# A Denotational Semantics for Continuous Queries over Streams and Relations*

**Arvind Arasu**
Stanford University
arvinda@cs.stanford.edu

**Jennifer Widom**
Stanford University
widom@cs.stanford.edu

## 1 Introduction

Continuous queries over data streams are an important new class of queries motivated by a number of applications [BBD+02, Geh03, GO03], and several languages for continuous queries have been proposed recently [ABW03, CCC+02, CCD+03, WZL03]. To date the semantics of these languages have been specified fairly informally, sometimes solely through illustrative examples.

Semantics of continuous queries can be surprisingly tricky, and the exact meaning of all possible queries often is not obvious from an informal description. For example, one approach for defining a continuous query language is to modify a conventional relational language: treat streams as append-only relations and produce answer tuples as soon as they are available [ABB+04]. This approach yields queries with easily understood semantics when the corresponding relational queries are *monotonic*, but the semantics become murky for more complex queries, e.g., queries with aggregation or windowing constructs.

A formal semantic specification assigns an unambiguous meaning to every query in the language. Furthermore, a formal semantics enables us to reason about the language as a whole, e.g., it can be used to prove equivalences of queries in the language. Extensive motivation for defining a formal semantics is provided in [Sch97].

This paper presents a formal *denotational semantics* for a generic continuous query language that we proposed in earlier work [ABW03]. The generic language consists of two data types—streams and relations—and three classes of operators over these data types: operators that produce a relation from a stream (*stream-to-relation*), operators that produce a relation from other relations (*relation-to-relation*), and operators that produce a stream from a relation (*relation-to-stream*). There are no *stream-to-stream* operators in the language—they are composed from operators belonging to the three classes above. The three classes of operators are "black box" components of the generic language, and specific continuous query languages can be derived by instantiating the black boxes in different ways. Reference [ABW03] discusses one instantiation in detail.

A denotational semantics [Sch97, Sto77] for a query language is specified by defining a *meaning function* $\mathcal{M}$. Function $\mathcal{M}$ takes any query $Q$ belonging to the language and returns the "input-output" function, denoted $\mathcal{M}[\![Q]\!]$, computed by $Q$. For the generic continuous query language we consider in this paper, $\mathcal{M}[\![Q]\!]$ takes as input the streams and relations referenced in $Q$, along with a time instant $\tau$, and produces as output the new relation values or stream tuples corresponding to time $\tau$.

The remainder of this short paper consists of two primary sections. In Section 2 we define terms and present a brief, informal description of the generic continuous query language we consider. In Section 3 we specify a formal denotational semantics for the language. Examples are sprinkled throughout, and we conclude in Section 4.

*This work was supported by the National Science Foundation under grants IIS-0118173 and IIS-9817799.

## 2 Preliminaries and Informal Language Description

Our generic continuous query language, hereafter denoted $\mathcal{L}$, contains two data types: *streams* and *relations*. Each stream and relation has an associated *schema*, which is a set of named attributes as in the standard relational model. Streams and relations are defined based on some discrete, ordered time domain $\mathcal{T}$. A *time instant* (or simply *instant*) is a value belonging to $\mathcal{T}$. For concreteness, we represent $\mathcal{T}$ as the set $\{0, 1, \ldots\}$. Note that 0 stands for the earliest time instant in this representation.

**Definition 2.1** (**Stream**) A stream $S$ is a possibly infinite bag (multiset) of *elements* $\langle s, \tau \rangle$, where $s$ is a tuple belonging to the schema of $S$ and $\tau \in \mathcal{T}$ is the *timestamp* of the element. □

**Definition 2.2** (**Relation**) A relation $R$ is a mapping from $\mathcal{T}$ to a finite but unbounded bag of tuples belonging to the schema of $R$. □

A stream is a collection of timestamped tuples. Intuitively, the element $\langle s, \tau \rangle$ of stream $S$ indicates that tuple $s$ arrives on $S$ at time $\tau$. A relation is a time-varying bag of tuples. For a relation $R$, $R(\tau)$ denotes the bag of tuples in the relation at time instant $\tau$. Note that our definition of relation is different from the standard one: In the standard interpretation, a relation is simply a set (or bag) of tuples with no notion of time as far as the semantics of standard relational query languages are concerned.

