A Scalable Reasoning and Learning Approach for Neural-Symbolic Stream Fusion

Danh Le-Phuoc,[1] **Thomas Eiter,** [2] **Anh Le-Tuan** [1]

[1] Open Distributed Systems, Technical University Berlin, Germany
[2] Institute of Logic and Computation, Vienna University of Technology (TU Wien), Austria
danh.lephuoc@tu-berlin.de, eiter@kr.tuwien.ac.at, anh.letuan@tu-berlin.de

Abstract

Driven by deep neural networks (DNN), the recent development of computer vision makes vision sensors such as stereo cameras and Lidars ubiquitous in autonomous cars, robotics and traffic monitoring. However, a traditional DNN-based data fusion pipeline like object tracking has to hard-wire an engineered set of DNN models to a fixed processing logic, which makes it difficult to infuse new models to that pipeline. To overcome this, we propose a novel neural-symbolic stream reasoning approach realised by semantic stream reasoning programs which specify DNN-based data fusion pipelines via logic rules with learnable probabilistic degrees as weights. The reasoning task over this program is governed by a novel incremental reasoning algorithm, which lends itself also as a core building block for a scalable and parallel algorithm to learn the weights for such program. Extensive experiments with our first prototype on multi-object tracking benchmarks for autonomous driving and traffic monitoring show that our flexible approach can considerably improve both accuracy and processing throughput compared to the DNN-based counterparts.

Introduction

The recent development of computer vision (CV) driven by deep neural networks (DNN) makes visual sensors such as stereo cameras and Lidars ubiquitous in autonomous cars, robotics and traffic monitoring. In particular, many DNN models for object detection (Liu et al. 2019) and tracking (Ciaparrone et al. 2019) are available. For instance, many object tracking algorithms use the Hungarian method to heuristically optimise the likelihood of accurately inferring object locations and trajectories. To achieve an expected performance (accuracy and throughput), one has to manually test several association hypotheses and tune many parameters to find the right set of DNN models and wiring logics for a specific application or dataset. In most of the cases, it is difficult to explain the outputs of the final algorithms and to reuse some association hypotheses that might deliver better results for another parameter configuration or processing pipeline.

To overcome this, we propose a novel neural-symbolic stream reasoning approach to specify general DNN-based data fusion pipelines in a semantic stream reasoning program via logic rules that have learnable probabilistic degrees as weights. In a nutshell, our approach is realized in an integrated framework, *Semantic Stream Reasoning* (SSR), that (a) can emulate common computer vision algorithms fusing multiple deep learning models via imperative programming; (b) allows one to translate heuristics, multiple hypotheses (Kim et al. 2015; Rezatofighi et al. 2015) and domain-specific observations (Ciaparrone et al. 2019) into rules; and (c) can fit rules with/learn them from labeled data as weight rules (clustering to if-then-else or classification logics with DNN models).

Contributions and Novelty

The novelty and advance of our approach consists in the interwoven combination of the following key contributions.

Integrated framework The first contribution of the paper is a general integrated framework for semantic stream fusion and reasoning on top that supports automated learning. The framework is realized with a sophisticated prototype implementation. A first benefit of having these capabilities in one framework is that the user is relieved from specifying rule weights as required in other approaches such as the one in (Suchan, Bhatt, and Varadarajan 2019). In particular, (Suchan, Bhatt, and Varadarajan 2019) requires users to have a good knowledge on ASP (Answer Set Programming) weak constraint rules and to know the right weights to specify the association hypotheses corresponding to the application logic. In contrast, our framework can learn such weights from labeled data automatically. This seems to be easy to achieve by simply using an off-the-self ASP-based learning algorithm like in (Lee and Wang 2018). However, aligning the semantics of programs in (Suchan, Bhatt, and Varadarajan 2019) with the one in (Lee and Wang 2018) is not trivial, and scalability is an issue. Our unified, clear formalization of a data model, reasoning program and learning task for neural-symbolic stream data reveals temporal and modularity properties allowing us to design more efficient and scalable algorithms.

Incremental reasoning algorithm The second contribution is a novel incremental reasoning algorithm that addresses the shortcomings of previous works (Suchan, Bhatt, and Varadarajan 2019; Aditya, Yang, and Baral 2019) which are similar to ours in incorporating common sense and domain

knowledge to exploit the semantic information of the data in combination with DNNs to achieve better accuracy in object tracking (Suchan, Bhatt, and Varadarajan 2019) and in image understanding (Aditya, Yang, and Baral 2019), respectively. However, they used canonical (one-shot) ASP solving and other episodic evaluation methods, respectively. For streaming cycles of 50-100ms, their solving mechanism has to restart and finish each time anew, not to mention that such an ASP/ILP solving program may easily fall into exponential complexity if not carefully translated. Notably, Suchan et al. reported for their system a drop from real-time speed (31 FPS) for 10 objects in one image frame (no. tracks) to only 2 FPS for 50 objects; thus a 5-fold processing states/program size caused a 15 times slower processing speed. To overcome this shortcoming, our incremental reasoning algorithm builds on multi-shot solving mechanism (Gebser et al. 2019). Moreover, it exploits the temporal, succinct and volatile nature of the stream data and the semantic stream reasoning program to establish links to module theory (Oikarinen and Janhunen 2006) that allow us to boost performance, as shown in our experiments.

Scalable learning Our third contribution is a parallel learning algorithm that uses the incremental reasoning algorithm as a core building block, geared to overcome the grounding and solving bottlenecks of ASP-learning algorithms such as (Lee and Wang 2018), when dealing with large and repetitive programs, especially over stream data. In particular, the learning algorithm of (Lee and Wang 2018) does not scale to stream data with 100 times more samples than its evaluated scenarios; the available implementation can only learn from dozens of samples. Our parallel learning algorithm incorporates a sophisticated use of module theory to split time-series data into succinct partitions based on sliding window operations, to exploit computation at previous timesteps via the incremental computing capability powered by our incremental reasoning algorithm; we believe this is not only a novel but also a considerable advancement to (Lee and Wang 2018).

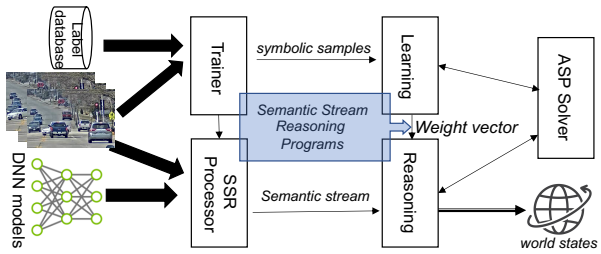
Our experiments with real data on traffic monitoring from the AI City Challenge (Tang et al. 2019) and autonomous driving from the KITTI dataset (Geiger, Lenz, and Urtasun 2012) show that our approach can deliver not only better accuracy (5%-15%) but also higher processing throughput than traditional DNN counterparts. Moreover, our learning algorithm can cope with their training data, which induce thousands of symbolic ground truths in facts and rules.

