Plausible Description Logic Programs for Stream Reasoning

Ioan Alfred Letia and Adrian Groza

Department of Computer Science
Technical University of Cluj-Napoca, Romania
Adrian.Groza@cs.utcluj.ro

ICAART, 6 February 2012, Vilamoura, Portugal

Outline

It's a Streaming World

- sensor networks[a]
- urban computing
- social networking
- financial markets

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[b]

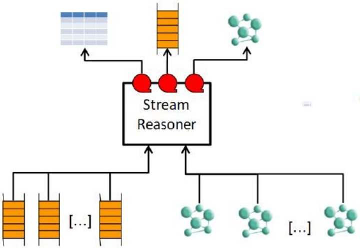
---

[a] [LPPHH10] D. Le-Phuoc, J. Parreira, M. Hausenblas, and M. Hauswirth. Unifying stream data and linked open data. Technical report, DERI, 2010.

[b] [VCvHF09] E. D. Valle, S. Ceri, F. van Harmelen, and D. Fensel. It's a streaming world! reasoning upon rapidly changing information. IEEE Intelligent Systems, 24:83â89, 2009.

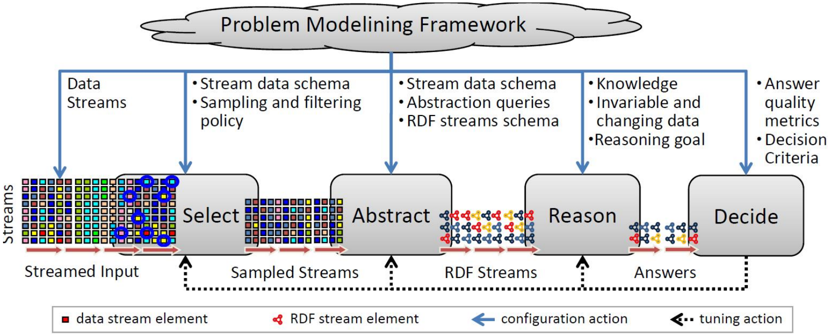
Stream Reasoning

- Real time logical reasoning on huge, possible infinite, noisy data streams, aiming to support the decision process of large numbers of concurrent querying agents.
- Continous semantics
  1. *streams are volatile* - they are consumed on the fly and not stored forever;
  2. *continuous processing* - queries are registered and produce answers continuously



Conceptual Architecture of Stream Reasoning

LARK perspective (The Large Knowledge Collider)



Outline

Plausible Logic

- Non-monotonic reasoning concerned with the problem of deducing conclusions from incomplete or uncertain information.
- The expressivity of Defeasible Logic is limited by its inability to represent or prove disjunctions.
- Extends Defeasible Logic by accomodating disjunction.

Plausible Theory

A reasoning situation is defined by a plausible description made of

- a set of indiputable facts, each represented by a formula.
- a set of plausible rules (example: $\{bird\} \Rightarrow flies$ which might have a few exceptions.
- a set of defeater rules ($\leadsto$) which can prevent a conclusion without supporting its negation. (if the buyer is a regular one and he has a short delay for paying, we might not ask for penalties $regular \leadsto \sim penalty$)
- a priority relation $\succ$ from all rules R to the plausible and defeater rules R_{pd}. $\succ$ must not be cyclic.

Formulas are proved at different levels of certainty.

Level of Proofs

In decreasing certainty they are: the definite level, the defeasible levels or and the supported level.

- The definite level is like classical monotonic proof in that modus ponens is used and so more information cannot defeat a previous proof.
- Proof at the defeasible level is non-monotonic, that is more information may defeat a previous proof.
- A more cautious defeasible level of proof can be defined by changing the level of proof required to eliminate counter-evidence from not δ-provable to not even supported.

Inference in Defeasible Logic

Notation

- $P = (P_1, ..., P_n)$ is a formal proof (derivation)
- q is a literal, F the set of facts
- $A(r)$ the antecedent of the rule r
- $R[q]$ the set of rules with consequent q
- $R_s[q]$ the set of strict rules with consequent q
- $R_{sd}[q]$ the set of strict and defeasible rules with consequent q
- $r \succ s$ means that a rule r beats rule s

The inference conditions come in pairs: a proof $-\Delta f$ proves that $+\Delta f$ can not be proven.

