DESCRIPTION PLAUSIBLE LOGIC PROGRAMS FOR STREAM REASONING

Ioan Alfred Letia and Adrian Groza
Department of Computer Science, Technical University of Cluj-Napoca, Cluj-Napoca, Romania

Keywords: Stream reasoning, Description logic, Plausible logic, Lazy evaluation, Sensors.

Abstract: Stream reasoning is defined as real time logical reasoning on large, noisy, heterogeneous data streams, aiming to support the decision process of large numbers of concurrent querying agents. In this research we exploit nonmonotonic rule-based systems for handling inconsistent or incomplete information and also ontologies to deal with heterogeneity. Data is aggregated from distributed streams in real time and plausible rules fire when new data is available. This study also investigates the advantages of lazy evaluation on data streams.

1 INTRODUCTION

Sensor networks are estimated to drive the formation of a new Web, by 2015 (Le-Phuoc et al., 2010). The value of the Sensor Web is related to the capacity to aggregate, analyse and interpret this new source of knowledge. Currently, there is a lack of systems designed to manage rapidly changing information at the semantic level (Valle et al., 2009). The solution given by data-stream management systems (DSMS) is limited mainly by the incapacity to perform complex reasoning tasks.

Stream reasoning is defined as real time logical reasoning on huge, possible infinite, noisy data streams, aiming to support the decision process of large numbers of concurrent querying agents. In order to handle blocking operators on infinite streams (like min, mean, average, sort), the reasoning process is restricted to a certain window of concern within the stream, whilst the previous information is discharged (Barbieri et al., 2010). This strategy is applicable only for applications where recent data have higher relevance (e.g. average water debit in the last 10 minutes). In some reasoning tasks, tuples need to be joined arbitrarily far apart from different streams. Stream Reasoning adopts the continuous processing model, where reasoning goals are continuously evaluated against a dynamic knowledge base. This leads to the concept of transient queries, opposite to the persistent queries in a database. Typical applications of stream reasoning are: traffic monitoring, urban computing, patient monitoring, weather monitoring from satellite data, monitoring financial transactions (Valle et al., 2009) or stock market. Real time events analysis is conducted in domains like seismic incidents, flu outbreaks, or tsunami alert based on a wide range of sensor networks starting from the RFID technology to the Twitter dataflow (Savage, 2011). Decisions should be taken based on plausible events. Waiting to have complete confirmation of an event might be to risky action.

Streams of sensor data are often characterised by heterogeneity, noise and contradictory data. In this research we exploit nonmonotonic rule-based systems for handling inconsistent or incomplete information and also ontologies to deal with heterogeneity. Data is aggregated from distributed streams in real time and plausible rules fire when new data is available. This study investigates the advantages of lazy evaluation on data streams, as well.

2 INTEGRATING PLAUSIBLE RULES WITH ONTOLOGIES

2.1 Plausible Logic

Plausible logic is an improvement of defeasible logic (Rock, 2010; Billington and Rock, 2001). A clause $\vee a_1, a_2, ...a_n$ is the disjunction of positive or negative atoms a_i. If both an atom and its negation appear, the clause is a tautology. A *contingent* clause is a clause which is neither empty nor a tautology (Rock, 2010).

Letia I. and Groza A..
DESCRIPTION PLAUSIBLE LOGIC PROGRAMS FOR STREAM REASONING.
DOI: 10.5220/0003887405600566
In *Proceedings of the 4th International Conference on Agents and Artificial Intelligence* (IWSI-2012), pages 560-566
ISBN: 978-989-8425-95-9
Copyright © 2012 SCITEPRESS (Science and Technology Publications, Lda.)

Definition 1. *A plausible description of a situation is a tuple $PD = (Ax, R_p, R_d, \succ)$, where Ax is a set of contingent clauses, called axioms, characterising the aspects of the situation that are certain, R_p is a set of plausible rules, R_d is a set of defeater rules, and $\succ$ is a priority relation on $R_p \cup R_d$.*

A plausible theory is computed from a plausible description by deriving the set R_s of strict rules from the definite facts Ax. Thus, a plausible knowledge base consists of strict rules ($\rightarrow$), plausible rules ($\Rightarrow$), defeater (warning) rules ($\rightharpoondown$), and a priority relation on the rules ($\succ$). Strict rules are rules in the classical sense, that is whenever the premises are indisputable, then so is the conclusion. An atomic fact is represented by a strict rule with an empty antecedent. The plausible rule $a_i \Rightarrow c$ means that if all the antecedents a_i are proved and all the evidence against the consequent c has been defeated then c can be deduced. The plausible conclusion c can be defeated by contrary evidence.

