Answer Set Programming for Stream Reasoning

Thang M. Do, Seng W. Loke, and Fei Liu

Dept. of CSCE, La Trobe University, Australia
mtdo@students.latrobe.edu.au,
{S.Loke,F.Liu}@latrobe.edu.au

Abstract. This paper explores Answer Set Programming (ASP) for stream reasoning with data retrieved continuously from sensors. We describe a proof-of-concept with an example of using declarative models to recognize car on-road situations.

Keywords: ASP, stream reasoning, semantic sensor based applications.

1 Introduction

A new concept of "stream reasoning" has been proposed in [8]. Recently, dlvhex, an extension of ASP, has been introduced as one candidate for rule-based reasoning for the Semantic Web [6]. dlvhex uses the semantic reasoning approach which makes it fully declarative and always terminating. dlvhex can deal with uncertain data (via using disjunctions rules to generate multiple answer sets), interoperate with arbitrary knowledge bases (to query data) and different reasoning frameworks (e.g., higher-order logic, for more reasoning power). However, to our knowledge, using dlvhex and ASP for stream reasoning is new.

Our research has three aims i) to introduce a prototype of dlvhex stream reasoning, ii) to formalize ASP for building stream reasoning systems, and iii) to further apply Semantic Web techniques (OWL) for sensor-based applications. The contribution is to propose a framework and theoretical formulation for building ASP-based stream reasoning systems with a focus on sensor stream applications.

There has been research to apply ASP in wireless sensor networks applications such as home-based health care services for aged people [7] or dealing with ambiguous situations [4], but the concept of stream reasoning was not introduced. To implement declarative stream processing, the logic framework of LarKC [5] basaed on aggregate functions (rather than logic programming semantic) of the C-SPARQL [3] language. The logic framework in [1] reasons with a stream of pairs of RDF triples and a timestamp but the ability to deal with unstable data was not mentioned. Therefore, we see the necessity to have a foundation for ASP-based stream reasoning and to investigate its feasibility.

2 ASP-Based Stream Reasoning: A Conceptual Model

In this section, we describe a conceptual model that formalizes ASP-based stream reasoning that processes streams of data into answer sets. We describe: i) a general abstract architecture of a stream reasoning system, ii) a formal model of data streams, and iii) a formalization of an ASP based stream reasoner.

Abstract Architecture. A stream reasoning system has three main components, which are sensor system, data stream management system (DSMS) [2] and stream reasoner illustrated in Figure 1 (SSN for Semantic Sensor Network).

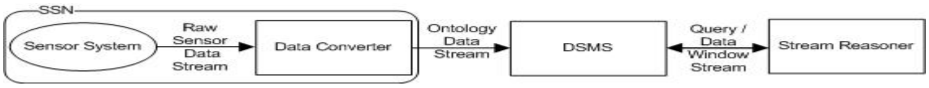

Fig. 1. Simple stream reasoning system

Notation. We introduce the notation which is used in the next two sections.

- *dr*: is the time period between the starting time and the finishing time of a reasoning process which is always terminates.
- *ds*: is the time period between the starting time and the finishing time of a sensor taking a data sample (usually very small).
- Δs: is the time period between the two start times of taking two consecutive data samples of a sensor. The sample rate f_s is: $f_s = 1/\Delta s$.
- Δr: is the time period between the two start times of two consecutive reasoning processes of the reasoner. The reasoning rate f_r is: $f_r = 1/\Delta r$.

There are two communication strategies between the DSMS and the stream reasoner: push and pull. In the pull method, when the reasoner needs sensor data sample(s), it sends a query to the DSMS which will perform the query and return the data sample(s) to the reasoner. In the push method, the reasoner registers with the DSMS the sensor name from which it wants to have the data sample. The DSMS returns to the reasoner the data sample whenever it is available.

We use the pull method in our prototype to discover the maximum reasoning speed of the reasoner when continuously running as fast as possible.

Data Stream Formalization. This section introduces the formalization of the data stream provided to the stream reasoner. The time when a sample is taken is assumed to be very close to the time when that sample is available for reasoning, otherwise the reasoner will give its result with a consistent delay.

