UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR INFORMATIONSSYSTEME

Özgür L. Özçep

Stream Processing 1

Lecture 11: Temporal OBDA, Relational Stream Processing

3 February, 2016

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

Solutions for Exercise 8

Solution for Exercise 8.1 (4 Bonus points)

Belief Revision has strong connections to Non-monotonic reasoning: For any (say consistent) belief set K one can define an entailment relation $\models_K$ as follows:

$$\alpha \models_K \beta \text{ iff } \beta \in K * \alpha$$

Answer the question whether $\models_K$ is a monotonic entailment relation, i.e., whether it fulfills:

$$\text{If } X \models_K \alpha \text{ and } Y \subseteq Y, \text{ then } Y \models_K \alpha$$

Solution: Clearly the entailment relation is non-monotonic. Consider $K = Cn(p \rightarrow q)$, $X = \{p\}$, $X' = \{p, \neg q)$. We have $X \models_K q$, but not $X' \models_K q$.

Exercise 8.2 (4 Bonus points)

An alleged weakness of AGM belief revision is dealt under the term "Ramsey Test". Inform yourself on this test and explain how it works.

Solution: Define counterfactual conditionals $\alpha \triangleright \beta$ using the above entailment relation. The Ramsey test gives an acceptability criterion for the acceptance of counterfactual condition stating: counterfactual $\alpha \triangleright \beta$ is accepted in K iff β belongs to revision result with α. If the language in which the belief sets and the triggers are described contains a connective for the counterfactual—i.e. if the counterfactual is part of the object language, then the Ramsey test reads as

$$\alpha \triangleright \beta \in K \text{ iff } \beta \in K * \alpha$$

Gärdenfors showed that in this case there cannot be a non-trivial AGM belief revision fulfilling the Ramsey test (because such a revision operator would be monotonic in the left argument).

Exercise 8.3 (4 Bonus Points)

Consider the following postulate for belief bases B:

(R) If $\beta \in B$ and $\beta \notin B * \alpha$, then there is some B' with

1. $B * \alpha \subseteq B' \subseteq B \cup \{\alpha\}$
2. B' is consistent
3. $B' \cup \{\beta\}$ is inconsistent

First describe this postulates in natural language. What would be a good name for this postulate (which was invented following a criticisms of AGM revision)?

Solution: If a sentence (β) does not survive the revision, then this is because it would lead to an inconsistency with a consistent subset of the belief base and the trigger. This says that only sentences of the belief base that are **relevant** for the (inconsistency with the) trigger, are allowed to be eliminated.

Recap of/Continuing Lecture 10

Ontology Change

- Considered ontology change from BR perspective
- Required adaptations and extensions for BR
  - non-classical logics
  - revision of finite belief bases
  - multiple revision
  - iterated revision

Infinite Iteration and Learning

Formal Learning Theory for Infinite Revision

- Iterable revision operators applied to potentially infinite sequence of triggers
- Define principles (postulates) that describe adequate behaviour
- The minimality ideas and relevant principles of BR not sufficient
- Let you guide by principles of inductive learning and **formal learning theory**
- Indeed, we need good principles for induction :)

  http://www.der-postillon.com/2015/10/autofahrer-entlarvt-geheimen.html

The Scientist-Nature-Scenario

- Class of possible worlds (one of them the real world = nature)
- Scientist has to answer queries regarding the real world
- He gets stream of data compatible with the real world
- Conjectures according to some strategy at every new arrival of trigger a hypothesis on the correct answer
- Success: Sequence of answers stabilizes to a correct hypothesis.

The Scientist-Nature-Scenario for Orders

- Class of possible worlds
- Scientist answers query regarding the real world (problem)
- He gets stream of data compatible with the real world
- Conjectures according to some strategy at every new arrival of trigger a hypothesis on the correct answer
- Success: Sequence of answers stabilizes to a correct hypothesis.

Example (Component of Order Example)

Strict orders $<$ on $\mathbb{N}$

- 0,1,2,3, ...
- 1,0,2,3, ...
- ...3,2,1,0
- 0,2,4,6, ..., 1,3,5,7

The Scientist-Nature-Scenario for Orders

- Class of possible worlds
- Scientist answers query regarding the real world (problem)
- He gets stream of data compatible with the real world
- Conjectures according to some strategy at every new arrival of trigger a hypothesis on the correct answer
- Success: Sequence of answers stabilizes to a correct hypothesis.

Example (Component of Order Example)

Stream of dat made up by facts (called environments)

- R(2,3), R(1,2), R(0,2), R(1,4) ...
  (for world: 0,1,2,3, ...)
- R(4,3), R(5,2), ...
  (for world: ...3,2,1,0)

The Scientist-Nature-Scenario for Orders

- Class of possible worlds
- Scientist answers query regarding the real world (problem)
- He gets stream of data compatible with the real world
- Conjectures according to some strategy at every new arrival of trigger a hypothesis on the correct answer
- Success: Sequence of answers stabilizes to a correct hypothesis.

Example (Component of Order Example)

Problem set: orders isomorphic to $\omega \cup \omega^*$

- 0,1,2,3, $\ldots$ is isomorphic to ω
- $\ldots$ 3,2,1,0 is isomorphic to ω^*.
- Problem query: Has order a least element?

The Scientist-Nature-Scenario for Orders

- Class of possible worlds
- Scientist answers query regarding the real world (problem)
- He gets stream of data compatible with the real world
- Conjectures according to some strategy at every new arrival of trigger a hypothesis on the correct answer
- Success: Sequence of answers stabilizes to a correct hypothesis.

Example (Component of Order Example)

Scientist solves problem P iff for every $<\in P$ and every environment e:

- If $<$ has least element, then cofinitely often scientist says yes on $e(n)$ (on n-prefix of environment)
- If $<$ has no least element, then for cofinitely many n scientist says no on $e(n)$

The Scientist-Nature-Scenario for Orders

- Class of possible worlds
- Scientist answers query regarding the real world (problem)
- He gets stream of data compatible with the real world
- Conjectures according to some strategy at every new arrival of trigger a hypothesis on the correct answer
- Success: Sequence of answers stabilizes to a correct hypothesis.

Example (Component of Order Example)

Problem $P = \{<\in \omega \cup \omega^* \mid < \text{has least element}\}$ is solvable

- Consider L-score: For any finite sequence it is the smallest number not occurring in right argument of R
- G-score: smallest number not occurring in first argument of R
- Scientist: If L-score lower than G-score on given prefix, say yes, otherwise no.

Choosing Revision as Strategy

- Kelly investigates learning based on various revision operators defined for epistemic states
- Hypotheses = sentences in the belief sets
- Main (negative) result in (Kelly 98)

Theorem

*Revision operators implementing a minimal (one-step) revision suffer from **inductive amnesia**: If and only if some of the past is forgotten, stabilization is guaranteed.*

Lit: K. T. Kelly. Iterated belief revision, reliability, and inductive amnesia. Erkenntnis, 50:11–58, 1998.