The only use of defeaters is to prevent some conclusions, as in "if the buyer is a regular one and he has a short delay for paying, we might not ask for penalties". This rule does not provide sufficient evidence to support a "non penalty" conclusion, but it is strong enough to prevent the derivation of the penalty consequent. The priority relation $\succ$ allows the representation of preferences among non-strict rules.

Decisive plausible logic consists of a plausible theory and a proof function $P(\lambda f,)$, which given a proof algorithm λ and a formula f in conjunctive normal form, returns $+1$ if λf was proved, -1 if there is no proof for λf, or 0 when λf is undecidable due to looping. Plausible Logic has five proof algorithms $\{\mu, \alpha, \pi, \beta, \gamma\}$, one is monotonic and four are non-monotonic: μ monotonic, strict, like classical logic; $\alpha = \beta \wedge \pi$; π plausible, propagating ambiguity; β plausible, blocking ambiguity; and $\gamma = \pi \vee \beta$.

2.2 Translating from DL to Plausible Logic Programs

Facing the challenge to reason on huge amount of noise and heterogeneous data, the OWL fragment corresponding to Horn clauses, known as Description Logic Programs (Grosof et al., 2003), can be a suitable choice. This section exploits the work in (Gomez et al., 2010) in order to translate description logic based ontologies into plausible logic axioms.

Conjunctions and universal restrictions in the right hand side of inclusion axioms are converted into rule heads (L_h classes), whilst conjunction, disjunction and existential restriction appearing in the left-hand side are translated into rule bodies (L_b classes). Figure 1 presents the mapping function $\mathcal{T}$ from DL to strict rules in a plausible knowledge base, where A, C, and D are concepts such that $A, C \in L_b$, $D \in L_h$, A is an atomic concept, X, Y, Z variables, and P, Q roles.

$$
\begin{array}{lcl}
\mathcal{T}(C \sqsubseteq D) & = & T_b(C,X) \rightarrow T_h(D,X) \\
\mathcal{T}(\top \sqsubseteq \forall P.D) & = & P(X,Y) \rightarrow T_h(D,Y) \\
\mathcal{T}(\top \sqsubseteq \forall P^-.D) & = & P(X,Y) \rightarrow T_h(D,X) \\
\mathcal{T}(a : D) & = & T_h(D,a) \\
\mathcal{T}((a,b) : P) & = & P(a,b) \\
\mathcal{T}(P \sqsubseteq Q) & = & P(X,Y) \rightarrow Q(X,Y) \\
\mathcal{T}(P^+ \sqsubseteq P) & = & P(X,Y) \wedge P(Y,Z) \\
 & & \rightarrow P(X,Z) \\
 & \textit{where} & \\
T_h(A,X) & = & A(X) \\
T_h(C \sqcap D, X) & = & T_h(C,X) \wedge T_h(D,X) \\
T_h(\forall R.C) & = & R(X,Y) \rightarrow T_h(C,Y) \\
T_b(A,X) & = & A(X) \\
T_b(C \sqcap D, X) & = & T_b(C,X) \wedge T_b(D,X) \\
T_b(C \sqcup D, X) & = & T_b(C,X) \vee T_b(D,X) \\
T_h(\exists R.C) & = & R(X,Y) \rightarrow T_b(C,Y)
\end{array}
$$

Figure 1: Mapping from DL ontologies into strict rules.

3 DATA STREAM MANAGEMENT SYSTEM IN HASKELL

This section details the system architecture, as depicted in figure 2. The user is responsible to define the priorities and the plausible rules in order to handle contradictory data for the problem in hand.

