SR Hackathon 2021 - Overview

Daniel Alvarez-Coello[1,2], Danh Le-Phuoc[3], Anh Le-Tuan[3], Manh Nguyen-Duc[3], Patrik Schneider[4,5]

[1] BMW Technologies E/E Architecture, Wire Harness, Garching, Germany
[2] University of Oldenburg, Oldenburg, Germany
[3] Technical University of Berlin, Berlin, Germany
[4] Vienna University of Technology, Vienna, Austria
[5] Siemens AG Österreich, Vienna, Austria

1 Introduction

Stream reasoning is an emerging area that focuses on inference (by deduction or induction) over data streams. It has been actively evolving for more than a decade now, and according to [1] and [7], there is a wide range of approaches to reason over streams. Since the type of tasks for stream reasoning can be considerably diverse, it has become desirable to have a well-defined set of general tasks accompanied by the corresponding data sets and tooling to *"compare"* the different approaches. So that different stakeholders (*e.g.,* reasoner developer) can use them as a starting point to showcase and validate their tools. Hence, this hackathon is an initiative to provide such resources to the community.

The hackathon is designed as a *model and solve* challenge, where participants have the freedom to solve the presented tasks with the approaches and techniques, which are most familiar to them. The principal points are the following:

– Two well-defined Intelligent Transport Systems (ITS) scenarios, where participants can show their skills and tools. The scenarios consist of (A) a simulation-generated traffic flow scenario with two intersections and many vehicles, and (B) a driving trace from the perspective of an ego vehicle's camera moving in a city.
– A simple platform for stream generation and a background model (KB) that is given beforehand to the participants.[6]
– Different *model and solve* tasks are provided for each scenario, where tasks are increasing in difficulty. Until the start of the hackathon, only introductory tasks are given. At the official start, we will provide the more challenging tasks to the teams.

The rest of this document is organized as follows: In Section 2, we give a brief overview of the stream generation platform. Then, we introduce Scenario A in Section 3, and Scenario B in (which will be fully detailed before the event starts). Lastly, the details of the hackathon, such as procedures, possible prices, and rules, are described in Section 5.

[6] `https://github.com/patrik999/stream-reasoning-challenge`

2 Platform

2.1 Installation

We provide the stream generation platform (SGP) as a Docker container that can be executed using the Docker Desktop tool.[7] After installing Docker Desktop, one can find the installation of the container on the hackathon website.[8]

The SGP can be controlled from the outside via a simple server API (described below) and is designed to send messages using the WebSocket protocol. The syntax of the message can be set on initialization, currently we support JSON-LD triples,[9] RDF-NT triples, and Datalog facts (a subset of the ASP-Core-2 standard).[10] An example client using WebSocket protocol is also provided on our website.

If someone has any problems, please contact us in our Slack channel[11] and/or open an issue on Github.

2.2 Server API

We have defined the simple server API to set up, start, and stop the stream generator via REST API calls. A session for stream generation needs first to be initialized giving the *stream type*, *stream id*, and and *output template*. After initialization a new stream can be started, re-started, stopped. Furthermore, the update frequency of a running stream can be modified. In the following, we give a list of available REST-API calls:

– *Initialize stream generation*:
  `/init?streamtype=TYPE&streamid=ID&templatetype=TEMPL`,
  where `TYPE` is the overall scenario, currently `sumo` and `kitti` are provided, `ID` is the particular stream in the scenario that will be played, and `TEMPL` has to be replaced by the output format for the generated messages, currently `traffic-json`, and `traffic-asp` are selectable.
– *Start stream generation*:
  `/start` uses the default parameter to start streaming, or
  `/start?frequency=100&replay=true` changing that that messages are updated every 100ms (instead of 500ms as default) and the stream is infinitely replayed (instead of replay=false as default).
– *Modify frequency of a stream*: `/modify?frequency=100`
– *Stop stream generation*: `/stop`
– *Get the actvie background KB*: `/getkb`

[7] https://www.docker.com/products/docker-desktop
[8] https://github.com/patrik999/stream-reasoning-challenge
[9] https://json-ld.org/
[10] https://www.mat.unical.it/aspcomp2013/files/ASP-CORE-2.03c.pdf
[11] https://srhackatonorganizers.slack.com/archives/C02E59NQ59T

Note that the arguments in `/init` correspond to keys in `config.yaml`. Here is an example for the initialization and start of a SUMO traffic stream that sends RDF messages in JSON:

