Stream Reasoning
For Linked Data
J-P Calbimonte, D. Dell'Aglio, E. Della Valle, M.I. Ali and A. Mileo
http://streamreasoning.org/events/sr4ld2015

Stream Reasoning with ASP
Alessandra Mileo
alessandra.mileo@insight-centre.org
http://www.insight-centre.org/users/alessandra-mileo
http://www.linkedin.com/in/alessandramileo
Share, Remix, Reuse — Legally

- This work is licensed under the Creative Commons Attribution 3.0 Unported License.

- **Your are free:**
  - **to Share** — to copy, distribute and transmit the work
  - **to Remix** — to adapt the work

- **Under the following conditions**
  - **Attribution** — You must attribute the work by inserting a credits slide stating
    - These slides are partially based on “Streaming Reasoning for Linked Data 2015” by J-P Calbimonte, D. Dell'Aglio, E. Della Valle, M. I. Ali and A. Mileo

- To view a copy of this license, visit
  - [http://creativecommons.org/licenses/by/3.0/](http://creativecommons.org/licenses/by/3.0/)
Agenda

- ASP semantics:
  - Basic notion and incremental computation
  - Examples and hands-on (online)

- StreamRule: A combined approach
  - Basics
  - StreamRule Examples

- Stream Reasoning with Probabilistic ASP
  - Probabilistic ASP basics
  - Streaming Probabilistic ASP: approximation and optimization
  - Examples and Hands-on

- Ongoing work and open issues
Answer Set Programming

- Declarative problem solving approach
  - “what is the problem?” vs “how to solve the problem?”
  - Problem is modeled using a logic program (set of logic rules)
  - Correct interpretations (or Answer Sets) correspond to problem solutions

- ASP combines:
  - Rich yet simple modeling language
    - Negation, disjunction, integrity constraints, weak constraints, aggregates, ...
  - High-performance solving capabilities
    - Based on guess/check/optimize strategy
    - Relies on CWA

- ASP has its roots in:
  - Deductive databases
  - Logic programming (with negation)
  - KR and NMR
  - Constraint solving (mostly SAT)
(Disjunctive) Rule or Cardinality Constraint

\[ l\{a_1,...,a_n\}u :- b_1,...,b_k, \text{ not } b_{k+1},...,\text{ not } b_m \]

```
head                      body
```

“if all \( b_1,...,b_k \) are true and none of the \( b_{k+1},...,b_m \) is true, then at least \( l \) and at most \( u \) among \( a_1,...,a_n \) are true”

Consistency constraint (rule with empty head):

\[ :- b_1,...,b_k, \text{ not } b_{k+1},...,\text{ not } b_m \]

“it is not possible that all \( b_1,...,b_k \) are true and none of the \( b_{k+1},...,b_m \) is true”

Fact (rule with empty body):

\[ a_1 \cdot \quad \text{“}a_1 \text{ is true.”} \]

Other constructs allow for aggregates and preferences (soft constraints)
Example: graph colouring (1/2)

- **Problem encoding P:**

  % generating plausible colorings
  1{col_V(V,C) : col(C)}1 :- vertex(V).
  % defining constraints
  :- col_V(V,C), col_V(V1,C), edge(V,V1).

- **Input instance I:**

  vertex(a). vertex(b). vertex(c). vertex(d). vertex(e).
  edge(a,b). edge(a,c). edge(a,d).
  edge(b,e). edge(c,d). edge(d,e).

- **Answer Set(s) of \( P \cup I \) (one solution of the problem)**

  col_V(a,red). col_V(b,blue). col_V(c,blue). col_V(d,green). col_V(e,red).
Incremental Answer Set Programming idea

- **Problem**
  - many real-world applications, like planning or model checking, have solutions size depending on increasing parameter \( t \)

- **ASP Limitation**
  - Each problem instance needs to be (re)considered entirely every time \( t \) increases
  - This results in highly inefficient grounding procedure

- **Goal**
  - Avoiding redundancy by gradually processing the extensions to a problem bounded by \( t \) to the same problem bounded by \( t+1 \), only by doing grounding and solving incrementally, thus avoiding re-processing the entire extended problem.