Stabilization for Ontology Learning

Example (Book Shopping Agent)

$$\begin{array}{lcll} O_{rec} & \models & \mathit{cheap} \equiv \mathit{costs} < 5\$, & \neg \mathit{costs} < 5\$(\text{`}\mathit{Faust}') \\ O_{send} & \models & \mathit{cheap} \equiv \mathit{costs} < 6\$, & \mathit{costs} < 6\$(\text{`}\mathit{Faust}') \end{array}$$

- Receiver: "List all cheap books by Goethe"
- Sender stream: $\alpha_1 = \mathit{cheap}(\text{`}\mathit{Faust}')$ $, \alpha_2, \alpha_3, \ldots$
- Integrating stream elements by revision operator $\circ$ gives Output stream $(O^i_{rec})_{i \in \mathbb{N}}$:

$$(O_{rec},\ O_{rec} \circ \alpha_1,\ (O_{rec} \circ \alpha_1) \circ \alpha_2,\ \ldots)$$

Stabilization for (Amnesic) Ontology Learning

- Properties of $(O^i_{rec})_{i\in\mathbb{N}}$ depend on $\circ$
- Special Case: $\circ =$ weak type-2 operator (forgets quite a lof of from "old ontology")
  - Prioritize incoming terminology
  - Simple mappings for disambiguation
    Example: $cheap_{rec} \sqsubseteq cheap_{send}$, $cheap \equiv cheap_{send}$

Theorem (Eschenbach & Ö., 2011)

For a (internally consistent) stream of atomic assertions the output streams of ontologies produced with weak type-2 operator stabilizes.

Lit: Eschenbach and Ö. Ontology revision based on reinterpretation. Logic Journal oft the IGPL, 18(4):579?616, 2010.

Stabilization for (Amnesic) Ontology Learning

- Properties of $(O^i_{rec})_{i \in \mathbb{N}}$ depend on $\circ$
- Special Case: $\circ =$ weak type-2 operator (forgets quite a lof of from "old ontology")
  - Prioritize incoming terminology
  - Simple mappings for disambiguation
    Example: $cheap_{rec} \sqsubseteq cheap_{send}$, $cheap \equiv cheap_{send}$

Theorem (Eschenbach & Ö., 2011)

For a (internally consistent) stream of atomic assertions the output streams of ontologies produced with weak type-2 operator stabilizes.

Lit: Eschenbach and Ö. Ontology revision based on reinterpretation. Logic Journal oft the IGPL, 18(4):579?616, 2010.

Non-Stabilization for (Non-Amnesic) Ontology Learning

- Special Case: $\circ$ = strong type-2 operator (remembers "old ontology")
  - Prioritize incoming terminology
  - Advanced mappings for disambiguation
    Example: $cheap_{rec} \sqsubseteq cheap_{send}$,
    $cheap_{send} \sqsubseteq cheap_{rec} \sqcup DifferConcept_{rec,send}$, $cheap \equiv cheap_{send}$

Theorem (Eschenbach & Ö., 2011)

*There is an ontology and a (internally consistent) stream of atomic assertions s.t. the output stream of ontologies produced with the strong type-2 operator does **not** stabilize.*

Lit: Eschenbach and Ö. Ontology revision based on reinterpretation. Logic Journal oft the IGPL, 18(4):579?616, 2010.

Non-Stabilization for (Non-Amnesic) Ontology Learning

- Special Case: $\circ$ = strong type-2 operator (remembers "old ontology")
  - Prioritize incoming terminology
  - Advanced mappings for disambiguation
    Example: $\mathit{cheap}_{\mathit{rec}} \sqsubseteq \mathit{cheap}_{\mathit{send}}$,
    $\mathit{cheap}_{\mathit{send}} \sqsubseteq \mathit{cheap}_{\mathit{rec}} \sqcup \mathit{DifferConcept}_{\mathit{rec},\mathit{send}}$, $\mathit{cheap} \equiv \mathit{cheap}_{\mathit{send}}$

Theorem (Eschenbach & Ö., 2011)

*There is an ontology and a (internally consistent) stream of atomic assertions s.t. the output stream of ontologies produced with the strong type-2 operator does **not** stabilize.*

Lit: Eschenbach and Ö. Ontology revision based on reinterpretation. Logic Journal oft the IGPL, 18(4):579?616, 2010.

Choosing Revision as Strategy

- Martin/Osherson investigate learning based revision operators defined for finite sequences
- So their revision operators have always the whole history within the trigger
- This leads to positive results

Theorem

Revision operators provide ideal learning strategies:There is a revision operator a scientist can use to solve every (solvable) problem.

Lit: E. Martin and D. Osherson. Scientific discovery based on belief revision. Journal of Symbolic Logic, 62(4):1352–1370, 1997.

Next Slides

- Infinite sequence from stream processing perspective
  - Additional aspects: temporality of data, recency, data-driveness, velocity

- Resume OBDA and consider how to lift them to temporal OBDA and streaming OBDA
  - Temporal OBDA: Add time aspect (somewhere)
  - Stream OBDA: Higher-level stream w.r.t. ontology (and mappings)

Temporalized OBDA

A Confession

- Ontology-Based Data Access on temporal and **Streaming** Data
- But: Streams are temporal streams and we talk about local temporal reasoning

Adding a Temporal Dimension to OBDA

- Most conservative strategy: handle time as "ordinary" attribute *time*

$$\left\{ \begin{array}{c} \mathit{meas}(x) \wedge \\ \mathit{val}(x, y) \wedge \\ \mathit{time}(x, z) \end{array} \right\} \longleftarrow$$

```
SELECT f(MID) AS m, Mval AS y, MtimeStamp AS z
FROM MEASUREMENT
```

- Classical Mapping
- Pro: Minimal (no) adaptation
- Contra:
  - No control on "logic of time"
  - Need reification
    - sometimes necessary (because DLs provided only predicates up to arity 2)
    - but not that "natural"

Flow of Time

- Flow of time $(T, \leq_T)$ is a structure with a time domain T and a binary relation $\leq_T$ over it.
- Flow metaphor hints on directionality and dynamic aspect of time
- But still different forms of flow are possible

- One can consider concrete structures of flow of (time), as done here
- Or investigate them additionally axiomatically
- An early model-theoretic and axiomatic treatise:

  Lit: J. van Benthem. The Logic of Time: A Model-Theoretic Investigation into the Varieties of Temporal Ontology and Temporal Discourse. Reidel, 2. edition, 1991.