Definition 1 (Data Stream). *Data stream DS is a sequence of sensor data samples d_i ordered by timestamps. $DS = \{(t_1, d_{t_1}), (t_2, d_{t_2}), \ldots, (t_i, d_{t_i}), \ldots\}$ where d_{t_i} is the sensor data sample taken at time t_i, and $t_1 < t_2 < \ldots < t_i < \ldots$.*

Definition 2 (Data Window). *A data window available at time t, W_t, is a finite subsequence of a data stream DS and has the latest data sample taken at time t. The number of data samples, $|W_t|$, of this subset is the size of the window. For $W_t \subseteq DS$, and $t_s = t$: $W_t = \{(t_1, d_{t_1}), (t_2, d_{t_2}), \ldots, (t_s, d_{t_s})\}$ where W_t is data window at time t, $s = |W_t|$: is the size of the window, $t_1 < t_2 < \ldots < t_s$, t_s is the time when the latest sample of the data window is taken, and $d_{t_i} (1 \leq i \leq s)$ is the sensor data sample taken at time t_i.*

The data window can also be defined by a time period, for example, a data window that includes all data samples taken in the last 10 seconds.

Definition 3 (Window Slide Samples). *Window slide samples l is the number of samples counted from the latest sample (inclusive) of one data window to the latest sample (exclusive) of the next data window.*

Definition 4 (Window Slide Time). *Given two continuous data windows W_{t_1} at time t_1 and W_{t_2} at time t_2 ($t_2 \geq t_1$), the time period between t_1 and t_2 is called window slide time Δw, i.e. we have $\Delta w = t_2 - t_1$.*

From Definition 3 we can calculate the window slide time with the formula: $\Delta w = l * \Delta s$. When we use the term "window slide", it means window slide samples or window slide time depending on context.

Definition 5 (Data Window Stream). *Given a data stream DS, a data window stream WS is a sequence of data windows W in time order. $WS = \{(t_1, W_{t_1}), (t_2, W_{t_2}), \ldots, (t_i, W_{t_i}), \ldots\}$ where W_{t_i} is a data window at time t_i, $t_1 < t_2 < \ldots < t_i < \ldots$, and $W_{t_i} \subseteq DS$.*

In dlvhex, we use $\&qW$ to represent a predicate which query a data window from a DSMS. This predicate is extended from the external atoms of dlvhex-dlplugin[1]: $\&qW[|W|, URI, sn](X, V)$ where $\&qW$ is an external predicate that queries a data window from the DSMS, $|W|$ (input) is window size, URI (input) is a Unique Resource Identifier or the file path of the OWL ontology data source, sn (input) is the name of the sensor providing data sample, X (output) is the name of the returned instance of the ontology class that describes the sensor, and V (output) is data sample value returned.

ASP-Based Stream Reasoner. This section introduces a formalization of the stream reasoner of a system model that has one data stream and one reasoner. This is easily extendible to models that have: one data stream providing data for multiple reasoners, one reasoner using data from multiple data streams, and many reasoner using data from multiple data streams.

Definition 6 (Data Window Reasoner). *An* ASP-based data window reasoner *AWR is a function that maps every data window $W \subseteq DS$ to a set SA of answer sets. $AWR : W_S \rightarrow 2^{\Sigma}$ where AWR denotes a ASP-based data window reasoner, W_S is the set of all data windows from data stream DS, Σ denotes the set of all possible answer sets S for any input, and 2^{Σ} is the power set of Σ.*

The reasoner AWR has input data window W and gives a set SA of answer sets: $AWR(W) = SA$, where $SA = \{S_1, S_2, \ldots, S_n\}$, and $S_i \in \Sigma, (1 \leq i \leq n)$.

When using pull communication, the reasoner AWR runs continuously with an interval of Δr, queries input data (not waitting for the data comming like in push method) from a data window stream WS, and gives a stream of sets of

[1] `http://www.kr.tuwien.ac.at/research/systems/dlvhex/download.html`

answer sets SA: $\&aSR(AWR, WS, \Delta r) = \{SA_{t_1}, SA_{t_2}, \ldots, SA_{t_i}, \ldots\}$ $\&aSR$ is the meta operator (or external predicate in dlvhex) that triggers the reasoner AWR to run continuously, Δr is the interval at which the meta operator $\&aSR$ repeatedly executes AWR, and SA_{t_i} is the set of answer sets output at times t_i.