- **Proposal**
  - **Incremental ASP**
Incremental to Reactive ASP (1/2)

- **Parametrized problem description**
  - **Base B**: static knowledge (independent of parameter $t$)
  - **Cumulative $P[t]$**: knowledge cumulating with increasing $t$
  - **Volatile $Q[t]$**: knowledge that is specific for each value of $t$

- **Example: Elevator controller**
  - At each step, the elevator can move one floor up or down
  - The elevator controller is accepting requests to go to a certain floor when it’s not at that floor already
  - If the elevator reaches a floor where a request exists, it serves the request...
  - ... until all requests are served (i.e. there are no more requests)

- Note that a request is coming from outside the system, and their occurrences cannot be foreseen within an incremental program

- **Reactivity**: deal with external requests $E[t]$ to apply incremental grounding and solving
Incremental to Reactive ASP (2/2)

Encoding as Incremental ASP Program:

- When \( t=1 \), the answer set \( AS \) of the program is
  \[
  AS = B U \{\text{atFloor}(2,1), \text{goal}(1)\}
  \]

- Assume a request to serve floor 3 occurs at time 1:
  \[
  E[t] = \{\text{request}(3,1)\}
  \]

- When \( t=1 \) we have no answer sets (elevator cannot serve the request in 1 move), but but when \( t=2 \) we get
  \[
  AS = B U E[1] U \{\text{requested}(3,1), \text{atFloor}(2,1), \text{atFloor}(3,2), \text{goal}(2)\}
  \]
Stream Reasoning with ASP

- **Problem**
  - time-decaying data poses a major challenge to ASP given that fixed encodings must tolerate emerging as well as expiring data.

- **Solution**
  - **Time-Decaying Logic Programs** extends reactive ASP
    - Base $B$: static knowledge (independent of parameter $t$)
    - Cumulative $P[t]$: knowledge cumulating with increasing $t$
    - External $E[t]$: external requests coming at time $t$
    - Volatile $Q_n[t]$ consider changes for each time $t$ considering a time span of $n$ steps

- **Example: Access Control**
Example: Access Control (1/3)

- Users attempting to log into a service
- Access attempts can be denied or granted
- A user account is temporarily closed after three access denials in a row (note this is simulating a tuple-based window of 3)
- Lifespan of access data is limited to three incremental steps, so that a closed account is re-opened after some time has elapsed
- Note that there is an offset of at most 2 steps from the current step for the time of the access attempt (the program receives the external information two timestamps after it occurred)
Example: Access Control (2/3)

- Encoding

```prolog
1  #const window=3. #const offset=2. #const denial=3. #iinit 1-offset.

3  #base.
4  user(bob;alice;claude). % some users
5  signal(denied;granted). % some signals
6  { account(U,closed) : user(U) }.
7  account(U,open) :- user(U), not account(U,closed).

9  #cumulative t.
10 #external access(U,S,t+offset) : user(U) : signal(S).
11  denied(U,1, t) :- access(U,denied,t+offset).
12  denied(U,N+1, t) :- access(U,denied,t+offset),
13     denied(U,N,t-1), N < denial.
14  denied(U,denial,t) :- denied(U,denial,t-1).
15     :- denied(U,denial,t), not account(U,closed).

17  #volatile t.
18     :- account(U,closed), not denied(U,denial,t).
```
Example: Access Control (3/3)

Stream Segment: user access in 3 steps

1  #step 1. #volatile : 3.
2  access(alice, granted, 1).
3  #step 2. #volatile : 3.
4  access(alice, denied, 3).
5  access(bob, denied, 3).
6  #step 3. #volatile : 3.
7  access(alice, denied, 2).
8  access(claude, granted, 5).
9  #step 4. #volatile : 3.
10 access(bob, denied, 2). access(bob, denied, 4).
11 access(claude, denied, 2).
12 #step 5. #volatile : 3.
13 access(alice, denied, 4).
14 access(claude, denied, 3). access(claude, denied, 4).
15 #step 6. #volatile : 3.
16 access(alice, denied, 6).
17 #step 7. #volatile : 3.
18 access(alice, denied, 8).
19 #step 8. #volatile : 3.
20 access(alice, denied, 7).