3.1 The Haskell Platform

The advantages which Haskell brings in this landscape, lazy evaluation and implicit parallelism, are significant features when dealing with huge data streams which are parallel in nature. The parralel performing of reasoning tasks is of significant importance in order to provide answers in due time (Valle et al., 2009). The Haskell's polymorphism allows to write generic code to process streams, which is particularly useful due to the different exploitation of the same data stream. The absence of side effects, means that the order of expression evaluation is of no importance, which is extremely desirable in the context of data streams coming from different sources. One challenge when answering in real-time to many continuously queries is query optimisation. Allowing equational reasoning, can be exploited for automatic program and query optimisation. A premise specified as lazy is matched only when its variables are antici-

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

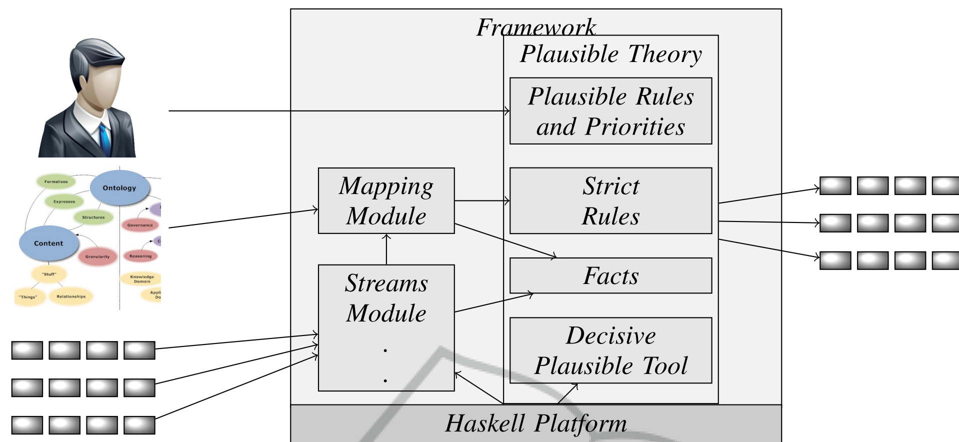

Figure 2: System architecture.

pating to participate in the answer to a pending conclusion or query.

The continuous semantics of data streams assumes that: i) *streams are volatile* - they are consumed on the fly and not stored forever; and ii) *continuous processing* - queries are registered and produce answers continuously (Barbieri et al., 2010). In our case, the rules are triggered continuously in order to produce streams of consequents. The stateless feature of pure Haskell facilitates the conceptual model of networks of stream reasoners as envisaged in (Stuckenschmidt et al., 2010), where data is processed on the fly, without being stored. The lazy evaluation in Haskell provides answers perpetually, when the queries are executed against infinite streams. One does not have to specify the timesteps when the query should be executed. By default, the tuples are consumed when they become available, and only in case they contribute to a query answer.

The computational efficiency is supported by the fact that a function is not forced to wait for a data to arrive - the possible computation are executed instead. Moreover one can use the about-to-come data by borrowing it from the future, as long as no function tries to change its value. The non strict semantics of Haskell, allows the functions to not produce errors in case these errors can be avoided. Consequently, some noise data can be avoided, without disturbing the computations.

There is no constraint on the nature of data fed by a stream. The functions can be applied on RDF streams as follows: A triple object is created by the *triple* function

```
data Triple    = triple !Node !Node !Node
triple         :: Subject -¿ Predicate -¿ Object -¿ Triple
```

Definition 2. *An RDF stream is an infinite list of tuples of the form $\langle subj, pred, obj \rangle$ annotated with their timestamps τ.*

$$type\ RDFStream = [((subj, pred, obj), \tau)]$$

Example 1. *An RDF stream of auction bids states the bidder agent, its action, and the bid value:* $[(:a_1, :sell, :30E), 14.32), (:a_2, :sell, :28E), 14.34), (:a_3, :buy, :26E), 14.35), (:a_1, :sell, :27E), 14.36)]$

3.2 Streams Module

Table 1 illustrates the operators provided by Haskell to manipulate infinite streams. Considering one wants to add the corresponding values from two financial data streams s_1 and s_2, expressed by two different currencies:

$$zipWith\ +\ s_1\ (map\ conversion\ s_2)$$

where the conversion function is applied on each element from s_2. For computing at each step the sum of a string of transactional data, the following expression can be used: $scan\ +\ 0\ [2,4,5,3,...]$, providing as output the infinite stream $[0,2,6,11,14,...]$.

