UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR INFORMATIONSSYSTEME

Özgür L. Özçep

Stream Processing 2

Lecture 12: STARQL
10 February, 2016

Foundations of Ontologies and Databases for Information Systems
CS5130 (Winter 2015)

Recap

- Talked about stream basics
- Hinted on higher-level declarative stream processing
  - Declarative: streams have assertional status
  - High-level: Have to incorporate (reason over) a background KB
- There are many interesting systems for stream processing w.r.t. an ontology—which we will not consider here
  - See activities of the RDF stream community
    `https://www.w3.org/community/rsp/`
  - See also one of the tutorials of Emanuelle Della Valle, e.g.
    `http://emanueledellavalle.org/Teaching/srt2015.html`

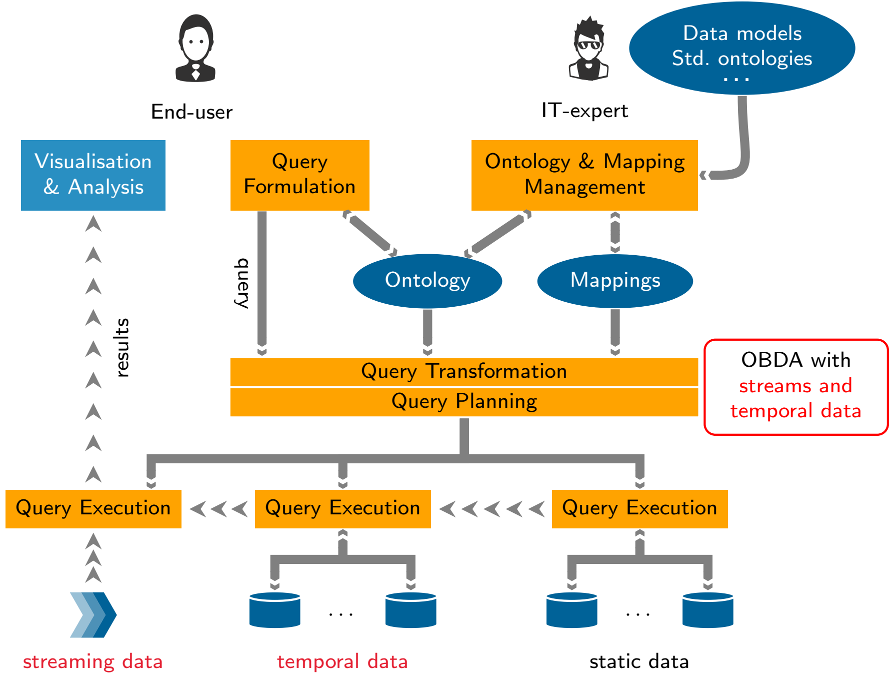
The query framework STARQL

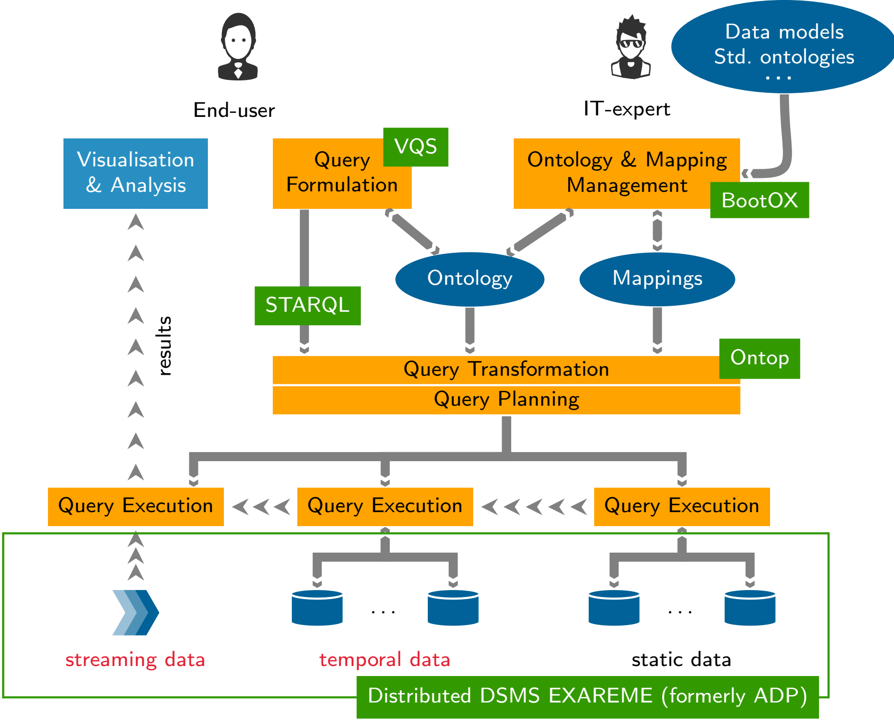
STARQL: Overview

- Started development within OPTIQUE
- Uses non-reified approach
- Use local temporal reasoning on finite state sequences
- Has framework character: embed different condition languages
- Convention for the following
  - Use logical ABox/TBox notation for RDF assertions (also in streams.), i.e.,
    - `{s0 rdf:type TempSensor}` written as TempSens(s0).
    - `{s0 val 90}` written as val(s0,90)
  - Use SPARQL notation within STARQL queries.

STARQL: Overview

- Started development within OPTIQUE
- Uses non-reified approach
- Use local temporal reasoning on finite state sequences
- Has framework character: embed different condition languages
- Convention for the following
  - Use logical ABox/TBox notation for RDF assertions (also in streams.), i.e.,
    - `{s0 rdf:type TempSensor}` written as TempSens(s0).
    - `{s0 val 90}` written as val(s0,90)
  - Use SPARQL notation within STARQL queries.


End-user
IT-expert
Data models
Std. ontologies
...
Visualisation
& Analysis
Query
Formulation
Ontology & Mapping
Management
query
Ontology
Mappings
results
Query Transformation
Query Planning
OBDA with
streams and
temporal data
Query Execution
Query Execution
Query Execution
...
...
streaming data
temporal data
static data


Data models
Std. ontologies
...
End-user
IT-expert
Visualisation & Analysis
Query Formulation
VQS
Ontology & Mapping Management
BootOX
Ontology
Mappings
STARQL
results
Query Transformation
Ontop
Query Planning
Query Execution
Query Execution
Query Execution
streaming data
temporal data
static data
Distributed DSMS EXAREME (formerly ADP)

Structure of STARQL queries

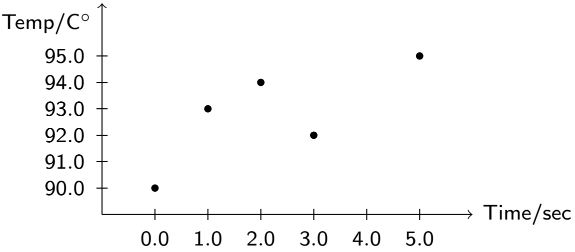
STARQL Query Template

`CREATE STREAM`	(initializes new stream)
`CREATE PULSE`	(create pulse for output times)
`SELECT/ CONSTRUCT`	(specifies output format)
`FROM`	(specifies the input streams)
`USING`	(specifies the static input)
`WHERE`	(selection w.r.t. static data)
`SEQUENCE BY`	(sequencing strategy)
`HAVING`	(FOL template for local temporal reasoning on states)

A Basic STARQL Example

Input: Stream S_{Msmt} of measurement assertions.

$$
\begin{aligned}
S_{Msmt} \quad = \quad \{ \; & val(s_0, 90^\circ C)\langle 0s \rangle, \\
& val(s_0, 93^\circ C)\langle 1s \rangle, \\
& val(s_0, 94^\circ C)\langle 2s \rangle, \\
& val(s_0, 92^\circ C)\langle 3s \rangle, \\
& val(s_0, 95^\circ C)\langle 5s \rangle \\
& \ldots \}
\end{aligned}
$$



Information Needs in STARQL

Information Need for Monotonicity (IN-Mon)

Tell every 1s whether the temperature in sensor *s*0 increased monotonically in the last 2s.