The Family of Flows of Time

- Domain T
  - points (atomic time instances)
  - pairs of points (application time, transaction time)
  - intervals etc.
- Properties of the time relation $\leq_T$
  - Non-branching (linear) vs. branching
    Linearity:
    - reflexive: $\forall t \in T$: $t \leq_T t$
    - antisymmetric: $\forall t_1, t_2 \in T$: $(t_1 \leq t_2 \land t_2 \leq_T t_1) \Rightarrow t_1 = t_2$
    - transitive: $\forall t_1, t_2, t_3 \in T : (t_1 \leq_T t_2 \land t_2 \leq t_3) \Rightarrow t_1 \leq t_3$.
    - total: $\forall t_1, t_2 \in T$: $t_1 \leq t_2 \lor t_2 \leq t_1 \lor t_1 = t_2$.
- Existence of first or last element
- discreteness (Example: $T = \mathbb{N}$); also used for modeling state sequences;
- density (Example: $T = \mathbb{Q}$);
- continuity (Example: $T = \mathbb{R}$)

The Family of Flows of Time

- Domain T
  - points (atomic time instances)
  - pairs of points (application time, transaction time)
  - intervals etc.
- Properties of the time relation $\leq_T$
  - Non-branching (linear) vs. branching
    Linearity:
    - reflexive: $\forall t \in T$: $t \leq_T t$
    - antisymmetric: $\forall t_1, t_2 \in T$: $(t_1 \leq t_2 \wedge t_2 \leq_T t_1) \Rightarrow t_1 = t_2$
    - transitive: $\forall t_1, t_2, t_3 \in T : (t_1 \leq_T t_2 \wedge t_2 \leq t_3) \Rightarrow t_1 \leq t_3$.
    - total: $\forall t_1, t_2 \in T$: $t_1 \leq t_2 \vee t_2 \leq t_1 \vee t_1 = t_2$.
- Existence of first or last element
- discreteness (Example: $T = \mathbb{N}$); also used for modeling state sequences;
- density (Example: $T = \mathbb{Q}$);
- continuity (Example: $T = \mathbb{R}$)

The Family of Flows of Time

- Domain T
  - points (atomic time instances)
  - pairs of points (application time, transaction time)
  - intervals etc.
- Properties of the time relation $\leq_T$
  - Non-branching (linear) vs. branching
    Linearity:
    - reflexive: $\forall t \in T$: $t \leq_T t$
    - antisymmetric: $\forall t_1, t_2 \in T$: $(t_1 \leq t_2 \wedge t_2 \leq_T t_1) \Rightarrow t_1 = t_2$
    - transitive: $\forall t_1, t_2, t_3 \in T : (t_1 \leq_T t_2 \wedge t_2 \leq t_3) \Rightarrow t_1 \leq t_3$.
    - total: $\forall t_1, t_2 \in T$: $t_1 \leq t_2 \vee t_2 \leq t_1 \vee t_1 = t_2$.
- Existence of first or last element
- discreteness (Example: $T = \mathbb{N}$); also used for modeling state sequences;
- density (Example: $T = \mathbb{Q}$);
- continuity (Example: $T = \mathbb{R}$)

The Family of Flows of Time

- Domain T
  - points (atomic time instances)
  - pairs of points (application time, transaction time)
  - intervals etc.
- Properties of the time relation $\leq_T$
  - Non-branching (linear) vs. branching
    Linearity:
    - reflexive: $\forall t \in T$: $t \leq_T t$
    - antisymmetric: $\forall t_1, t_2 \in T$: $(t_1 \leq t_2 \wedge t_2 \leq_T t_1) \Rightarrow t_1 = t_2$
    - transitive: $\forall t_1, t_2, t_3 \in T$: $(t_1 \leq_T t_2 \wedge t_2 \leq t_3) \Rightarrow t_1 \leq t_3$.
    - total: $\forall t_1, t_2 \in T$: $t_1 \leq t_2 \vee t_2 \leq t_1 \vee t_1 = t_2$.
- Existence of first or last element
- discreteness (Example: $T = \mathbb{N}$); also used for modeling state sequences;
- density (Example: $T = \mathbb{Q}$);
- continuity (Example: $T = \mathbb{R}$)

Temporalized OBDA: General Approach

- Semantics rests on family of interpretations $(\mathcal{I}_t)_{t \in T}$
- Temporal ABox $\tilde{A}$: Finite set of T-tagged ABox axioms

Example

$val(s_0, 90°)\langle 3s \rangle$ holds in $(\mathcal{I}_t)_{t \in T}$ iff $\mathcal{I}_{3s} \models val(s_0, 90°)$
"sensor s_0 has value 90° at time point 3s"

- Alternative sequence representation of temporal ABox $\tilde{\mathcal{A}}$
  - $(\mathcal{A}_t)_{t \in T'}$ (where T' are set of timestamps in T)
  - $\mathcal{A}_t = \{ax \mid ax\langle t \rangle \in \tilde{\mathcal{A}}\}$

Definition (Adapted notion of OBDA rewriting)

$$cert(Q, (Sig, \mathcal{T}, (\mathcal{A}_t)_{t \in T'}) = ans(Q_{rew}, (DB(\mathcal{A}_t))_{t \in T'})$$

Temporalized OBDA:TCQs

- Different approaches based on modal (temporal) operators
- LTL (linear temporal logic) operators only in QL (Borgwardt et al. 13)

Example

$$
\begin{aligned}
Critical(x) &= \exists y. Turbine(x) \wedge showsMessage(x, y) \wedge \\
&\quad FailureMessage(y) \\
Q(x) &= \bigcirc^{-1} \bigcirc^{-1} \bigcirc^{-1} (\Diamond (Critical(x) \wedge \bigcirc \Diamond Critical(x)))
\end{aligned}
$$

"turbine has been at least two times in a critical situation in the last three time units"

- CQ embedded into LTL template
- Special operators taking care of endpoints of state sequencing
- Not well-suited for OBDA as non-safe
- Rewriting simple due to atemporal TBox

Lit: S. Borgwardt, M. Lippmann, and V. Thost. Temporal query answering in the

Temporalized OBDA: TQL

- LTL operators in TBox and T argument in QL

Example

TBox axiom : $\mathit{showsAnomaly} \sqsubseteq \Diamond \mathit{UnplanedShutDown}$
"if turbine shows anomaly (now)
then sometime in the future it will shut down"

Query : $\exists t.3s \leq t \leq 6s \wedge \mathit{showsAnomaly}(x, t)$

- Can formulate rigidity assumptions
- Rewriting not trivial

Lit: A. Artale, R. Kontchakov, F. Wolter, and M. Zakharyaschev. Temporal description logic for ontology- based data access. In IJCAI'13, pages 711–717. AAAI Press, 2013.

Stream Basics

Streams

Definition (Stream)

A stream S is a potentially infinite sequence of objects d over some domain D.

- "Streams are forever"

  Lit: J. Endrullis, D. Hendriks, and J. W. Klop. Streams are forever. Bulletin of the EATCS, 109:70–106, 2013.

- "Order matters!"

  Lit: E. D. Valle et al. Order matters! harnessing a world of orderings for reasoning over massive data. Semantic Web, 4(2):219–231, 2013.

- "It's a streaming world!"

  Lit: E. Della Valle, et al. It?s a streaming world! Reasoning upon rapidly changing information. Intelligent Systems, IEEE, 24(6):83–89, nov.-dec. 2009.

Streams

Definition (Stream)

A stream S is a potentially infinite sequence of objects d over some domain D.

