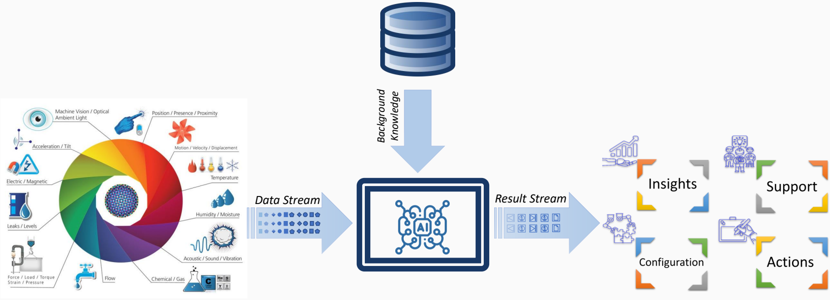
I-DLV-sr: a Stream Reasoning System based on I-DLV

Francesco Calimeri Marco Manna **Elena Mastria** Maria Concetta Morelli
Simona Perri Jessica Zangari

Department of Mathematics and Computer Science, University of Calabria, Italy

36th Italian Conference on Computational Logic

Motivations and Contributions



Stream Reasoning (SR)

Continuous application of inference techniques on highly dynamic data streams

- IoT, Smart Cities, Emergency Management...
  - Sources (*devices*, *sensors*, etc) produce high volume of data at each moment
  - Goals: *insight*, *knowledge*, *support* to decision-making process etc.
- A Stream Reasoner performs complex deduction tasks
  - Use some Background Knowledge of the domain
  - Use **window-based** processing to deal with infinite data streams

Motivations and Contributions

Answer Set Programming (ASP):

- **Declarative** paradigm for **K**nowledge **R**epresentation and **R**easoning
- Successfully employed in both academy and industry
  - Robust and efficient implementations
- A particularly attractive basis for SR

Motivations and Contributions

Answer Set Programming (ASP):

- **Declarative** paradigm for **K**nowledge **R**epresentation and **R**easoning
- Successfully employed in both academy and industry
  - Robust and efficient implementations
- A particularly attractive basis for SR

Goal: obtain a novel and reliable ASP-based stream reasoner, that:

- Inherits the highly declarative nature and ease of use from ASP;
- Can be easily extended with new constructs relevant for practical SR scenarios;
- Efficiently scales over real-world application domains.

Motivation and Contribution

I-DLV-sr: an ASP-based stream reasoner

- Support normal stratified ASP programs
- Provide a set of constructs for reason over streams

Motivation and Contribution

I-DLV-sr: an ASP-based stream reasoner

- Support normal stratified ASP programs
- Provide a set of constructs for reason over streams

⟨⟨How many cars have passed in the last 20 seconds?⟩⟩

```
carPassing(C,N) :- car(C) count N in [20].
tot(T) :- #sum {N,C: carPassing(C,N)} = T.
```

I-DLV-sr: an ASP-based stream reasoner

- Support normal stratified ASP programs
- Provide a set of constructs for reason over streams

⟨⟨How many cars have passed in the last 20 seconds?⟩⟩

```
carPassing(C,N) :- car(C) count N in [20].
tot(T) :- #sum {N,C: carPassing(C,N)} = T.
```

Note: *built-in atoms* and *aggregate literals* are supported (ASP-Core-2)

Input Language

Streaming Atoms

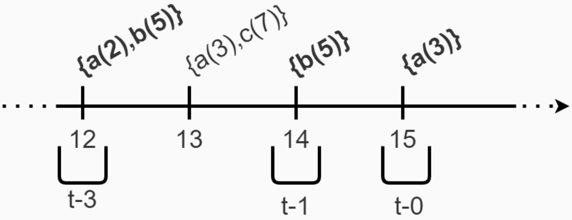
- *a* **at least** *c* **in** $\{d_1, ..., d_m\}$
- *a* **always in** $\{d_1, ..., d_m\}$
- *a* **count** *t* **in** $\{d_1, ..., d_m\}$

where *a* is an atom, *c* $\in \mathbb{N}^+$, *t* is either $\in \mathbb{N}^+$ or a variable, and $\{d_1, \ldots, d_m\} \subset \mathbb{N}$