STARQL Representation (STARQL-Mon)

```
CREATE STREAM S_out_1 AS
PULSE START = 0s, FREQUENCY = 1s
CONSTRUCT {s0 rdf:type RecMonInc}<NOW>
FROM S_Msmt [NOW-2s, NOW]->1s
SEQUENCE BY StdSeq AS SEQ1
HAVING FORALL i < j IN SEQ1,?x,?y:
       IF {s0 val ?x}<i>  AND {s0  val ?y}<j>
       THEN ?x <= ?y
```

Components of STARQL

Information Need for Monotonicity (IN-Mon)

Tell every 1s whether the temperature in sensor *s*0 increased monotonically in the last 2s in stream S_Msmt.

```
CREATE STREAM S_out_1 AS
PULSE START = 0s, FREQUENCY = 1s
CONSTRUCT {s0 rdf:type RecMonInc}<NOW>
FROM S_Msmt [NOW-2s, NOW]->1s
SEQUENCE BY StdSeq AS SEQ1
HAVING FORALL i < j IN SEQ1,?x,?y:
       IF {s0 :val ?x}<i>  AND {s0 :val ?y}<j>
       THEN ?x <= ?y
```

- Creates stream named S_{out_1}
- Can be referenced under this name within another query

Components of STARQL

Information Need for Monotonicity (IN-Mon)

Tell every 1s whether the temperature in sensor *s*0 increased monotonically in the last 2s in stream S_Msmt.

```
CREATE STREAM S_out_1 AS
PULSE START = 0s, FREQUENCY = 1s
CONSTRUCT {s0 rdf:type RecMonInc}<NOW>
FROM S_Msmt [NOW-2s, NOW]->1s
SEQUENCE BY StdSeq AS SEQ1
HAVING FORALL i < j IN SEQ1,?x,?y:
       IF {s0 :val ?x}<i>  AND {s0 :val ?y}<j>
       THEN ?x <= ?y
```

- CONSTRUCT is a SPARQL like constructor
- Alternatively, if one is interested only in the bindings one uses SELECT

Output Format

$$
\begin{aligned}
S_{Msmt} \; = \; \{ \; & val(s_0, 90^\circ C)\langle 0s\rangle, \\
& val(s_0, 93^\circ C)\langle 1s\rangle, \\
& val(s_0, 94^\circ C)\langle 2s\rangle, \\
& val(s_0, 92^\circ C)\langle 3s\rangle, \\
& \\
& val(s_0, 95^\circ C)\langle 5s\rangle \\
& \ldots \}
\end{aligned}
$$

$$
\begin{aligned}
S_{out_1} \; = \; \{ \; & RecMonInc(s_0)\langle 0s\rangle, \\
& RecMonInc(s_0)\langle 1s\rangle, \\
& RecMonInc(s_0)\langle 2s\rangle, \\
& \\
& \\
& RecMonInc(s_0)\langle 5s\rangle \\
& \ldots \}
\end{aligned}
$$

Components of STARQL

Information Need for Monotonicity (IN-Mon)

Tell every 1s whether the temperature in sensor *s*0 increased monotonically in the last 2s in stream S_Msmt.

```
CREATE STREAM S_out_1 AS
PULSE START = 0s, FREQUENCY = 1s
CONSTRUCT {s0 rdf:type RecMonInc}<NOW>
FROM S_Msmt [NOW-2s, NOW]-> 1s
SEQUENCE BY StdSeq AS SEQ1
HAVING FORALL i < j IN SEQ1,?x,?y:
       IF {s0 :val ?x}<i>  AND {s0 :val ?y}<j>
       THEN ?x <= ?y
```

- Pulse fixes output times (bindings of NOW variable)
- Needed also for synchronization of streams

Components of STARQL

Information Need for Monotonicity (IN-Mon)

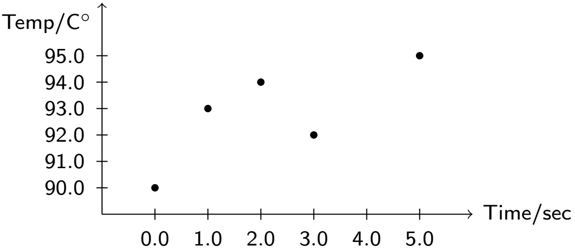
Tell every 1s whether the temperature in sensor $s0$ increased monotonically in the last 2s in stream S_Msmt.

```
CREATE STREAM S_out_1 AS
PULSE START = 0s, FREQUENCY = 1s
CONSTRUCT {s0 rdf:type RecMonInc}<NOW>
FROM S_Msmt [NOW-2s, NOW]-> 1s
SEQUENCE BY StdSeq AS SEQ1
HAVING FORALL i < j IN SEQ1,?x,?y:
       IF {s0 :val ?x}<i>  AND {s0 :val ?y}<j>
       THEN ?x <= ?y
```

- Window specification with window interval and slide parameter
- Applied to input stream S_Msmt of timestamped RDF assertions (= RDF quadruples)

Window Semantics

- `S_Msmt [NOW-2s,NOW]->1s`: stream of temporal ABoxes
- Sliding movement as in CQL but with timestamp preservation



Window sliding every second

Time	Window contents
$0s$	$val(s_0, 90^\circ)\langle 0s\rangle$
$1s$	$val(s_0, 90^\circ)\langle 0s\rangle$, $val(s_0, 93^\circ)\langle 1s\rangle$
$2s$	$val(s_0, 90^\circ)\langle 0s\rangle$, $val(s_0, 93^\circ)\langle 1s\rangle$, $val(s_0, 94^\circ)\langle 2s\rangle$
$3s$	$val(s_0, 93^\circ)\langle 1s\rangle$, $val(s_0, 94^\circ)\langle 2s\rangle$, $val(s_0, 92^\circ)\langle 3s\rangle$
$4s$	$val(s_0, 94^\circ)\langle 2s\rangle$, $val(s_0, 92^\circ)\langle 3s\rangle$
$5s$	$val(s_0, 92^\circ)\langle 3s\rangle$, $val(s_0, 95^\circ)\langle 5s\rangle$

Components of STARQL

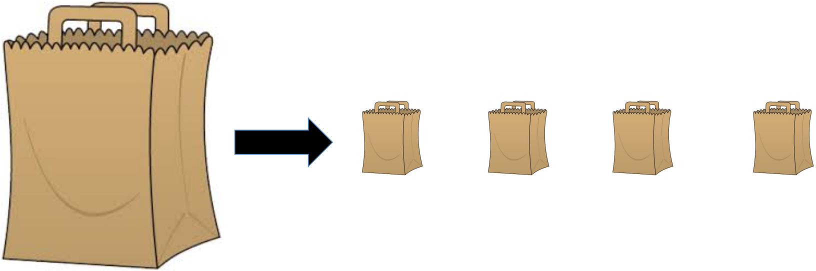
Information Need for Monotonicity (IN-Mon)

Tell every 1s whether the temperature in sensor $s0$ increased monotonically in the last 2s in stream S_Msmt.

```
CREATE STREAM S_out_1 AS
PULSE START = 0s, FREQUENCY = 1s
CONSTRUCT {s0 rdf:type RecMonInc}<NOW>
FROM S_Msmt [NOW-2s, NOW] -> 1s
SEQUENCE BY StdSeq AS SEQ1
HAVING FORALL i < j IN SEQ1,?x,?y:
       IF {s0 :val ?x}<i>  AND {s0 :val ?y}<j>
       THEN ?x <= ?y
```

- Generate every 1 second a sequence of states referred to by variables i, j
- States are annotated with ABoxes (RDF repositories)
- StdSeq = Standard Sequencing

STARQL Sequencing

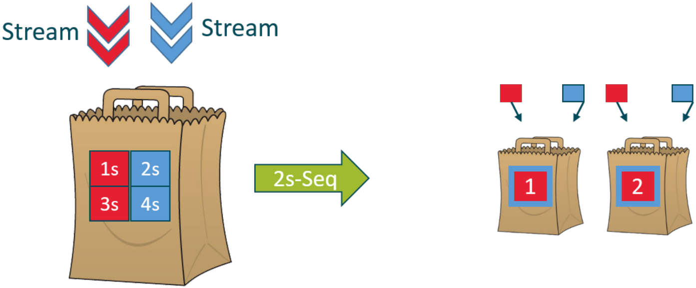


- Group elements according to specified criterion (including timestamps) into mini-bags
- Technically: Result is a sequence of ABoxes/RDF graphs

