the simulation can only change their behavior after receiving `accelerate` or `brake` messages from the Golog program. Thus a car that receives a `brake` message will slow down until the Golog program signals a clear path. Cars can also be set to the state `ignore`, which is done by the Java program if a car leaves the intersection and by the Golog program if a car's trajectory has no conflicts with any other trajectory. In this state, cars accelerate until they leave the area and are neglected in the reasoning process.

The Golog program is called by the Java program with a qualitative representation of the world, which can be easily generated from the complete simulation. For each car it contains the starting position, the current position and the goal position, all as STAR-relations relative to the center of the intersection. The drivers can get this information from watching the other cars and their turning signals. Whenever the Golog program detects a possible collision, it tries to avoid a crash by sending the correct movement messages back to the Java program using a device manager also written in Java. To detect collisions in the trajectories, the Golog program integrates the generic qualitative reasoner

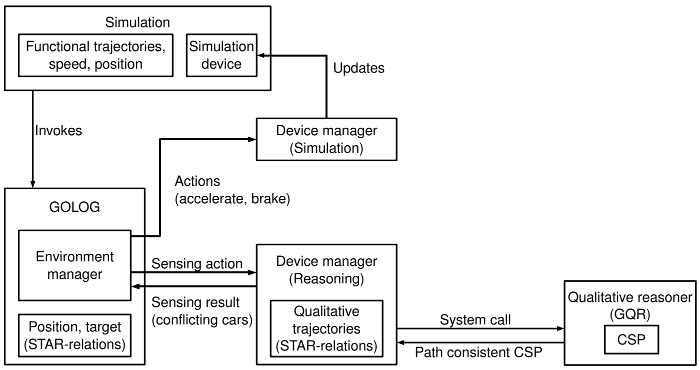
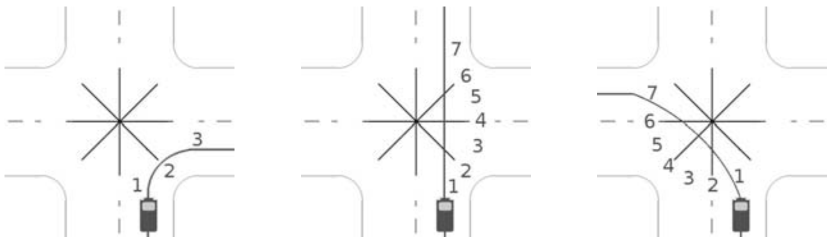

Simulation
Functional trajectories, speed, position
Simulation device
Updates
Invokes
Device manager (Simulation)
Actions (accelerate, brake)
GOLOG
Environment manager
Sensing action
Sensing result (conflicting cars)
Device manager (Reasoning)
Qualitative trajectories (STAR-relations)
Position, target (STAR-relations)
System call
Path consistent CSP
Qualitative reasoner (GQR)
CSP

Fig. 2. Communication of components

GQR [10] with a second device manager, which transforms the trajectories into a qualitative reasoning task, starts GQR, and sends the result back as a sensing result.

Encoding of Trajectories. To reason with the trajectory of a car, the Golog program first needs to recreate it from the current and goal position. The full (functional) trajectory from the simulation is not used here since it would be implausible for a driver to know the absolute coordinates of each car at every moment. Instead the trajectories are expressed qualitatively as a sequence of relations to the center of the intersection. The sequence describes the relations the car will pass through on its way from the current to the goal position.



(a) Right turn **(b)** No turn **(c)** Left turn

Fig. 3. Standard turns

For a standard 4-way intersection all possible turning maneuvers (a right or left turn or driving straight through) go through at most seven neighboring relations in the STAR$_4$(0) calculus (cf. Fig. 3). Since we consider only tangential turning, no car travels through the center of the intersection. The qualitative trajectory can thus be reconstructed by finding a path inside the CNG with at most seven nodes that transforms the current position to the goal position. For standard 4-way intersections such a path always exists and is unique for a given start and goal position.