Admitted Shortcuts

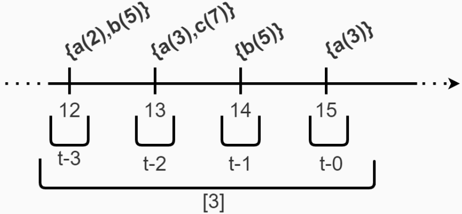
- *a* **at least** *1* **in** $\{d_1, ..., d_m\} \rightarrow$ **a in** $\{d_1, ..., d_m\}$
- **not** *a* **at least** *c* **in** $\{d_1, ..., d_m\} \rightarrow$ **a at most c′ in** $\{d_1, ..., d_m\}$
  where $c' = c - 1$
- *a* **at least** *1* **in** $\{0\} \rightarrow$ **a** (a standard ASP atom!)
- $\{d_1, ..., d_m\} \rightarrow$ **[d$_m$]**, if it is the set of natural numbers in the interval $[0, d_m]$

Example 1: $t = 15$



`b(5)` **at least** *2* **in** $\{0, 1, 3\}$.
holds at the time point 15

Example 2: $t = 15$



`b(5) always in [3].`
does not hold at the time point 15

Remark:
`b(5) always in [3].` is equivalent to `b(5) always in {0,1,2,3}.`

Modelling—Underground Traffic Monitoring

Goal: build a monitoring system for the underground trains in the city of Milan

For each underground station

- Identify irregularity in train arrivals
- Send alerts in case of recurrent irregularities
  - **mild alert**: from 2 to 5 irregularities
  - **severe alert**: more than 5 irregularities

Traffic regularity: passengers expect to see a train stopping every 3–6 minutes

```
r1: irregular :- train_pass, train_pass at least 1 in {1,2}.
r2: irregular  :- not train_pass in [6].
r3: #temp num_anomalies(X)  :- irregular count X in [30].
r4: mild_alert   :- num_anomalies(X), X>2, X<=5.
r5: severe_alert  :- num_anomalies(X), X>5.
```

I-DLV-SR is based on a continuous cooperation between two components:

- A Java application built on top of APACHE FLINK (FLINK) for processing data stream
- $\mathscr{I}$ncremental $\mathscr{I}$-DLV ($\mathscr{I}^2$-DLV) for performing complex reasoning tasks

Apache Flink

A **distributed** stream processor for efficiently managing data streams

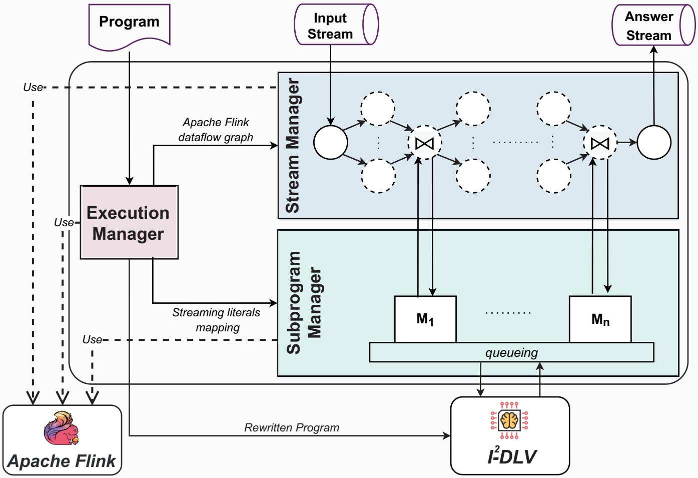
- both batch and realtime stream data processing
- high throughput and low latency

$\mathscr{I}^2$– DLV

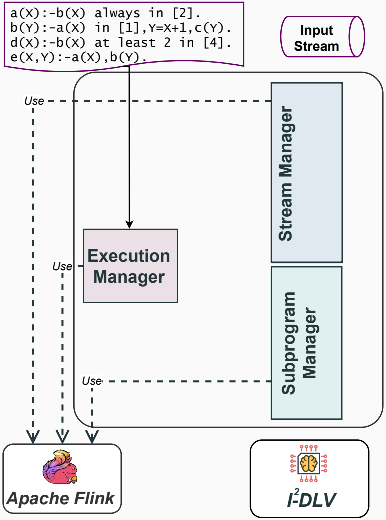
An ASP grounder and a full-fledged deductive database system

- incremental ASP evaluation via **overgrounding** techniques
- service-oriented behavior
    - given a fixed input program, it remains "listening" for input facts