STARQL Sequencing

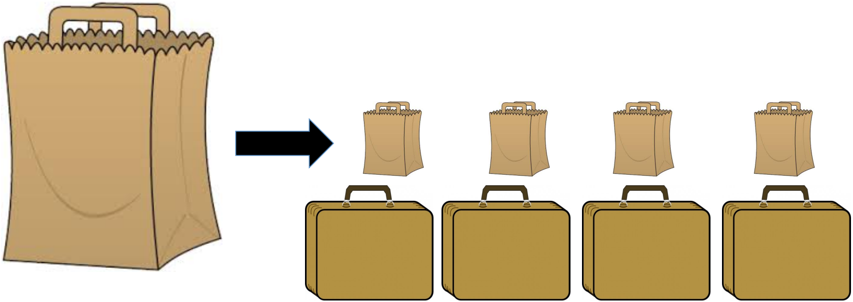
- Group elements according to specified criterion (operator based on timestamps) into mini-bags
- Technically: Result is a sequence of ABoxes/RDF graphs

STARQL Sequencing and Multi-Streams



- Multi Streams are joined in the big bag and grouped together in mini-bags
- Non-standard sequences as grouping criteria

Don't Forget the Suitcase



- At every state: Incorporate background knowledge
- Semantically clear; how to achieve feasibility not

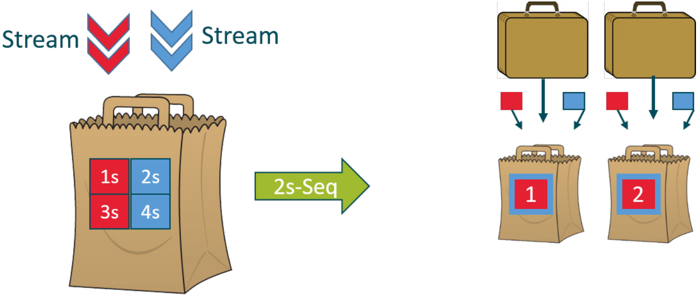
Components of STARQL

Extended Information Need for Monotonicity (IN-Mon)

Tell every 1s for every temperature sensor *s* whether the temperature increased monotonically in the last 2s in stream S_Msmt.

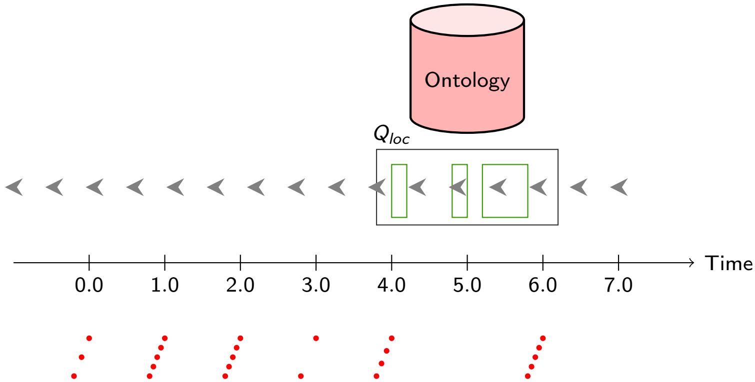
```
CREATE STREAM S_out_1 AS
PULSE START = 0s, FREQUENCY = 1s
CONSTRUCT {s rdf:type RecMonInc}<NOW>
FROM S_Msmt [NOW-2s, NOW] -> 1s
WHERE  { s rdf:type TemperatureSensor }
SEQUENCE BY StdSeq AS SEQ1
HAVING FORALL i < j IN SEQ1,?x,?y:
       IF {s0 :val ?x}<i>  AND {s0 :val ?y}<j>
       THEN ?x <= ?y
```

- One has to incorporate the background knowledge on sensor types at every state
- Semantically clear: Add static Abox to every state ABox

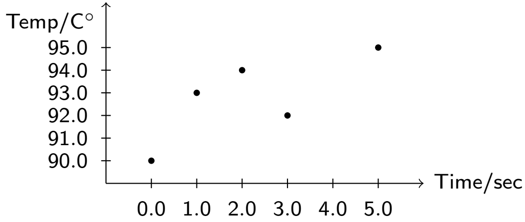


- Multi streams are joined in the big bag and group together in mini-bags
- Non standard sequences as grouping criteria
- Static data is added to every ABox

STARQL: Mini-Bags



The Sequence View



Time	Window contents before sequencing	
0s	$val(s_0, 90^\circ)\langle 0s\rangle$	
1s	$val(s_0, 90^\circ)\langle 0s\rangle$, $val(s_0, 93^\circ)\langle 1s\rangle$	
2s	$val(s_0, 90^\circ)\langle 0s\rangle$, $val(s_0, 93^\circ)\langle 1s\rangle$, $val(s_0, 94^\circ)\langle 2s\rangle$	
3s	$val(s_0, 93^\circ)\langle 1s\rangle$, $val(s_0, 94^\circ)\langle 2s\rangle$, $val(s_0, 92^\circ)\langle 3s\rangle$	
4s	$val(s_0, 94^\circ)\langle 2s\rangle$, $val(s_0, 92^\circ)\langle 3s\rangle$	
5s	$val(s_0, 92^\circ)\langle 3s\rangle$, $val(s_0, 95^\circ)\langle 5s\rangle$	
Time	**Window contents after standard sequencing**	***SEQ*1**
0s	$\{val(s_0, 90^\circ)\}\langle 0\rangle$	$\{0\}$
1s	$\{val(s_0, 90^\circ)\}\langle 0\rangle$, $\{val(s_0, 93^\circ)\}\langle 1\rangle$	$\{0, 1\}$
2s	$\{val(s_0, 90^\circ)\}\langle 0\rangle$, $\{val(s_0, 93^\circ)\}\langle 1\rangle$, $\{val(s_0, 94^\circ)\}\langle 2\rangle$	$\{0, 1, 2\}$
3s	$\{val(s_0, 93^\circ)\}\langle 0\rangle$, $\{val(s_0, 94^\circ)\}\langle 1\rangle$, $\{val(s_0, 92^\circ)\}\langle 2\rangle$	$\{0, 1, 2\}$
4s	$\{val(s_0, 94^\circ)\}\langle 0\rangle$, $\{val(s_0, 92^\circ)\}\langle 1\rangle$	$\{0, 1\}$
5s	$\{val(s_0, 92^\circ)\}\langle 0\rangle$, $\{val(s_0, 95^\circ)\}\langle 1\rangle$	{0,1}

Timestamps to Sequences

Time	Window contents before sequencing	
...	...	
$5s$	$val(s_0, 92°)\langle 3s\rangle,\ \ val(s_0, 95°)\langle 5s\rangle$	
Time	Window contents after standard sequencing	$SEQ1$
...	...	
5s	$\{val(s_0, 92°)\}\langle 0\rangle,\ \ \{val(s_0, 95°)\}\langle 1\rangle$	{0,1}

- Timestamped assertions are grouped to ABoxes with state index
- Information on timestamps and on their distance gets lost
- The index set SEQ may be different at every time point NOW
- One may think of SEQ as a dynamic relation giving for every time point the set of states
- For unfolding: Additionally SEQ may contain for every state also the corresponding timestamp.

Timestamps to Sequences

Time	Window contents before sequencing	
...	...	
5s	$val(s_0, 92^\circ)\langle 3s\rangle,\ \ val(s_0, 95^\circ)\langle 5s\rangle$	
Time	Window contents after standard sequencing	$SEQ1$
...	...	
5s	$\{val(s_0, 92^\circ)\}\langle 0\rangle,\ \ \{val(s_0, 95^\circ)\}\langle 1\rangle$	{0,1}

- Timestamped assertions are grouped to ABoxes with state index
- Information on timestamps and on their distance gets lost
- The index set SEQ may be different at every time point NOW
- One may think of SEQ as a dynamic relation giving for every time point the set of states
- For unfolding: Additionally SEQ may contain for every state also the corresponding timestamp.

Why at all Bother with State Sequences?