Queries in $\mathcal{L}$ are composed from operators belonging to three classes: stream-to-relation, relation-to-relation, and relation-to-stream.

- A stream-to-relation operator takes an input stream and produces a relation. The bag of tuples in the output relation at time instant $\tau$ depends only on input stream elements with timestamps $\leq \tau$.
- A relation-to-relation operator takes one or more relations as input and produces another relation as output. The bag of tuples in the output relation at time $\tau$ depends only on the bag of tuples in the input relations at $\tau$.
- A relation-to-stream operator takes an input relation and produces a stream. The elements of the stream with timestamp $\tau$ depend only on the relation state at time instants $\leq \tau$.

Conceptually, a query or operator runs continuously within the time domain. When time advances from $\tau - 1$ to $\tau$, the query or operator produces new output corresponding to $\tau$: either a new value for its output relation at $\tau$, or the elements of its output stream with timestamp $\tau$. (We refer the reader to [ABW03, SW04] for discussions of the subtleties of time in data stream systems.)

Language $\mathcal{L}$ does not prescribe specific operators belonging to the three classes—the three classes are "black boxes" of the language, and they can be instantiated using any set of operators as long as the operators satisfy the properties mentioned above. We have designed and implemented one concrete instantiation: a language called *CQL* (for *Continuous Query Language*) [ABW03]. Briefly, the stream-to-relation operators of CQL are derived from the *sliding window* operators of SQL-99. The relation-to-relation operators of CQL are the standard SQL operators. There are three relation-to-stream operators: *Istream*, which streams inserts to its input relation, *Dstream*, which streams deletes from its input relation, and *Rstream*, which streams the entire bag of tuples in its input relation at every time instant.

**Example 2.1** Consider a hypothetical network monitoring application. The application has one input stream, `Packet(srcIp,destIp,len,router)`, which represents a stream of network packets, and one input relation `DomainOf(ip,domainName)`, which contains domain names corresponding to IP addresses. Note that the relation is time-varying: new domain names might be added, or the domain names of existing IP addresses might change. The following example CQL query, explained below, produces the stream of packets originating from domain "`stanford.edu`":

```
Select Rstream(srcIp, destIp, len, router)
From   Packet [Now], DomainOf
Where  srcIp = ip and domainName = "stanford.edu"
```

Consider any time instant $\tau$. The query applies a "`Now`" window over the `Packet` stream, which is a stream-to-relation operator that selects those packet tuples with current timestamp $\tau$, i.e., packets arriving at the current timestep. The `Select-From-Where` clause effectively performs a semijoin of these packet tuples with the current state of the `DomainOf` relation, to identify packets belonging to the "`stanford.edu`" domain. Finally, the `Rstream` operator streams the current packet tuples satisfying the semijoin filter. □

Although language $\mathcal{L}$ is fairly generic, some high-level choices have been made. For example, we chose to have two data types in $\mathcal{L}$—streams and relations—rather than, say, just streams. Also, we chose not to include native stream-to-stream operators. These issues are discussed and justified in detail in [ABW03].

# 3 Formal Semantics

Before presenting the denotational semantics in Section 3.2, we first specify a syntax for our generic language and list the domains used in the semantic specification.

## 3.1 Abstract Syntax and Domains

Figure 1 presents an abstract syntax for our generic language $\mathcal{L}$ using BNF-style rules. This syntax is specified solely for the purpose of presenting the semantics; the actual syntax of a language derived from $\mathcal{L}$ could be significantly different, e.g., the syntax of CQL queries as in Example 2.1. The syntax is best understood by reading the accompanying table that describes the terms used in the syntax. (The **Domain** column of the table will be used in Section 3.2.)