Legenda
[n] granted
_n_ denied
_account closed

Data

<table>
<thead>
<tr>
<th>i</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
</table>
Data Streams

Stream Reasoning

Semantic Complex Event Processing

Stream Query Processing

Applications

Scalability

Expressivity

14
Stream Reasoning with ASP: What’s new?

- Normal Answer Set Programs are written to work with static knowledge and rules for non-monotonic datalog

\[ h \leftarrow a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n \]

- Streaming ASP allows to externally input data into logic programs and reason upon them to produce dynamic solutions (answer sets) to dynamic problems

- Implements time-decaying, incremental, logic inference
The StreamRule idea

• 2-tier approach: not all dynamic data streams are relevant for complex reasoning
• Enrich the ability of complex reasoning over data streams
• Keep the solution scalable
• Leverage existing engines from both stream processing and non-monotonic reasoning research areas
The StreamRule idea

- Fully-fledged system
  - from data streams to complex problem solving
- RSP engine as pre-processor
  - to select, filter, aggregate, integrate streaming data
- Streaming Answer Set Programming as stream reasoning layer for complex problem solving
  - recursion, defaults, constraint checking, solution enumerations, abduction, planning, etc.

in other words...
The StreamRule idea

StreamRule is coupling:

- the linked data stream query processing power of RSP engines
- the expressivity and reasoning capabilities of Answer Set Programming with the CLINGO4 stream reasoning solver
- ... in a 2-tier approach so that the size of the input is reduced as the reasoning task becomes more computationally intensive.

in other words...
Limitations

- The more expressive the inference task, the longer it takes to perform reasoning
- Bottleneck when results are returned not as fast as the next input arrives
StreamRule

RDF Files (e.g. maps)

Web of Data

LSD Wrappers

Query Processing

Filtered Stream

Stream Rule

Clingo

Logic Program

Rule-based Expressive Reasoning

Application

Sensor Streams

Scalability requires adaptation!

Query

C-SPARQL CQELS

Facts

Controller

Rule-based Expressive Reasoning

Web of Data

LSD Wrappers

Query Processing

Filtered Stream

Stream Rule

Clingo

Logic Program

Rule-based Expressive Reasoning

Application

Sensor Streams

Scalability requires adaptation!
StreamRule: Potentials

- Complex reasoning over dynamic streams and their temporal dependencies, makes StreamRule suitable for:
  - Dynamic optimal planning/routing
  - Spatial reasoning, geofencing, access control, tracking
  - Inconsistency checking or constraint-based programming (e.g. configuration, diagnosis)
  - ...

- BUT we need to investigate further (among others):
  - Window size vs program complexity
  - Information flow between the components, more flexible coupling, e.g. to adapt window size
  - Parallel, distributed computation (e.g. via STORM/SPARK framework, orchestrated Logic Programs,...)
StreamRule Running Examples
**Geofencing example**

- **Scenario:**
  - People wear RFID tags and move around in a building equipped with RFID readers producing streams of position information.
  - Within the building, we have defined “geo-fences”, i.e., virtual perimeter for a real-world area (restricted access).
  - Particular areas are marked “off-limits”, and an area is at risk when it is close to an off-limit area.

- **Problem’s solutions:**
  - detect users who have traversed outside the geo-fence or are close to off-limits areas, and
  - detect inconsistencies in movement patterns of people.
PREFIX lv: <http://deri.org/floorplan/>
SELECT ?p1 ?p2
FROM NAMED <http://deri.org/floorplan/> WHERE
{
GRAPH <http://deri.org/floorplan/>
{?loc1 lv:connected ?loc2}
STREAM <http://deri.org/streams/rfid> [NOW]
{?p1 lv:detectedAt ?loc1}
STREAM <http://deri.org/streams/rfid> [RANGE 3s]
{?p2 lv:detectedAt ?loc2}
}
Geofencing example

```plaintext
#const window=3. #const offset=2.
#base.
% people IDs, areas IDs and connected areas are also part of the base program.
offbounds(area17).
risk(Y) :- offbounds(X), connected(X,Y), area(X), area(Y).

#cumulative t.
% generating plausible positions
0{position(P,A,t) : area(A)}1 :- person(P).

% detecting times when when risks and violations occur
breach(P,A,t) :- position(P,A,t), offbounds(A).

% identify inconsistent movements:
noisy_data(P,A2,t) :- position(P,A1,t-1),
    position(P,A2,t), not connected(A1,A2), neq(A1,A2).

#external detectedat(P,A,t+offset).
#volatile t.
% eliminate positions for which no detection exists (reduces the instance size).
:- detectedat(P,A,t), not position(P,A,t), person(P), area(A).
:- not detectedat(P,A,t), position(P,A,t), person(P), area(A).
```
Adaptation Heuristics: future work

- More than an engineering problem
  How to model interactions between RSP and ASP components, including different semantics, input split, window-size tuning,...

- Design and runtime features
  - E.g. operational semantics (design) and throughput (runtime)

- Streaming rate and window size: where’s the tradeoff?