- Building microcosm for LTL like temporal reasoning on states
- But note
  - Temporal logic frameworks presuppose state sequences
  - In contrast, sequence construction is part of STARQL query
- Can, if needed, regain information by timestamp function on states
- With state approach one can handle non-standard sequencing techniques
  - for advanced machine learning techniques
  - in order to realize pre-processing: Filter out inconsistent ABoxes
  - in order to realize pre-processing: Roughen time granularity

Non-Standard Sequencing

- Use arbitrary congruence $\sim$ on time domain for sequencing
- Example: $x \sim y$ iff $\lfloor x/2 \rfloor = \lfloor y/2 \rfloor$ for all $x, y \in T = \mathbb{N}$.

Time	Window contents before sequencing
$0s$	$val(s_0, 90°)\langle 0s \rangle$
$1s$	$val(s_0, 90°)\langle 0s \rangle$, $val(s_0, 93°)\langle 1s \rangle$
$2s$	$val(s_0, 90°)\langle 0s \rangle$, $val(s_0, 93°)\langle 1s \rangle$, $val(s_0, 94°)\langle 2s \rangle$
3s	$val(s_0, 93°)\langle 1s \rangle$, $val(s_0, 94°)\langle 2s \rangle$, $val(s_0, 92°)\langle 3s \rangle$
$4s$	$val(s_0, 94°)\langle 2s \rangle$, $val(s_0, 92°)\langle 3s \rangle$
$5s$	$val(s_0, 92°)\langle 3s \rangle$, $val(s_0, 95°)\langle 5s \rangle$
Time	**Window contents after $\sim$ sequencing**
$0s$	$\{val(s_0, 90°)\}\langle 0 \rangle$
$1s$	$\{val(s_0, 90°), val(s_0, 93°)\}\langle 0 \rangle$
$2s$	$\{val(s_0, 90°), val(s_0, 93°)\}\langle 0 \rangle$, $\{val(s_0, 94°)\}\langle 1 \rangle$
3s	$\{val(s_0, 93°)\}\langle 0 \rangle$, $\{val(s_0, 94°), val(s_0, 92°)\}\langle 1 \rangle$
$4s$	$\{val(s_0, 94°), val(s_0, 92°)\}\langle 0 \rangle$
$5s$	$\{val(s_0, 92°)\}\langle 0 \rangle$, $\{val(s_0, 95°)\}\langle 1 \rangle$

Components of STARQL

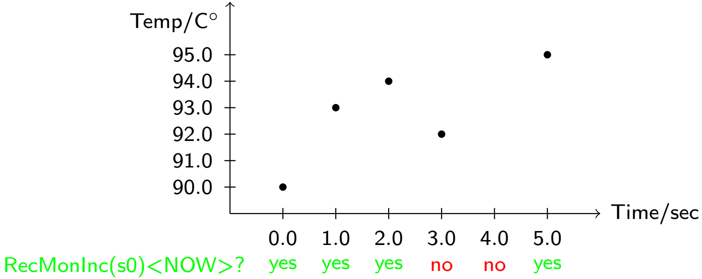
Information Need for Monotonicity (IN-Mon)

Tell every 1s whether the temperature in sensor *s*0 increased monotonically in the last 2s in stream S_Msmt.

```
CREATE STREAM S_out_1 AS
PULSE START = 0s, FREQUENCY = 1s
CONSTRUCT {s0 rdf:type RecMonInc}<NOW>
FROM S_Msmt [NOW-2s, NOW] -> 1s
SEQUENCE BY StdSeq AS SEQ1
HAVING FORALL i < j IN SEQ1,?x,?y:
       IF {s0 :val ?x}<i>  AND {s0 :val ?y}<j>
       THEN ?x <= ?y
```

- First order condition over states with special "atoms"
- Informal epistemic semantics of {s0 :val ?x}<i>:
  it is (provably) the case that in state i s0 has value ?x.

Testing the Conditions



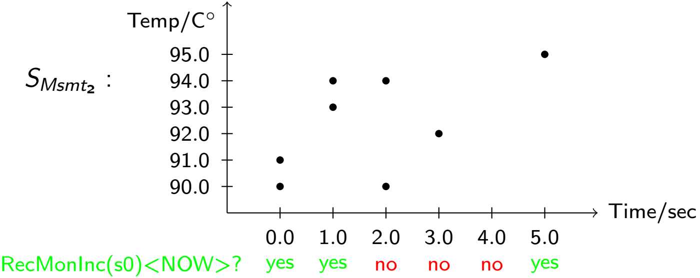
$$
\begin{aligned}
S_{Msmt} &= \{\, val(s_0, 90^\circ C)\langle 0s\rangle, \\
&\quad\;\; val(s_0, 93^\circ C)\langle 1s\rangle, \\
&\quad\;\; val(s_0, 94^\circ C)\langle 2s\rangle, \\
&\quad\;\; val(s_0, 92^\circ C)\langle 3s\rangle, \\
&\quad\;\; val(s_0, 95^\circ C)\langle 5s\rangle \\
&\quad\;\; \ldots \}
\end{aligned}
$$

$$
\begin{aligned}
S_{out_1} &= \{\, RecMonInc(s_0)\langle 0s\rangle, \\
&\quad\;\; RecMonInc(s_0)\langle 1s\rangle, \\
&\quad\;\; RecMonInc(s_0)\langle 2s\rangle, \\
&\quad\;\; RecMonInc(s_0)\langle 5s\rangle \\
&\quad\;\; \ldots \}
\end{aligned}
$$

Intricacies of the Monotonicity Condition

Information Need for Monotonicity (IN-Mon)

```
...
HAVING FORALL i < j IN SEQ1,?x,?y:
       IF {s0 :val ?x}<i>  AND {s0 :val ?y}<j>
       THEN ?x <= ?y
```



Expressive Strength of HAVING Clauses

Information Need for Monotonicity (IN-Mon)

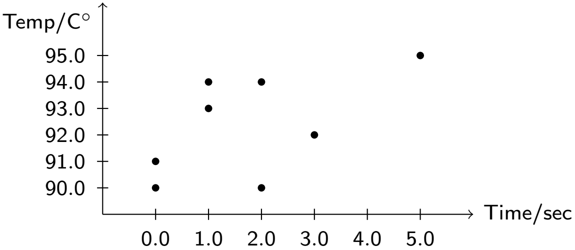
Tell every 1s whether the temperature in sensor *s*0 increased monotonically in the last 2s in stream S_Msmt and whether the value is functional.

```
CREATE STREAM S_out_1 AS
CONSTRUCT {s0 rdf:type RecMonInc}<NOW>
FROM S_Msmt [NOW-2s, NOW] -> 1s
SEQUENCE BY StdSeq AS SEQ1
HAVING FORALL i  <=  j IN SEQ1,?x,?y:
       IF {s0 :val ?x}<i>  AND {s0 :val ?y}<j>
       THEN ?x <= ?y
```

- Check for $i = j$ means checking of "functionality" of *val* in ABox $i = j$
- then "monotonicity" in the usual sense on non-empty bindings (which must be unique at every time point)

Monotonicity Variant

S_{Msmt_2} :



RecMonInc(s0)<NOW>? no no no no no yes

Pulse Declarations

- STARQL uses window operator as in CQL
- but mitigates CQLs "problems" with continuous time flows on the query language level
- STARQL uses pulse declaration for well-defined output stream with `CREATE PULSE`
- Pulse synchronizes multiples streams
- Pulse defines output times

Operational Semantics

Example template for operational semantics

```
CREATE STREAM S_out
CREATE PULSE START = st, FREQUENCY = fr
...
FROM S_MSMt [NOW-wr, NOW] -> sl
...
```

- Pulse time vs. stream time
- Pulse time t_{pulse} regular according to FREQUENCY
  $t_{pulse} = st \longrightarrow st + fr \longrightarrow st + 2fr \longrightarrow \ldots$
- Stream time t_{str} determined by trace of endpoint of sliding window
- Stream time jumping/sliding

How Streaming Time Evolves I

Example template for operational semantics

```
CREATE STREAM S_out
CREATE PULSE START = st, FREQUENCY = fr
...
FROM S_MSMt [NOW-wr, NOW] -> sl
...
```

- Evolvement of t_{str}:
  $$t_{str} \xrightarrow{\text{IF } t_{str} + m \times sl \leq t_{pulse} \text{ (for } m \in \mathbb{N} \text{ maximal)}} t_{str} + m \times sl$$
- Window contents at t_{pulse}:
  $\{ax\langle t\rangle \in S_{Msmt} \mid t_{str} - wr \leq t \leq t_{str}\}$
- Always $t_{str} \leq t_{pulse}$.