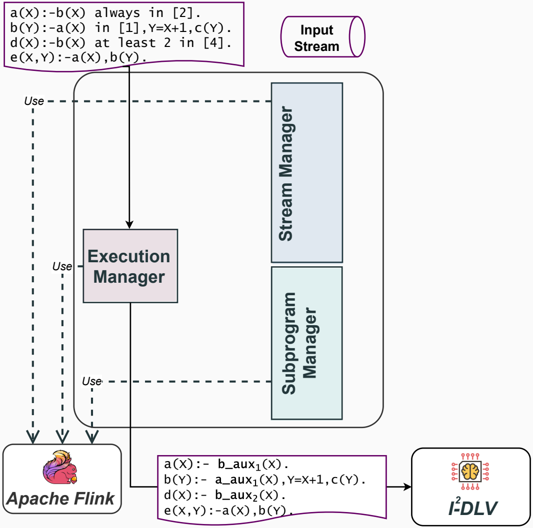
System Architecture



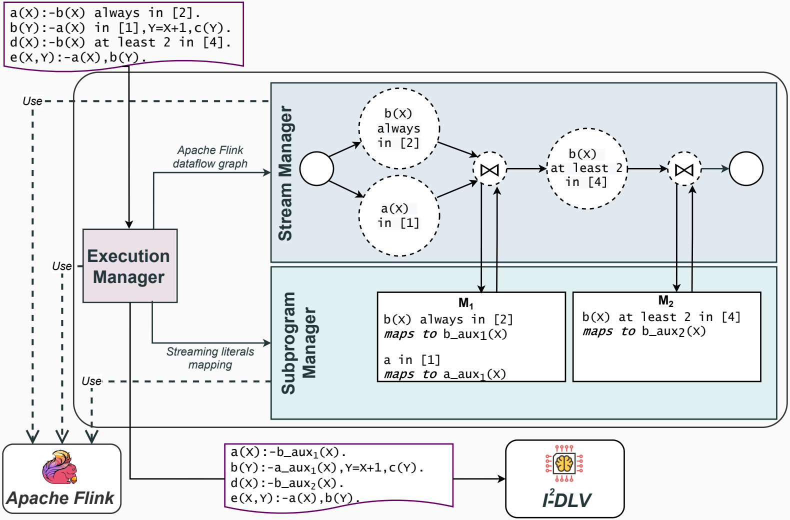
A Practical Example



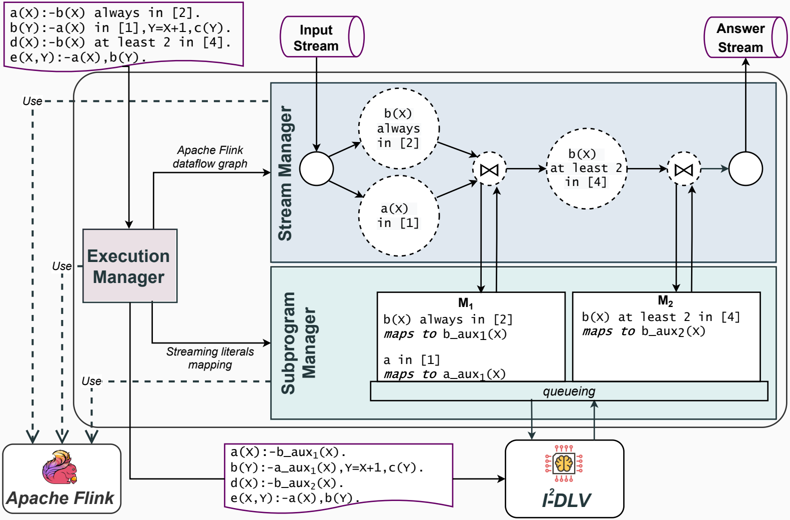
A Practical Example



A Practical Example



A Practical Example



Description (1/2)

Tested Systems:

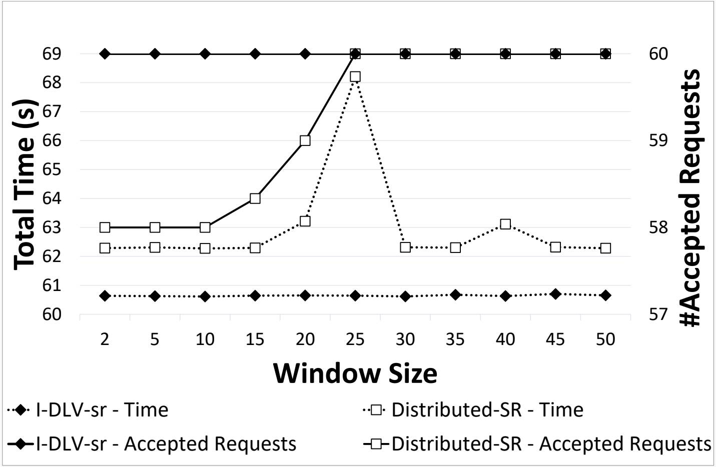
- I-DLV-sr
- Distributed-SR:
  - the most recent LARS-based implementation
  - supports a large set of features
  - relies on a distributed architecture

Benchmarks:

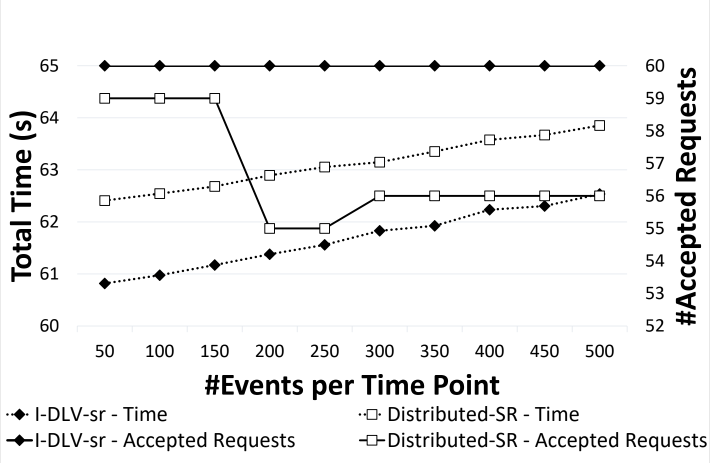
- Content Caching
- Heavy Join

Performance:

- Total Time (s)
- Number of Accepted Requests



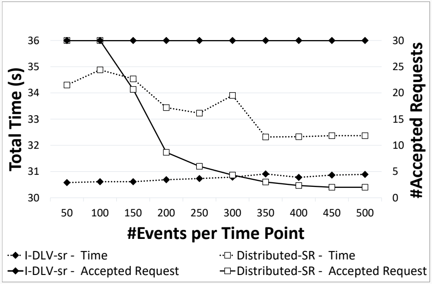
Window size: from 2s to 50s — **Total number of requests**: 60 (1 per second) — **Events per time point**: 1



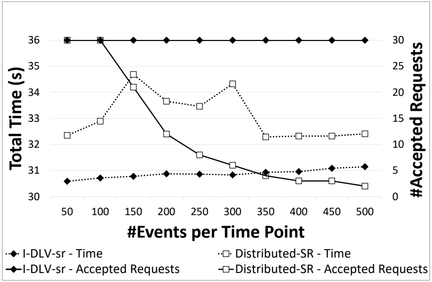
Window size: 5s — **Total number of requests**: 60 (1 per second) — **Events per time point**: from 50 to 500

Program

```
a(X,Y):-b(X,Z) in [w],c(Z,Y) in [w]
```



Window size(`[w]`): 2s



Window size(`[w]`): 20s

Total number of requests: 30 (1 per second) — **Events per time point**: from 50 to 500

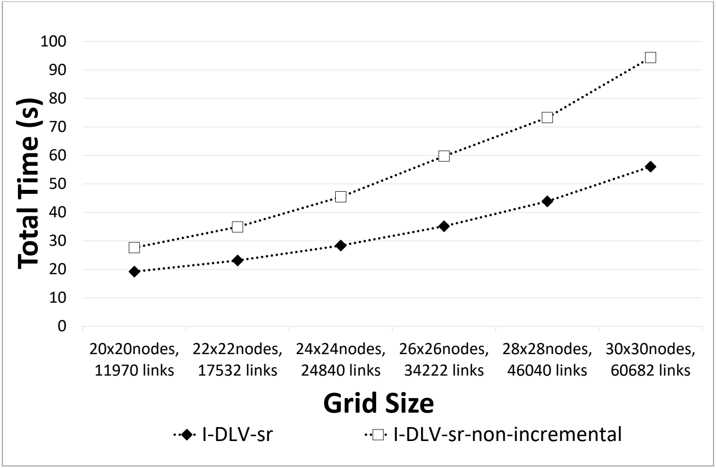
Tested Systems:

- I-DLV-SR: relies on the incremental $\mathscr{I}^2$-DLV system
- I-DLV-SR-NON-INCREMENTAL: relies on the non-incremental $\mathscr{I}$-DLV engine

Benchmark: Photo-voltaic System

Performance: Total Time (s)

I-DLV-sr vs I-DLV-sr-non-incremental



Period of incoming requests: 0.1s — **Total number of requests**: 60 (1 each 0.1 second) — **Events per time point**: vary with the grid size (eg. 900 for a 30x30 grid)

Conclusions and Future Works

I-DLV-sr: an ASP-based stream reasoner

- Tight interaction between $\mathscr{I}^2$-DLV and a FLINK-based application
- Easily extendable by design
- Good performance and scalability in complex domains

Future goal: move towards a more complete SR reasoner

- Add the support to additional language constructs
- Study proper means for the management of **noise** and **incompleteness**
- Investigate new real-world domains

Thank you!