Encoding of rules in IndiGolog. To encode the right-of-way rule, the Golog program needs additional information on the turning maneuver performed by each car and the notion of prioritized and opposing lanes, all of which can be extracted from the CNG. Each of the turning maneuvers can be identified by the relative position of starting and goal position in the CNG. For example, a right turn moves through three relations counterclockwise (cf. Fig. 3a), whereas a left turn moves through seven relations clockwise (cf. Fig. 3c). To identify prioritized and oncoming lanes for two cars it is sufficient to compare the starting positions of both cars. With this information Golog can decide which of two possibly conflicting cars has the right of way and, accordingly, slow down the other car.

Qualitative Reasoning with GQR. In order to apply GQR, the reasoning problem has to be transformed into a constraint satisfaction problem in the

STAR calculus. This is done by introducing one variable to represent the center of the intersection and one for each entry in the qualitative trajectory of each car. Since the trajectories of the cars are generated from a CNG, we can add an inequality constraint between each two consecutive elements from a car's trajectory. All other added constraints are constraints between a trajectory variable and the center of the intersection determined by the entry in the trajectory.

GQR is called to calculate a path-consistent equivalent CSP, which contains the possible relations between each two trajectory variables. If all constraints between the trajectory variables of two cars do not mention equality, the reasoner can guarantee that these two cars will never be in the same relation to the center of the intersection.

The device manager returns a list of car tuples to the Golog program, containing the pairs of cars where the reasoner cannot guarantee a conflict-free trajectory. The reasoning process done here basically amounts to checking if two trajectories share a relation and only demonstrates the integration of GQR as a high-performance tool for qualitative reasoning within the situation calculus.

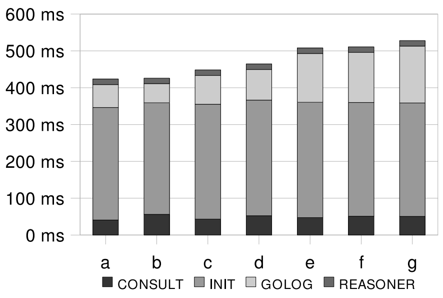
4 Evaluation

Whether the behavior found by the program is crash-free and rule-compliant crucially depends on the speed of the cars and how often the Golog program can be started to determine an action for the cars. Obviously, crashes might happen if the first call to the reasoner returns an action too late for a car to brake in time. The timing currently implemented allows for approximately six calls to the reasoner in the time it takes one car to approach an intersection, cross it and reach a safe distance (all numbers with respect to a 2 GHz PC with 3 GB RAM). With this timing most of the combinations of up to four cars can be resolved in a crash-free and rule-compliant way[2].

For the runtime measurement seven different test cases were executed until all cars reached their goal. The test cases are listed in Fig. 4b and ordered by the number of cars involved and the total number of possible crash situations, i.e., the sum of common STAR-relations for each pair of cars. Each test case called the Golog program between 11 and 23 times. For the evaluation we measured the execution times averaged over all calls. Fig. 4a shows the time it took to consult the prolog files through JPL (CONSULT), the time to initialize IndiGolog and set up all device managers (INIT), the time spent in the main IndiGolog procedure (GOLOG; without external reasoning) as well as the time needed by the external reasoner GQR (REASONER). The time needed to shut down IndiGolog and close all device managers is not depicted.

The interesting components are *GOLOG* and *REASONER*. *GOLOG* strongly reacts to the number of cars in the situation and is mainly responsible for the increase in overall running time of a reasoning step. *REASONER* on the other

[2] We currently do not handle deadlock situations in which no car has priority, e.g., four cars arriving at an intersection simultaneously from the four cardinal directions, where each car wants to cross the intersection in a straight line.



Test	# Cars	# Conflicts
a	1	0
b	2	0
c	2	1
d	2	3
e	3	4
f	3	6
g	3	9

(a) Average runtime of components **(b)** Test case scenarios

Fig. 4. Runtime evaluation

hand only shows slight changes despite the quadratic growth of the CSP. This could be the effect of the small problem sizes considered here. Seeing how the runtime of the Golog program scales with the problem size explains the demand for outsourcing parts of Golog programs to external reasoners. The Golog program presented here is only responsible for a very limited part of the calculation, but still makes up a significant amount of the overall runtime. By outsourcing problems to external reasoners, Golog can use the additional computation time to work on a high-level progam that uses the results of those reasoning processes to reach its goal.