From Definition 6, we have $AWR(W_{t_{i'}}) = SA_{t_i}$, where t_i is the time when the reasoner gives the output, and $t_{i'}$ is the time when the input data becomes available. The input data was available before the reasoning process, so: $t_{i'} < t_i - d_r$. The reasoner use the latest data window, so: $t_{i'} = max(t_j), t_0 \leq t_j < (t_i - d_r)$ where t_l $(l \geq 0)$ is the time when a data window W_{t_l} becomes available.

3 Prototype Implementation and Experimentation

Prototype Implementation. As a proof-of-concept of the model introduced in Section 2, we built a prototype to detect driving situations of a car travelling in public traffic conditions. The system has main components illustrated in Figure 2 and uses models of car situations (e.g., turning left or right) as constraints on sensor data values, defined declaratively as dlvhex programs.

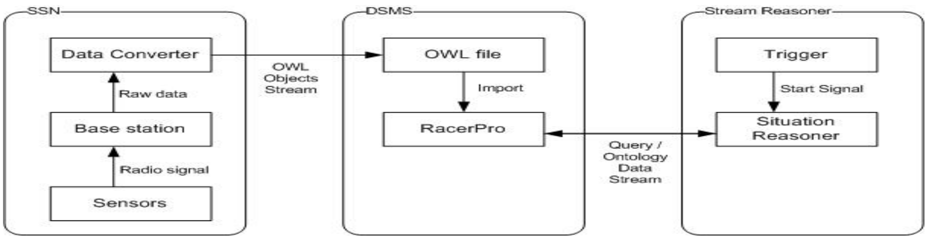

Fig. 2. Prototype design

The sensor system is built using the SunSPOT tool kit version 4.0[2]. We build a simple ontology called Sensor Ontology in the OWL language. Sensor data is placed in a queue in an OWL file which is fed to RacerPro version 1.9.0[3]. This setup simulates a DSMS mentioned in Section 2. The reasoner AWR is built from a dlvhex program[4] and we use a Unix shell script to realize the meta predicate $\&aSR$. The prototype is installed in Ubuntu 9.10 which runs on a Sun virtual box version 3.1.6 on our Windows Vista Fujitsu Lifebook T 4220.

Experimental Setup. We attach the sensor kit in the middle of a car with the sampling rate of the accelerometer was 0.3s/sample, as the maximum speed of the reasoner. AcceX and AcceZ are the horizontal and vertical direction respectively; AcceY is along the forward direction. The AcceX and AcceY values are mapped to the scale [0, 100]. A AcceX data sample of a data window is created as below:

[2] `http://www.sunspotworld.com`
[3] `http://www.racer-systems.com`
[4] `http://www.kr.tuwien.ac.at/research/systems/dlvhex/download.html`

```
<AcceX rdf:ID="AcceX1"><acceValue rdf:datatype="&xsd;positiveInteger">55
</acceValue></AcceX>
```

To query for a data window, we use the dlvhex atom $\&dlDR[URI, a, b, c, d, Q]$ (X, Y):

```
  url("../ontology/driving.owl").
  acceX1(X,Y) :- &dlDR[U,a,b,c,d,"acceValue"](X,Y), X="sensorOnto:AcceX1", url(U).
```

We have to query every single data sample in the data window. With our proposed formula $(\&qW[|W|, URI, sn](X, V))$ we only use one rule to obtain the most recent data window (of size five):

```
acceX(X, Y) :- &qW[5, U, ‘‘sensorOnto:AcceX’’](X, Y).
```

The code below, which is a declarative model of a right turn situation, reasons to recognize “right turn” situations. It implements an ASP-based window reasoner defined conceptually earlier.

```
 % right turn:
 doingRightTurn :- acceX1(X1,Y1), acceX2(X2,Y2), acceX3(X3,Y3), acceX4(X4,Y4),
    acceX5(X5,Y5), #int(S1), #int(S2), #int(S3), #int(S4), #int(Y1), #int(Y2),
    #int(Y3), #int(Y4), #int(Y5), S1=Y1+Y2, S2=S1+Y3, S3=S2+Y4, S4=S3+Y5, S4>277.
```

The bounds (e.g., 277) were obtained via experiments done. Similar rules model other car on-road situations. Because we used a UNIX shell script to trigger the reasoner continuously, the Operating System has to repeatedly load dlvhex, run it, and then unload it. This is resource consuming and can reduce reasoning speed, but provided a fast, though crude implementation of the meta-predicate, adequate for reasoning about car on-road behaviours.