Overview of the Stream Fusion Framework

While this paper focuses on stream fusion in computer vision with demonstrations around multi-object tracking scenarios, our proposed semantic stream fusion framework is more general. In an abstract view, it allows one to specify the flow of uncertain data coming in streams from subsymbolic models (DNNs) or sensors in a data fusion and reasoning pipeline, which efficiently outputs a stream of current states of the world that are regarded as most likely given the input.

The framework comprises several components, visible in Figure 1. At the base is a *neuro-symbolic stream model*,

Figure 1: The neural-symbolic stream fusion framework



which describes the elements of DNN models and sensors in a semantic data model, resorting to a standard for sensors; the *SSR Processor* component maps the data to the symbolic representation. On top of this model is a *semantic stream reasoning program* that specifies the fusion pipeline and the decision logic for singling out the most likely state of the world at each evaluation. It consists of *reasoning rules*, with *hard rules* for background knowledge given by (non-monotonic) common-sense and domain knowledge that is regarded as "always true", and *soft rules* to express association hypotheses with weights corresponding to probability degrees. The SSR program is evaluated by the *Reasoning* component, which employs an ASP solver. The weights of the rules are determined by the *Learning* component using an ASP solver, based on the symbolic training samples provided by the *Trainer* component.

The workflow of the framework is as follows. For the setup, the Trainer constructs symbolic training samples from labeled data and feeds them to the Learning component, which computes a vector of weights for the soft rules and passes it to the Reasoning component, which sets the weights in the SSR program; our realization uses the novel parallel learning algorithm from above.

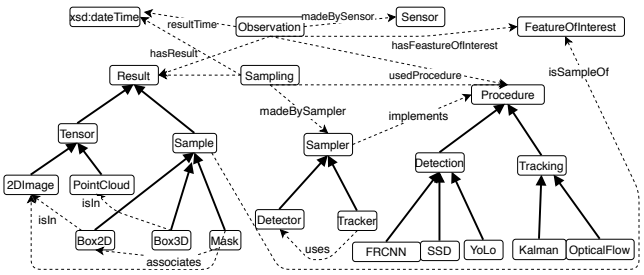
In operation, the SSR Processor produces a semantic stream from the sensor and DNN data streams using a stream processor, which it feeds into the Reasoning component. The latter uses a translation of the SSR program and the stream input into an ASP program that it evaluates on the ASP solver, and outputs a stream of answer sets describing the most likely *world state* at each evaluation. In our realization, the novel incremental reasoning algorithm plays a central role here.

In the following sections, we first describe the model for stream data from DNNs and sensors. We then present semantic stream reasoning programs, which equip a restricted class of temporal rules from (Beck, Dao-Tran, and Eiter 2018) with a Markov logic semantics as in (Lee and Wang 2016), and illustrate how association hypotheses can be expressed by rules in SSR programs. Next, we present our incremental reasoning and learning algorithms, which are followed by the evaluation and addressing related work.

Neural-Symbolic Stream Data Model

To fuse sensor data with DNN models, we describe relations between elements of sensor fusion models, e.g. the JDL data fusion model proposed since the 1990s (Steinberg, Bowman, and White 1999), using a symbolic representation formalism.

Figure 2: Ontology for Neural-Symbolic Stream Fusion



To this end, we extend the standardised Semantic Sensor Network Ontology (SSN) (Haller et al. 2019), which provides vocabularies to specify the semantics of sensors, observations, and samplings as shown in Figure 2. Note that using URI-based symbols from such vocabularies with the associated meaning will facilitate semantic interoperability across distributed sensor sources and reasoners.

Semantically, a multi-sensor fusion data pipeline will consume the data that is observed by a Sensor as *a stream of observations* (represented as an Observation). The example in Figure 3 illustrates this with six image frames observed by a camera. These image frames are represented as instances of the subclass 2DImage of the class Tensor that inherit from the generic Result class of SSN. These observations will then be fed into a probabilistic inference process such as a DNN model or a CV algorithm (represented as a *Procedure*) to provide derived stream elements which then are representing Sampling instances. For instance, a detection model generates an output as a fact $det(b_1, car, 0.8)$ consisting of bounding box b_1, object type car and confidence score 0.8. Similarly, $trk(b_2, 5)$ represents an output of the tracking algorithm (a Tracker that associates bounding box b_2 with the tracklet 5; a tracklet T consists of a series of predicted bounding boxes of an object O that is represented as $trklet(T, O)$).

With this data representation, Figure 3 illustrates how our approach can emulate a typical DNN-based algorithm for multi-object tracking (MOT) via soft rules.

For such algorithms, the tracking-by-detection approach (Ciaparrone et al. 2019) is dominating, with the following key operations: 1) detection of objects, 2) propagating object states (location, velocity, ..) into future frames, 3) associating current detections with existing objects, and 4) managing the lifespan of tracked objects. For example, SORT (Bewley et al. 2016) is a simple object tracking algorithm based on existing DNN detectors such as SSD (Liu et al. 2016) or YOLO (Redmon and Farhadi 2017). To associate resultant detections with existing targets, SORT uses a Kalman filter to predict the new locations of targets in the current frame. Based on this, SORT computes in an association cost matrix between detections and targets based on the intersection-over-union (IOU) distance between each detection and all predicted bounding boxes from the existing targets. Then, the Hungarian algorithm is used to compute the optimal association assignment.

In case some detection is associated with a target, the detected bounding box is used to update the target state via the Kalman filter. Otherwise, the target state is simply predicted without correction using the linear velocity model. For example, in Figure 3 there are no detections of the tracked white car at time points 3-5 because it is occluded by another car; thus, the bounding boxes b_7 and b_{10} are wrong predictions of tracklet 5. To emulate SORT, we later represent these association hypotheses by soft rules that are translated into an optimization problem solved by an ASP Solver.

As in practice the DNN detection models are noisy (e.g. trained on the most popular dataset, COCO (Lin et al. 2014), they have mean precision below 70%), approaches that use association hypotheses with object trajectories akin to SORT often create ID switches, i.e. an object is assigned to different tracking numbers or different tracklets in situation when detection is missing (e.g., by occlusion). For example, when in Figure 3 the white car reappears in frame 6 after being occluded in frames 3-5, a new tracklet 34 is created for it as a new tracked object. To remedy this problem and improve accuracy, one may build more sensitive DNN models. For instance, DeepSORT (Wojke, Bewley, and Paulus 2017) replaces to this end the trajectory-based association metric with a richer metric that combines motion and appearance information. In particular, DeepSORT extends SORT with a DNN providing *discriminative metrics* that is trained on a re-identification dataset to identify targeted objects based on visual appearance. Notably, a semantic stream reasoning program makes it easy to emulate DeepSORT with a DNN model or a traditional CV algorithm by adding some rules.

Semantic Stream Reasoning Programs

To formalise the reasoning process with a semantic representation of stream data, we need a temporal model that allows us to reason about the properties and features of objects. This model must account for the laws of the physical world movement and in particular enable to fill gaps of incomplete information (e.g., if we do not see objects appearing in observations, or camera reads are missing), based on commonsense principles. We start with formalizing semantic streams as above following (Beck, Dao-Tran, and Eiter 2018).