- "Streams are forever"

  Lit: J. Endrullis, D. Hendriks, and J. W. Klop. Streams are forever. Bulletin of the EATCS, 109:70–106, 2013.

- "Order matters!"

  Lit: E. D. Valle et al. Order matters! harnessing a world of orderings for reasoning over massive data. Semantic Web, 4(2):219–231, 2013.

- "It's a streaming world!"

  Lit: E. Della Valle, et al. It?s a streaming world! Reasoning upon rapidly changing information. Intelligent Systems, IEEE, 24(6):83–89, nov.-dec. 2009.

Streams

Definition (Stream)

A stream S is a potentially infinite sequence of objects d over some domain D.

- "Streams are forever"

  Lit: J. Endrullis, D. Hendriks, and J. W. Klop. Streams are forever. Bulletin of the EATCS, 109:70–106, 2013.

- "Order matters!"

  Lit: E. D. Valle et al. Order matters! harnessing a world of orderings for reasoning over massive data. Semantic Web, 4(2):219–231, 2013.

- "It's a streaming world!"

  Lit: E. Della Valle, et al. It?s a streaming world! Reasoning upon rapidly changing information. Intelligent Systems, IEEE, 24(6):83–89, nov.-dec. 2009.

Streams

Definition (Stream)

A stream S is a potentially infinite sequence of objects d over some domain D.

- "Streams are forever"

  Lit: J. Endrullis, D. Hendriks, and J. W. Klop. Streams are forever. Bulletin of the EATCS, 109:70–106, 2013.

- "Order matters!"

  Lit: E. D. Valle et al. Order matters! harnessing a world of orderings for reasoning over massive data. Semantic Web, 4(2):219–231, 2013.

- "It's a streaming world!"

  Lit: E. Della Valle, et al. It?s a streaming world! Reasoning upon rapidly changing information. Intelligent Systems, IEEE, 24(6):83–89, nov.-dec. 2009.

Adding a Time Dimension

Definition (Temporal Stream)

A temporal stream S is a potentially infinite sequence of timestamped objects $d\langle t\rangle$ over some domain D and flow of time $(T, \leq_T)$.

- Consider non-branching (or: linear) time, i.e., $\leq_T$ is
- We assume that there is no last element in T
- We do not restrict T further, so it may be
  - discrete or
  - dense or
  - continuous

Adding a Time Dimension

Definition (Temporal Stream)

A temporal stream S is a potentially infinite sequence of timestamped objects $d\langle t\rangle$ over some domain D and flow of time $(T, \leq_T)$.

- Consider non-branching (or: linear) time, i.e., $\leq_T$ is
- We assume that there is no last element in T
- We do not restrict T further, so it may be
  - discrete or
  - dense or
  - continuous

Arrival Ordering

- Sequence fixed by arrival ordering fixed $<_{ar}$
- $<_{ar}$ is a strict total ordering on the elements of S
- Synchronous streams: $\leq_T$ compatible with $<_{ar}$
- Compatibility: For all $d_1\langle t_1\rangle, d_2\langle t_2\rangle \in S$: If $d_1\langle t_1\rangle <_{ar} d_2\langle t_2\rangle$, then $t_1 \leq_T t_2$.
- Asynchronous streams: $\leq_T$ not necessarily compatible with $<_{ar}$

Convention for the following

- Consider only temporal streams
- Consider only synchronous streams $\Longrightarrow$ neglect $<_{ar}$.
- Represent streams as a potentially infinite multi-set (bag) of elements

Arrival Ordering

- Sequence fixed by arrival ordering fixed $<_{ar}$
- $<_{ar}$ is a strict total ordering on the elements of S
- Synchronuous streams: $\leq_T$ compatible with $<_{ar}$
- Compatibility: For all $d_1\langle t_1\rangle, d_2\langle t_2\rangle \in S$: If $d_1\langle t_1\rangle <_{ar} d_2\langle t_2\rangle$, then $t_1 \leq_T t_2$.
- Asynchronous streams: $\leq_T$ not necessarily compatible with $<_{ar}$

Convention for the following

- Consider only temporal streams
- Consider only synchronous streams $\Longrightarrow$ neglect $<_{ar}$.
- Represent streams as a potentially infinite multi-set (bag) of elements

Arrival Ordering

- Sequence fixed by arrival ordering fixed $<_{ar}$
- $<_{ar}$ is a strict total ordering on the elements of S
- Synchronuous streams: $\leq_T$ compatible with $<_{ar}$
- Compatibility: For all $d_1\langle t_1\rangle, d_2\langle t_2\rangle \in S$: If $d_1\langle t_1\rangle <_{ar} d_2\langle t_2\rangle$, then $t_1 \leq_T t_2$.
- Asynchronous streams: $\leq_T$ not necessarily compatible with $<_{ar}$

Convention for the following

- Consider only temporal streams
- Consider only synchronous streams $\Longrightarrow$ neglect $<_{ar}$.
- Represent streams as a potentially infinite multi-set (bag) of elements

Stream Stack and Stream Research

- **Low-level sensor perspective** (semantic sensor networks)
  - Develop fast algorithms on high-frequency streams with minimal space consumption
  - Considers approximate algorithms for aggregation functions
  - See lecture "Non-standard DBs" by Ralf Möller

- **Data stream management system (DSMS)** perspective
  - Provide whole DB systems for streams of structured (relational) data
  - Deals with all aspects relevant in static DBMS adapted to stream scenario
  - See lecture "Non-standard DBs" by Ralf Möller and this lecture
  - Stream Query Language

- **High-level and Ontology layer streams**
  - Processing stream of assertions (RDF triples) w.r.t. an ontology
  - Related: Complex Event Processing (CEP)
  - this and next lecture

Stream Stack and Stream Research

- **Low-level sensor perspective** (semantic sensor networks)
  - Develop fast algorithms on high-frequency streams with minimal space consumption
  - Considers approximate algorithms for aggregation functions
  - See lecture "Non-standard DBs" by Ralf Möller
- **Data stream management system (DSMS)** perspective
  - Provide whole DB systems for streams of structured (relational) data
  - Deals with all aspects relevant in static DBMS adapted to stream scenario
  - See lecture "Non-standard DBs" by Ralf Möller and this lecture
  - Stream Query Language
- **High-level and Ontology layer streams**
  - Processing stream of assertions (RDF triples) w.r.t. an ontology
  - Related: Complex Event Processing (CEP)
  - this and next lecture

Stream Stack and Stream Research

- **Low-level sensor perspective** (semantic sensor networks)
  - Develop fast algorithms on high-frequency streams with minimal space consumption
  - Considers approximate algorithms for aggregation functions
  - See lecture "Non-standard DBs" by Ralf Möller
- **Data stream management system (DSMS)** perspective
  - Provide whole DB systems for streams of structured (relational) data
  - Deals with all aspects relevant in static DBMS adapted to stream scenario
  - See lecture "Non-standard DBs" by Ralf Möller and this lecture
  - Stream Query Language
- **High-level and Ontology layer streams**
  - Processing stream of assertions (RDF triples) w.r.t. an ontology
  - Related: Complex Event Processing (CEP)
  - this and next lecture

Local vs. global stream proessing

- **Global aim: Learn** about the whole by looking at the parts
  - Examples: inductive learning, ontology change, iterated belief revision (see slides before), robotics oriented stream processing with plan generation
  - May produce also an output stream
  - But in the end the whole stream counts
- **Local aim: Monitor** window contents with time-local
  - Examples: Real-time monitoring, simulation for reactive diagnostics
- Categories not exclusive
  - In learning one applies operation on (NOW)-window to learn about stream
  - In predictive analytics one monitors with wndow in order to predcit upcoming events

Domain Objects for Streams

Definition (Temporal Stream)

A stream S is a sequence of timestamped objects $d\langle t\rangle$ over some domain D and flow of time $(T, \leq_T)$.