– `⟨IP_ADDRESS⟩:⟨PORT⟩/init?streamtype=sumo &streamid=streamSumo1&templatetype=traffic-json`
– `⟨IP_ADDRESS⟩:⟨PORT⟩/start`

3 Scenario A - Traffic management

The first scenario is in the domain of urban traffic management and involves traffic management for Cooperative Intelligent Transportation Systems (C-ITS). Traffic is observed from a third-person, top-down perspective, and streams of vehicle movements and signal phases of traffic lights in a given road network are generated. Additionally, unexpected events are triggered, *e.g.,* vehicle breakdowns, which lead to possible traffic disruptions.

In this scenario, we have identified the following (possible) tasks to be tackled:

1. Gathering traffic statistics, *e.g.,* counting the number of vehicles passing;
2. Event detection, *e.g.,* detecting, accidents or traffic jams;
3. Diagnosis, *e.g.,* finding the cause for a traffic jam;
4. Motion planning, *e.g.,* routing the vehicles optimally through the network.

The sources of the provided data streams are generated on the fly by different simulation runs of the Simulation of Urban MObility (SUMO) framework,[12] and are described below. Note that the simulation runs are taken from the experiments in [2].

3.1 Simulation Environment

Figure 1a shows the scenario road network in SUMO with two intersections that connect three roads (one horizontal and two vertical) with two incoming and two outgoing lanes for each road. Figure 1b is the graph representation of the road network in Figure 1a including nodes for intersections, links, sources, and sinks. As shown in the figure, the street layout is as follows:

– 2 intersections and 3 roads with 2 in/outgoing lanes each
– Road segments between intersections
– Each intersection has two traffic light controller with static signal plans

We provide different traffic scenarios to generate traffic stream of a varying number of vehicles, where at the end we give the stream id for the initialization in our API:

– Light traffic with free flow (30 vehicles): `&streamid=streamSumo1`;
– Medium traffic with free flow (120 vehicles): `&streamid=streamSumo2`;
– Heavy traffic with traffic jam (180 vehicles): `&streamid=streamSumo3`.

[12] `https://www.eclipse.org/sumo/`

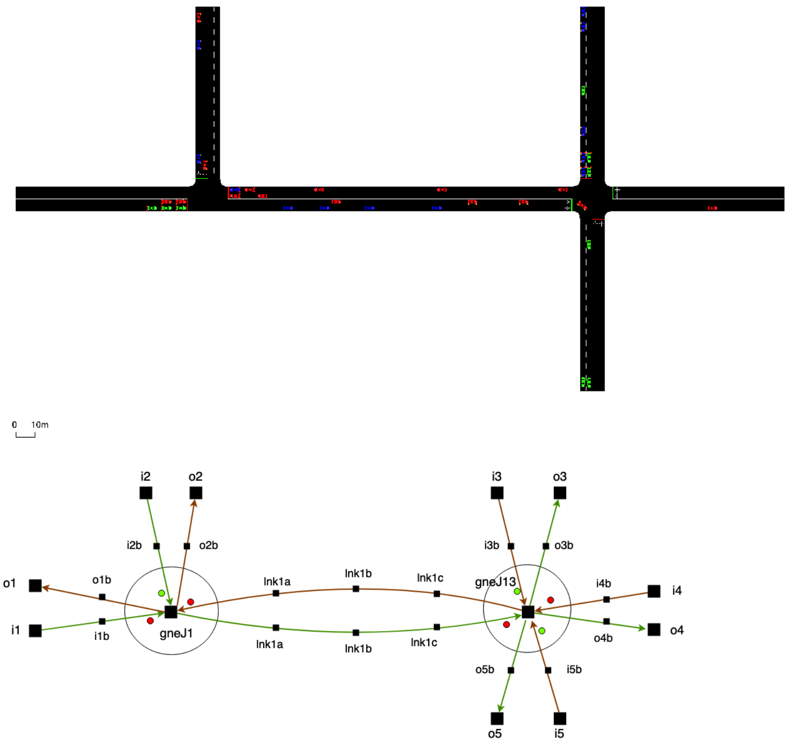

Fig. 1: (a) SUMO rendering of two intersections and (b) corresponding abstract flow model, where green/brown edges are the "we" or "ew" traffic orientation.