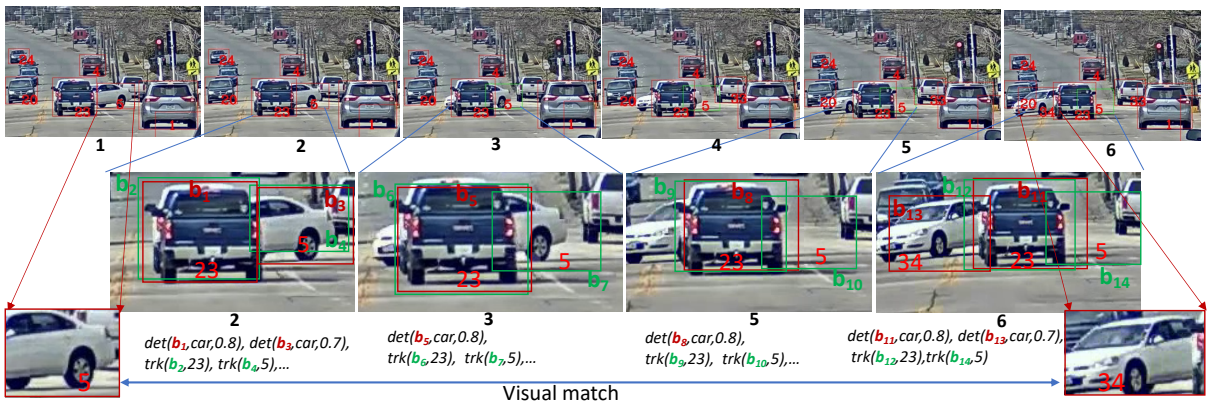
We assume an underlying set $\mathcal{A}$ of propositional atoms and *timelines* T, which are closed intervals $T = [t_s, t_e] \subseteq \mathbb{N}$ of the non-negative integers called *time points*.

Definition 1 *A semantic stream $S = (T, \upsilon)$ consists of a timeline T and an* evaluation function $\upsilon\colon \mathbb{N} \mapsto 2^{\mathcal{A}}$.

Intuitively, a semantic stream S associates with each time point a set of symbolic atoms. The current time point (now) is typically the end of T, which describes a window to the (recent) past; however, now might be inside T, which then describes a bounded horizon into the future. If $S = (T, \upsilon)$ and $S' = (T', \upsilon')$ are streams such that $T' \subseteq T$ and $\upsilon'(t') \subseteq \upsilon(t')$ for all $t' \in T'$, we write $S' \subseteq S$ and call S' a *substream* or *window* of S. To achieve a certain processing throughput or delay for an online stream fusion pipeline, we introduce a window mechanism via functions that omit atoms.

Definition 2 (Window Function) *Any (computable) function w that returns, given a stream $S = (T, \upsilon)$ and a time point $t \in T$, a substream $S' \subseteq S$ of S such that $S' = w(S, t)$ is called a* window function.

Figure 3: A Semantic Visual Stream Snapshot



In particular, we will use *(sliding) time-based* window functions $w = \tau(k)$, where $k \geq 0$, which select the atoms at time t in S down to $t - k$.

To express properties over streams, we will borrow restricted LARS formulas (Beck, Dao-Tran, and Eiter 2018) that allow for window and temporal operators, more specifically formulas of the following form ($a \in \mathcal{A}$ and $t \in \mathbb{N}$):

$$\alpha ::= a \mid \neg\alpha \mid \alpha \wedge \alpha \mid \Diamond\alpha \mid \Box\alpha \mid @_t\alpha \mid \boxplus^w\alpha \qquad (1)$$

Here $\Diamond\alpha$ ($\Box\alpha$) means that α is true somewhere (everywhere) in the stream; $@_t\alpha$ that α is true at time t; and $\boxplus^w\alpha$ that α is true in the substream obtained by the window function w.

Example 1 *Suppose an atom $inFoV(car_1)$ states that car_1 is in the "Field of View" of a camera. Then $\boxplus^3\Box inFoV(car_1)$ holds at time t, if car_1 has always been the field of view during the time points $t-3$, $t-2$, $t-1$, t.*

To define semantics formally, we consider structures $M = \langle S, W, B\rangle$ where $S = (T, \upsilon)$ is a stream, W is a set of window functions (which we fix to $W = \{\tau(k) \mid k \geq 0\}$), and $B \subseteq \mathcal{A}$ is a set of atoms modeling static *background data*. We now define when a ground formula holds in a structure.

Definition 3 (Entailment) *Let $M = \langle S^\star, W, B\rangle$ be a structure, where $S^\star = (T^\star, \upsilon^\star)$, and let $S = (T, \upsilon)$ be a substream of $S^\star$. Moreover, let $t \in T^\star$. The* entailment *relation $\Vdash$ between (M, S, t) and ground formulas is as follows:*

$$\begin{array}{lll}
M,S,t \Vdash a & \textit{iff} & a \in \upsilon(t) \textit{ or } a \in B, \textit{ for atom } a \in \mathcal{A}\\
M,S,t \Vdash \neg\alpha & \textit{iff} & M,S,t \nVdash \alpha,\\
M,S,t \Vdash \alpha \wedge \beta & \textit{iff} & M,S,t \Vdash \alpha \textit{ and } M,S,t \Vdash \beta,\\
M,S,t \Vdash \Diamond\alpha & \textit{iff} & M,S,t' \Vdash \alpha \textit{ for some } t' \in T,\\
M,S,t \Vdash \Box\alpha & \textit{iff} & M,S,t' \Vdash \alpha \textit{ for all } t' \in T,\\
M,S,t \Vdash @_{t'}\alpha & \textit{iff} & M,S,t' \Vdash \alpha \textit{ and } t' \in T,\\
M,S,t \Vdash \boxplus^w\alpha & \textit{iff} & M,S',t \Vdash \alpha, \textit{where } S' = w(S,t)
\end{array}$$

Intuitively, $M, S, t \Vdash \alpha$, in words (M, S, t) *entails* α, states that α evaluates on S in the context of $S^\star$ at time t to be true. We say that M *satisfies* α (or *is a model of* α) at time t, denoted $M, t \Vdash \alpha$, if $(M, S^\star, t)$ entails α. Satisfaction and the notion of a model are extended to sets of formulas as usual. For convenience, we abbreviate $\boxplus^{\tau(k)}$ with $\boxplus^k$.

We then define programs as follows. We call a formula α an *extended atom*, if it is of one of the forms α', $@_t\alpha'$, $\boxplus^w\Diamond\alpha'$, $\boxplus^w\Box\alpha'$, where $\alpha' \in \mathcal{A}$.