Streamified OBDA has to deal with **different types of domains**

Domain Objects for Streams

Definition (Temporal Stream)

A stream S is a sequence of timestamped objects $d\langle t\rangle$ over some domain D and flow of time $(T, \leq_T)$.

Streamified OBDA has to deal with **different types of domains**

$D =$ a set of **typed relational tuples** adhering to a relational schema

- Streams at the backend sources
- $S_{rel} = \{(s_1, 90^\circ)\langle 1s\rangle, (s_2, 92^\circ)\langle 2s\rangle, (s_1, 94^\circ)\langle 3s\rangle, \dots\}$
- Schema: hasSensorRelation(Sensor:string, temperature:integer)

Domain Objects for Streams

Definition (Temporal Stream)

A stream S is a sequence of timestamped objects $d\langle t\rangle$ over some domain D and flow of time $(T, \leq_T)$.

Streamified OBDA has to deal with **different types of domains**

$D =$ set of **untyped tuples** (of the same arity)

- Stream of tuples resulting as bindings for subqueries

Domain Objects for Streams

Definition (Temporal Stream)

A stream S is a sequence of timestamped objects $d\langle t\rangle$ over some domain D and flow of time $(T, \leq_T)$.

Streamified OBDA has to deal with **different types of domains**

$D =$ set of **assertions (RDF tuples)**.

- $S_{rdf} = \{\ val(s_0, 90^\circ)\langle 1s\rangle, val(s_2, 92^\circ)\langle 2s\rangle,\ val(s_1, 94^\circ)\langle 3s\rangle, \ldots\ \}$

Domain Objects for Streams

Definition (Temporal Stream)

A stream S is a sequence of timestamped objects $d\langle t\rangle$ over some domain D and flow of time $(T, \leq_T)$.

Streamified OBDA has to deal with **different types of domains**

$D =$ set of **RDF graphs**

Taming the Infinite

Nearly all stream provide a fundamental means to cope with potential infinity of streams, namely ...

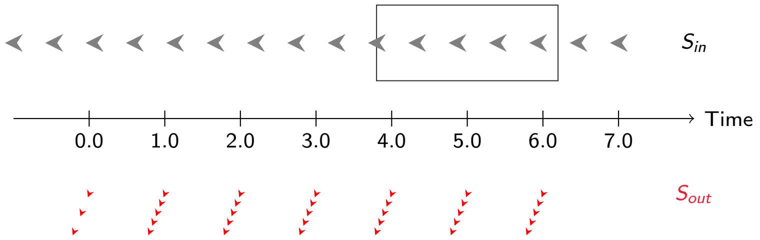
Taming the Infinite

Nearly all stream provide a fundamental means to cope with potential infinity of streams, namely ...

- Stream query continuous, not one-shot activity
- Window content continuosly updated

Taming the Infinite

Nearly all stream provide a fundamental means to cope with potential infinity of streams, namely ...



- Here a time-based window of width 3 seconds
- and slide 1 second is applied

Window Operators: Classification

- Direction of movement of the endpoints
  - Both endpoints fixed (needed for "historical" queries)
  - Both moving/sliding
  - One moving the other not
- Window size
  - Temporal
  - Tuple-based
  - Partitioned window
  - Predicate window
- Window update
  - tumbling
  - sampling
  - overlapping

Window Operators: Classification

- Direction of movement of the endpoints
  - Both endpoints fixed (needed for “historical” queries)
  - Both moving/sliding
  - One moving the other not

- Window size
  - Temporal
  - Tuple-based
  - Partitioned window
  - Predicate window

- Window update
  - tumbling
  - sampling
  - overlapping

Window Operators: Classification

- Direction of movement of the endpoints
  - Both endpoints fixed (needed for “historical” queries)
  - Both moving/sliding
  - One moving the other not

- Window size
  - Temporal
  - Tuple-based
  - Partitioned window
  - Predicate window

- Window update
  - tumbling
  - sampling
  - overlapping

Relational Stream Processing with CQL

Relational Data Stream Processing

- Different groups working on DSMS around 2004
  - Academic prototypes: STREAM and CQL (Stanford); TelgraphCQ (Berkeley) (extends PostGreSQL); Aurora/Borealis (Brandeis, Brown and MIT); PIPES from Marburg University
  - Commercial systems: StreamBase, Truviso (Standalone), extensions of commercial DBMS (MySQL, PostgreSQL, DB2 etc.)
- Though well investigated and many similarities there is no streaming SQL standard
- First try for standardization:

  Lit: N. Jain et al. Towards a streaming sql standard. Proc. VLDB Endow., 1(2):1379–1390, Aug. 2008.
- But if development speed similar to that for introducing temporal dimension into SQL, then we have to wait...

Relational Data Stream Processing

- Different groups working on DSMS around 2004
  - Academic prototypes: STREAM and CQL (Stanford); TelgraphCQ (Berkeley) (extends PostGreSQL); Aurora/Borealis (Brandeis, Brown and MIT); PIPES from Marburg University
  - Commercial systems: StreamBase, Truviso (Standalone), extensions of commercial DBMS (MySQL, PostgreSQL, DB2 etc.)

- Though well investigated and many similarities there is no streaming SQL standard
- First try for standardization:

  Lit: N. Jain et al. Towards a streaming sql standard. Proc. VLDB Endow., 1(2):1379–1390, Aug. 2008.
- But if development speed similar to that for introducing temporal dimension into SQL, then we have to wait...

CQL (Continuous Query Language)

- Early relational stream query language extending SQL
- Developed in Stanford as part of a DSMS called STREAM
- Semantics theoretically specified by denotational semantics
- Practically, development of CQL was accompanied by the development the Linear Road Benchmark (LRB) (`http://www.cs.brandeis.edu/~linearroad/`)
- Had immense impact also on development of early RDF streaming engines in RSP community `https://www.w3.org/community/rsp/`)

Lit: A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: semantic foundations and query execution. The VLDB Journal, 15:121–142, 2006.
Lit: A. Arasu et al. Linear road: A stream data management benchmark. In VLDB, pages 480–491, 2004.