3.2 Static Model (Background KB)

A static background Knowledge Base (KB) adds additional immutable information to streaming data. The background KB captures the SUMO model, including the road network, simple traffic regulations, traffic light signal plans, and a simple vehicle taxonomy. The road network of the SUMO model is (manually) rendered into a graph representation either as Datalog facts or as JSON-LD triples. Note that the road network is split into segments of the same length as shown in Figure 1b, where `node`(`X`) defines a connection point in between two edges `link`(`Y`, `X`, `we`) and `link`(`X`, `Z`, `we`) and `we` is constant that describes the direction of (possible) traffic flow.

Traffic regulations only come with speed limits that are given by `maxSpeed`(`X`, `Y`, `D`), where the tuple (`X`, `Y`) is an edge, and D is the speed limit in m/s. Streams generate traffic light signal plan state, but conflicts between traffic signal (e.g., not allowed to be red at the same time) are given by `conflict_tl`(`I`, `X`, `Y`), where `I` is the intersection and (`X`, `Y`) is defined as before.

Finally, we provide a simple vehicle taxonomy, which adds super-types by `isSubType(T1, T2)`, where `T1` is a sub-type of `T2`. In streams, only the base types `carA`, `carB`, `carC`, and `carD` are generated.

If we have a RDF-based representation, the background KB also contains the namespace abbreviations used in the messages. E.g., for `sosa`, we have the following `@context` definition `"sosa" : "http://www.w3.org/ns/sosa/"`.

The background KB can either be retrieved using the REST-API by `/getkb` or can be download from the hackathon website as defined in `config.yaml` with the path `templates : traffic−json : backgroundKB :` directly.

3.3 Streams

The given streams are the means that participants should use to solve the given task. The traffic streams are directly extracted from the SUMO simulation, where we distinguish between *vehicle* and traffic light *signal streams*.

The generation of stream messages in this scenario is driven by each simulation step in SUMO. Each simulation step results in a single message for each vehicle and each traffic light signal. For instance, if we have 10 vehicles and 4 traffic light signals, 14 messages are generated for one simulation step.

We also provide for each stream a predefined template that allows to render the message in JSON-LD triples or Datalog facts; the rendering can be set in the stream initialization by `templatetype=traffic-json` or `templatetype=traffic-asp`. Figure 2 provides an overview of the model used for the traffic streams.

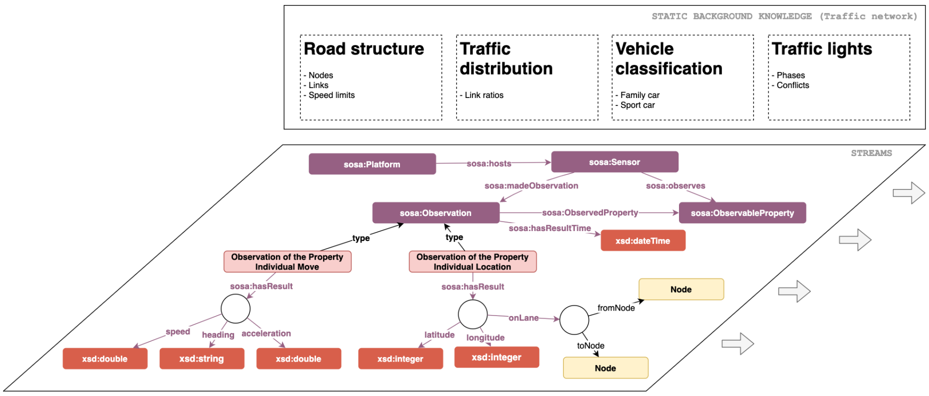

Fig. 2: Overview of the JSON-LD model of (semantic) streams. The background KB is static, whereas the traffic streams capture changing objects (the vehicles) and their values as observations. Two observable properties (*i.e.,* Move and Location) have slightly different annotation patterns for their observations.