Definition 4 (Program) *A* semantic reasoning program Π *as finite sets of* weighted rules r *of the form*

$$\omega : \alpha \leftarrow \beta \qquad (2)$$

where α has the form a or $@_t a$ with $a \in \mathcal{A}$, $\beta = \beta_1 \wedge \cdots \wedge \beta_n$ is a conjunction of possibly negated extended atoms β_i, and $\omega \in \mathbb{R}$ is an optional weight *of the rule, denoted $w(r)$; if ω is missing, r is a* hard *rule, otherwise a* soft *rule. We call α the* head*, denoted $H(r)$, and β the* body*, denoted $B(r)$, of r.*

Example 2 *The rules (3) and (4) trigger the events "a car enters resp. leaves the FoV of a camera". We use here and in other rules predicates with placeholders (variables), written in upper case, that range over concrete values; "," is conjunction and "not" negation.* [1]

$$30 : @_T enters(O) \leftarrow @_T det(B, car, S), iSO(B, O), \\ not\ \boxplus^5 \Diamond inFoV(O), S >= 0.8 \qquad (3)$$

$$10 : @_T leaves(O) \leftarrow @_T not\ \boxplus^5 \Diamond det(B, car, S), \\ iSO(B, O), inFoV(O), S >= 0.8 \qquad (4)$$

Here $det(B, car, S)$ and $inFoV(O)$ are the detection-with-score resp. in-camera-focus predicates from above, while $iSO(B, O)$ associates a detected bounding box B with an object O. The trigger conditions are set via the time windows which test whether detected bounding boxes associated with the object exists within 5 time points.

Stable models We next define the semantics of Π over data streams, which are streams $D = (T, \upsilon_D)$ where all atoms in D are from a designated set $\mathcal{A}^\mathcal{E} \subseteq \mathcal{A}$ of *extensional* atoms for sensor data; the atoms in $at^\mathcal{I} = at \setminus at^\mathcal{E}$ are *intensional* facts. A stream $I = (T, \upsilon)$ such that $D \subseteq I$ is an *interpretation stream for D*, if at every time point $t \in T$ no atom in $\upsilon(t) \setminus \upsilon_D(t)$ is from $\mathcal{A}^\mathcal{E}$, and the structure $M = \langle I, W, B\rangle$ is an *interpretation (for D)*.

Then M is *model* of Π for D and time point t, written $M, t \Vdash \Pi$, if $M, t \Vdash B(r) \rightarrow H(r)$, where "$\rightarrow$" is defined by "$\neg$" and "$\wedge$" as usual.

[1] User-friendly syntax for ASP and RDF/SPARQL developers is provided in the supplementary material.

To define stable models, we use a reduct $red(\Pi, M, t)$ that retains all rules $r \in \Pi$ such that $M, t \models B(r)$ (intuitively, the rule "fires") and, if r is a soft rule, in addition $M, t \models H(r)$; this amounts to treating soft rules as in LP^{MLN} (Lee and Wang 2016), where violated soft rules are dropped.

Definition 5 *An interpretation $M = \langle I, W, B\rangle$ is a* stable model *of a program Π for data stream D at time t, if (1) $M, t \Vdash red(\Pi, M, t)$ and (2) no interpretation $M' = \langle I', W, B\rangle$ such that $I \subset I'$ fulfills $M', t \Vdash red(\Pi, M, t)$.*

Intuitively, (2) ensures that the retained rules must logically entail each intensional atom in I. By $AS(\Pi, D, t)$ we denote the set of all stable models M of Π for D at time t; we may omit "for D" and/or "at t" if this is clear from the context.

Each stable model $M \in AS(\Pi, D, t)$ is assigned a probability degree $P_\Pi(M, t)$ as follows. Let $v(\Pi, M, t)$ be the set of all soft rules r in Π violated by M at t, i.e., $M, t \models B(r) \wedge \neg H(r)$; then the *weight* of M at t is

$$W_\Pi(M, t) = exp(-\textstyle\sum_{r \in v(\Pi, M, t)} w(r)) \quad (5)$$

and

$$P_\Pi(M, t) = W_\Pi(M, t) \,/\, \textstyle\sum_{M', t \in AS(\Pi, D, t)} W_\Pi(M', t) \quad (6)$$

That is, the probability degree is the weight normalized by the total weight of stable models. Among the stable models M, the most probables (with maximum $P_\Pi(M, t)$) are selected.

Rules with variables as in Example 2 are reduced to variable-free (propositional) rules by grounding them over a (finite) set $\mathcal{C}$ of constant symbols, as customary in ASP, i.e., r is replaced by $r\theta$ for each mapping θ of the variables in r to $\mathcal{C}$; for sorted variables (e.g., for time), different sets $\mathcal{C}$ may be used. Rules must be safe, i.e., every variable must occur in a non-negated extended atom in the rule body. The most probable models of a program Π for a data stream D and time t are then those of the grounded version $gr(\Pi, \mathcal{C})$ of all rules in Π over $\mathcal{C}$.

Association Hypotheses as Reasoning Rules

Next, we will demonstrate how to construct association hypotheses according to CV algorithms, i.e. MOT algorithms (Ciaparrone et al. 2019), using the introduced semantic reasoning programs. To associate a detected bounding box B with an object O, we use soft rules that assert $iSO(B, O)$ based on *explained spatial, temporal, and visual appearance evidences*. Such rules can be used to represent *hypotheses* on temporal relations among detected objects in video frames following a tracking trajectory. When the object's movement is consistent with the constant velocity model, e.g., the Kalman filter used in SORT, and there is a detection associated with its trajectory, an iSO fact is generated by the following rule: [2]

$$\begin{aligned}\omega_3 : iSO(B_1, O) \leftarrow\ & @_T trk(T_1, B_1), @_T det(B_2, OT, S),\\ & trklet(T_1, O), iou(B_1, B_2) \end{aligned} \quad (7)$$

Here, $iou(B_1, B_2)$ states the IOU (intersection over union) condition of the bounding boxes B_1 and B_2 satisfies. As mentioned above, we can also emulate DeepSORT via soft rules that can search for supporting evidence to link a newly detected bounding box from an occluded tracklet using visual appearance associations, e.g. frames 1 and 6 of Figure 3. For this, we search for pairs of bounding boxes from recently occluded tracklets w.r.t. visual appearance. As the search space of possible matches is large, we limit it by filtering the candidates based on their temporal and spatial properties. To this end, we use rules with windows to reason about disconnected tracklets that have bounding boxes visually matched within a window of δ_M time points that are aligned with DeepSORT's gallery of associated appearance descriptors for each tracklet. Based on this gallery of previously tracked boxes, the appearance-based discriminative metrics are computed to recover the identities after long-term occlusions, where the motion is less discriminative. Hence, to connect a newly detected bounding box B_1 that has a visual appearance match with another bounding box B_2 (represented by $vMatch(B_1, B_2)$) of a discontinued tracklet T_2 (represented by $@_{T_e} ends(T_2)$) that ended 3 time points before, we use the following rule:

$$\begin{aligned}\omega_4 : iSO(B_1, O) \leftarrow\ & @_T trk(T_1, B_1), vMatch(B_1, B_2),\\ & iSO(B_2, O), trklet(T_2, O), @_{T_e} ends(T_2),\\ & T < T_e + 3, @_{T_e} \boxplus^{\delta_M} \Diamond trk(T_2, B_2) \end{aligned} \quad (8)$$

To demonstrate how to leverage the available commonsense knowledge for eliminating noisy data and unlikely hypotheses, we employ axioms from the Event Calculus (EC) (Mueller 2015) to enforce consistent values of fluents F, i.e., changeable facts and properties, in time by hard rules. Fluent value formulas $holdsAt(F, \tau)$ in EC amount to $@_\tau F$ (where $F = \alpha$). Similarly, the event occurrence $happens(E, \tau)$ of an event E in EC amounts to $@_\tau E$. This makes employing EC axioms for our common-sense reasoning rules fairly easy.