The following domains are used in the semantics specification:

- *Time domain* ($\mathcal{T}$): $\mathcal{T} = \{0, 1, \ldots\}$ as described in Section 2.
- *Tuple domain* ($\mathcal{TP}$): The domain of tuples. A tuple is a finite sequence of atomic values. We do not need to distinguish among tuples with different schemas for our semantics.
- *Tuple multiset domain* ($\Sigma$): The domain of finite, but unbounded, bags of tuples.
- *Relation domain* ($\mathcal{R}$): $\mathcal{R} = \mathcal{T} \rightarrow \Sigma$, i.e., the domain of functions that map time instants to bags of tuples (Definition 2.2).
- *Stream domain* ($\mathcal{S}$): The domain of (possibly infinite) multisets over $\mathcal{TP} \times \mathcal{T}$ (Definition 2.1).
- *Relational operator domain* ($\mathcal{R}_{op}$): $\mathcal{R}_{op} = \Sigma \times \cdots \times \Sigma \rightarrow \Sigma$, i.e., the domain of functions that produce a bag of tuples from one or more bags of tuples. For example, the standard relational algebra operators (e.g., $\sigma$, $\pi$, $\bowtie$) and SQL queries belong to this domain.

| | | |
|---|---|---|
| Q | ::= | $\texttt{Q}_\texttt{R} \mid \texttt{Q}_\texttt{S}$ |
| $\texttt{Q}_\texttt{R}$ | ::= | $\texttt{RName} \mid \texttt{R2R-Op}(\texttt{Q}_\texttt{R}^1, \ldots, \texttt{Q}_\texttt{R}^n) \mid \texttt{S2R-Op}(\texttt{Q}_\texttt{S})$ |
| $\texttt{Q}_\texttt{S}$ | ::= | $\texttt{SName} \mid \texttt{R2S-Op}(\texttt{Q}_\texttt{R})$ |
| RName | ::= | Id |
| SName | ::= | Id |

| Symbol | Description | Domain |
|---|---|---|
| Q | Continuous Query (CQ) in $\mathcal{L}$ | *Query* |
| $\texttt{Q}_\texttt{R}$ | CQ producing a relation | *RelQuery* |
| $\texttt{Q}_\texttt{S}$ | CQ producing a stream | *StrQuery* |
| S2R-Op | Stream-to-Relation Operator | *S2ROp* |
| R2R-Op | Relation-to-Relation Operator | *R2ROp* |
| R2S-Op | Relation-to-Stream Operator | *R2SOp* |
| RName | Relation Name | *Identifier* |
| SName | Stream Name | *Identifier* |
| Id | Identifier | *Identifier* |

Figure 1: Abstract syntax of $\mathcal{L}$; symbol descriptions and domains

- *Syntactic domains*: The domains associated with the syntactic terms listed in Figure 1. For example, *Query* denotes the domain of valid continuous queries according to the syntax in Figure 1, and *R2ROp* denotes the domain of relation-to-relation operators.
- *Relation Lookup domain* (*RelLookup*): The domain of functions that map an identifier (relation name) to its corresponding relation, i.e., $\mathit{RelLookup} = \mathit{Identifier} \rightarrow \mathcal{R}$.
- *Stream Lookup domain* (*StrLookup*): The domain of functions that map an identifier (stream name) to its corresponding stream, i.e., $\mathit{StrLookup} = \mathit{Identifier} \rightarrow \mathcal{S}$.