How Streaming Time Evolves II

Instantiation of example template

```
CREATE STREAM S_out
CREATE PULSE START = 0s, FREQUENCY = 2s
...
FROM S_MSMt [NOW-3s, NOW] -> 3s
...
```

$$t_{pulse}: \quad 0s \rightarrow 2s \rightarrow 4s \rightarrow 6s \rightarrow 8s \rightarrow 10s \rightarrow 12s \rightarrow$$

$$t_{str}: \quad 0s \rightarrow 0s \rightarrow 3s \rightarrow 6s \rightarrow 6s \rightarrow 9s \rightarrow 12s \rightarrow$$

Example

Multiple streams

```
CREAT STREAM Sout AS
PULSE START = 0s, FREQUENCY = 2s
CONSTRUCT  ?sens rdf:type RecentMonInc <NOW>
FROM    S_Msmt_1 0s<-[NOW-3s, NOW]->3s,
        S_Msmt_2 0s<-[NOW-3s, NOW]->2s
SEQUENCE BY StdSeq AS SEQ
HAVING (...)
```

$$t_{pulse}: \quad 0s \rightarrow 2s \rightarrow 4s \rightarrow 6s \rightarrow 8s \rightarrow 10s \rightarrow 12s \rightarrow$$

$$t_{S_{Msmt_1}}: \quad 0s \rightarrow 0s \rightarrow 3s \rightarrow 6s \rightarrow 6s \rightarrow 9s \rightarrow 12s \rightarrow$$

$$t_{S_{Msmt_2}}: \quad 0s \rightarrow 2s \rightarrow 4s \rightarrow 6s \rightarrow 8s \rightarrow 10s \rightarrow 12s \rightarrow$$

Reasoning w.r.t. TBox and Static ABox

Extended Information Need (IN-Emon)

Tell every 1s whether the temperature in all temperature sensors increased monotonically in the last 2s in stream S_Msmt

```
CREATE STREAM S_out_2 AS
PULSE START = 0s, FREQUENCY = 1s
SELECT  { ?s rdf:type RecentMonInc }<NOW>
FROM S_Msmt [NOW-2s, NOW]->1s
USING STATIC ABOX <http://Astatic>,
      TBOX <http://TBox>
WHERE { ?s rdf:type TempSens }
SEQUENCE BY StdSeq AS SEQ
HAVING
 FORALL i < j IN SEQ,?x,?y:
 IF ({ ?s val ?x }<i>  AND { ?s  val ?y }<j>)
 THEN ?x <= ?y
```

Reasoning

- TBox $\mathcal{T}$
  - No temporal constructors
  - Example: $\mathit{BurnerTipTempSensor} \sqsubseteq \mathit{TempSens}$
    "At every time point: a burner tip temperature sensor is a temperature sensor"
- Static ABox $\mathcal{A}_{st}$
  - Assertions assumed not to change in time
  - i.e., to hold at every time point
  - Example: $\mathit{BurnerTipTempSens}(s0)$, $\mathit{hasComponent}(\mathit{turb}, s1)$

WHERE Clause

Extended Information Need (IN-Emon)

```
...
USING STATIC ABOX <http://Astatic>,
      TBOX <http://TBox>
WHERE { ?s rdf:type TempSens }
SEQUENCE BY StdSeq AS SEQ
HAVING
 FORALL i < j IN SEQ,?x,?y:
 IF { ?s  val ?x }<i>  AND { ?s   val ?y }<j>
 THEN ?x <= ?y
```

- Answering WHERE clause by certain answer semantics
  - $\psi_{WHERE}(?s) = TempSens(?s)$
  - $cert(\psi_{WHERE}, \mathcal{T} \cup \mathcal{A}_{st})$
  - Example: Captures also BurnerTipTempSensors
- Gives preselection of constants for instantiation in HAVING clause

Semantics of HAVING Clause

Extended Information Need (IN-Emon)

```
...
HAVING
 FORALL i < j IN SEQ,?x,?y:
 IF { ?s val ?x }<i>  AND { ?s  val ?y }<j>
 THEN ?x <= ?y
```

- In original STARQL semantics $\langle i \rangle$ is interpreted as epistemic operator
- Motivated by framework approach
- $val(?s, ?x)\langle i \rangle$ holds if it is provably the case in ith ABox that $val(?x, ?y)$
- Note the different uses of $\langle \cdot \rangle$
- $cert(val(?s, ?x), \mathcal{A}_i \cup \mathcal{T} \cup \mathcal{A}_{st})$

Rewritability of HAVING Clauses

- Rewritability of HAVING clause becomes almost trivial for epistemic semantics
  - One perfectly rewrites embedded queries in state indexed atoms w.r.t. $\mathcal{T}$
  - Resulting HAVING clause can be formulated in FOL with $<, +$
- Example
  - HAVING clause
    `...EXISTS i {?s val ?x} <i> ...`
  - TBox axiom: $\mathit{tempVal} \sqsubseteq \mathit{val} \in \mathcal{T}$
  - Rewritten HAVING clause
    ... `EXISTS i ({?s val ?x} UNION {?s tempVal ?x})<i>`
    ...
- Works only for $\mathcal{T}$ without temporal operators

Rewritability of HAVING Clauses

- Non-epistemic semantics of $\langle i \rangle$
  - Read $\langle i \rangle$ not as operator but as state-index attachment
  - $val(s, x)\langle i \rangle$ read as $val(s, x, i)$
- Same rewriting as for epistemic semantics works for some fragment of HAVING clauses
  - No negation
  - No FORALL over domain variables

Inconsistency Handling

- Want to express that at every time point a sensor has at most one value
- Non-reified view with classical TBox $\mathcal{T}$: $(\mathit{func}\ \mathit{val}) \in \mathcal{T}$
- No home-made inconsistencies in STARQL window semantics
  - Window operator conserves timestamps

Time	Window contents before sequencing
...	...
$5s$	$val(s_0, 92^\circ)\langle 3s\rangle,\ \ val(s_0, 95^\circ)\langle 5s\rangle$

  - Otherwise we could have: $val(s_0, 90), val(s_0, 91)$
  - This was the reason to change CQL window semantics to STARQL window semantic
- In reified view no similar problem with window semantics
- But more difficult to express functionality
  "There are no two measurements having the same sensor but different times"

Inconsistency Handling

- Want to express that at every time point a sensor has at most one value
- Non-reified view with classical TBox $\mathcal{T}$: $(func\ val) \in \mathcal{T}$
- No home-made inconsistencies in STARQL window semantics
  - Window operator conserves timestamps

Time	Window contents before sequencing
...	...
$5s$	$val(s_0, 92^\circ)\langle 3s \rangle, \ \ val(s_0, 95^\circ)\langle 5s \rangle$

  - Otherwise we could have: $val(s_0, 90), val(s_0, 91)$
  - This was the reason to change CQL window semantics to STARQL window semantic
- In reified view no similar problem with window semantics
- But more difficult to express functionality
  "There are no two measurements having the same sensor but different times"

Non-Standard Sequencing Again

- $S_{Msmt} = \{\dots val(s0, 90)\langle 3s \rangle, val(s0, 95)\langle 3s \rangle \dots\}$
- With standard sequencing leads to an ABox not consistent with (*func val*)
- Can test for inconsistencies by FOL query (Consistency is FOL rewritable for DL-Lite)
- How to handle inconsistent ABoxes?
  1. Use repair semantics (not classical certain answer semantics) (perhaps in the future)
  2. Use non-standard sequencing eliminating non-consistent ABoxes

Detecting Inter-Temporal Inconsistencies

- Remember: No temporal operators in $\mathcal{T}$

$$\begin{aligned} \exists tempVal \sqsubseteq TempSens, \\ \exists pressVal \sqsubseteq PressSens \\ TempSens \sqsubseteq \neg PressSens \end{aligned}$$