First, we introduce the attributes, i.e., the changing values, in a *vehicle* message, where in the column we show the representation of Datalog facts and JSON-LD triples, where `v` represents an attribute value and `s` represents a blank node for an entity such as vehicle or observation (see for full nesting of some JSON-LD triples in the Appendix example):

Attribute	Type	Datalog	JSON-LD	Description
Vehicle ID	String	In every atom as `i`	$(\mathtt{s}_1,\mathtt{@id},\mathtt{i})$	Unique ID of a vehicle
Message Time	Long	In every atom as `t`	$(\mathtt{s}_2,\mathtt{sosa:resultTime},\mathtt{v})$	Time on message send
Vehicle Model	String	`vehModel(i,v)`	$(\mathtt{s}_1,\mathtt{its:vehModel},\mathtt{v})$	Model of vehicle
Vehicle Speed	Long	`speed(i,v,t)`	$(\mathtt{s}_2,\mathtt{its:speed},\mathtt{v})$	Speed a vehicle in m/s
Vehicle Position	Tuple	$\mathtt{position(i,v_x,v_y,t)}$	$(\mathtt{s}_3,\mathtt{schema:latitude},\mathtt{v_x})$, $(\mathtt{s}_3,\mathtt{schema:longitude},\mathtt{v_y})$	Coordinates as (v_x, v_y)
Vehicle Lane	String	$\mathtt{onLane(i,v_s,v_e,t)}$	$(\mathtt{s}_3,\mathtt{its:onLane},\{\ldots\})$	Vehicle is on lane $(v_s,\, v_e)$ with orientation v_o
Vehicle Heading	Long	`heading(i,v,t)`	$(\mathtt{s}_2,\mathtt{its:heading},\mathtt{v})$	Orientation of vehicle in degree
Vehicle Acceleration	Long	`accel(i,v,t)`	$(\mathtt{s}_2,\mathtt{its:acceleration},\mathtt{v})$	Acceleration in m/s

Note that the lane id, vehicle position, and vehicle model (e.g., carA) also refer to the facts in the background KB.

Next, we introduce the attributes of a *traffic signal* message, where the Datalog and JSON-LD column is as before with `v` as an attribute value, `s` is a blank node:

Attribute	Type	Datalog	JSON-LD	Description
Intersection ID	String	In the atom as `i`	$(\mathtt{s}_1,\mathtt{@id},\mathtt{i})$	Intersection of signal
Traffic light ID	String	In the atom as `j`	$(\mathtt{s}_2,\mathtt{@id},\mathtt{j})$	ID of signal itself
Message Time	Long	In the atom as `t`	$(\mathtt{s}_3,\mathtt{sosa:resultTime},\mathtt{t})$	System time on message send
Signal state	String	`tlight(i,j,v,t)`	$(\mathtt{s}_3,\mathtt{its:state},\mathtt{v})$	Current state of a signal

Note that a signal state is only unique in the combination of intersection and traffic light ID, which both refer also to the background KB. A signal state can be either green as `G` or red `r`. Other states, e.g., yellow, would be possible but currently not defined in the signal plans.

As mentioned before, the generation of messages is based on predefined templates that map the attributes defined above. For vehicle streams, the following set of variables can be used in the template: `$VehicleID$`, `$Type$`, `$Timestamp$`, `$Speed$`, `($Position_X$,$Position_Y$)`, `($LaneID$,$LaneOrient$)`, `$Orient_Heading$`, and `$Accel$`. For traffic light streams the following variables can be used: `$IntersectionID$`, `$Timestamp$`, `$TrafficLightID$`, and `$SignalState$`.

In the following, we give a (simplified) example message rendered in Datalog (a full example is given in the Appendix): `speed(vehicle:20, 20, 1001).` stating that vehicle:20 moves at the speed of 20 at time-point 1001. The same message is sent as JSON-LD triples as follows:

```
{
    "@id": "vehicle:20",
    "sosa:madeObservation":[
        {
            "@type" : "sosa:Observation",
            "sosa:hasResult":{ "its:speed": "20" },
            "sosa:resultTime": "1001"
        }
     ]
}
```


3.4 Tasks

The defined tasks are the main focus of the *"model and solve"* challenge, and participants are encouraged to solve them starting with the simpler Task 1, and later move to more difficult Task 2. Task 3 is given as an extra challenge, if the first two tasks could be "solved" during the hackathon.

Task 1. Collect traffic statistics and try to update the statistics frequently (how frequent will vary between the different approaches):

1. Calculating the number of vehicles (NoV) and average speed of all vehicles on each edge;
2. Separated by vehicle super-type, calculate the NoV and their average speed;
3. Find the time intervals, where vehicles exceeds the top speed defined in `maxSpeed(X,Y,D)`. Note that this sub-task can be reused in the next task.