CQL Operators

- Special data structure next to streams: relations R
  - R maps times t to ordinary (instantaneous) relations $R(t)$
  - Motivation: Use of ordinary SQL operators on instantaneous relations
- Operators
  - Stream-to-relation = window operator
  - Relation-to-relation = standard SQL operators at every single time point
  - relation-stream = for getting streams agains
- Non-predictability condition for operators *op*:
  - If two inputs S_1 , S_2 are the same up to t, then $op(S_1)(t) = op(S_2)(t)$.

CQL Windows

- Window operators are stream-to-relation operators
- CQL knows tuple-based, partition based, and time-based windows

Definition (Semantics of Window Operator)

R = S [Range wr Slide sl]

- with slide parameter sl and range wr
- $t_{start} = \lfloor t/sl \rfloor \cdot sl$
- $t_{end} = max\{t_{start} - wr, 0\}$

$$R(t) = \begin{cases} \emptyset & \text{if } t < sl \\ \{s \mid s\langle t' \rangle \in S \text{ and } t_{end} \leq t' \leq t_{start}\} & \text{else} \end{cases}$$

- Standard slide = 1: [RANGE wr]
- Left end fixed: [Range UNBOUND]
- Width 0: [NOW]

CQL Windows

- Window operators are stream-to-relation operators
- CQL knows tuple-based, partition based, and time-based windows

Definition (Semantics of Window Operator)

R = S [Range wr Slide sl]

- with slide parameter sl and range wr
- $t_{start} = \lfloor t/sl \rfloor \cdot sl$
- $t_{end} = max\{t_{start} - wr, 0\}$

$$R(t) = \begin{cases} \emptyset & \text{if } t < sl \\ \{s \mid s\langle t' \rangle \in S \text{ and } t_{end} \leq t' \leq t_{start}\} & \text{else} \end{cases}$$

- Standard slide = 1: [RANGE wr]
- Left end fixed: [Range UNBOUND]
- Width 0: [NOW]

CQL Windows

- Window operators are stream-to-relation operators
- CQL knows tuple-based, partition based, and time-based windows

Definition (Semantics of Window Operator)

R = S [Range wr Slide sl]

- with slide parameter sl and range wr
- $t_{start} = \lfloor t/sl \rfloor \cdot sl$
- $t_{end} = max\{t_{start} - wr, 0\}$

$$R(t) = \begin{cases} \emptyset & \text{if } t < sl \\ \{s \mid s\langle t' \rangle \in S \text{ and } t_{end} \leq t' \leq t_{start}\} & \text{else} \end{cases}$$

- Standard slide = 1: [RANGE wr]
- Left end fixed: [Range UNBOUND]
- Width 0: [NOW]

Sliding Window Example in CQL

- Flow of time $(\mathbb{N}, \leq)$
- Input stream

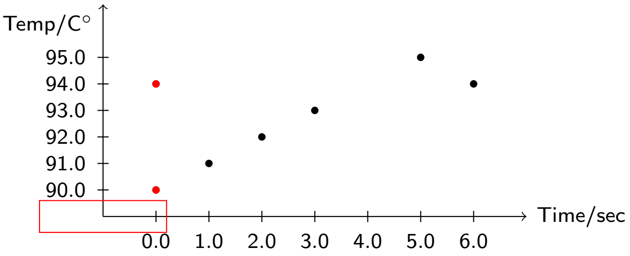
$$S \;=\; \{(s_0, 90^\circ)\langle 0\rangle, (s_1, 94^\circ)\langle 0\rangle, (s_0, 91^\circ)\langle 1\rangle, (s_0, 92^\circ)\langle 2\rangle, (s_0, 93^\circ)\langle 3\rangle, (s_0, 95^\circ)\langle 5\rangle, (s_0, 94^\circ)\langle 6\rangle \ldots\}$$

- Output relation $R =$ `S [Range 2 Slide 1]`

t :	0	1	2	3	4	5	6
$R(t)$:	$\{(s_0, 90), (s_1, 94)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91), (s_0, 92)\}$	$\{(s_0, 91), (s_0, 92), (s_0, 93)\}$	$\{(s_0, 92), (s_0, 93)\}$	$\{(s_0, 92), (s_0, 93), (s_0, 95)\}$	$\{(s_0, 93), (s_0, 95), (s_0, 94)\}$

Sliding Window Example in CQL

$S = \{(s_0, 90)\langle 0\rangle, (s_1, 94)\langle 0\rangle, (s_0, 91)\langle 1\rangle, (s_0, 92)\langle 2\rangle, (s_0, 93)\langle 3\rangle, (s_0, 95)\langle 5\rangle, (s_0, 94)\langle 6\rangle\}$

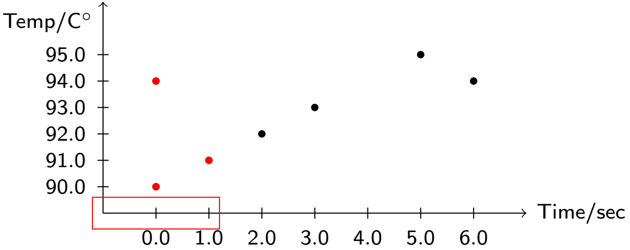


Output relation $R =$ `S [Range 2 Slide 1]`

t :	0	1	2	3	4	5	6
$R(t)$:	$\{(s_0, 90), (s_1, 94)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91), (s_0, 92)\}$	$\{(s_0, 91), (s_0, 92), (s_0, 93)\}$	$\{(s_0, 92), (s_0, 93)\}$	$\{(s_0, 93), (s_0, 95)\}$	$\{(s_0, 95), (s_0, 94)\}$

Sliding Window Example in CQL

$S = \{(s_0, 90)\langle 0\rangle, (s_1, 94)\langle 0\rangle, (s_0, 91)\langle 1\rangle, (s_0, 92)\langle 2\rangle, (s_0, 93)\langle 3\rangle, (s_0, 95)\langle 5\rangle, (s_0, 94)\langle 6\rangle\}$

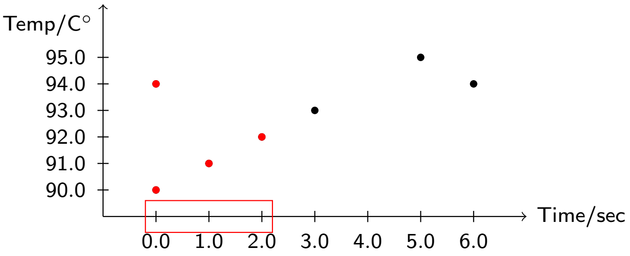


Output relation $R =$ `S [Range 2 Slide 1]`

t :	0	1	2	3	4	5	6
$R(t)$:	$\{(s_0, 90), (s_1, 94)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91), (s_0, 92)\}$	$\{(s_0, 91), (s_0, 92), (s_0, 93)\}$	$\{(s_0, 92), (s_0, 93)\}$	$\{(s_0, 93), (s_0, 95)\}$	$\{(s_0, 95), (s_0, 94)\}$

