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

1. Stream Reasoning
2. Integrating Plausible Rules with Ontologies
   - Plausible Logic
   - Translating from DL to PLP
3. DSMS in Haskell
   - Haskell Platform
   - System Architecture
4. Running Scenario
5. Ongoing Work
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\(^b\)

---


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.

- **Continuous 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

1. Stream Reasoning
2. Integrating Plausible Rules with Ontologies
   - Plausible Logic
   - Translating from DL to PLP
3. DSMS in Haskell
   - Haskell Platform
   - System Architecture
4. Running Scenario
5. Ongoing Work

ICAART, 6 February 2012, Vilamoura, Portugal
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 accommodating disjunction.
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 (\sim\sim) 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 \textit{regular} \sim\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.
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 \(\delta\)-provable to not even supported.
Plausible Logic

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
\(\neg \Delta \neg q \in P(1..i)\) 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 \(t \not\succ s\)
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
   \(Radio \sqsubseteq \exists \text{hasPart}.Tuner\)

Examples of LP beyond DLP

A rule involving multiple variables.
\[\text{Man}(X) \land \text{Woman}(Y) \rightarrow PotentialLoveInterestBetween(X, Y)\]

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 \(\text{type}(a, C)\), 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. \(\text{sameClassAs}(C, D)\) 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 \(P(X, Y), P(Y, Z) \rightarrow P(X, Z)\)
Translating from DL to PLP

Expressing RDFS/OWL into Horn logic

1. The intersection of classes $C_1$ and $C_2$ is a subclass of $D$: $C_1(X), C_2(X) \rightarrow D(X)$

2. $C$ is a subclass of the intersection of $D_1$ and $D_2$ as: $C(X) \rightarrow D_1(X), C(X) \rightarrow D_2(X)$

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 available in Plausible Logic.
Outline

1. Stream Reasoning
2. Integrating Plausible Rules with Ontologies
   - Plausible Logic
   - Translating from DL to PLP
3. DSMS in Haskell
   - Haskell Platform
   - System Architecture
4. Running Scenario
5. Ongoing Work
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 parallel in nature.

2. **polymorphism**: same code processing heterogeneous streams.

3. **equational reasoning**: query optimisation for answering in real time to many continuous queries.
System Architecture
## System Architecture

### Streams Module

<table>
<thead>
<tr>
<th>Type</th>
<th>Function</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>constructor</td>
<td><code>&lt;:: ::a -&gt; S a -&gt; Sa</code></td>
</tr>
<tr>
<td></td>
<td>extract the first element</td>
<td>head:: S a -&gt; a</td>
</tr>
<tr>
<td></td>
<td>extract the sequence following the stream's head</td>
<td>tail:: S a -&gt; S a</td>
</tr>
<tr>
<td></td>
<td>take a stream and returns all its finite prefixes</td>
<td>inits :: S a -&gt; S ([a])</td>
</tr>
<tr>
<td></td>
<td>take a stream and returns all its suffixes</td>
<td>tails :: S a -&gt; S (S a)</td>
</tr>
<tr>
<td>Transformation</td>
<td>apply a function over all elements</td>
<td>map :: (a -&gt; b) -&gt; S a -&gt; S b</td>
</tr>
<tr>
<td></td>
<td>interleave 2 streams</td>
<td>inter :: Stream a -&gt; Stream a -&gt; S a</td>
</tr>
<tr>
<td></td>
<td>yield a stream of successive reduced values</td>
<td>scan :: (a -&gt; b -&gt; a) -&gt; a -&gt; S b -&gt; S a</td>
</tr>
<tr>
<td></td>
<td>computes the transposition of a stream of streams</td>
<td>transp :: S (S a) -&gt; S (S a)</td>
</tr>
<tr>
<td>Building streams</td>
<td>repeated applications of a function</td>
<td>iterate :: (a -&gt; a) -&gt; a -&gt; S a</td>
</tr>
<tr>
<td></td>
<td>constant streams</td>
<td>repeat :: a -&gt; S a</td>
</tr>
<tr>
<td></td>
<td>return the infinite repetition of a set of values</td>
<td>cycle :: [a] -&gt; S a</td>
</tr>
<tr>
<td>Extracting sublists</td>
<td>take the first elements</td>
<td>take :: Int -&gt; S a -&gt; [a]</td>
</tr>
<tr>
<td></td>
<td>drop the first elements</td>
<td>drop :: Int -&gt; S a -&gt; S a</td>
</tr>
<tr>
<td></td>
<td>return the longest prefix for which the predicate p holds</td>
<td>takeWhile :: (a -&gt; Bool) -&gt; S a -&gt; [a]</td>
</tr>
<tr>
<td></td>
<td>return the suffix remaining after takeWhile</td>
<td>dropWhile :: (a -&gt; Bool) -&gt; S a -&gt; S a</td>
</tr>
<tr>
<td></td>
<td>removes elements that do not satisfy p</td>
<td>filter :: a -&gt; Bool) -&gt; S a -&gt; S a</td>
</tr>
<tr>
<td>Index</td>
<td>return the element of the stream at index n</td>
<td><code>!! :: S a -&gt; Int -&gt; a</code></td>
</tr>
<tr>
<td></td>
<td>return the index of the first element equal to the query element</td>
<td>elemIndex :: Eq a =&gt; a -&gt; S a -&gt; Int</td>
</tr>
<tr>
<td></td>
<td>return the index of the first element satisfying p</td>
<td>findIndex :: (a -&gt; Bool) =&gt; S a -&gt; Int</td>
</tr>
<tr>
<td>Aggregation</td>
<td>return a list of corresponding pairs from 2 streams</td>
<td>zip :: S a -&gt; S b -&gt; S (a,b)</td>
</tr>
<tr>
<td></td>
<td>combine two streams based on a given function</td>
<td>ZipWith :: (a -&gt; b -&gt; c) -&gt; S a -&gt; S b -&gt; S c</td>
</tr>
</tbody>
</table>
An RDF stream of auction bids states the bidder agent, its action, and the bid value:

\[
\text{type RDFStream} = [((\text{subj, pred, obj}), \tau)]
\]

\[
[({a_1, sell, 30}, 14.32), ({a_2, sell, 28}, 14.34), ({a_3, buy, 26}, 14.35)]
\]