- Reasoning Complexity: how far can we go? How can we parametrize the complexity to estimate the execution time
Core concepts

- **Unit of time** ([U](#))
  - How often collected input events are sent for processing by the RSP engine

- **Reasoning complexity** ([C](#))
  - Computational complexity

- **Streaming size** ([S](#))
  - Input elements *(to the reasoning component)* per unit of time

- **(Tuple-based) Window size** ([W](#))
  - Input elements processed *(by the reasoning component)* per computation
Notation

- **$T(N)$**
  Time needed *(by the reasoning component)* to process $N$ input elements

- **$T_\omega(S,W)$**
  Time needed *(by the reasoning component)* to process $S$ events using windows of size $W$

  $$T_\omega = \left\lceil \frac{S}{W} \right\rceil \times T(W)$$

- **$S_u$**
  Number of elements that can be processed *(by the reasoning component)* in one unit of time

  $$S_u = N \text{ s.t. } T(N) = U$$

- **$S_l$**
  Maximum number of elements that can be processed *(by the reasoning component)* within one unit of time using a proper windows size
Given a fixed streaming size $S$ with fixed complexity $C$ and unit of time $U$, find a window size $W$ such that the time required to process $S$ events using windows of size $W$ is less than or equal to one unit of time

$$T_\omega(S,W) \leq U$$
Experimental setup

- **Dataset:**
  - Simulated randomly generated events of the type
    \(\text{event}(\text{type}, \text{name}, \text{value}, \text{latitude}, \text{longitude})\)
  - *E.g. event*(weather, strong-wind, 2014-11-26T13:00:00, 38.011736, 12.186724)

- **Reasoning tasks:**
  - Ranking event criticality
  - Contextualizing events based on user status
  - Default rule to detect changes in event criticality

- **Run:**
  - Streaming size up to 30000
  - Reasoner triggered 20 times for each \(S\)
Empirical Results

\[ T_\omega(20000, 2000) = \frac{20000}{2000} \times T(2000) = 10 \times 72 \text{ ms} = 720 \text{ ms} \]

\[ T_\omega(20000, 2000) = 4 \times 216 \text{ ms} = 864 \text{ ms} \]

\[ T(5000) = 216 \text{ ms} \]

\[ T(20000) = 1232 \text{ ms} \]
Open issues and ongoing work

- **Limitations**
  - Independence assumption
  - Experiments with a fixed logic program (i.e. fixed complexity)

- **How to relax the independence assumption?**
  - Input dependency graph
  - Event duplication

- **Is it not just about streaming size and window size:**
  - What’s the correlation between complexity and window-size?
  - Are there other dimensions that makes the difference?
Dealing with Uncertainty and learning relational structures
IoT data are messy: deal with uncertainty

- Expressive inference
  - non-monotonicity, noisy, partial and inconsistent data
- “ease of” declarative logic-based reasoning to model a problem/domain. Still we need to manage uncertainty and non-monotonicity
- Probabilistic rules for uncertain knowledge and learning by example
  - represent, use, infer and learn probabilistic knowledge (PrASP)

Can we (learn the) answer to questions about uncertain knowledge using qualitative (declarative) inference in dynamic environments?
What is Streaming PrASP

A framework that uses:

1. PrASP as an uncertainty reasoning server to reason over Streaming Web Data

2. Continuous Query Processing over Linked Data Streams for data filtering

What is PrASP then?
PrASP is...

... an experimental Statistical Relational Learning (SRL) reasoner based on *Answer Set Programming* (ASP)

PrASP can...

... represent, use, infer and learn probabilistic knowledge
How does it work?

- Language (Syntax)
- Semantics
- Inference
- Learning
How does it work? Language

- **ASP or FOL syntax**
  - `s(X) <- r(X) & not ![X]: (p(X,Y) & q(X,Y))`.  
  - `p(X) :- q(X), not r(X)`

- **Just add probabilities as weights:**
  - Single probability weights: `[0.7] p(X) :- q(X)`.  
  - Conditional probabilities of formulas: `[p|c] f`.  
  - Rules with no probability have probability `1` by default.