### 3.2 Denotational Semantics

Recall from Section 1 that a denotational semantics for a query language specifies a meaning function $\mathcal{M}$ that maps queries in the language to the input-output function that they compute. Following convention, we specify the meaning function recursively, using subsidiary meaning functions for subcomponents of a query. Figure 2 lists the meaning functions that we use in our specification. $\mathcal{M}$ is the "main" meaning function, which assigns a meaning to the entire query. The other functions assign meaning to (sub)components of a query; for example, $\mathcal{M}_{R2R}$ maps a relation-to-relation operator to a function over conventional relations.

We use *lambda calculus* [Pie97] for defining functions. The expression $\lambda x_1 \ldots \lambda x_n . E$ defines a function that takes arguments $v_1, \ldots, v_n$, and returns the result of evaluating expression $E$ with all free occurrences of $x_i$ in $E$ replaced by $v_i$, $1 \leq i \leq n$. The arguments $v_i$ and the returned result could themselves be functions, i.e., lambda calculus expressions.

The semantics of $\mathcal{L}$ treats the semantics of the three classes of operators as black boxes. In other words, we assume that the meaning functions $\mathcal{M}_{S2R}$, $\mathcal{M}_{R2R}$, and $\mathcal{M}_{R2S}$ are given. Section 3.3 specifies these meaning functions for some example operators.

The specifications of the remaining three meaning functions are given below. Each meaning function is specified in separate parts, one part for each BNF rule for its corresponding syntactic term (Figure 1). For example, $\mathcal{M}$ is specified in two parts: one corresponding to the derivation of

| Query component | Meaning Function | Signature |
|---|---|---|
| Q | $\mathcal{M}$ | $Query \rightarrow (RelLookup \times StrLookup \times \mathcal{T} \rightarrow (\Sigma \cup \mathcal{S}))$ |
| $\mathtt{Q_R}$ | $\mathcal{M}_R$ | $RelQuery \rightarrow (RelLookup \times StrLookup \times \mathcal{T} \rightarrow \Sigma)$ |
| $\mathtt{Q_S}$ | $\mathcal{M}_S$ | $StrQuery \rightarrow (RelLookup \times StrLookup \times \mathcal{T} \rightarrow \mathcal{S})$ |
| S2R-Op | $\mathcal{M}_{S2R}$ | $S2ROp \rightarrow (\mathcal{S} \times \mathcal{T} \rightarrow \Sigma)$ |
| R2R-Op | $\mathcal{M}_{R2R}$ | $R2ROp \rightarrow \mathcal{R}_{op}$ |
| R2S-Op | $\mathcal{M}_{R2S}$ | $R2SOp \rightarrow (\mathcal{R} \times \mathcal{T} \rightarrow \mathcal{S})$ |

Figure 2: Meaning functions in the denotational semantics

Q from $\mathtt{Q_R}$ and the other to the derivation from $\mathtt{Q_S}$. The complete meaning function can be thought of as a combination of its parts using appropriate "if-then-else" statements. In these functions, and the remainder of the paper, we use the $\in$ symbol as if we are considering sets rather than multisets. In the presence of duplicates, each duplicate must be considered separately when evaluating $\in$—in all cases the interpretation is obvious from context.

- $\mathcal{M}$ : The input-output function $\mathcal{M}[\![\mathtt{Q}]\!]$ produced by $\mathcal{M}$ for a query Q takes three parameters: the first two parameters are functions that are used to map relation or stream names in the query to corresponding relations and streams, and the third parameter is a time instant. $\mathcal{M}[\![\mathtt{Q}]\!](r, s, \tau)$ specifies the output produced by Q at time instant $\tau$. $\mathcal{M}[\![\mathtt{Q}]\!](r, s, \tau)$ invokes $\mathcal{M}_R[\![\mathtt{Q_R}]\!](r, s, \tau)$ if $\mathtt{Q} = \mathtt{Q_R}$ produces a relation as output, or it invokes $\mathcal{M}_S[\![\mathtt{Q_S}]\!](r, s, \tau)$ if $\mathtt{Q} = \mathtt{Q_S}$ produces a stream as output. An additional filtering operation is required for the latter case since $\mathcal{M}_S[\![\mathtt{Q_S}]\!](r, s, \tau)$ returns all elements of its output stream with timestamp $\leq \tau$. (See the definition of $\mathcal{M}_S$ below.)