Sliding Window Example in CQL

$S = \{(s_0, 90)\langle 0\rangle, (s_1, 94)\langle 0\rangle, (s_0, 91)\langle 1\rangle, (s_0, 92)\langle 2\rangle, (s_0, 93)\langle 3\rangle, (s_0, 95)\langle 5\rangle, (s_0, 94)\langle 6\rangle\}$

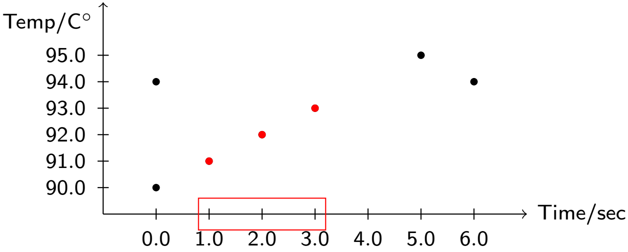


Output relation $R =$ `S [Range 2 Slide 1]`

t :	0	1	2	3	4	5	6
$R(t)$:	$\{(s_0, 90), (s_1, 94)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91), (s_0, 92)\}$	$\{(s_0, 91), (s_0, 92), (s_0, 93)\}$	$\{(s_0, 92), (s_0, 93)\}$	$\{(s_0, 93), (s_0, 95)\}$	$\{(s_0, 95), (s_0, 94)\}$

Sliding Window Example in CQL

$$S = \{(s_0, 90)\langle 0\rangle, (s_1, 94)\langle 0\rangle, (s_0, 91)\langle 1\rangle, (s_0, 92)\langle 2\rangle, (s_0, 93)\langle 3\rangle, (s_0, 95)\langle 5\rangle, (s_0, 94)\langle 6\rangle\}$$

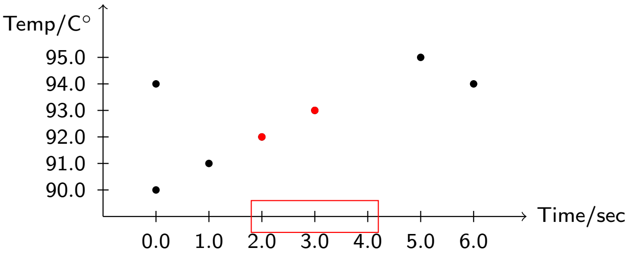


Output relation $R =$ `S [Range 2 Slide 1]`

t :	0	1	2	3	4	5	6
$R(t)$:	$\{(s_0, 90), (s_1, 94)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91), (s_0, 92)\}$	$\{(s_0, 91), (s_0, 92), (s_0, 93)\}$	$\{(s_0, 92), (s_0, 93)\}$	$\{(s_0, 93), (s_0, 95)\}$	$\{(s_0, 95), (s_0, 94)\}$

Sliding Window Example in CQL

$$S = \{(s_0, 90)\langle 0\rangle, (s_1, 94)\langle 0\rangle, (s_0, 91)\langle 1\rangle, (s_0, 92)\langle 2\rangle, (s_0, 93)\langle 3\rangle, (s_0, 95)\langle 5\rangle, (s_0, 94)\langle 6\rangle\}$$

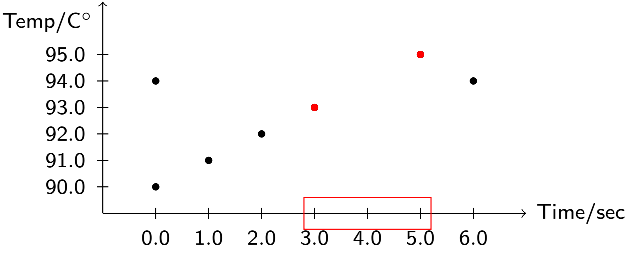


Output relation $R =$ `S [Range 2 Slide 1]`

t :	0	1	2	3	4	5	6
$R(t)$:	$\{(s_0, 90), (s_1, 94)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91), (s_0, 92)\}$	$\{(s_0, 91), (s_0, 92), (s_0, 93)\}$	$\{(s_0, 92), (s_0, 93)\}$	$\{(s_0, 93), (s_0, 95)\}$	$\{(s_0, 95), (s_0, 94)\}$

Sliding Window Example in CQL

$S = \{(s_0, 90)\langle 0\rangle, (s_1, 94)\langle 0\rangle, (s_0, 91)\langle 1\rangle, (s_0, 92)\langle 2\rangle, (s_0, 93)\langle 3\rangle, (s_0, 95)\langle 5\rangle, (s_0, 94)\langle 6\rangle\}$

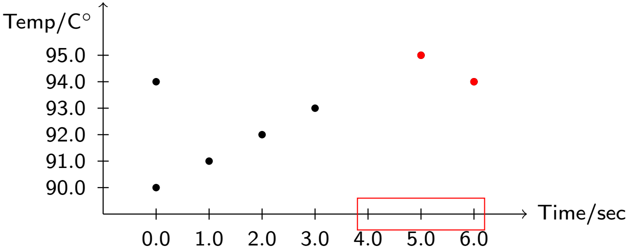


Output relation R = `S [Range 2 Slide 1]`

t :	0	1	2	3	4	5	6
$R(t)$:	$\{(s_0, 90), (s_1, 94)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91), (s_0, 92)\}$	$\{(s_0, 91), (s_0, 92), (s_0, 93)\}$	$\{(s_0, 92), (s_0, 93)\}$	$\{(s_0, 93), (s_0, 95)\}$	$\{(s_0, 95), (s_0, 94)\}$

Sliding Window Example in CQL

$S = \{(s_0, 90)\langle 0\rangle, (s_1, 94)\langle 0\rangle, (s_0, 91)\langle 1\rangle, (s_0, 92)\langle 2\rangle, (s_0, 93)\langle 3\rangle, (s_0, 95)\langle 5\rangle, (s_0, 94)\langle 6\rangle\}$



Output relation $R =$ `S [Range 2 Slide 1]`

t :	0	1	2	3	4	5	6
$R(t)$:	$\{(s_0, 90), (s_1, 94)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91), (s_0, 92)\}$	$\{(s_0, 91), (s_0, 92), (s_0, 93)\}$	$\{(s_0, 92), (s_0, 93)\}$	$\{(s_0, 93), (s_0, 95)\}$	$\{(s_0, 95), (s_0, 94)\}$

Relation vs. Stream

$$S = \{(s_0, 90^\circ)\langle 0\rangle, (s_1, 94^\circ)\langle 0\rangle, (s_0, 91^\circ)\langle 1\rangle, (s_0, 92^\circ)\langle 2\rangle, (s_0, 93^\circ)\langle 3\rangle, (s_0, 95^\circ)\langle 5\rangle, (s_0, 94^\circ)\langle 6\rangle \ldots\}$$

- Output relation $R =$ `S [Range 2 Slide 1]`

t :	0	1	2	3	4	5	6
$R(t)$:	$\{(s_0, 90), (s_1, 94)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91)\}$	$\{(s_0, 90), (s_1, 94), (s_0, 91), (s_0, 92)\}$	$\{(s_0, 91), (s_0, 92), (s_0, 93)\}$	$\{(s_0, 92), (s_0, 93)\}$	$\{(s_0, 92), (s_0, 93), (s_0, 95)\}$	$\{(s_0, 93), (s_0, 95), (s_0, 94)\}$