The aggregation of two streams takes place according to an aggregation policy, depending on the time or the configuration of the new tuples. Here, the policy is a function provided as input argument for the high order function *zipWith policy stream stream*. Similarly, generating new stream is done based on a policy. The incoming streams can be dynamically split into two streams, based on a predicate p.

3.3 The Mapping Module

The ontologies are translated based on the conceptual instrumentation introduced in section 2.2. Two sources of knowledge are exploited to reason on data

Table 1: Stream operators in Haskell (*S* stands for the *Stream* datatype).

Type	Function	Signature
Basic	constructor	<:> ::a -> S a -> Sa
	extract the first element	head:: S a -> a
	extracts the sequence following the stream's head	tail:: S a -> S a
	takes a stream and returns all its finite prefixes	inits :: S a -> S ([a])
	takes a stream and returns all its suffixes	tails :: S a -> S (S a)
Transfor mation	applies a function over all elements	map :: (a -> b) -> S a -> S b
	interleaves 2 streams	inter :: Stream a -> Stream a -> S a
	yields a stream of successive reduced values	scan :: (a -> b -> a) -> a -> S b -> S a
	computes the transposition of a stream of streams	transp :: S (S a) -> S (S a)
Building streams	repeated applications of a function	iterate :: (a -> a) -> a -> S a
	constant streams	repeat :: a -> S a
	returns the infinite repetition of a set of values	cycle :: [a] -> S a
Extracting sublists	takes the first elements	take :: Int -¿ S a -> [a]
	drops the first elements	drop :: Int -> S a -> S a
	returns the longest prefix for which the predicate p holds	takeWhile :: (a -> Bool) -> S a -> [a]
	return the suffix remaining after takeWhile	dropWhile :: (a -> Bool) -> S a -> S a
	removes elements that do not satisfy p	filter :: a -> Bool) -> S a -> S a
Index	return the element of the stream at index n	!! :: S a -> Int -> a
	return the index of the first element equal to the query element	elemIndex :: Eq a => a -> S a -> Int
	return the index of the first element satisfying p	findIndex :: (a -> Bool) -> S a -> Int
Aggregation	return a list of corresponding pairs from 2 streams	zip :: S a -> S b -> S (a,b)
	combine two streams based on a given function	ZipWith :: (a -> b -> c) -> S a -> S b -> S c

$$
\begin{array}{l}
Sensor \sqsubseteq \forall measure.PhysicalQuality \\
Sensor \sqsubseteq \forall hasLatency.Time \\
Sensor \sqsubseteq \forall hasLocation.Location \\
Sensor \sqsubseteq \forall hasFrequency.Frequency \\
Sensor \sqsubseteq \forall hasAccuracy.MeasureUnit \\
WirelessSensor \sqsubseteq Sensor \\
RFIDSensor \sqsubseteq WirelessSensor \\
ActiveRFID \sqsubseteq RFIDSensor
\end{array}
$$

$$
\begin{array}{l}
Sensor(X), Measures(X,Y) \rightarrow PhysicalQuality(Y) \\
Sensor(X), HasLatency(X,Y) \rightarrow Time(Y) \\
Sensor(X), HasLocation(X,Y) \rightarrow Location(Y) \\
Sensor(X), HasFrequency(X,Y) \rightarrow Frequency(Y) \\
Sensor(X), HasAccuracy(X,Y) \rightarrow MeasureUnit(Y) \\
WirelessSensor(X) \rightarrow Sensor(X) \\
RFIDSensor(X) \rightarrow WirelessSensor(X) \\
ActiveRFID(X) \rightarrow WirelessSensor(X)
\end{array}
$$

Figure 3: Translating the sensor ontology.

collected by the sensors. On the one hand, one needs detailed information about sensors, measurements domain and units, or accuracy (see figure 3). On the other hand domain specific axioms are exploited when reasoning on a specific scenario.

The rapid development of the sensor technology rises the problem of continuously updating the sensor ontology. The system is able to handle this situation by treating the ontology as a stream of description logic axioms. When applying the high order function *map* on the transformation function $\mathcal{T}$, each axiom in description logic is converted to strict rules as soon as it appears:

$$map\ \mathcal{T}\ [A \sqsubseteq B, C \sqsubseteq \forall r.D, ...]$$

ouputs the infinite list:

$$[r_1 : A(X) \rightarrow B(X)), r_2 : C(X), r(X,Y) \rightarrow D(Y), ...]$$

The main advantage consists in the possibility to dynamically include new background knowledge in the system.