Strict inference

$+\Delta$:
If $P(i+1) = +\Delta q$ then either
$q \in F$
$\exists r \in R_s[q] \; \forall a \in A(r) : +\Delta a \in P(1..i)$

$-\Delta$:
If $P(i+1) = -\Delta q$ then either
$q \notin F$
$\forall r \in R_s[q] \; \exists a \in A(r) : -\Delta a \in P(1..i$

Inference in Defeasible Logic

Defeasible inference

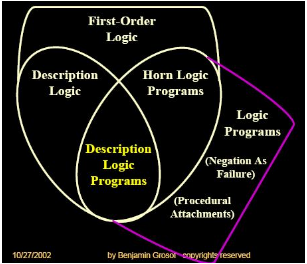
$+\partial$:

If $P(i+1) = +\partial q$ then either
$+\Delta q \in P(1..i)$ or
$\exists r \in R_{sd}[q] \forall a \in A(r) : +\partial a \in P(1..i)$ and
$-\Delta \neg q \in P(1..i)$ and
$\forall s \in R[\neg q]$ either
$\exists a \in A(s) : -\partial a \in P(1..i)$ or
$\exists t \in R_{sd}[q]$ such that $\forall a \in A(t) : +\partial a \in P(1..i)$ and $t \succ s$

$-\partial$:

If $P(i+1) = -\partial q$ then
$-\Delta q \in P(1..i)$ and either
$\forall r \in R_{sd}[q] \exists a \in A(r) : -\partial a \in P(1..i)$ or
$+\Delta \neg q \in P(1..i)$ or
$\exists s \in R[\neg q]$ either
$\forall a \in A(s) : +\partial a \in P(1..i)$ and
$\forall t \in R_{sd}[q]\ \exists a \in A(t) : -\partial a \in P(1..i)$ or $t \not\succ s$

$DL \setminus LP \sqcup LP \setminus DL$



Examples of DL beyond DLP

1. State a subclass of a complex class expression which is a disjunction $(Human \sqcap Adult) \sqsubseteq (Man \sqcup Woman)$
2. State a subclass of a complex class expression which is an existential $Radio \sqsubseteq \exists hasPart.Tuner$

Examples of LP beyond DLP

A rule involving multiple variables.

$$Man(X) \wedge Woman(Y) \rightarrow PotentialLoveInterestBetween(X, Y)$$

DL's not used to represent "more than one free variable at a time"

Expressing OWL into Horn logic

1. A triple of the form (a, P, b) can be expressed as a fact P(a, b)
2. Instance declaration of the form $type(a, C)$, stating that a is an instance of class C, can be expressed as C(a)
3. The fact that C is a subclass of D ($C \sqsubseteq D$) is expressed as $C(X) \rightarrow D(X)$
4. Domain and range restrictions can be expressed in Horn logic: the following rule states that C is the domain of the property P: $P(X, Y) \rightarrow C(X)$
5. $sameClassAs(C, D)$ can be expressed by the pair of rules $C(X) \rightarrow D(X)$, $D(X) \rightarrow C(X)$
6. Transitivity of a property P is expressed as $P(X, Y), P(Y, Z) \rightarrow P(X, Z)$

Expressing RDFS/OWL into Horn logic

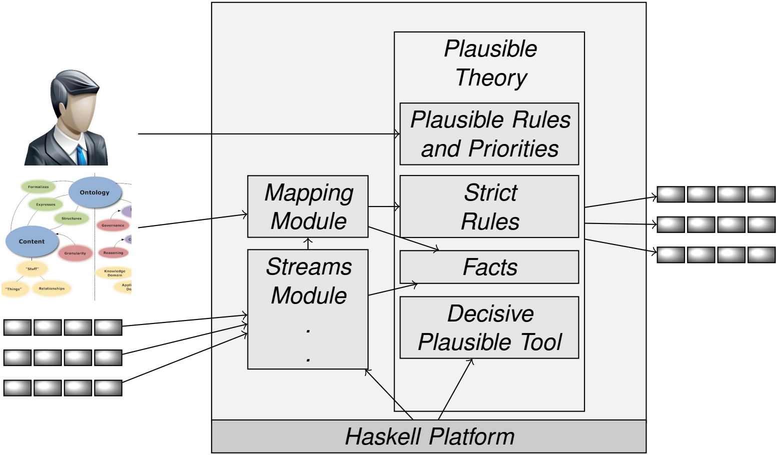
1. The intersection of classes C_1 and C_2 is a subclass of D: $C1(X), C2(X) \rightarrow D(X)$
2. C is a subclass of the intersection of D_1 and D_2 as: $C(X) \rightarrow D1(X)$, $C(X) \rightarrow D2(X)$
3. the union of C_1 and C_2 is a subclass of D: $C_1(X) \rightarrow D(X)$, $C_2(X) \rightarrow D(X)$
4. $C \sqsubseteq \forall P.D$: $C(X), P(X,Y) \rightarrow D(Y)$
5. $\exists P.D \sqsubseteq C$:$P(X,Y), D(Y) \rightarrow C(X)$
6. C is a subclass of the union of D_1 and D_2 would require a disjunction in the head of the corresponding rule, not available in Horn Logic, but availalbe in Plausible Logic.