5 Conclusion

The two main results presented here are a working example that uses qualitative reasoning to find rule-compliant crash-free behavior for a number of cars, and a reusable method to integrate external reasoning in IndiGolog. The main limitations of the current implementation is that actions for the individual vehicles are selected from the point of a traffic controller that has complete knowledge about the approaching vehicles. Therefore the current framework cannot be considered a multi-agent framework for reasoning about action and change in spatial environments. It would be interesting to integrate external reasoning into other Golog variants such as MIndiGolog [11] and combine their features with qualitative reasoning. To that end one would have to add features like sensing actions that allow for an execution of actions outside of the Golog context.

Moreover, the current implementation could be extended by adding further sensing actions to request information that the device manager can calculate concurrently to the execution of the Golog program. An interesting example of this would be to ignore conflicts that are rendered invalid by other conflicts because the car causing the conflict will brake anyway. This could also be used to act on incomplete knowledge. If, for example, the intersection is not completely visible from all streets, one could add relations between cars as soon as they are visible and restart the reasoning process to react to the new situation.

Finally, the method for integrating external reasoners in Golog could be simplified by an extension of Golog implementations. The method presented in the paper works well on a per-application basis, but has to be reimplemented in each application. Some of the basic features of this implementation (especially the base class of the Java device manager) can be used to develop a language extension of the IndiGolog syntax that provides an easy-to-use interface to external reasoning services as provided by constraint solvers.

References

1. Levesque, H.J., Pirri, F., Reiter, R.: Foundations for the situation calculus. Electronic Transactions on Artificial Intelligence 2(18), 159–178 (1998)
2. McCarthy, J.: Situations, actions and causal laws. Technical Report Memo 2, Stanford University Artificial Intelligence Laboratory, Stanford, CA (1963)
3. de Weghe, N.V., Cohn, A.G., Maeyer, P.D., Witlox, F.: Representing moving objects in computer-based expert systems: the overtake event example. Expert Syst. Appl. 29(4), 977–983 (2005)
4. Giacomo, G.D., Levesque, H.J.: An incremental interpreter for high-level programs with sensing. In: Levesque, H.J., Pirri, F. (eds.) Logical Foundations for Cognitive Agents: Contributions in Honor of Ray Reiter, pp. 86–102. Springer, Heidelberg (1999)
5. Dylla, F., Frommberger, L., Wallgrün, J.O., Wolter, D., Nebel, B., Wölfl, S.: SailAway: Formalizing navigation rules. In: Proceedings of the Artificial and Ambient Intelligence Symposium on Spatial Reasoning and Communication, AISB 2007 (2007)
6. Dylla, F., Wallgrün, J.O.: Qualitative spatial reasoning with conceptual neighborhoods for agent control. Journal of Intelligent and Robotic Systems 48(1), 55–78 (2007)
7. Dylla, F., Moratz, R.: Exploiting qualitative spatial neighborhoods in the situation calculus. In: Freksa, C., Knauff, M., Krieg-Brückner, B., Nebel, B., Barkowsky, T. (eds.) Spatial Cognition IV. LNCS (LNAI), vol. 3343, pp. 304–322. Springer, Heidelberg (2005)
8. Renz, J., Mitra, D.: Qualitative direction calculi with arbitrary granularity. In: Zhang, C., Guesgen, H.W., Yeap, W.-K. (eds.) PRICAI 2004. LNCS (LNAI), vol. 3157, pp. 65–74. Springer, Heidelberg (2004)
9. Freksa, C.: Conceptual neighborhood and its role in temporal and spatial reasoning. In: Singh, M., Travé-Massuyès, L. (eds.) Decision Support Systems and Qualitative Reasoning, pp. 181–187. North-Holland, Amsterdam (1991)
10. Gantner, Z., Westphal, M., Wölfl, S.: GQR - A fast reasoner for binary qualitative constraint calculi. In: Proceedings of the AAAI 2008 Workshop on Spatial and Temporal Reasoning. AAAI Press, Menlo Park (2008)
11. Kelly, R.F., Pearce, A.R.: Towards high-level programming for distributed problem solving. In: IAT 2006: Proceedings of the IEEE/WIC/ACM international conference on Intelligent Agent Technology, Washington, DC, USA, pp. 490–497. IEEE Computer Society, Los Alamitos (2006)