3.4 Efficiency

The system incorporates the Decisive Plausible Logic tool[1]. A Haskell glue module that exports functions requesting proofs (Rock, 2010) is used to make the connection with the other modules. The efficiency is mandatory when one needs to reason on huge data in real time. The efficiency of the proposed solution is based on the following vectors: i) The implementation of a family of defeasible logic is polynomial (Maher et al., 2001). Plausible logic being a particular case of defeasible reasoning belongs to this efficiency class. The possibility to select the current inference algorithm among $\{\mu, \alpha, \pi, \beta, \gamma\}$ can be exploited to adjust the reasoning task to the complexity of problem in hand. ii) DLP are subfragments of Horn logics and their complexity is polynomial, as reported in (Krötzsch et al., 2007).

[1] Available at http://www.ict.griffith.edu.au/arock/DPL/

$Milk \sqsubseteq Item$
$Item \sqsubseteq \forall HasPeak.Time$
$WholeMilk \sqsubseteq Milk$
$LowFatMilk \sqsubseteq Milk$
$fm_1 : WholeMilk$
$sm_1 : LowFatMilk$
$sm_1 : LowFatMilk.$

Figure 4: Domain Knowledge for the Milk Monitoring.

$r_1: \quad Milk(X) \rightarrow Item(X)$
$r_2: \quad Item(X), HasPeak(X,Y) \rightarrow Time(Y)$
$r_3: \quad WholeMilk(X) \rightarrow Milk(X)$
$r_4: \quad LowFatMilk(X) \rightarrow Milk(X)$
$f_1: \quad WholeMilk(fm1)$
$f_2: \quad LowFatMilk(sm1)$
$f_3: \quad LowFatMilk(sm2)$
$r_{10}: \quad Milk(X), Stock(X,Y), Less(Y,c1) \Rightarrow NormalSupply(X,c2)$
$r_{11}: \quad HasPeak(X,Y) \rightharpoonup NormalSupply(X,c2)$
$r_{12}: \quad Milk(X), Stock(X,Y), Less(Y,c1), hasPeak(X,Z), now(Z) \Rightarrow PeakSupply(X,c3)$
$r_{13}: \quad AlternativeItem(X,Z), Milk(X), Stock(Z,Y), Greater(Y,c4) \Rightarrow \neg PeakSupply(X,c3)$
$r_{14} \quad LastMeasurement(S,Y), HasLatency(S,Z), Greater(Y,Z) \Rightarrow BrokenSensor(S)$
$r_{15} \quad BrokenSensor(S), Measur(S,X) \rightharpoonup Stock(X,_)$
$r_{13} \succ r_{12}$

Figure 5: Plausible Knowledge Base.

4 RUNNING SCENARIO

The scenario regards supporting real-time supply chain decisions based on RFID streams. Consider the stock management of a retailer. RFID sensors are used to count the items entering on the shelves from two locations. The clients leave the supermarket from three payment points, corresponding to three output streams. Monitoring an item like *Milk* implies monitoring several subcategories like *WholeMilk* and *LowFatMilk*. The retailer sells a specific item fm_1 of whole milk, and two types of low fat milk sm_1 and sm_2. Some peak periods are associated to each commercialised item. This background knowledge is formalised in figure 4. The corresponding strict rules are depicted in the upper part of the figure 5. During peak periods for an item the usual supply action is blocked by the defeater r_{11}.

The plausible rule r_{10} says that if the milk stock Y is bellow the alert threshold $c1$, the *normalSupply* action should be executed. *NormalSupply* assures a stock value of $c2$. Instead, the *PeakSupply* action is derived by the rule r_{11}.

If there is an alternative item Z for the *Milk* product and the stock of the alternative is larger than the threshold $c4$, this implies not to supply the higher quantity $c2$ (the rule r_{12}). Depending on the priority relation between the rules r_{12} and r_{13}, the action is executed or not.

The sensor related information can be integrating when reasoning. If the sensor S seems not to function according to the specifications in the ontology, it is plausible to be broken (the rule r_{14}). A broken sensor defeats the stock information asserted in the knowledge base related to the measured item (the defeater r_{15}).