- $S_{Msmt} = \{\dots tempVal(s0, 90)\langle 3s\rangle, pressVal(s0, 70)\langle 4s\rangle \dots\}$
- Intuitively: Information regarding $s0$ not consistent
- Not detected if s_0 not classified in static ABox
- Reasoning: Cannot express rigidity on sensor concepts

Querying Historical Data

- Different approaches to handle historical data in STARQL
  1. Put slide = 0 and fix window ends
  2. Stream historical data according to timestamps

Example solution 1

Return all sensor values of a specific sensor $s0$ within a specific time interval $[0s, 60s]$

```
CREATE GRAPH Solution-One AS
CONSTRUCT   { s0 :val ?x }
FROM S_Msmt[0s, 60s]->0s
USING  STATIC ABOX <http://ABox>,
       TBOX <http://TBox>
SEQUENCE BY StdSeq AS SEQ
HAVING EXISTS i  { s0 :val ?x } <i>
```

Querying Historical Data

Example solution 2

Return all sensor values of a specific sensor $s0$ within a specific time interval $[0s, 60s]$

```
CREATE STREAM Solution-TWO AS
CREATE PULSE AS
       START = 0s, FREQUENCY = 1s, END = 60s
CONSTRUCT   { s0 :val ?x }<NOW>
FROM S_Msmt[NOW, NOW]->1s
USING  STATIC ABOX <http://ABox>,
       TBOX <http://TBox>
SEQUENCE BY StdSeq AS SEQ
HAVING EXISTS i  { s0 :val ?x } <i>
```

Window Semantics Again

- No difference whether S_Msmt is real-time stream or streamed historical data
- Due to $t_{str} \leq t_{pulse}$
- Assume otherwise ($t_{str} > t_{pulse}$)
  - Historical query: window may contain future elements from $[t_{pulse}, t_{str}]$
  - Stream query: window cannot contain future elements from $[t_{pulse}, t_{str}]$

Mapping Temporal and Streaming Data

- Mapping historical data

$m1 : val(x, y)\langle z\rangle \longleftarrow$

```
SELECT f(SID) AS x, Mval AS y, MtimeStamp AS z
FROM MEASUREMENT-TABLE
```

  - $\mathcal{A}(m1, DB)$ is a temporal ABox
  - where `MEASUREMENT-TABLE` in DB
- Mapping streams

$m2 : val(x, y)\langle z\rangle \longleftarrow$

```
SELECT Rstream(f(SID) AS x, Mval AS y,
               MtimeStamp AS z)
FROM MEASUREMENT-REL-STREAM
```

  - $\mathcal{A}(m2, Str - DB)$ is a stream of timestamped ABox assertions
  - where `MEASUREMENT-REL-STREAM` in Str-DB

Mapping Temporal and Streaming Data

- Mapping historical data

$m1 : val(x, y)\langle z\rangle \quad \longleftarrow$

```
SELECT f(SID) AS x, Mval AS y, MtimeStamp AS z
FROM MEASUREMENT-TABLE
```

  - $\mathcal{A}(m1, DB)$ is a temporal ABox
  - where `MEASUREMENT-TABLE` in DB
- Mapping streams

$m2 : val(x, y)\langle z\rangle \quad \longleftarrow$

```
SELECT Rstream(f(SID) AS x, Mval AS y,
               MtimeStamp AS z)
FROM MEASUREMENT-REL-STREAM
```

  - $\mathcal{A}(m2, Str - DB)$ is a stream of timestamped ABox assertions
  - where `MEASUREMENT-REL-STREAM` in Str-DB

Challenges of Unfolding

- `HAVING` clause language refers to state tagged not time stamped assertions
- Solution
  - Use simple sequencing mechanisms such as standard sequencing
  - Keep track of time window processing by stream of SEQ-entries
- `HAVING` clause language uses domain calculus, CQL tuple calculus
- Solution: Use safety mechanisms by adornments for variables to guarantee domain independence
- CQL looses timestamps in window contents
- Solution: Assume Stream-To-Stream Operator duplicating timestamps as time attributes: $d\langle t\rangle \mapsto (d, t)\langle t\rangle$

Challenges of Unfolding

- HAVING clause language refers to state tagged not time stamped assertions
- Solution
  - Use simple sequencing mechanisms such as standard sequencing
  - Keep track of time window processing by stream of SEQ-entries
- HAVING clause language uses domain calculus, CQL tuple calculus
- Solution: Use safety mechanisms by adornments for variables to guarantee domain independence
- CQL looses timestamps in window contents
- Solution: Assume Stream-To-Stream Operator duplicating timestamps as time attributes: $d\langle t\rangle \mapsto (d,t)\langle t\rangle$

Challenges of Unfolding

- `HAVING` clause language refers to state tagged not time stamped assertions
- Solution
  - Use simple sequencing mechanisms such as standard sequencing
  - Keep track of time window processing by stream of SEQ-entries
- `HAVING` clause language uses domain calculus, CQL tuple calculus
- Solution: Use safety mechanisms by adornments for variables to guarantee domain independence
- CQL looses timestamps in window contents
- Solution: Assume Stream-To-Stream Operator duplicating timestamps as time attributes: $d\langle t\rangle \mapsto (d,t)\langle t\rangle$

Safety Mechanism

- `HAVING` clause $?y > 3$ is not safe: Infinite binding set for $?y$
- $val(s_0, ?y)\langle i\rangle \wedge (?y > 3)$ is safe
- Adornments for variables ensure not only finiteness but domain independence
- Domain independence
  - Query answer depends only on the interpretations of the predicates mentioned in the query or the DB but not the domain
  - A query ϕ is domain independent iff for all interpretations $\mathcal{I}, \mathcal{J}$ such that $\mathcal{I}$ is a substructure of $\mathcal{J}$: $ans(\phi, \mathcal{I}) = ans(\phi, \mathcal{J})$.

Safety Mechanism

- `HAVING` clause $?y > 3$ is not safe: Infinite binding set for $?y$
- $val(s_0, ?y)\langle i \rangle \wedge (?y > 3)$ is safe
- Adornments for variables ensure not only finiteness but domain independence
- Domain independence
  - Query answer depends only on the interpretations of the predicates mentioned in the query or the DB but not the domain
  - A query ϕ is domain independent iff for all interpretations $\mathcal{I}, \mathcal{J}$ such that $\mathcal{I}$ is a substructure of $\mathcal{J}$: $ans(\phi, \mathcal{I}) = ans(\phi, \mathcal{J})$.

Domain Independence

- Counterexample
  - $\phi(x, y) = A(x) \vee B(y)$
  - $\mathcal{I} = (\{\alpha\}, (\cdot)^{\mathcal{I}})$, $\mathcal{J} = (\{\alpha, \beta\}, (\cdot)^{\mathcal{J}})$
  - $A^{\mathcal{I}} = A^{\mathcal{J}} = \{\alpha\}$
  - $B^{\mathcal{I}} = B^{\mathcal{J}} = \emptyset$:
  - $(\alpha, \beta) \in ans(\phi, \mathcal{J})$ but $(\alpha, \beta) \notin ans(\phi, \mathcal{I})$.