$$\begin{aligned} \mathcal{M}[\![\mathtt{Q_R}]\!] &= \lambda r.\lambda s.\lambda\tau.\mathcal{M}_R[\![\mathtt{Q_R}]\!](r, s, \tau) \\ \mathcal{M}[\![\mathtt{Q_S}]\!] &= \lambda r.\lambda s.\lambda\tau.\{\langle e, \tau'\rangle : \langle e, \tau'\rangle \in \mathcal{M}_S[\![\mathtt{Q_S}]\!](r, s, \tau) \wedge \tau' = \tau\} \end{aligned}$$

- $\mathcal{M}_R$ : If $\mathtt{Q_R}$ is a (sub)query producing a relation, $\mathcal{M}_R[\![\mathtt{Q_R}]\!](r, s, \tau)$ specifies the bag of tuples in the output relation at time $\tau$. Parameters $r$ and $s$, as before, are stream and relation lookup functions. $\mathcal{M}_R$ calls functions $\mathcal{M}_R$ (recursively), $\mathcal{M}_S$, $\mathcal{M}_{R2R}$, $\mathcal{M}_{S2R}$, and/or $r$, depending on the structure of $\mathtt{Q_R}$. For example, if $\mathtt{Q_R} = \mathtt{RName}$, $\mathcal{M}_R[\![\mathtt{Q_R}]\!](r, s, \tau)$ uses function $r$ to look up the time-varying relation corresponding to RName, and applies the relation to identify the bag of tuples at time $\tau$.

$$\begin{aligned} \mathcal{M}_R[\![\mathtt{RName}]\!] &= \lambda r.\lambda s.\lambda\tau.r(\mathtt{RName})(\tau) \\ \mathcal{M}_R[\![\mathtt{R2R\text{-}Op}(\mathtt{Q_R^1}, \ldots, \mathtt{Q_R^n})]\!] &= \lambda r.\lambda s.\lambda\tau.\mathcal{M}_{R2R}[\![\mathtt{R2R\text{-}Op}]\!](\mathcal{M}_R[\![\mathtt{Q_R^1}]\!](r, s, \tau), \ldots, \mathcal{M}_R[\![\mathtt{Q_R^n}]\!](r, s, \tau)) \\ \mathcal{M}_R[\![\mathtt{S2R\text{-}Op}(\mathtt{Q_S})]\!] &= \lambda r.\lambda s.\lambda\tau.\mathcal{M}_{S2R}[\![\mathtt{S2R\text{-}Op}]\!](\mathcal{M}_S[\![\mathtt{Q_S}]\!](r, s, \tau), \tau) \end{aligned}$$

- $\mathcal{M}_S$ : If $\mathtt{Q_S}$ is a (sub)query producing a stream, $\mathcal{M}_S[\![\mathtt{Q_S}]\!](r, s, \tau)$ specifies the bag of stream elements in the output stream with timestamp $\leq \tau$. In the specification of $\mathcal{M}_S[\![\mathtt{Q_S}]\!]$ for the case $\mathtt{Q_S} = \mathtt{R2S\text{-}Op}(\mathtt{Q_R})$, the lambda calculus expression "$\lambda\tau'.\ \mathcal{M}_R[\![\mathtt{Q_R}]\!](r, s, \tau')$" defines a function that takes a single parameter $\tau'$ and returns the bag of tuples at time $\tau'$ in the relation produced by subquery $\mathtt{Q_R}$, which is just a formal representation for the relation produced by $\mathtt{Q_R}$.