Task 2. Detection normal and wrong behaviour of individual vehicles:

1. Task 2 is an event detecting task, where different driving patterns that suggest normal/wrong behaviour have to be detected;
2. Detect the vehicles that perform standard maneuvers: vehicles appears/disappears in the network, vehicle turns left/right, vehicle makes a short stop (< 5 time steps);
3. Detect the vehicles that make the following traffic violations: speeding, accident (stop longer than $2o$ time steps), u-turn (since not allowed in this network);
4. Detect the vehicles that have to stop because of another vehicle's accident.

Task 3 (Bonus). Will be announced at the second part of the hackathon.

4 Scenario B - Autonomous driving

The second scenario consists of data streams produced from the perspective of one vehicle. In this sense, at each time step, graph data about the driving context is generated. The driving context can be thought of as the environment where a vehicle is being driven. Although the driving context involves data from several domains (*e.g.,* hazards, weather, infrastructure, driver's mental state, and others [?]), we focus on two specific sources: (1) vehicle data streams that result from the direct observation of its measurements (*e.g.,* speed, acceleration, etc.), and (2) streams of detected objects surrounding the vehicle.

In this scenario, we find the following tentative tasks to be solved:

1. Finding relevant behavioral patterns from the driving context, *e.g.,* the flow of the traffic around;
2. Finding possible reasons of particular situations, *e.g.,* what was the reason for a particular maneuver?;
3. Detection of complex events, *e.g.,* dangerous situations on the road.

The scenario uses a few traces from the Kitti dataset[13] [3], a well-known dataset that has been extensively used for benchmark comparisons in tasks related to autonomous driving. More specifically, the platform, discussed in section 2, internally performs object detection on the video frames to obtain labels of objects that surround the vehicle.

4.1 Data model

The second scenario concerns with the object scene flow for autonomous vehicles. It is based on computer vision datasets for autonomous driving taken from KITTI. The data consists of images captured from a camera attached to a car, its GPS location, and its speed and acceleration. The data is semantically annotated with SSN [4] and SemKG vocabulary [5] and is provided as stream using the streaming platform described in Section 3.3.

Figure 3 illustrates the semantic schema of the stream data. A vehicle is annotated as a *sosa:Platform* which hosts few sensors (*sosa:Sensor*) and *sosa:Sampler*. The camera attached to a vehicle is annotated as a sensor which captures images as frames (*Frame*). Thus a frame is a *sosa:Observation* which has a result is an *Image2D* and a result time stamp. The result of each *InvidualMove* observation includes the speed (m/s), the acceleration in X axis and Y axis of the vehicle. The *InvidualLocation* gives the latitude and longitude GPS coordinator of the vehicle.

To detect the objects from the images captured by the camera, we use a detector which is annotated as a sampler (sosa:Sampler). A detection is a sampling of a detector which can be implemented with different computer vision algorithms (*sosa:Procedure*) such as SDD [6], FRCNN [8] or YOLO. In our data, the object detection result is sampled by Yolo4 algorithm [14]. The result of a detection contains a box (*Box2D*, an object (*DetectedObject*) and a confidence score. The box is 1:1 associated with an image by the property *isIn*, the x and y are the coordinator of the center of the box in the image. The width and height values

[13] `http://www.cvlibs.net/datasets/kitti/`
[14] https://pjreddie.com/darknet/yolo/

are the size of the box. The visual sample of the boxes from the object detection can be found on `https://vision.semkg.org/`. Furthermore, the detected object is linked to Wikidata. The example of a stream message can be found in Appendix B and Appendix C.