Recall that $inFoV(O)$ asserts whether object O is in the FOV of the camera. Without loss of generality, we may assume that a time point τ corresponds to a synchronised video frame; we then can enforce consistent states of $inFoV$ via the *law of inertia*. For example, the following rules set the conditions to apply EC axioms to the fluent $inFOV$ with two events corresponding to objects entering and leaving a FOV:

$$initiates(enters(O), inFoV(O), T) \leftarrow @_T enters(O) \quad (9)$$

$$terminates(leaves(O), inFoV(O), T) \leftarrow @_T leaves(O) \quad (10)$$

When an object enters a FoV, its fluent states must be consistent in terms of trajectory and spatial relations, which is checked by further rules. Notably, the rules about occlusions and reappearance after occlusions proposed in (Suchan, Bhatt, and Varadarajan 2019) can be expressed by employing the trajectory axioms of EC. Moreover, we use also the well-known RCC-5 axioms (Skiadopoulos and Koubarakis 2004) in an efficient ASP-encoding from (Izmirlioglu and Erdem 2018) to put constraints on the object movements. The possibility to plug in avaiable ASP encodings for employing theories such as EC and RCC-5 underlines the flexibility of our approach in modifying the application logic of a data fusion pipeline.

Incremental Reasoning Algorithm

In the streaming setting, the reasoning task on the program Π is to *continuously compute at each timestep t the most probable model M for the data stream D at t*, in a *MAP (maximum*

[2] All rules of the program used are in the supplementary material.

Algorithm 1 Incremental Reasoning

Input: program $\Pi = \Pi^h \cup \Pi^s$ with hard rules Π^h, soft rules Π^s; data stream D; time point t
Output: optimal answer set M^* of Π for D at t

1: $\Pi^{sc} \leftarrow \emptyset$
2: **for** $r \in \Pi^s$ **do**
3: $\Pi^{sc} \leftarrow \Pi^{sc} \cup \{v(r) \leftarrow B(r), not\ H(r)\}$ // guess for
$\cup \{H(r) \leftarrow B(r), not\ v(r)\}$ // soft rule
$\cup \{v(r)[\omega(r)@0, i^r, var(r)]\}$ // sum up weights
4: **end for**
5: $\Pi^b \leftarrow rewrite\_Base(\Pi^h \cup \Pi^{sc})$
6: $R \leftarrow \{\#external\ q^\alpha(\mathbf{X}) \mid \alpha(\mathbf{X})$ occurs in $\Pi^h \cup \Pi^{sc}\}$
7: $S \leftarrow createSolver(\Pi^b, R)$
8: $S.ground(\Pi^b)$
9: **while** true **do**
10: **for** $a \in E(R,t)$ **do**
11: $S.set\_external(a)$
12: **end for**
13: $M^* \leftarrow$ S.solve()
14: **end while**

a posteriori) approach. To fulfill the subsecond delay requirement of stream processing, we introduce a novel incremental stream reasoning algorithm described in Algorithm 1.

In lines(1)-(4), we rewrite the soft rules Π^s to their weak constraint counter-part (Lee, Talsania, and Wang 2017), where i^r is the index and $var(r)$ are the variable of soft rule r, respectively. Intuitively, lines (2)-(4) translate r into its weight constraint version that allows to violate r, where $v(r)$ accumulates the number of violated ground instances under the intermediate atom structure $unsat(i^r, \omega(r), var(r))$.

Now, the program $\Pi^l = \Pi^h \cup \Pi^{sc} \cup D$ (viewing the data stream D as facts) must be translated into an ASP program that is fed into an ASP solver to find an optimal answer set, from which we obtain an optimal stable model of Π. To avoid the notorious grounding bottleneck of ASP solvers in reasoning tasks over stream data, we translate Π^l to an evolving ASP program for incremental evaluation similar as in (Beck, Eiter, and Folie 2017), where rules are added resp. deleted at each time point. However, different from Beck et al. we exploit multi-shot solving (Gebser et al. 2019) to avoid the continuous grounding overhead, as well as to reuse the solving effort and state from previous time steps as in lines (5)-(14).

The key idea of multi-shot solving is to divide a big program into smaller subprograms that can be joined by means of module theory (Oikarinen and Janhunen 2006), which defines a module as a triple $\mathbb{P} = (P, I, O)$ consisting of a ground logic program P and sets I and O of ground input and output atoms, respectively, such that $I \cap O = \emptyset$, $A(P) \subseteq I \cup O$, and $H(P) \subseteq O$ where $A(P)$ (resp. $H(P)$) is the set of atoms occurring in the rules (resp. rule heads) from P; we let $P(\mathbb{P}) = P$, $I(\mathbb{P}) = I$ and $O(\mathbb{P}) = O$.

We thus construct a base program Π^b and an extensive program R from Π^l as input for the ASP solver in line(7). Specifically, in line(5) the method $rewrite\_Base$ constructs Π^b from $\Pi^h \cup \Pi^{sc}$ by replacing every extended atom $\alpha(\mathbf{X})$ that occurs in $\Pi^h \cup \Pi^{sc}$ and involves @, $\Diamond$ or $\Box$ with an external atom $q^\alpha(\mathbf{X})$ where $\mathbf{X}$ are the variables in α. Then, line(6) builds R based on the $\#external$ directive. For each atom $q^\alpha(\mathbf{X})$, we then adopt the translation approach from Ticker (Beck, Eiter, and Folie 2017) to rewrite it to continuous sub-queries over the stream D of ground facts, where we delegate grounding the external atoms to an efficient stream processing engine such as CQELS (Le-Phuoc et al. 2011). This underlying engine will emit external ground atoms at each time step t via a shared buffer $E(R,t)$, which the incremental algorithm reads in lines (10)-(12) to set the values for the external atoms at every time step t (line(11)).

Algorithm 1 issues successive grounding instructions that result in modules to be joined with the modules of the previous processing state to be solved at line(13). According to (Gebser et al. 2019), such state is built progressively by

$$\mathbb{P}_{t+1} = \mathbb{P}_t \sqcup \mathbb{R}_{t+1}(I(\mathbb{P}_t) \cup O(\mathbb{P}_t))$$

where "$\sqcup$" is the join of modules; the initial module is $\mathbb{P}_0 = (\emptyset, \emptyset, \emptyset)$ and $\mathbb{R}_{t+1}(I(\mathbb{P}_t) \cup O(\mathbb{P}_t))$ represents the grounding state of R at time $t+1$ relative to the current atom base $I(\mathbb{P}_t) \cup O(\mathbb{P}_t)$. We adopt here the usual condition on module joins (Oikarinen and Janhunen 2006), viz. that no cyclic positive dependencies between $\mathbb{R}_{t+1}$ and $\mathbb{P}_t$ exist.