Adding two financial streams:

\[
\text{zipWith } + \ s_1 (\text{map conversion } s_2)
\]

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

\[
\text{scan } + \ 0 [2, 4, 5, 3, \ldots]
\]

providing as output the infinite stream \([0, 2, 6, 11, 14, \ldots]\).

Policy-based aggregation: \text{zipWith policy stream stream}
Sensor ⊑ ∀measure. PhysicalQuality
Sensor ⊑ ∀hasLatency. Time
Sensor ⊑ ∀hasLocation. Location
Sensor ⊑ ∀hasFrequency. Frequency
Sensor ⊑ ∀hasAccuracy. MeasureUnit
WirelessSensor ⊑ Sensor
RFIDSensor ⊑ WirelessSensor
ActiveRFID ⊑ RFIDSensor

Sensor(X), Measures(X, Y) → PhysicalQuality(Y)
Sensor(X), HasLatency(X, Y) → Time(Y)
Sensor(X), HasLocation(X, Y) → Location(Y)
Sensor(X), HasFrequency(X, Y) → Frequency(Y)
Sensor(X), HasAccuracy(X, Y) → MeasureUnit(Y)
WirelessSensor(X) → Sensor(X)
RFIDSensor(X) → WirelessSensor(X)
ActiveRFID(X) → 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:

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

outputs:

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

1. Stream Reasoning
2. Integrating Plausible Rules with Ontologies
   - Plausible Logic
   - Translating from DL to PLP
3. DSMS in Haskell
   - Haskell Platform
   - System Architecture
4. Running Scenario
5. Ongoing Work

ICAART, 6 February 2012, Vilamoura, Portugal
Real-time Stock Management
Plausible Knowledge Base

\( \text{Milk} \sqsubseteq \text{Item} \)
\( \text{Item} \sqsubseteq \forall \text{HasPeak}.\text{Time} \)
\( \text{WholeMilk} \sqsubseteq \text{Milk} \)
\( \text{LowFatMilk} \sqsubseteq \text{Milk} \)
\( \text{fm}_1 : \text{WholeMilk} \)
\( \text{sm}_1 : \text{LowFatMilk} \)
\( \text{fm}_2 : \text{LowFatMilk} \)
\( \text{sm}_2 : \text{LowFatMilk} \)

\( r_1 : \text{Milk}(X) \rightarrow \text{Item}(X) \)
\( r_2 : \text{Item}(X), \text{HasPeak}(X, Y) \rightarrow \text{Time}(Y) \)
\( r_3 : \text{WholeMilk}(X) \rightarrow \text{Milk}(X) \)
\( r_4 : \text{LowFatMilk}(X) \rightarrow \text{Milk}(X) \)
\( f_1 : \text{WholeMilk}(\text{fm}_1) \)
\( f_2 : \text{LowFatMilk}(\text{sm}_1) \)
\( f_3 : \text{LowFatMilk}(\text{sm}_2) \)

\( r_{10} : \text{Milk}(X), \text{Stock}(X, Y), \text{Less}(Y, c_1) \Rightarrow \text{NormalSupply}(X, c_2) \)
\( r_{11} : \text{HasPeak}(X, Y) \rightsquigarrow \text{NormalSupply}(X, c_2) \)
\( r_{12} : \text{Milk}(X), \text{Stock}(X, Y), \text{Less}(Y, c_1), \text{hasPeak}(X, Z), \text{now}(Z) \Rightarrow \text{PeakSupply}(X, c_3) \)
\( r_{13} : \text{AlternativeItem}(X, Z), \text{Milk}(X), \text{Stock}(Z, Y), \text{Greater}(Y, c_4) \Rightarrow \neg \text{PeakSupply}(X, c_3) \)
\( r_{14} : \text{LastMeasurement}(S, Y), \text{HasLatency}(S, Z), \text{Greater}(Y, Z) \Rightarrow \text{BrokenSensor}(S) \)
\( r_{15} : \text{BrokenSensor}(S), \text{Measures}(S, X) \rightsquigarrow \text{Stock}(X, _) \)
\( r_{13} \triangleright r_{12} \)
Simulating infinite streams: $s_1 = (\text{randomItem itemsList}) : s_1$

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

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

Monitoring milk items (either whole or low fat)

$$MI = \text{filter } (\backslash x = \text{prove } \Delta (\text{milk } x)) (\text{map first } (\text{merge } s_1, s_2))$$
Outline

1. Stream Reasoning
2. Integrating Plausible Rules with Ontologies
   - Plausible Logic
   - Translating from DL to PLP
3. DSMS in Haskell
   - Haskell Platform
   - System Architecture
4. Running Scenario
5. Ongoing Work

ICAART, 6 February 2012, Vilamoura, Portugal
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!