- Note that there are also entries for second 4
- Note that timestamps are lost in the bags
- Slides are local to streams and may be different over different streams

Relation-To-Stream Operators

- Output stream of input relation R:

$$
\begin{aligned}
\mathit{Istream}(R) &= \bigcup_{t \in T} (R(t) \setminus R(t-1)) \times \{t\} \\
&\quad \text{stream of inserted elements} \\
\mathit{Dstream}(R) &= \bigcup_{t \in T} (R(t-1) \setminus R(t)) \times \{t\} \\
&\quad \text{stream of deleted elements} \\
\mathit{Rstream}(R) &= \bigcup_{t \in T} R(t) \times \{t\} \\
&\quad \text{stream of all elements}
\end{aligned}
$$

- In CQL *IStream* and *DStream* are syntactic sugar

Sensor Measurement CQL Example

Example

```
SELECT Rstream(m.sensorID)
FROM Msmt[Range 1] as m, Events[Range 2] as e
WHERE m.val > 30 AND
      e.category = Alarm AND
      m.sensorID = e.sensorID
```

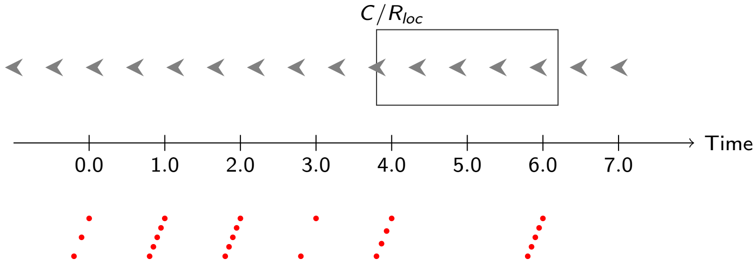
- Stream join realized by join of window contents
- Output is a stream

Non-discrete Time Flows

- Taken literally, CQL window definitions work only for discrete flows of times
- Time flow: $(T, \leq) = (\mathbb{R}, \leq)$
- Input stream: $S = \{i\langle i\rangle \mid i \in \mathbb{N}\}$
- $RStream(S[RANGE\ 1\ SLIDE\ 1])$ is "stream" with cardinality of $\mathbb{R}$
- "Solution" in CQL hidden in stream engine layer
- Heartbeat with smallest possible time granularity

High-Level Declarative Stream Processing

Local Reasoning Service



- Need to apply calculation/reasoning CR_{loc} locally, e.g.
  - arithmetics, timeseries analysis operations
  - SELECT querying, CONSTRUCT querying, abduction, revision, planning

High-Level and Declarative

- **Declarative:**
  Stream elements have "assertional status" and allow for symbolic processing

Example (Relational data streams)

Stream element $(sensor, val)\langle 3sec \rangle$ "asserts" that sensor shows some value at second 3

- **High-Level:**
  Streams are processed with respect to some background knowledge base such as a set of rules or an ontology.

Example (Streams of time-tagged ABox assertions)

Streams elements of form $val(sensor, val)\langle 3sec \rangle$ evaluated w.r.t. to an ontology containing, e.g., axiom $tempVal \sqsubseteq val$

High-Level and Declarative

- **Declarative:**
  Stream elements have "assertional status" and allow for symbolic processing

Example (Relational data streams)

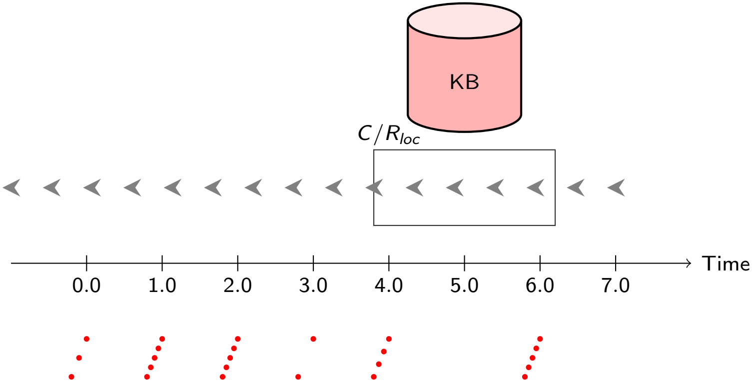
Stream element $(sensor, val)\langle 3sec\rangle$ "asserts" that sensor shows some value at second 3

- **High-Level:**
  Streams are processed with respect to some background knowledge base such as a set of rules or an ontology.

Example (Streams of time-tagged ABox assertions)

Streams elements of form $val(sensor, val)\langle 3sec\rangle$ evaluated w.r.t. to an ontology containing, e.g., axiom $tempVal \sqsubseteq val$

Local Reasoning Service



- Need to apply calculation/reasoning CR_{loc} locally, e.g.
  - arithmetics, timeseries analysis operations
  - SELECT querying, CONSTRUCT querying, abduction, revision, planning ($\Longrightarrow$ high-level & declarative)

Streamified OBDA

- Nearly ontology layer stream processing
  - CEP (Complex event processing)
  - EP-SPARQL/ETALIS, T-REX/ TESLA, Commonsens/ESPER
- RDF-ontology layer stream processing
  - C-SPARQL (della Valle et al. 09), CQELS
- Classical OBDA stream processing
  - $SPARQL_{Stream}$ (Calbimonte et al. 12) and MorphStream
- All approaches rely on CQL window semantics
- extend SPARQL or use some derivative of it
- Treat timestamped RDF triples but use reification

Example of Reified Handling

Example

```
SELECT ?windspeed ?tidespeed
FROM NAMED STREAM <http://swiss-experiment.ch/
                   data#WannengratSensors.srdf>
[NOW-10 MINUTES TO NOW-0 MINUTES]
WHERE
  ?WaveObs a ssn:Observation;
           ssn:observationResult ?windspeed;
           ssn:observedProperty sweetSpeed:WindSpeed.
  ?TideObs a ssn:Observation;
           ssn:observationResult ?tidespeed;
           ssn:observedProperty sweetSpeed:TideSpeed.
  FILTER (?tidespeed<?windspeed)
```

SRBench (Zhang et al. 2012)

- Benchmark for RDF/SPARQL Stream Engines
- Contains data from LinkedSensorData, GeoNames, DBPedia
- Mainly queries for functionality tests, with eye on SPARQL 1.1. functionalities

Example (Example Query (to test basic pattern matching))

Q1. Get the rainfall observed once in an hour.

- Tested on CQELS, $\text{SPARQL}_{\mathit{Stream}}$ and C-SPARQL

- Test results (for engine versions as of 2012)
  - Basic SPARQL features supported
  - SPARQL 1.1 features (property paths) rather not supported
  - Only C-SPARQL supports reasoning (on RDFS level) (tested subsumption and sameAs)
  - Combined treatment of static data plus streaming data only for CQELS and C-SPARQL