Scalable Learning Algorithm

To free developers from specifying weights, the soft rules in a program Π can be rewritten to create a template program $\hat{\Pi}$ with parametric weights $\mathbf{w} = w_1, \ldots, w_n$ for the soft rules $r_1, \ldots, r_n$ such that concrete weights $\mathbf{w}$ induce a semantic stream reasoning program $\hat{\Pi}(\mathbf{w})$. Following the approach in (Lee and Wang 2018), a weight vector $\mathbf{w}$ can be learned from a set of learning samples $\overline{M}$ generated from a labeled data stream $\overline{D}$. Formally, given a template program $\hat{\Pi}$ and a set of learning samples $\overline{M}$, the learning task is to find an optimal $\mathbf{w}$ using maximum likelihood estimation (MLE), captured by the optimization problem

$$\mathbf{w} = \operatorname*{argmax}_{\mathbf{w}} \prod_{t \in T} P_{\hat{\Pi}(\mathbf{w})}(\overline{M}, t).$$

To optimize the expression via the gradient ascent method, we take the partial derivative of its base-e logarithm w.r.t. w_i:

$$\frac{\partial \ln \prod_{t \in T} P_{\hat{\Pi}(\mathbf{w})}(\overline{M}, t)}{\partial w_i} = \sum_{t \in T} -n_i \cdot (\overline{M^*}, t) + E^i[\overline{M}] \quad (11)$$

where for any stable model M at t, $n_i(M,t)$ is the number of violated ground instances of r_i and $\overline{M^*}$ is the most probable at t; furthermore, $E^i[\overline{M}]$ is the expected number of violated ground instances of r_i in all stable models w.r.t $\overline{M}$, given by $E^i[\overline{M}] = \sum_{t \in T} \sum_{M, t \in AS(\hat{\Pi}(\mathbf{w}), \overline{D}, t)} P_{\hat{\Pi}(\mathbf{w})}(M,t) n_i(M,t)$.

From this equation, we develop Algorithm 2 our scalable learning algorithm based on the one in (Lee and Wang 2018). Similarly as in the latter, the optimisation loop in lines (2)-(12) repeats until the weight vector fulfills the condition on line (12), i.e. the change of every weight component is below a threshold δ. *The key difference is that our algorithm exploits the incremental solving technique* from Algorithm 1 to do

Algorithm 2 Parallel Learning

Input: template program $\hat{\Pi}$, learning samples $\overline{M}$, number of parallel threads N and learning rate λ
Output: weight vector $\mathbf{w}$

1: initialize weight vector $\mathbf{w}^0$, $j = 0$
2: **repeat**
3: $\{\overline{M}^k | k \in 1..N\} \leftarrow split(\overline{M}, N)$
4: **for** $k = 1..N$ **do**
5: $\{\overline{M^*}, t | t \in T\} \leftarrow incr\_solve(\hat{\Pi}(\mathbf{w}^j), \overline{M}^k)$
6: **end for**
7: **for** $w_i^j \in \mathbf{w}$ **do**
8: $\dot{E} \leftarrow par\_approximate(E^i[\overline{M}])$
9: $w_i^{j+1} \leftarrow w_i^j + \lambda \cdot (\sum_{t \in T} -n_i(\overline{M^*}, t) + \dot{E})$
10: **end for**
11: $j \leftarrow j + 1$
12: **until** $max|w_i^j - w_i^{j-1}| < \delta$

line (5) faster and in parallel on N threads, to update then in line (9) the weights in each iteration of the optimisation loop.

In more detail, line(3) splits $\overline{M}$ into N mini-batch substreams $\overline{M}^1$, ..., $\overline{M}^N$ based on the splitting theorem for LP^{MLN} in (Wang et al. 2018). The mini-batches are then fed in line (5) into parallel solving processes with the method $incr\_solve$ using different ASP solver instances.

As regards computing $E^i[\overline{M}]$, our empirical studies showed that there is a significant bottleneck in approximating $E^i[\overline{M}]$ via the MC-ASP algorithm (Lee and Wang 2018) which is an ASP variant of MC-SAT for Markov Logic (Poon and Domingos 2006) to sample stable models. For instance, it might take nearly an hour to generate a small set of samples from a small ground truth program with a few dozens of atoms and rules. Our observation is in line with authors's that the serial sampling procedure via Xorro (Everardo et al. 2019) is the main source for this bottleneck. To overcome this, we develop the parallel sampling method $par\_approximate$ on line (8) using similar XOR-parity constraints as in (Gomes, Sabharwal, and Selman 2006) to modify the original MC-ASP algorithm for approximating $E^i[\overline{M}]$ in lines (7)-(10) in an incremental and parallel fashion similar as in lines (3)-(6). More details are given in the supplementary material.

The key improvement in terms of performance is the tight integration with theory propagators (Gebser et al. 2016) for parity-constraints which can directly access the processing states of the concurrent solvers via Clingo's API for C. This also paves the way for further performance improvements using the novel technique to overcome the grounding bottleneck due to the constraints in (Cuteri et al. 2020).

Experimental Evaluation

Implementation. We have implemented the reasoning and learning algorithms in Java to exploit the code bases of CQELS and Ticker, which are key elements in the open source prototype system SSR that has been realized based on the stream fusion framework described in the Introduction. We use the Java native interface to wrap C/C++ libraries of Clingo 5.4.0 as ASP Solver and NVidia CUDA 10.2 as DNN inference engine. The solving and inference tasks are coordinated in an asynchronous multi-threading fashion on multiple CPU&GPU cores. We extended CQELS (Le-Phuoc et al. 2011) as the underlying stream processing engine to enable DNN inference on GPUs as built-in functions of its query language CQELS-QL. Following (Gebser et al. 2019), the external atoms of R in Algorithm 1 are expressed as CQELS-QL queries to delegate incremental grounding operations to CQELS's incremental evaluation algorithms. The built-in functions also generate the symbolic counterparts of the outputs of DNN models resp. CV algorithms. Adopting the idea of over-grounding (Calimeri et al. 2019), which pre-grounds rules for later use, continuous grounding overhead is avoided by using incremental materialised views with efficient data structures and caching mechanisms (Le-Phuoc 2017) for dynamic rule activation.

Table 1: Evaluation datasets: KITTI and AIC

Datasets	no. labeled frames	no. streams	avg./max. no. obj/fr	no. ground truth atoms	learning time
KITTI	6k	20	4/10	10k	35 hrs
AIC	7.5k	25	10/30	50k	54 hrs

Empirical Evaluation. We evaluated SSR with MOT pipelines for autonomous cars on the KITTI dataset (Geiger, Lenz, and Urtasun 2012), which is a well-known benchmark for autonomous cars, and for traffic surveillance on the AI City Challenge (AIC) dataset (Tang et al. 2019), from which we picked Scenario 4 that tracks vehicles in 25 cameras; we used 13 of them for training and 12 for evaluation. Profile information of the datasets is shown in Table 1.

We have conducted all experiments on a workstation with 2 Intel Xeon Silver 4114 processors having 10 physical cores each, 1TB RAM, 2 NVIDIA Tesla V100 16GB running Centos 7.0. In weight learning, we used starting weights 1, $\delta = 0.001$, and learning rate $\lambda = 0.01$ for both learning pipelines.