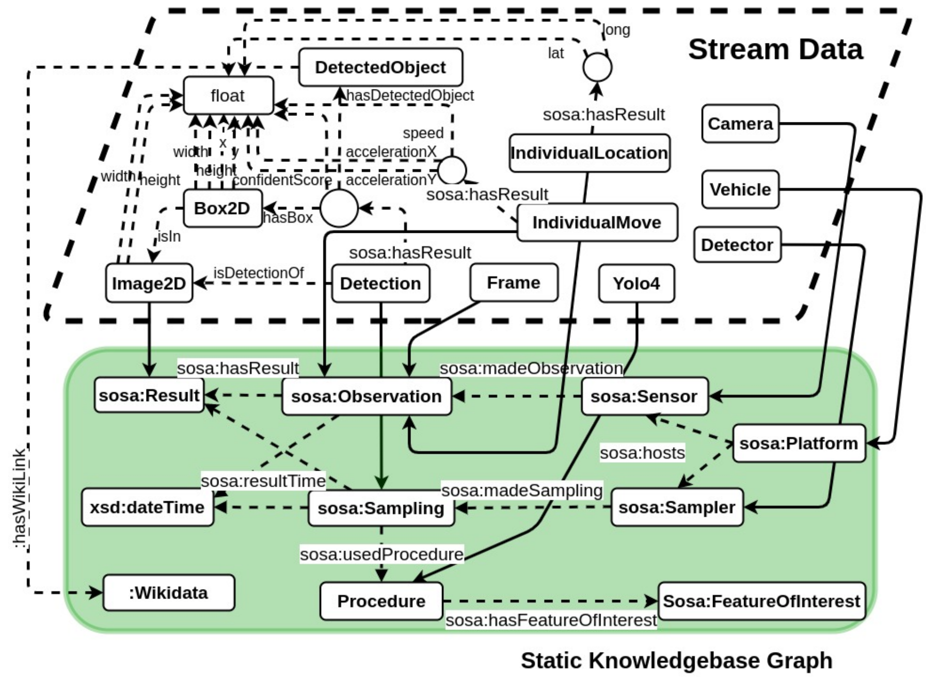

Fig. 3: Schema used for the semantic annotation of objects detected from the video frames of the kitti dataset

4.2 Streams

Since raw image processing is not possible, annotated data with bounding boxes and the labels should be used. It is still to be decided, if we used the ´´gold standard" annotation of labels, or the YOLO object detection is used.

4.3 Tasks

Task 1. Query (detect) other vehicles behaviour in the stream of labelled objects collected by the ego-vehicle. Possible task are detect:

1. All oncoming traffic or all crossing traffic.

2. Detect if one object (vehicle) is stationary or moving.

Scenes: Road 2011_09_26_drive_0015 dataset.

Task 2. Driving scence understanding, understanding the explanations for certain observations, e.g., stopping because of pedestrians, traffic lights, other cars.

Scenes: 2011_09_26_drive_0017, 2011_09_26_drive_0018, and 2011_09_29_drive_0071 datasets.

Recognise detected

Task 3 (Bonus). Detection of dangerous situations, e.g., person/bike crossing

5 Hackathon Procedure

The procedure is just preliminary, and will still change until the hackathon itself, but should give already an indication on how it will be realized.

5.1 Process

Before the hackathon:

1. Announcement on the 16.9.2021;
2. Everybody is encouraged to get familiar with Scenario A, the stream generation platform, and Task 1;
3. Feedback can be given to us using the mentioned Slack channel or send us an email.

On the hackathon day (4.10.2021):

1. Introduction and introducing;
2. Discussion of scenarios and tasks;
3. Hackathon starts;
4. Mid report and lunch;
5. Hackathon continues;
6. At the end, the participants give a presentation of their solutions;
7. Discussion and feedback of solutions;
8. Evaluation of most "best" solutions (details below).

After the hackathon, a summary of the solution will be presented at the workshop. Important, the hackathon will be held online and offline simultaneously.

5.2 Prices

We are happy to announce that Siemens AG, Austria (https://new.siemens.com/at) and the Zentrum für Informatikforschung (http://zif.or.at/) are each sponsoring one Amazon gift card of 100 Euro. The prices will be awarded to the teams that score highest in development effort, understandability, problem coverage, and originality of their solutions.

5.3 Rules and Evaluation

We suggest the following rules (open for discussion):

- The organizers give two scenarios and some problem tasks, which are given in the introduction of the hackathon;
- The organizers provide a stream generation platform, and users are allowed to bring their own (development) tools to reprocess the generated stream messages;
- Teams can either be assembled before the conference or are on-demand set-up at the begining of the challenge;
- For each task, teams are allowed to use a specific solver or their own solving tools, and are encouraged to work out their problem encoding;
- Solutions are to be presented at the end of the competition.

After the solution presentation, a jury of all participants from the hackathon give scores according to the following criteria:

- Development effort,
- Understandability and easiness of use,
- Problem coverage,
- Originality of the solutions.