In dlvhex, using rules with disjunctive heads can give several possible answer sets representing several possible situations given the same sensor data readings.

```
doingRightTurn v doingLeftTurn :- acceX1(X1,Y1), acceX2(X2,Y2), ....
```

Results and Evaluation. The maximum reasoning speed of the system is nine (three) times/s without (with) querying ontology data. We tested the system in two running states (normal and delayed) when our car’s speed range is 25-50km/h for straight driving and 25-40km/h for turning, turning angles approximately greater than 30^o, with three data window sizes (one, two and five). The system recognizes turning situations at higher accuracy with higher speed.

With window size five, the system detected 15 left turns and 15 right turns with no error. With window size two, the system detected 10 left turns and 10 right turns with no error. With window size one, the system is very sensitive and often mis-recognizes because accelerometer sensor’s data fluctuates. When the reasoner ran with a small deliberate delay, with window size five, the system detected 10 left turns and 10 right turns with six errors. With window size two, the system detected eight left turns and eight right turns with seven errors.

This result means that, using smaller data window sizes makes the system more sensitive and reduces accuracy, but it can more quickly recognize fine-grained situations also different fine-grained car manoeuvres such as start turning, doing turning, and finish turning. When using larger data window sizes, the system returns more accurate results and deals better with unstable sensor data.

Overall, our system detects turning, stopping, going straight and going over a ramp with high accuracy. The bound values in the rules can be adjusted to change the sensitivity of the system. We could use machine learning to process such sensor data, but our aim is to illustrate a simply proof-of-concept of ASP-based stream reasoning where the stream comprises a sequence of time-stamped OWL objects. This prototype suggests the potential for applications that require up to three reasoning processes per second such as driving assistant.

4 Conclusion and Future Work

This paper has provided a conceptual model of ASP-based stream reasoning, and showed the feasibility of stream reasoning with dlvhex for semantic sensor based applications. This project successfully used OWL objects to represent sensor data (which is more general than even time-stamped RDF triples) and utilized dlvhex to reason with this data. Our future work will: (i) implement repeated reasoning within dlvhex programs themselves to improve the system's performance, and ii) research hybrid ASP-machine learning for stream reasoning.

References

1. Barbieri, D., Braga, D., Ceri, S., Della Valle, E., Grossniklaus, M.: Incremental reasoning on streams and rich background knowledge. In: Aroyo, L., Antoniou, G., Hyvönen, E., ten Teije, A., Stuckenschmidt, H., Cabral, L., Tudorache, T. (eds.) ESWC 2010. LNCS, vol. 6088, pp. 1–15. Springer, Heidelberg (2010)
2. Barbieri, D., Braga, D., Ceri, S., Valle, E.D., Huang, Y., Tresp, V., Rettinger, A., Wermser, H.: Deductive and inductive stream reasoning for semantic social media analytics. IEEE Intelligent Systems 25(6), 32–41 (2010)
3. Barbieri, D.F., Braga, D., Ceri, S., Valle, E.D., Grossniklaus, M.: C-sparql: Sparql for continuous querying. In: Proceedings of the 18th International Conference on World Wide Web, pp. 1061–1062. ACM, New York (2009)
4. Buccafurri, F., Caminiti, G., Rosaci, D.: Perception-dependent reasoning and answer sets (2005), `http://www.ing.unife.it/eventi/rcra05/articoli/BuccafurriEtAl.pdf`
5. Della Valle, E., Ceri, S., Barbieri, D.F., Braga, D., Campi, A.: A first step towards stream reasoning. In: Domingue, J., Fensel, D., Traverso, P. (eds.) FIS 2008. LNCS, vol. 5468, pp. 72–81. Springer, Heidelberg (2009)
6. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: dlvhex: A prover for semantic-web reasoning under the answer-set semantics. In: IEEE / WIC / ACM International Conference on Web Intelligence, pp. 1073–1074 (2006)
7. Mileo, A., Merico, D., Pinardi, S., Bisiani, R.: A logical approach to home healthcare with intelligent sensor-network support. Comput. J. 53, 1257–1276 (2010)
8. Della Valle, E., Ceri, S., van Harmelen, F., Fensel, D.: It's a streaming world! reasoning upon rapidly changing information. IEEE Intelligent Systems 24(6), 83–89 (2009)