- **Variable domains need to be finite**
How does it work? Semantics

- Defined as a probability distribution over the set of possible worlds of the Spanning Program $P'$ of $P$

- Spanning Program of a PrASP program is obtained by
  - Removing all probabilities
  - Transforming each weighted formula $[w] f.$ into $f \mid not f.$
  - Transforming each weighted formula $[0] f.$ into $not f.$

- A discrete probability function $\mu$ over the set of possible worlds is defined using
  - as a set of constraints which represent a system $S$ of linear equalities
  - System $S$ ensures $\mu$ is consistent w.r.t. the given formula weights
How does it work? Inference

- Inference is a model counting task
  - Each possible world (or answer set) has a weight (=probability)
  - The probability of a query formula $\varphi$ is the sum of the probabilities of the possible worlds where $\varphi$ is true

- Sampling step
  - High cost for very large sets of possible worlds
    - In solving the system of inequalities
    - In counting weighted answer sets
  - We need a uniformly distributed sample without computing all answer sets (hard in ASP due to the solving heuristics)
  - XOR constraints for near-uniform sampling:
    - Works with any ASP solver, can be obtained with parallel calls to the ASP solver
PrASP Inference Core

PrASP 0.6 inference core

- PrASP program (given knowledge base)
- Pre-processing (translation of FOL syntax, ...)
  - Simplification of rules with independent events
  - Spanning program (disjunctions from weighted formulas)
  - Filtering of formulas with no or very little influence on queries
- Query probabilities
  - Probability distribution over possible worlds (maximum entropy solution of equations)
- System of linear equations (or inequalities)
- Possible worlds (sampled answer sets)
- PrASP learning core
  - PrASP learning core
    - Native solver for linear systems
      - CVC4 (SMT solver; optional)
    - ASP grounder/solver (default: Clingo 3)
- Examples
- Queries
- Hypotheses

F2LP (optional)
Example (PrASP standalone)

- **Dice game (propositional version)**
  1. `face(1..6).`
  2. `[[::]] result(F) :- face(F).`
  3. `1{result(F):face(F)}1.`
  4. `win :- result(6).`
  5. `[0.8|win] :- happy.`
  6. `:- happy, not win.`

- **Spanning program transforms lines 2 and 5**
  2. `result(1) :- {not result(1)}0, true.`
  2. `...`
  2. `result(6) :- {not result(6)}0, true.`
  5. `happy :- {not happy}0, true.`
Example (PrASP standalone) contd.

- **Answer sets and weights**
  - \{result(1)\} [0.167]
  - \{result(2)\} [0.167]
  - \{result(3)\} [0.167]
  - \{result(4)\} [0.167]
  - \{result(5)\} [0.167]
  - \{result(6), win\} [0.033]
  - \{result(6), win, happy\} [0.133]

- **Queries**
  - \[?\] happy. [0.133]
  - \[?|result(6)\] happy. [0.8] = Pr(happy \& result(6))/Pr(result(6)) = 0.133/(0.033+0.133)
How does it work? Learning

- **Learning task:**
  - Hypothesis $H$ and (optional) background knowledge $B$ provided by an expert
  - Example data (fully observable) provided as a set of formulas $E=\{e_1, e_2, \ldots\}$
  - The objective is to discover weights $w$ of $H$ such that the likelihood of $E$ is maximized
  - Maximization target is $Pr( E | H_w U B)$

- Any maximization target can be used (not only $E$)
- Learning algorithm based on Barzilai and Borwein method (a form of gradient descent)
Why Streaming PrASP

- Challenges in Stream Reasoning for Linked Data
  - Expressivity: non-monotonicity, noisy, partial and inconsistent data
  - Probabilistic rules for uncertain knowledge and learning by example

- “ease of” declarative logic-based reasoning to model a problem/domain. Still we need to manage uncertainty and non-monotonicity

- Growing interests in Linked Data Streams for Big Data (IoT, WoT, W3C involvement)
Streaming new beliefs are added incrementally to a loaded PrASP program

Streaming new learning examples are added to the set of learning examples $E$

Assert/Retract, time decay and sliding windows supported

Windows prefixes realized by a caching mechanism (no reactive ASP used) for faster processing

Preprocessing based on RDF query processing over streams (SPARQL 1.1 + streaming operators)
Streaming PrASP framework
PrASP Running Examples
Streaming new triples to PrASP

- CQELS queries contains a pattern language to feed matching patterns to PrASP as
  - New beliefs \( f \) valid for \( n \) timesteps \( \rightarrow B_n[\text{weight}] f \)
  - New learning examples \( e \) valid for \( n \) timesteps \( \rightarrow E_n e \)