References

1. Dell'Aglio, D., Della Valle, E., van Harmelen, F., Bernstein, A.: Stream reasoning: A survey and outlook: A summary of ten years of research and a vision for the next decade 1(1), 59–83
2. Eiter, T., Falkner, A.A., Schneider, P., Schüller, P.: Asp-based signal plan adjustments for traffic flow optimization. In: Giacomo, G.D., Catalá, A., Dilkina, B., Milano, M., Barro, S., Bugarín, A., Lang, J. (eds.) ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020). Frontiers in Artificial Intelligence and Applications, vol. 325, pp. 3026–3033. IOS Press (2020)
3. Geiger, A., Lenz, P., Stiller, C., Urtasun, R.: Vision meets robotics: The kitti dataset. The International Journal of Robotics Research 32(11), 1231–1237 (2013)
4. Haller, A., Janowicz, K., Cox, S.J.D., Lefrançois, M., Taylor, K., Phuoc, D.L., Lieberman, J., García-Castro, R., Atkinson, R., Stadler, C.: The modular SSN ontology: A joint W3C and OGC standard specifying the semantics of sensors, observations, sampling, and actuation. Semantic Web 10(1), 9–32 (2019), `https://doi.org/10.3233/SW-180320`
5. Le-Tuan, A., Kien-Tran, T., Nguyen-Duc, M., Yuan, J., Hauswirth, M., Yuan, J.: Visionkg: Towards a unified vision knowledge graph. In: Proceedings of the ISWC 2021 Posters & Demonstrations Track. CEUR Workshop Proceedings (2021)
6. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S.E., Fu, C., Berg, A.C.: SSD: single shot multibox detector. In: ECCV. pp. 21–37 (2016)
7. Margara, A., Urbani, J., van Harmelen, F., Bal, H.E.: Streaming the web: Reasoning over dynamic data. J. Web Semant. 25, 24–44 (2014)

8. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. Advances in neural information processing systems 28, 91–99 (2015)

A Message Examples for Scenario A

Two example messages, one from the vehicle and one from the traffic light stream rendered as Datalog facts:

```
speed(vehicle:20, 20.5, 1001) .
position(vehicle:20, (13,-25), 1001) .
onLane(vehicle:20, lnk1b, lnk1a, ew, 1001) .
heading(vehicle:20, 43.44, 1001) .
acceleration(vehicle:20, 1.2, 1001) .
vehModel(vehicle:20, carA) .

tlight(is1, tl2, r, 1001) .
```

An example messages for the vehicle stream rendered in JSON-LD:

```
{
        "@id": "vehicle:20",
        "@type": ["sosa:Platform", "its:carA"],
        "sosa:hosts":[
         {
          "@id": "vehicle:SensorMove_20",
          "@type": "sosa:Sensor",
          "sosa:madeObservation":[
           {
                "@id"   : "obs:Move_20",
                "@type" : "sosa:Observation",
                "sosa:observedProperty":{
                    "@id":"obsProp:Move"
                },
                "sosa:hasResult":{
                    "its:speed": 20.5,
                    "its:heading": 43.44,
                    "its:acceleration": 1.2
                },
                "sosa:resultTime": "1001"
            }
           ]
          },
          {
            "@id": "vehicle:SensorGPS_20",
            "@type": "sosa:Sensor",
            "sosa:madeObservation":[
                {
                "@id" : "obs:GPS_20",
                "@type" : "sosa:Observation",
                "sosa:observedProperty":{
```

```
                "@id":"obsProp:Location"
            },
            "sosa:hasResult":{
                "@id" : "result:GPS_20",
                "schema:latitude": 13,
                "schema:longitude": -25,
                "its:onLane": {
                  "its:fromNode": "lane1Start",
                  "its:toNode": "lane1End",
                  "its:orientation": "we"
                }
            }
          },
          "sosa:resultTime": "1001"
        }
       ]
      }
     ]
}
```