Outline

Haskell Advantages

1. purity: no side effects
   - the order of expression evaluation is of no importance: extremely desirable in the context of streams coming from different sources
   - implicit parallelism: significant when dealing with huge data which are parralel in nature.
2. polymorphism: same code processing eterogeneous streams.
3. equational reasoning: query optimisation for answering in real time to many continous queries.

System Architecture



Streams Module

Type	Function	Signature
Basic	constructor	<:> ::a -> S a -> Sa
	extract the first element	head:: S a -> a
	extract the sequence following the stream's head	tail:: S a -> S a
	take a stream and returns all its finite prefixes	inits :: S a -> S ([a])
	take a stream and returns all its suffixes	tails :: S a -> S (S a)
Transfor mation	apply a function over all elements	map :: (a -> b) -> S a -> *S* b
	interleave 2 streams	inter :: Stream a -> Stream a -> S a
	yield 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
	return the infinite repetition of a set of values	cycle :: [a] -> S a
Extracting sublists	take the first elements	take :: Int -> S a -> [a]
	drop the first elements	drop :: Int -> S a -> S a
	return 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

Stream Processing Examples

An RDF stream of auction bids states the bidder agent, its action, and the bid value:

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

$[(a_1, sell, 30), 14.32), (a_2, sell, 28), 14.34), (a_3, buy, 26), 14.35)]$
Adding two financial streams:

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

Computing at each step the sum of a stream of transactional data:

$$scan\ +\ 0\ [2, 4, 5, 3, ...]$$

providing as output the infinite stream $[0, 2, 6, 11, 14, ...]$.
Policy-based aggregation: *zipWith policy stream stream*

Mapping Module

$Sensor \sqsubseteq \forall measure.PhysicalQuality$

$Sensor \sqsubseteq \forall hasLatency.Time$

$Sensor \sqsubseteq \forall hasLocation.Location$

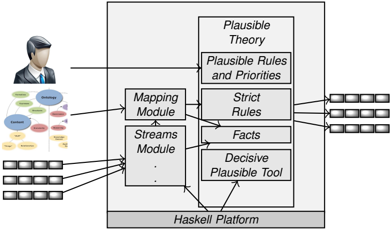
$Sensor \sqsubseteq \forall hasFrequency.Frequency$

$Sensor \sqsubseteq \forall hasAccuracy.MeasureUnit$

$WirelessSensor \sqsubseteq Sensor$

$RFIDSensor \sqsubseteq WirelessSensor$

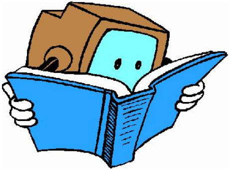
$ActiveRFID \sqsubseteq RFIDSensor$



$$Sensor(X), Measures(X, Y) \rightarrow PhysicalQuality(Y)$$
$$Sensor(X), HasLatency(X, Y) \rightarrow Time(Y)$$
$$Sensor(X), HasLocation(X, Y) \rightarrow Location(Y)$$
$$Sensor(X), HasFrequency(X, Y) \rightarrow Frequency(Y)$$
$$Sensor(X), HasAccuracy(X, Y) \rightarrow MeasureUnit(Y)$$
$$WirelessSensor(X) \rightarrow Sensor(X)$$
$$RFIDSensor(X) \rightarrow WirelessSensor(X)$$
$$ActiveRFID(X) \rightarrow WirelessSensor(X)$$

Dynamic Knowledge

Dynamic domains: the rapid development of the sensor technology rises the problem of continuously updating the sensor ontology.



The ontology is treated as a stream of description logic axioms:

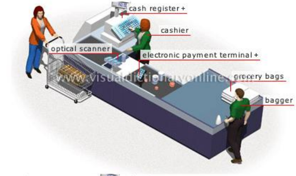
$$map\ \mathcal{T}\ [A \sqsubseteq B, C \sqsubseteq \forall r.D, ...]$$

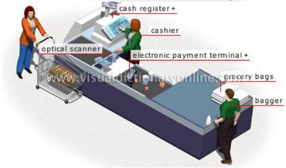
ouputs:

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

Outline

Real-time Stock Management





Plausible Knowledge Base

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

$r_1 : \quad Milk(X) \rightarrow Item(X)$
$r_2 : \quad Item(X), HasPeak(X, Y) \rightarrow Time(Y)$
$r_3 : \quad WholeMilk(X) \rightarrow Milk(X)$
$r_4 : \quad LowFatMilk(X) \rightarrow Milk(X)$
$f_1 : \quad WholeMilk(fm1)$
$f_2 : \quad LowFatMilk(sm1)$
$f_3 : \quad LowFatMilk(sm2)$