$$\begin{aligned} \mathcal{M}_S[\![\mathtt{SName}]\!] &= \lambda r.\lambda s.\lambda\tau.\{\langle e, \tau'\rangle : \langle e, \tau'\rangle \in s(\mathtt{SName}) \wedge \tau' \leq \tau\} \\ \mathcal{M}_S[\![\mathtt{R2S\text{-}Op}(\mathtt{Q_R})]\!] &= \lambda r.\lambda s.\lambda\tau.\mathcal{M}_{R2S}[\![\mathtt{R2S\text{-}Op}]\!]((\lambda\tau'.\ \mathcal{M}_R[\![\mathtt{Q_R}]\!](r, s, \tau')), \tau) \end{aligned}$$

$$
\begin{array}{lcl}
\texttt{S2R-Op} & ::= & \texttt{Now} \mid \texttt{Range}(T) \mid \texttt{Row}(N) \\
\texttt{R2R-Op} & ::= & \texttt{SemiJoin}(i,j) \mid \texttt{Filter}(i,v) \\
\texttt{R2S-Op} & ::= & \texttt{IStream} \mid \texttt{DStream} \mid \texttt{RStream}
\end{array}
$$

Figure 3: Abstract syntax for example operators

### 3.3 Semantics for Example Operators

This section presents formal semantics for several of the instantiated operators in the CQL language. Figure 3 lists the example operators we consider and their abstract syntax using the same BNF-style rules we used for $\mathcal{L}$. (We emphasize that the abstract syntax is for illustrative purposes only—it does not correspond to the actual syntax of these operators in CQL.)

- $\mathcal{M}_{S2R}$: We consider three kinds of sliding windows as examples of stream-to-relation operators. All three operators take a stream $S$ and a timestamp $\tau$ as input and return a bag of tuples as output: the `Now` window operator returns the tuples of $S$ with timestamp $\tau$; the `Range` window operator, specified using a parameter $T$, returns the tuples of $S$ with timestamps in the range $[\tau - T, \tau]$; the `Row` window operator, specified using an integer parameter $N$, returns the $N$ most recent tuples of $S$ with timestamps $\leq \tau$.[1]

$$
\begin{array}{rcl}
\mathcal{M}_{S2R}[\![\texttt{Now}]\!] & = & \lambda S.\lambda\tau.\{e : \langle e,\tau\rangle \in S\} \\
\mathcal{M}_{S2R}[\![\texttt{Range}(T)]\!] & = & \lambda S.\lambda\tau.\{e : \langle e,\tau'\rangle \in S \wedge \max(\tau - T, 0) \leq \tau' \leq \tau\} \\
\mathcal{M}_{S2R}[\![\texttt{Row}(N)]\!] & = & \lambda S.\lambda\tau.\{e : \langle e,\tau'\rangle \in S \wedge (\tau' \leq \tau) \wedge (N \geq |\,\{\langle e,\tau''\rangle \in S : \tau' \leq \tau'' \leq \tau\}\,|\,)\,\}
\end{array}
$$

- $\mathcal{M}_{R2R}$: We present formal semantics for restricted versions of two standard relational operators: semijoin and filter (selection). $\texttt{SemiJoin}(i,j)$ performs a semijoin on the $i^{th}$ attribute of its first input with the $j^{th}$ attribute of its second input, where both inputs are bags of tuples. $\texttt{Filter}(i,v)$ returns all tuples from its input bag having value $v$ in the $i^{th}$ attribute. In the definitions, $e.i$ abuses standard notation to denote the value in the $i^{th}$ attribute of a tuple $e$.

$$
\begin{array}{rcl}
\mathcal{M}_{R2R}[\![\texttt{SemiJoin}(i,j)]\!] & = & \lambda E_1.\lambda E_2.\{e_1 : e_1 \in E_1 \wedge (\exists\, e_2 \in E_2 : e_1.i = e_2.j)\,\} \\
\mathcal{M}_{R2R}[\![\texttt{Filter}(i,v)]\!] & = & \lambda E.\{e : e \in E \wedge e.i = v\}
\end{array}
$$