The merchandise flow is simulated by generating infinite input and output streams. Assuming that the function $randomItem :: [Item] -> Item$, based on the list of availalbe items returns a random item. The output stream for the payment point $out1$ would be:

$$out1 = (randomItem\ l) : out1$$

where l represents the available items in the simulation. Assume a stream of sold items and the time of measurement s_1 : $[(sm_1,1),(m_1,2),(fm_1,3),(m_2,4),(m_3,5),(sm_2,6),(m_4,7),....]$.

The *updateStock* function continously computes the current stocks based on the s_1 stream. Based on the fact f_1 and the rule r_3, one can conclude that $fm1$ is a milk item. Similarly, based on the facts f_2 and f_3, the rule r_4 cathegorises the instances sm_1 and sm_2 as milk items. The filter function is used to monitor each milk item, either low fat or not:

$$milkItems = filter\ milk\ (map\ first\ s1)$$

Here, the predicate *milk* returns true if the input is of type *Milk* according to the rules r_3 or r_4. The *map* function is used to select only the first element from the tuples $(item, time)$ from the stream $s1$. The stream *milkItems* collects all the items of type milk, and everytime an item occurs, the $updateStock :: Item -> Stream -> Int$ function is activated to compute the available stock for a specific cathegory. Thus, by combining ontological knowledge with plausible rules one can reason with generic products (*Milk*), even if the streams report data regarding instances of specific products (*WholeMilk* and *LowFatMilk*), minimising the number of business rules that should be added within the system.

5 DISCUSSION AND RELATED WORK

Stream integration is considered an ongoing challenge for the stream management systems (Valle

et al., 2009; Calbimonte et al., 2010; Le-Phuoc et al., 2010; Palopoli et al., 2003). There are several tools available to perform stream reasoning.

DyKnow (Fredrik Heintz and Doherty, 2009) introduces the knowledge processing language KPL to specify knowledge processing applications on streams. We exploit the Haskell stream operators to handle streams and list comprehension for querying this streams. The SPARQL algebra is extended in (Bolles et al., 2008) with time windows and pattern matching for stream processing. In our approach we exploit the existing list comprehension and pattern matching in Haskell, aiming at the same goal of RDF streams processing. Comparing to C-SPARQL, Haskell provides capabilities to aggregate streams before querying them Etalis tool performs reasoning tasks over streaming events with respect to background knowledge (Anicic et al., 2010). In our case the background knowledge is obtained from ontologies, translated as strict rules in order to reason over a unified space.

The research conducted here can be integrated into the larger context of Semantic Sensor Web, where challenges like abstraction level, data fusion, application development (Corcho and Garcia-Castro, 2010) are adressed by several research projects like Aspire[2] or Sensei[3]. By incapsulating domain knowledge as description logic programs, the level of abstraction can be adapted for the application in hand by importing a more refined ontologly into DLP.

Streams being approximate, omniscient rationality is not assumed when performing reasoning tasks on streams. Consequently, we argue that plausible reasoning for real time decision making is adequate. One particularity of our system consists of applying an efficient non-monotonic rule based system (Maher et al., 2001) when reasoning on gradually occurring stream data. The inference is based on several algorithms, which is in line with the proof layers defined in the Semantic Web cake. Moreover, all the Haskell language is available to extend or adapt the existing code. The efficiency of data driven computation in functional reactive programming is supported by the lazy evaluation mechanism which allows to use values before they can be known.

The strength of plausibility of the consequents is given by the superiority relation among rules. One idea of computing the degree of plausibility is to exploit specific plausible reasoning patterns like *epagoge*: "If A is true, then B is true, B is true. Therefore, A becomes more plausible", "If A is true, then B is true. A is false. Therefore, B becomes less plausible.", or "If A is true, then B becomes more plausible. B is true. Therefore, A becomes more plausible."

[2]http://www.fp7-aspire.eu/

[3]http://www.ict-sensei.org/

6 CONCLUSIONS

Our semantic based stream management system is characterised by: i) continuous situation awareness and capability to handle theoretically infinite data streams due to the lazy evaluation mechanism, ii) aggregating heterogeneous sensors based on the ontologies translated as strict rules, iii) handling noise and contradictory information inherently in the context of many sensors, due to the plausible reasoning mechanism. Ongoing work regards conducting experiments to test the efficiency and scalability of the proposed framework, based on the results reported in (Maher et al., 2001) and on the reduced complexity of description logic programs (Krötzsch et al., 2007).