$r_{10} : \quad Milk(X), Stock(X, Y), Less(Y, c1) \Rightarrow NormalSupply(X, c2)$
$r_{11} : \quad HasPeak(X, Y) \leadsto NormalSupply(X, c2)$
$r_{12} : \quad Milk(X), Stock(X, Y), Less(Y, c1), hasPeak(X, Z), now(Z) \Rightarrow PeakSupply(X, c3)$
$r_{13} : \quad AlternativeItem(X, Z), Milk(X), Stock(Z, Y), Greater(Y, c4) \Rightarrow \neg PeakSupply(X, c3)$
$r_{14} \quad LastMeasurement(S, Y), HasLatency(S, Z), Greater(Y, Z) \Rightarrow BrokenSensor(S)$
$r_{15} \quad BrokenSensor(S), Measures(S, X) \leadsto Stock(X, \_)$
$r_{13} \succ r_{12}$

Continous Queries

Simulating infinite streams: $s1 = (randomItem\ itemsList) : s1$

$$s_1 : [(lm, 1), (a, 2), (wm, 3), (b, 4), (c, 5), (lm, 6), (b, 7), ...]$$

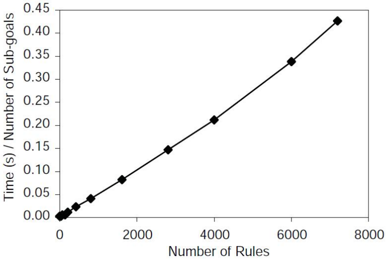
$$s_2 : [(a, 1), (lm, 2), (lm, 3), (noItem, 4), (d, 5), (lm, 6), (a, 7), ..].$$

Monitoring milk items (either whole or low fat)

$MI\ =\ filter\ (\backslash x\ =\ prove\ \Delta\ (milk\ x))\ (map\ first\ (merge\ s_1\ s_2))$

Outline

Decisive Plausible Logic Tool[2]



The proof in each case used all of the rules and one priority for every four rules

Defeasible Logic - handles hundreds of thousands of rules[1].

Plausible Logic - disjunction introduces exponential complexity

- In practice the number of disjuncts is small

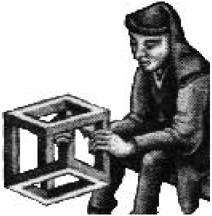
DLP is polynomial

---

[1] Results reported by A. Rock and D. Billington, An Implementation of Propositional Plausible Logic, 23rd Australasian Computer Science Conference, 2000, pp 204-210.

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

Handling Complexity



- Selecting the inference algorithm can be exploited to adjust the reasoning task to the complexity of problem in hand
- The level of abstraction can be adapted for the current scenario by importing a more refined ontology into PDLP

Computing the Degree of Plausibility

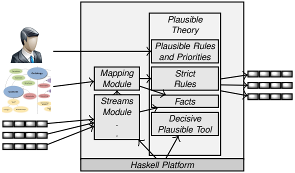
The strength of plausibility of the consequents is given by the superiority relation among rules.

Exploiting specific plausible reasoning patterns:

”If A is true, then B is true, B is true. Therefore, A becomes more plausible” (*epagoge*)

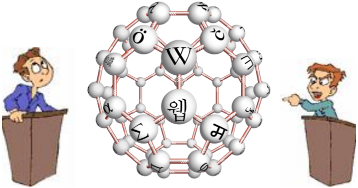
”If A is true, then B is true. A is false. Therefore, B becomes less plausible.”,

”If A is true, then B becomes more plausible. B is true. Therefore, A becomes more plausible.”



Supporting Decisions Under Contradictory Information

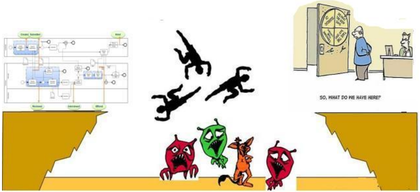
Argumentative Semantics of Plausible Logic



Rebuttal Argument
Undercutting Argument
Exploit the connection between plausible reasoning and argumentation theory.

Role of Ontologies

Gap between high level knowledge for management decisions and process models or low level streams.



Conclusion

Our semantic based stream management system is characterised by:

- aggregating heterogeneous sensors based on the ontologies translated as strict rules
- handling noise and contradictory information inherently in the context of many sensors, due to the plausible reasoning mechanism.

Thank you!

Danh Le-Phuoc, Josiane Xavier Parreira, Michael Hausenblas, and Manfred Hauswirth.
Unifying stream data and linked open data.
Technical report, DERI, 2010.

Emanuele Della Valle, Stefano Ceri, Frank van Harmelen, and Dieter Fensel.
It's a streaming world! reasoning upon rapidly changing information.
IEEE Intelligent Systems, 24:83–89, 2009.