With the weights learned, we compared SSR against SORT and DeepSORT (DSORT) using different detectors, viz. Faster-RCNN (Ren et al. 2015), Yolov3 (Redmon and Farhadi 2017), and SSD (Liu et al. 2016) on the data sets w.r.t. three MOT metrics: IDF1 (the ratio of correctly identified detections), MOTA (MOT Accuracy in %), and FPS (frames/sec throughput).

Results & Discussions. The last column of Table 1 reports the learning time on each dataset; they amount to the learning time of approximately 1000-2000 epochs with the Siamese network (Koch, Zemel, and Salakhutdinov 2015) that provides appearance-based discriminative metrics of DeepSORT.

The results in Table 2 show that SSR achieved consistently better IDF1 and MOTA (10-15% more accurate) than SORT, which serves as a baseline, and DSORT (5-7%). Regarding the throughput, SSR can deliver real-time speed with 15-30 FPS, which is 4-6 times the throughput of DSORT and just a little lower than that of SORT (80-90%); this is outweighed by significantly higher detection and tracking accuracy.

To analyse the ASP solving overhead of Algorithm 1 compared to the DNN inference counterpart, we report in Table 3 the average processing time for each batch of inputs of these processes on the datasets. The results show that the time

Table 2: MOT metrics and throughputs on AIC and KITTI

Detectors	System	AIC			KITTI		
		IDF1	MOTA	FPS	IDF1	MOTA	FPS
FRCNN	SORT	54	52	**22**	51	49	**22**
	DSORT	61	59	4	60	55	5
	SSR	**70**	**67**	19	**68**	65	20
YOLOV3	SORT	57	53	**30**	55	51	**25**
	DSORT	63	60	5	63	59	6
	SSR	**71**	**69**	27	**69**	**67**	22
SSD	SORT	51	48	**32**	49	45	**37**
	DSORT	55	51	6	57	52	8
	SSR	**62**	**58**	31	**65**	**63**	**30**

Table 3: DNN Inference vs. ASP Solving Overhead

Overhead	AIC			KITTI		
	FRCNN	YOLOV3	SSD	FRCNN	YOLOV3	SSD
DNN	95 ms	85 ms	70 ms	90 ms	82 ms	65 ms
ASP	45 ms	43 ms	39 ms	29 ms	30 ms	20 ms

spent on the ASP solver is not a dominant overhead of the processing pipeline, viz. 28-57% (avg. 41%) of the DNN counterparts. Moreover, SSR uses multiple instances of Clingo concurrently; hence, even if ASP solving takes 20-50 ms per processing batch (approx. 60-120kB program size), by using 40 parallel solving threads (2 threads per physical core) SSR can produce 50-200 learning samples per second; this outperforms the original algorithm in (Lee and Wang 2018) by an order of magnitude.

Accessibility. All source code and experiments will be released at https://github.com/cqels/SSR/ as a part of the open source project CQELS Framework (Le-Phuoc et al. 2011).[3]

Related Work

In Computer Vision, a DNN-based system heavily depends on expert knowledge about the data distribution of the targeted scenario for fitting it to the proper model. However, many factors can seriously interfere with its performance, such as occlusion, illumination, fuzziness, or noise interference (cf. Figure 3). To jointly deal with those factors in the same pipeline, one has to encode the joint distributions via cascading DNN models and/or manually tuned lists of parameters based on the association hypotheses that emerged from the data and empirical analysis. In fact, like our method, traditional ones such as the Joint Probabilistic Data Association Filter (JPDAF) (Rezatofighi et al. 2015) and Hypothesis Tracking (MHT) (Kim et al. 2015) use multiple hypotheses. These methods perform data association on a frame-by-frame basis. The JPDAF generates a single state hypothesis by weighting individual measurements with their association likelihoods. In MHT, all possible hypotheses are tracked, but pruning schemes must be applied for tractability. Hence, considerable domain knowledge is needed for translating them to a specific optimization problem. Moreover, heuristic tricks and parameters tuning that are only useful for a certain application and dataset. To remedy the shortcomings of such a traditional programming approach, our reasoning programs offer not only modularity and portability, but also provide a logical basis for explaining the outputs they produce.

In the knowledge representation community, the closest work to ours is (Suchan, Bhatt, and Varadarajan 2019), which expresses the probability optimisation problems of MOT in ASP using weak constraints. Apart from our highlighted advantages (i.e supporting weight learning and more efficient with multi-shot solving) against this approach at the beginning of the paper, our approach with semantic reasoning rules is more compact and more flexible with *window-based search*. For instance, it is easy for our approach to incorporate additional DNNs to the DeepSORT tracking logic with just an additional rule in equation (8) which is not trivial to do with (Suchan, Bhatt, and Varadarajan 2019). Moreover, (Suchan, Bhatt, and Varadarajan 2019) and other approaches surveyed in (Aditya, Yang, and Baral 2019) have aforementioned shortcomings in using ASP or ILP solvers with the one-shot solving mechanism. On the nature of input data, while the work in (Aditya, Yang, and Baral 2019) only considered still images, we aim at visual streams to exploit the temporal relation and semantic information among a series of visual data, e.g video frames to apply the multi-shot solving mechanism. Similarly, DeepProbLog (Manhaeve et al. 2018) and NeurASP (Yang, Ishay, and Lee 2020) also integrate DNNs with rules; however, their focus is on static data while ours is dealing with stream data processed by neural networks to achieve performance and scalability via deriving incremental and parallel algorithms based on succinct natures of the stream data. Investigating how the latter can be transferred to these approaches remains for future work. For instance, a tighter integration of our learning algorithm with the inner DNN learning loops is the next foreseeable step in our development of SSR. Along this line, extending our framework with online learning and feedback of rules to the DNNs is suggestive; the latter is conceptually straightforward under temporal stratification.

Conclusion and Future Work

We have presented a novel semantic reasoning approach which seamlessly marries reasoning based on Answer Set Programming (ASP) with Deep Neural Network (DNN) inference for building advanced data fusion pipelines. We have demonstrated the flexibility and amenability of the approach for reuse of existing solutions by emulating MOT algorithms that constitute popular DNN-based stream fusion pipelines. Our extensive experiments on reasoning and learning showed that our prototype SSR can deliver higher processing throughput and yield better accuracy than DNN counterparts, and that SSR's weight learning algorithm can scale to deal with realistic benchmark datasets. We believe that the open source platform for SSR will create a timely bridge to integrate the strengths of ASP and DNN for building fast and scalable stream fusion pipelines, where rules and semantic information provide a valuable basis for addressing challenging issues like explainability and verification in future work. Furthermore, we plan to apply our work beyond computer vision to other application domains such as traffic forecasting and streaming perceptions for robotics.

[3]CQELS Execution Framework, https://cqels.org/

Acknowledgments

We appreciate the comments of the reviewers and the AAAI 2021 chairs, which helped us to improve the presentation of this paper.

This work was funded in part by the German Ministry for Education and Research as BIFOLD - Berlin Institute for the Foundations of Learning and Data (refs 01IS18025A and 01IS18037A), by the Marie Skodowska-Curie Programme H2020-MSCA-IF-2014 (SMARTER project) under Grant No. 661180 and supported by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) under the program ICT of the Future (FFG-PNr.: 861263, project DynaCon).