ACKNOWLEDGEMENTS

We are grateful to the anonymous reviewers for their useful comments. The work has been cofunded by the Sectoral Operational Programme Human Resources Development 2007-2013 of the Romanian Ministry of Labour, Family and Social Protection through the Financial Agreement POSDRU/89/1.5/S/62557 and PN-II-Ideas-170.

REFERENCES

Anicic, D., Fodor, P., Rudolph, S., Stühmer, R., Stojanovic, N., and Studer, R. (2010). A rule-based language for complex event processing and reasoning. In Pascal Hitzler, T. L., editor, *Web Reasoning and Rule Systems - Fourth International Conference*, volume 6333 of *LNCS*, pages 42–57. Springer.

Barbieri, D., Braga, D., Ceri, S., Della Valle, E., and Grossniklaus, M. (2010). Incremental reasoning on streams and rich background knowledge. In Aroyo, L., Antoniou, G., Hyvnen, E., ten Teije, A., Stuckenschmidt, H., Cabral, L., and Tudorache, T., editors, *The Semantic Web: Research and Applications*, volume 6088 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin / Heidelberg.

Billington, D. and Rock, A. (2001). Propositional plausible logic: Introduction and implementation. *Studia Logica*, 67(2):243–269.

Bolles, A., Grawunder, M., and Jacobi, J. (2008). Streaming sparql extending sparql to process data streams. In *Proceedings of the 5th European semantic web conference on The semantic web: research and applications*, ESWC'08, pages 448–462, Berlin, Heidelberg. Springer-Verlag.

Calbimonte, J.-P., Corcho, Ó., and Gray, A. J. G. (2010). Enabling ontology-based access to streaming data sources. In Patel-Schneider, P. F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J. Z., Horrocks, I., and Glimm, B., editors, *International Semantic Web Conference (1)*, volume 6496 of *Lecture Notes in Computer Science*, pages 96–111. Springer.

Corcho, Ó. and Garcia-Castro, R. (2010). Five challenges for the semantic sensor web. *Semantic Web*, 1(1-2):121–125.

Fredrik Heintz, J. K. and Doherty, P. (2009). Stream reasoning in dyknow: A knowledge processing middleware system. In *In Stream Reasoning Workshop, Heraklion, Crete*.

Gomez, S. A., Chesnevar, C. I., and Simari, G. R. (2010). A defeasible logic programming approach to the integration of rules and ontologies. *Journal of Computer Science and Technology*, 10(2):74–80.

Grosof, B. N., Horrocks, I., Volz, R., and Decker, S. (2003). Description logic programs: combining logic programs with description logic. In *WWW*, pages 48–57.

Krötzsch, M., Rudolph, S., and Hitzler, P. (2007). Complexity boundaries for horn description logics. In *AAAI*, pages 452–457. AAAI Press.

Le-Phuoc, D., Parreira, J. X., Hausenblas, M., and Hauswirth, M. (2010). Unifying stream data and linked open data. Technical report, DERI.

Maher, M. J., Rock, A., Antoniou, G., Billington, D., and Miller, T. (2001). Efficient defeasible reasoning systems. *International Journal on Artificial Intelligence Tools*, 10(4):483–501.

Palopoli, L., Terracina, G., and Ursino, D. (2003). A plausibility description logic for handling information sources with heterogeneous data representation formats. *Annals of Mathematics and Artificial Intelligence*, 39:385–430.

Rock, A. (2010). Implementation of decisive plausible logic. Technical report, School of Information and Communication Technology, Griffith University.

Savage, N. (2011). Twitter as medium and message. *Commun. ACM*, 54:18–20.

Stuckenschmidt, H., Ceri, S., Valle, E. D., and van Harmelen, F. (2010). Towards expressive stream reasoning. In Aberer, K., Gal, A., Hauswirth, M., Sattler, K.-U., and Sheth, A. P., editors, *Semantic Challenges in Sensor Networks*, number 10042 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

Valle, E. D., Ceri, S., van Harmelen, F., and Fensel, D. (2009). It's a streaming world! reasoning upon rapidly changing information. *IEEE Intelligent Systems*, 24:83–89.

SCIENCE AND TECHNOLOGY PUBLICATIONS