- Arbitrary use of disjunction has strange consequences for answering on DB
  - $\psi(?x, ?y) = TempSens(?x) \vee PressureSens(?y)$
  - gives finite set of bindings but is not domain independent

DB:	TempSens	PressureSens
	a_1	b_1

  - $ans(\psi(?x, ?y), DB) = \{(a_1, b_1), (a_1, a_1), (b_1, b_1)\}$

Adornments

- Safety conditions by variable adornments in $\{+, -, --, \emptyset\}$
  - x^+: x is safe variable
  - x^-: x is non-safe variable (but may become safe by negation)
  - x^{--}: x is non-safe variable
  - $x^\emptyset$: x does not occur in other formula
- Allowed adornment combinations fixed by table
- Grammar rules have form

$$hCl(\vec{z}^{\vec{g}^1 \vee \vec{g}^2}) \longrightarrow hCl(\vec{z}^{\vec{g}^1}) \texttt{ OR } hCl(\vec{z}^{\vec{g}^2})$$

g_1	g_2	$g_1 \vee g_2$	$\dots$
$--$	$--$	$--$	$\dots$
$--$	$-$	$-$	
$--$	$+$	$--$	
$--$	$\emptyset$	$--$	
$-$	$--$	$-$	
$-$	$-$	$-$	
$-$	$+$	$-$	
$-$	$\emptyset$	$-$	
$+$	$--$	$--$	
$+$	$-$	$-$	
$+$	$+$	$+$	
$+$	$\emptyset$	$--$	
$\emptyset$	$--$	$--$	
$\emptyset$	$-$	$-$	
$\emptyset$	$+$	$--$	
$\emptyset$	$\emptyset$	$\emptyset$	

Adornments

- Grammar rules have form

$$hCl(\vec{z}^{\vec{g}^1 \vee \vec{g}^2}) \longrightarrow hCl(\vec{z}^{\vec{g}^1}) \texttt{ OR } hCl(\vec{z}^{\vec{g}^2})$$

- Example

$$F(x_1^{--}, x_2^{+}, x_3^{-}) \longrightarrow$$
$$F_1(x_1^{--}, x_2^{+}, x_3^{-}) \texttt{ OR } F_2(x_1^{+}, x_2^{+}, x_3^{\emptyset})$$

g_1	g_2	$g_1 \vee g_2$	...
$--$	$--$	$--$	...
$--$	$-$	$-$	
$--$	$+$	$--$	
$--$	$\emptyset$	$--$	
$-$	$--$	$-$	
$-$	$-$	$-$	
$-$	$+$	$-$	
$-$	$\emptyset$	$-$	
$+$	$--$	$--$	
$+$	$-$	$-$	
$+$	$+$	$+$	
$+$	$\emptyset$	$--$	
$\emptyset$	$--$	$--$	
$\emptyset$	$-$	$-$	
$\emptyset$	$+$	$--$	
$\emptyset$	$\emptyset$	$\emptyset$	

Safety in Monotonicity Condition

Extended Information Need (IN-Emon)

```
HAVING
 FORALL i < j IN SEQ,?x,?y:
 IF { ?s val ?x }<i>  AND { ?s  val ?y }<j>
 THEN ?x <= ?y
```

- Unsafe variables in `?x <= ?y` ...
- ... are bound by antecedens of all quantifier

Transformation into SQL

- Safety mechanism guarantees: `HAVING` clauses transformable into formulas in safe range normal form (SRNF)
- Folklore theorem: SRNF is domain independent
- Transformation into SQL
  - Rewrite `FORALL` with `NOT EXISTS NOT`
  - Push `NOT` inwards (stopping at `EXISTS`) ...

Example (Part of the STARQL-to-Backend Transformation)

```
FORALL i < j IN SEQ,?x,?y:
IF { ?s val ?x }<i>  AND { ?s  val ?y }<j>
THEN ?x <= ?y

;; ==== transformed to  ====>

 NOT EXISTS i,j in SEQ, x,y:
 i < j AND { ?s val ?x }<i>  AND { ?s  val ?y }<j>
 AND x > y
```

Example (Monotonicity Query in STARQL)

```
CREATE STREAM S_out_1 AS
PULSE START = 0s, FREQUENCY = 1s
CONSTRUCT {s0 rdf:type RecMonInc}<NOW>
FROM S_Msmt [NOW-2s, NOW]->1s
SEQUENCE BY StdSeq AS SEQ1
HAVING FORALL i < j IN SEQ1,?x,?y:
       IF {s0 val ?x}<i>  AND {s0  val ?y}<j>
       THEN ?x <= ?y
```

Example (Outcome of Transformation in CQL)

```
CREATE VIEW windowRel as
SELECT *  FROM REL-STREAM-MEASUREMENT[RANGE 2s Slide 1s];

SELECT  Rstream(' s0 rdf:Type RecMonInc '||'<'||timestamp||'>')
FROM    windowRel
WHERE   windowRel.SID = 'TC255'  AND
        NOT EXISTS (
             SELECT * FROM
             (SELECT timestamp as i, value as x FROM windowRel),
             (SELECT timestamp as j, value as y FROM windowRel)
             WHERE   i < j AND x > y );
```

Complexity in Stream Processing

- Low-level stream processing: strict constraints on space complexity
- Very strict: exact O(log(n)) where n is length of stream (seen so far)
- Less strict:
  1. Approximate solutions
  2. O(polylog(n) n) ("semi-stream" in graph processing where *n* is number of vertices)
- Extensive use of synopses: data structure for storing relevant interim results

Questions

1. Isn't the space problem already solved by choosing a finite window?
2. Is this relevant for high-level (in particular STARQL) stream processing?

Ad Question 1 (Finite Window)

- Window may still be too big
- Small time based windows may still cause problems

  S: { val(s0,90)<3s>, val(s1,91)<3s>, val(s2,95)<3s>,
  val(s3,94)<3s>, val(s4,96)<3s>, ... }

  S[NOW,NOW] at (t = 3s): unbounded

Ad Question II (High-Level Streams)

- Answer: Want optimized version with small synopses (in particular for multiple query scenarios)
- Related problem in high-level data stream processing: Achieve memory-boundedness

Definition

A query is memory-bounded if there exists an algorithm using a constant number of registers as synopsis for producing answers.

Example: Monotonic Increase

Information Need for Monotonicity (IN-Mon)

Tell every 1s whether the temperature in sensor *s*0 increased monotonically in the last 10s.

STARQL Representation (STARQL-Mon)

```
...
FROM S_Msmt [NOW-10s, NOW]->1s
...
HAVING FORALL i < j IN SEQ1,?x,?y:
       IF {s0 val ?x}<i>  AND {s0  val ?y}<j>
       THEN ?x <= ?y
```

Example: Monotonic Increase

- Simple implementation
  - Every 1 second construct from scratch sequence in 10s-window and
  - test monotonicity (by iterating over all state pairs (i, j))
- Efficient implementation (alg-mon)
  - store max temp value for last and current time point
  - shift if incoming triple has new (bigger) timestamp
- Can this idea be generalized?

  Yes, but surely not for all queries

Example: not memory bounded

```
HAVING
EXISTS i,j { ?s :val ?x } <i> AND { ?r :val ?x }<j>
```

Example: Monotonic Increase

- Simple implementation
  - Every 1 second construct from scratch sequence in 10s-window and
  - test monotonicity (by iterating over all state pairs (i, j))
- Efficient implementation (alg-mon)
  - store max temp value for last and current time point
  - shift if incoming triple has new (bigger) timestamp
- Can this idea be generalized?

  Yes, but surely not for all queries

Example: not memory bounded

```
HAVING
EXISTS i,j { ?s :val ?x } <i> AND { ?r :val ?x }<j>
```

Testing for Memory Boundedness

Proposition

- There is a polynomial syntactic criterion on the existential positive fragment of HAVING clauses for testing whether a memory-bounded algorithm exists
  (uses a theorem of (Arasu et al. 04))
- In this case, the synopsis algorithm is a slightly generalized version of alg-mon

- But how to implement algorithm in STARQL?
  $\Longrightarrow$ user defined functions (UDFs)

Lit: A. Arasu, B. Babcock, S. Babu, J. McAlister, and J. Widom. Characterizing memory requirements for queries over continuous data streams. ACM Trans. Database Syst., 29(1):162–194, Mar. 2004.

Testing for Memory Boundedness

Proposition

- There is a polynomial syntactic criterion on the existential positive fragment of HAVING clauses for testing whether a memory-bounded algorithm exists
  (uses a theorem of (Arasu et al. 04))
- In this case, the synopsis algorithm is a slightly generalized version of alg-mon

- But how to implement algorithm in STARQL?
  $\Longrightarrow$ user defined functions (UDFs)

Lit: A. Arasu, B. Babcock, S. Babu, J. McAlister, and J. Widom. Characterizing memory requirements for queries over continuous data streams. ACM Trans. Database Syst., 29(1):162–194, Mar. 2004.

Multiple Queries and Streams

- **Reasons** for multiplicity in Sensor Measurement scenario
  - Monitor different components (e.g. turbines, sensors) in a system
  - Monitor different hand-crafted well-proven patterns
- Challenges
  - **Scalability**
  - Need specific and Generic optimization strategies

Example: Comparing sensor readings

What is it?