References

Aditya, S.; Yang, Y.; and Baral, C. 2019. Integrating Knowledge and Reasoning in Image Understanding. In *IJCAI*, 6252–6259.

Beck, H.; Dao-Tran, M.; and Eiter, T. 2018. LARS: A Logic-based framework for Analytic Reasoning over Streams. *Artif. Intell.* 261: 16–70.

Beck, H.; Eiter, T.; and Folie, C. 2017. Ticker: A system for incremental ASP-based stream reasoning. *TPLP* 17(5-6): 744–763.

Bewley, A.; Ge, Z.; Ott, L.; Ramos, F.; and Upcroft, B. 2016. Simple online and realtime tracking. In *ICIP*, 3464–3468.

Calimeri, F.; Ianni, G.; Pacenza, F.; Perri, S.; and Zangari, J. 2019. Incremental Answer Set Programming with Overgrounding. *TPLP* 19(5-6): 957–973.

Ciaparrone, G.; Sánchez, F. L.; Tabik, S.; Troiano, L.; Tagliaferri, R.; and Herrera, F. 2019. Deep learning in video multi-object tracking: A survey. *Neurocomputing* ISSN 0925-2312.

Cuteri, B.; Dodaro, C.; Ricca, F.; and Schüller, P. 2020. Overcoming the Grounding Bottleneck Due to Constraints in ASP Solving: Constraints Become Propagators. 1688–1694. ijcai.org.

Everardo, F.; Janhunen, T.; Kaminski, R.; and Schaub, T. 2019. The Return of xorro. volume 11481 of *Lecture Notes in Computer Science*, 284–297. Springer.

Gebser, M.; Kaminski, R.; Kaufmann, B.; Ostrowski, M.; Schaub, T.; and Wanko, P. 2016. Theory Solving Made Easy with Clingo 5. volume 52 of *OASICS*, 2:1–2:15.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2019. Multi-shot ASP solving with clingo. *Theory Pract. Log. Program.* 19(1): 27–82.

Geiger, A.; Lenz, P.; and Urtasun, R. 2012. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *CVPR*.

Gomes, C. P.; Sabharwal, A.; and Selman, B. 2006. Near-Uniform Sampling of Combinatorial Spaces Using XOR Constraints. 481–488. MIT Press.

Haller, A.; Janowicz, K.; Cox, S. J. D.; Lefrançois, M.; Taylor, K.; Le-Phuoc, D.; Lieberman, J.; García-Castro, R.; Atkinson, R.; and Stadler, C. 2019. 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.

Izmirlioglu, Y.; and Erdem, E. 2018. Qualitative Reasoning About Cardinal Directions Using Answer Set Programming. 1880–1887. AAAI Press.

Kim, C.; Li, F.; Ciptadi, A.; and Rehg, J. M. 2015. Multiple Hypothesis Tracking Revisited. In *ICCV*, ICCV '15, 4696–4704. ISBN 978-1-4673-8391-2.

Koch, G.; Zemel, R.; and Salakhutdinov, R. 2015. Siamese Neural Networks for One-shot Image Recognition. In *ICML*.

Le-Phuoc, D. 2017. Operator-aware approach for boosting performance in RDF stream processing. *J. Web Semant.* 42: 38–54.

Le-Phuoc, D.; Dao-Tran, M.; Parreira, J. X.; and Hauswirth, M. 2011. A Native and Adaptive Approach for Unified Processing of Linked Streams and Linked Data. In *ISWC*, 370–388.

Lee, J.; Talsania, S.; and Wang, Y. 2017. Computing LPMLN using ASP and MLN solvers. *Theory Pract. Log. Program.* 17(5-6): 942–960.

Lee, J.; and Wang, Y. 2016. Weighted Rules under the Stable Model Semantics. In *KR*, 145–154.

Lee, J.; and Wang, Y. 2018. Weight Learning in a Probabilistic Extension of Answer Set Programs. In *KR*, 22–31.

Lin, T.; Maire, M.; Belongie, S. J.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; and Zitnick, C. L. 2014. Microsoft COCO: Common Objects in Context. volume 8693, 740–755. Springer.

Liu, L.; Ouyang, W.; Wang, X.; Fieguth, P.; Chen, J.; Liu, X.; and Pietikäinen, M. 2019. Deep Learning for Generic Object Detection: A Survey. *IJCV* .

Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S. E.; Fu, C.; and Berg, A. C. 2016. SSD: Single Shot MultiBox Detector. In *ECCV*, 21–37.

Manhaeve, R.; Dumancic, S.; Kimmig, A.; Demeester, T.; and Raedt, L. D. 2018. DeepProbLog: Neural Probabilistic Logic Programming. 3753–3763.

Mueller, E. T. 2015. *Commonsense Reasoning: An Event Calculus Based Approach.* 2 edition.

Oikarinen, E.; and Janhunen, T. 2006. Modular Equivalence for Normal Logic Programs. volume 141 of *Frontiers in Artificial Intelligence and Applications*, 412–416. IOS Press.

Poon, H.; and Domingos, P. M. 2006. Sound and Efficient Inference with Probabilistic and Deterministic Dependencies. AAAI Press.

Redmon, J.; and Farhadi, A. 2017. YOLO9000: Better, Faster, Stronger. In *CVPR*, 6517–6525.

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *NIPS*, 91–99.

Rezatofighi, S. H.; Milan, A.; Zhang, Z.; Shi, Q.; Dick, A. R.; and Reid, I. D. 2015. Joint Probabilistic Data Association Revisited. In *ICCV*, 3047–3055. IEEE Computer Society. ISBN 978-1-4673-8391-2.

Skiadopoulos, S.; and Koubarakis, M. 2004. Composing cardinal direction relations. *Artif. Intell.* 152(2): 143–171.

Steinberg, A. N.; Bowman, C. L.; and White, F. E. 1999. Revisions to the JDL data fusion model. In *Sensor Fusion: Architectures, Algorithms, and Applications III*, 430 – 441.

Suchan, J.; Bhatt, M.; and Varadarajan, S. 2019. Out of Sight But Not Out of Mind: An Answer Set Programming Based Online Abduction Framework for Visual Sensemaking in Autonomous Driving. In *IJCAI-19*, 1879–1885.

Tang, Z.; Naphade, M.; Liu, M.-Y.; Yang, X.; Birchfield, S.; Wang, S.; Kumar, R.; Anastasiu, D.; and Hwang, J.-N. 2019. CityFlow: A City-Scale Benchmark for Multi-Target Multi-Camera Vehicle Tracking and Re-Identification. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Wang, B.; Zhang, Z.; Xu, H.; and Shen, J. 2018. Splitting an LPMLN Program. 1997–2004. AAAI Press.

Wojke, N.; Bewley, A.; and Paulus, D. 2017. Simple online and realtime tracking with a deep association metric. In *ICIP*, 3645–3649.

Yang, Z.; Ishay, A.; and Lee, J. 2020. NeurASP: Embracing Neural Networks into Answer Set Programming. 1755–1762. ijcai.org.