B Message Examples for Scenario B

An example of the message from an object detection stream in JSON-LD format:


```
{
  "@context": {
    "category": "http://vision.semkg.org/category/",
    "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
    "semkg": "http://vision.semkg.org/onto/v0.1/",
    "sosa": "http://www.w3.org/ns/ssn/",
    "srhackaton": "http://stream-reasoning-challenge.org/",
    "xsd": "http://www.w3.org/2001/XMLSchema#"
  },
  "@graph": [
    {
      "@id": "srhackaton:Camera",
      "rdfs:subClassOf": {
        "@id": "sosa:Sensor"
      }
    },
    {
      "@id": "srhackaton:camera2011_09_26_drive_0015",
      "@type": "srhackaton:Camera"
    },
    {
      "@id": "srhackaton:individual_move_0000000000_result",
      "rdfs:subClassOf": {
        "@id": "sosa:Result"
      },
      "srhackaton:accelerationX": {
        "@type": "xsd:float",
        "@value": "-0.431055573366"
      },
      "srhackaton:accelerationY": {
        "@type": "xsd:float",
        "@value": "0.22098659453951"
      },
      "srhackaton:speed": {
        "@type": "xsd:float",
        "@value": "17.657510351453"
      }
    },
    {
      "@id": "srhackaton:detection_0000000000",
      "@type": "srhackaton:Detection",
      "semkg:isDetectionOf": {
        "@id": "srhackaton:image_0000000000"
      },
      "sosa:hasResult": {
```

```
      "@id": "srhackaton:detection_00000000000000000000_0"
    },
    "sosa:useProcedure": {
      "@id": "srhackaton:cv_algorithm_YoloV4"
    }
  },
  {
    "@id": "srhackaton:individual_location_0000000000_result",
    "rdfs:subClassOf": {
      "@id": "sosa:Result"
    },
    "srhackaton:lat": {
      "@type": "xsd:float",
      "@value": "49.019505467042"
    },
    "srhackaton:long": {
      "@type": "xsd:float",
      "@value": "8.4432376969941"
    }
  },
  {
    "@id": "srhackaton:cv_algorithm_YoloV4",
    "@type": "sosa:Procedure"
  },
  {
    "@id": "srhackaton:Frame",
    "sosa:subClassOf": {
      "@id": "sosa:Observation"
    }
  },
  {
    "@id": "srhackaton:detection_00000000000000000000_0",
    "semkg:hasBox": {
      "@id": "srhackaton:detection_00000000000000000000_0_box"
    }
  },
  {
    "@id": "srhackaton:individual_move_0000000000",
    "@type": "srhackaton:IndividualLocation",
    "sosa:hasResult": {
      "@id": "srhackaton:individual_location_0000000000_result"
    }
  },
  {
    "@id": "srhackaton:vehicle2011_09_26_drive_0015",
    "sosa:hots": [
      {
        "@id": "srhackaton:camera2011_09_26_drive_0015"
      },
      {
```

```
      "@id": "srhackaton:object_detector"
    }
  ]
},
{
  "@id": "srhackaton:frame_0000000000",
  "@type": "srhackaton:Frame",
  "sosa:hashResult": {
    "@id": "srhackaton:image_0000000000"
  }
},
{
  "@id": "srhackaton:Detector",
  "rdfs:subClassOf": {
    "@id": "sosa:Sampler"
  }
},
{
  "@id": "srhackaton:IndividualLocation",
  "rdfs:subClassOf": {
    "@id": "sosa:Observation"
  }
},
{
  "@id": "srhackaton:detection_00000000000000000000_0_box",
  "semkg:box_x": {
    "@type": "xsd:float",
    "@value": "483.37110765356766"
  },
  "semkg:box_y": {
    "@type": "xsd:float",
    "@value": "176.59812851956016"
  },
  "semkg:hasDetectedObject": {
    "@id": "category:02961779-n"
  },
  "semkg:height": {
    "@type": "xsd:float",
    "@value": "6.596981890891727"
  },
  "semkg:isIn": {
    "@id": "srhackaton:image_0000000000"
  },
  "semkg:width": {
    "@type": "xsd:float",
    "@value": "10.083831634960676"
  }
},
{
  "@id": "srhackaton:Detection",
```

```
      "rdfs:subClassOf": {
        "@id": "sosa:Sampling"
      }
    },
    {
      "@id": "srhackaton:IndividualMove",
      "rdfs:subClassOf": {
        "@id": "sosa:Observation"
      }
    },
    {
      "@id": "srhackaton:object_detector",
      "@type": "srhackaton:Detector",
      "sosa:madeSampling": {
        "@id": "srhackaton:detection_0000000000"
      }
    },
    {
      "@id": "srhackaton:image_0000000000",
      "semkg:height": {
        "@type": "xsd:Integer",
        "@value": "375"
      },
      "semkg:width": {
        "@type": "xsd:Integer",
        "@value": "1242"
      },
      "sosa:subClassOf": {
        "@id": "sosa:Result"
      }
    },
    {
      "@id": "srhackaton:individual_location_0000000000",
      "@type": "srhackaton:IndividualLocation",
      "sosa:hasResult": {
        "@id": "srhackaton:individual_location_0000000000_result"
      },
      "sosa:resultTime": "2021-04-10T10:10:0.0"
    }
  ]
}
```


C Links to Wikidata