- CQELS query example

```html
PREFIX lv: <http://deri.org/floorplan/>
SELECT ?person1 ?loc1
PRASP E5 atPos(?person1,?loc1,TIMESTAMP).
[OMIT http://deri.org dblp/persons/]
FROM NAMED <http://deri.org/floorplan/> WHERE {
  GRAPH <http://deri.org/floorplan/>
  {?loc1 lv:connected ?loc2}
  STREAM <http://deri.org/streams/rfid> [NOW] {?person1 lv:detectedAt ?loc1}
```
Streaming PrASP Example

- Position estimation for a moving target

- Streaming Input:
  - Uncertain positions (from sensor data)
  - Sensed variation of speed w.r.t. a default speed

- PrASP program:
  - ASP encoding of the localization problem (generate & test)
  - Constraints on invalid locations and speed coherence
  - Uncertain background knowledge

- PrASP output:
  - Updated probabilities of hypothesis expressed as formulas
  - Hypothesis are, e.g., represented by a person being in a location at a particular time
Streaming PrASP Example encoding

1. `time(1..5). row(1..3). col(1..3).
2. location(loc(X,Y)) :- row(X), col(Y).
3. `1{atPos(L,T) : location(L) : not invalid(L)}1 :- time(T).` Generate
4. invalid(L) :- `1{wall(L), locked(L)}, location(L).` Test
5. :- atPos(L,T), not speed_coherent(L,T), time(T).` Speed Coherence
6. speed_coherent(L,1) :- atPos(L,1), location(L).
7. speed_coherent(L,T) :- speed(X,T-1), atPos(L1,T-1),
   distanceT(L,L1,X,T).
8. `distanceT(loc(X,Y),loc(U,V),#abs(X-U) + #abs(Y-V),T) :-
   atPos(loc(X,Y),T), atPos(loc(U,V),T-1).`
9. `[0.8] atPos(loc(1,1),1).` Probabilistic Background Knowledge
10. sensedspeed(1,1).
11. speed(S,T) :- sensedspeed(S,T).
12. speed(X,T) :- speed(X,T-1), not n_speed(X,T), time(T).
13. n_speed(X,T) :- sensedspeed(Z,T), speed(X,T-1), Z!=X.` Speed Inertia
Learning streaming PrASP example

- Default speed
  
sensedspeed(1,1).

- Queries:
  
  ![queries](image)

- Variation of speed (new beliefs)

<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>B1</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>0.7</td>
<td>0.5</td>
</tr>
<tr>
<td>B1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Results

  ![results](image)

- Updated Probabilities

  ![updated_probabilities](image)
PrASP 0.6 Implementation

- Supports Gringo and F2LP syntax

- Needs
  - Gringo and Clasp (or Clingo)
  - SMT solver CVC4
    - Used as a fall-back approach to constraint solving in addition to the built-in equation solver
  - F2LP if needed
    - Used to transform First-Order formulas into ASP

- Implemented using
  - Scala 2.11
  - JVM executing Scala bytecode, exception are native libraries for solving the system of linear equations
Streaming PrASP relates to…

- Like StreamRule
  - Uses RSP for pre-processing RDF Triples
  - Uses ASP for non-monotonic Reasoning

- Unlike StreamRule
  - Deals with Probabilistic Inference
  - Learns from new examples/new beliefs
  - Implements windows by caching and not using streaming ASP
Directions for Investigation

- Alternatives to the system of linear equations for finding the probability distribution:
  - Choice constraints (and helper literal), not as straightforward

- Further exploration and empirical evaluation
  - Optimization needed e.g. for XOR

- More ambitious goals
  - Structure Learning
  - Relation between Streaming ASP and PrASP
  - More experiments on Web Stream Reasoning with PrASP: scalability over real-world data streams
Open Challenges and next steps

- Streaming ASP: Adaptivity and cross-layer integration
  - Heuristics
  - Input dependency
  - Expressivity vs scalability tradeoff + complexity

- Distributed Stream Reasoning (Spark?)

- Probabilistic Streaming ASP:
  - Structured Learning
  - Experiments and Scalability
Stream Reasoning
For Linked Data
J-P Calbimonte, D. Dell'Aglio, E. Della Valle, M.I. Ali and A. Mileo
http://streamreasoning.org/events/sr4ld2015

Stream Reasoning with ASP
Alessandra Mileo
alessandra.mileo@insight-centre.org
http://www.insight-centre.org/users/alessandra-mileo
http://www.linkedin.com/in/alessandramileo