Measure the temperature at different (6 to 24) interduct thermocouples

How to spot problems?

- Do the readings really change in unison?
- Spot failing temperature probes: does one reading go out of sync with the others?

Bias drift solution

Compute average, monitor deviation from average for all individual temperatures, generate event if a certain absolute difference is exceeded.

$\Longrightarrow$ up to 24 queries (per turbine)

$\Longrightarrow$ using aggregation (average AVG)

Correlation

- Weakness of "Biased Drift" solution: Does not detect
  1. change of signal position (e.g., from coldest to hottest)
  2. when signals start spreading apart
- A fine-grained solution: using (Pearson) correlation

Definition

$$\rho(X, Y) = \frac{cov(X, Y)}{\sigma_X \sigma_Y} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y}$$

- Measure linear dependence of two random variables with value in [-1,1]
- μ = mean
- E = expectation value
- σ = standard deviation

Correlation

- Note: SQL allows only unary (one-column) aggregation
- Have to invent multicolumn user defined function
- Even more: we may be interested in calculating correlations among all sensors
- Challenge: Quadratic increase of correlation calculations
- Known optimization strategy for online correlation calculation: **Locality-sensitive hashing**

Local-Sensitive Hashing (LSH)

- Calculates most similar pairs above some correlation threshold
- Is an approximation technique for continuous calculation

- **Idea**: Hash into buckets and then calculate correlation only over bucket

  $\Longrightarrow$ Elements in different buckets are not similar
- But: An element is allowed to be in different buckets

- The unwanted cases
  - **False positives**: objects in same bucket but not similar
    Not severe: One recognizes it during correlation calculation, but may only lower performance
  - **False negatives**: similar object in different buckets
    Severe: not recognized.
  - Technique to lower false negatives: Allow objects to fall in more buckets.

STARQL Implementations(s)

- STARQL running as submodule in Optique platform
  - Uses stream extended version of EXAREME (formerly ADP)
  - `https://github.com/madgik/exareme`
    - Highly distributable DBMS
    - Extends SQL-Lite with window operators (as in CQL)
  - Mapping handling using ontop and hardcoded timestamp hook mechanism
  - Multiple streams
  - Nested queries
- But don't you stop here? Why more than one implementation?
  - Additional system for comparison with Optique submodule
  - Fast identification of errors, faults, unexpected behaviours
  - Fast change without dependencies
  - Pre-testing of desired features/requirements
  - Testing different paradigms: materialized vs. non-materialized

STARQL Implementations(s)

- STARQL running as submodule in Optique platform
  - Uses stream extended version of EXAREME (formerly ADP)
  - `https://github.com/madgik/exareme`
    - Highly distributable DBMS
    - Extends SQL-Lite with window operators (as in CQL)
  - Mapping handling using ontop and hardcoded timestamp hook mechanism
  - Multiple streams
  - Nested queries

- But don't you stop here? Why more than one implementation?
  - Additional system for comparison with Optique submodule
  - Fast identification of errors, faults, unexpected behaviours
  - Fast change without dependencies
  - Pre-testing of desired features/requirements
  - Testing different paradigms: materialized vs. non-materialized

STARQL implementations(s)

- STARQL+ PostgreSQL
  - Meant to be used for historical reasoning
  - Extended to STARQL+PipelineDB (`https://www.pipelinedb.com/`) in order to handle streams (work in progress)
- Prolog prototype
  - Translates STARQL queries into safe non-recursive datalog with negation
  - Uses mappings to SQL
  - Stream handling to be implemented
- LISP prototype
  - ABDEO approach
  - Uses materialization

Another Use Case

- Applying STARQL prototype to a use case which requires
  - handling timed data for reactive diagnosis and monitoring
  - coping with heterogeneous data
  - use of pattern recognition/machine learning
- Tiny Stream OBDA demo within FP7 Panoptesec (`www.panoptesec.eu`)
  - Intrusion detection within a cyber defence decision support system
  - Simple OWL ontology extracted from IDMEF XML file (Intrusion Detection Exchange Format)

Data

- Logs from components such as routers, hubs, firewalls, ids
- Example: IDS schema

timestamp	severity	src	dest	id	analyzer	description
09:41:28	1	...	...	21	ids0	url/rule-1111207/
05:52:56	1	...	...	22	ids0	url/rule-1111207/
02:10:34	2	...	...	23	ids0	url/rule-1111210/
17:44:50	2	...	...	24	ids0	url/rule-1111210/
06:49:49	2	...	...	25	ids0	url/rule-1111210/
...						
03:00:43	5	...	...	34	ids0	url/rule-1111203/

- url = http://www.digitalbond.com/tools/quickdraw/dnp3-rules

Rule: 1111207

<table>
<tr><td>GEN:SID</td><td>1:1111207</td></tr>
<tr><td>Message</td><td>DNP3 - Unauthorized Write Request to a PLC</td></tr>
<tr><td>Rule</td><td>alert tcp !$DNP3_CLIENT any -> $DNP3_SERVER $DNP3_PORTS (flow:from_client,established; content:"|05 64|"; depth:2; pcre:"/[\S\s]{10}(\x02|\x04|\x05|\x06|\x09|\x0A|\x0F|\x12)/iAR"; msg:"SCADA_IDS: DNP3 - Unauthorized Write Request to a PLC"; reference:url,digitalbond.com/tools/quickdraw/dnp3-rules; classtype:bad-unknown; sid:1111207; rev:1; priority:1;)</td></tr>
<tr><td>Summary</td><td>An unauthorized DNP3 client attempts to write information to a PLC or other field device.</td></tr>
<tr><td>Impact</td><td>System integrity. Denial of service.</td></tr>
<tr><td>Detailed Information</td><td>DNP3 is a protocol commonly used in SCADA and DCS networks for process control.The DNP3 protocol does not provide authentication of the source of a command. Most SCADA/DCS networks have a limited number of HMI or other control devices that should write information to a PLC. An adversary may attempt to corrupt a PLC or set in a state to negatively affect the process being controlled.</td></tr>
<tr><td>Affected Systems</td><td>DNP3 servers, such as PLC's and RTU's.</td></tr>
<tr><td>Attack Scenarios</td><td>An attacker with IP connectivity to the PLC issues DNP3 write requests. This could change the configuration of the PLC, make the PLC inoperable, send requests to actuators to change the state of the process being controlled, or overwrite important information.</td></tr>
</table>

- Rules described in specific language
- For demo: STARQL used on level above IDS
- Possible extension: STARQL implements rules (CEP style)

Complex Event Processing

- One stream with point-events
- Example: Recognize shipment chain of contaminated products

Example: Contamination (Agrawal et al 08)

```
PATTERN SEQ(Alert a, Shipment+ b[ ])
WHERE skip_till_any_match(a, b[ ])
      a.type = 'contaminated' and
      b[1].from = a.site and
      b[i].from = b[i-1].to
 WITHIN 3 hours
```

Complex Event Processing

- One stream with point-events
- Example: Recognize shipment chain of contaminated products

Example: Contamination (Agrawal et al 08)

```
PATTERN SEQ(Alert a, Shipment+ b[ ])
WHERE skip_till_any_match(a, b[ ])
      a.type = 'contaminated' and
      b[1].from = a.site and
      b[i].from = b[i-1].to
 WITHIN 3 hours
```

Complex Event Processing

- One stream with point-events
- Example: Recognize shipment chain of contaminated products

Example: Contamination (Agrawal et al 08)

```
PATTERN SEQ(Alert a, Shipment+ b[ ])
WHERE skip_till_any_match(a, b[ ])
      a.type = 'contaminated' and
      b[1].from = a.site and
      b[i].from = b[i-1].to
 WITHIN 3 hours
```

Complex Event Processing

- One stream with point-events
- Example: Recognize shipment chain of contaminated products

Example: Contamination (Agrawal et al 08)

```
PATTERN SEQ(Alert a, Shipment+ b[ ])
WHERE skip_till_any_match(a, b[ ])
      a.type = 'contaminated' and
      b[1].from = a.site and
      b[i].from = b[i-1].to
 WITHIN 3 hours
```