- $\mathcal{M}_{R2S}$: We present formal semantics for the three relation-to-stream operators in CQL. The `IStream` operator takes a (time-varying) relation $R$ and a time instant $\tau$, and streams the new tuples inserted into $R$ at time $\tau$, i.e., tuples that appear in $R(\tau)$ but not in $R(\tau - 1)$. Analogously, the `DStream` operator streams the tuples that were deleted from $R$ at time $\tau$, i.e., tuples that appear in $R(\tau - 1)$ but not in $R(\tau)$. Finally, the `RStream` operator (for "relation stream") streams all the tuples in $R(\tau)$. In the definitions, assume $R(-1) = \phi$.

$$
\begin{array}{rcl}
\mathcal{M}_{R2S}[\![\texttt{IStream}]\!] & = & \lambda R.\lambda\tau.\{\langle e,\tau'\rangle : \tau' \leq \tau \wedge e \in R(\tau) \wedge e \notin R(\tau - 1)\} \\
\mathcal{M}_{R2S}[\![\texttt{DStream}]\!] & = & \lambda R.\lambda\tau.\{\langle e,\tau'\rangle : \tau' \leq \tau \wedge e \in R(\tau - 1) \wedge e \notin R(\tau)\} \\
\mathcal{M}_{R2S}[\![\texttt{RStream}]\!] & = & \lambda R.\lambda\tau.\{\langle e,\tau'\rangle : \tau' \leq \tau \wedge e \in R(\tau)\}
\end{array}
$$

[1] If the stream has duplicate timestamps, our formal specification of the `Row` window operator may return fewer than $N$ elements. An alternative definition (used in our CQL implementation) introduces nondeterminism for `Row`-based windows in the presence of duplicate timestamps.

**Example 3.1** The CQL query in Example 2.1 is expressed as follows using the abstract syntax of Figures 1 and 3:

$$\texttt{Q} = \texttt{RStream}(\texttt{SemiJoin}(1,1)(\texttt{Now}(\texttt{Packet}), \texttt{Filter}(2, \text{“}\texttt{stanford.edu}\text{”})(\texttt{DomainOf})))$$

The reader can verify that according to our formal semantics, the meaning of `Q` is equivalent to the following expression, after some simplifications.

$$\begin{aligned}\mathcal{M}[\![\texttt{Q}]\!] \;=\; & \lambda r.\lambda s.\lambda\tau.\{\langle e,\tau'\rangle : \langle e,\tau'\rangle \in s(\texttt{Packet}) \wedge \tau' = \tau \wedge \\ & \quad (\exists e' \in r(\texttt{DomainOf})(\tau) : e'.2 = \text{“}\texttt{stanford.edu}\text{”} \wedge e'.1 = e.1)\,\}\end{aligned}$$ □

# 4 Conclusions

We specified a complete formal semantics for a generic continuous query language over streams and relations. Our semantics resolves any ambiguities present in informal language descriptions—it assigns an exact meaning to any query in the language, at any point in time, for any possible input streams and relations. Our generic language is built from a number of "black box" operators, whose semantics must be instantiated for a specific concrete language. As examples we instantiated a portion of CQL [ABW03], a language that includes SQL constructs, sliding windows over streams, and special-purpose relation-to-stream operators.

In addition to clarifying the potentially subtle meaning of continuous queries, our formal semantics provides a tool for discovering and reasoning about equivalences within a given continuous query language, and for comparing expressiveness across different proposed languages.

# Acknowledgments

We are grateful to Mayur Naik for helpful discussions on the basics of programming language semantics.

# References

[ABB+04] A. Arasu, B. Babcock, S. Babu, J. McAlister, and J. Widom. Characterizing memory requirements for queries over continuous data streams. *ACM Trans. on Database Systems*, 29(1):1–33, Mar. 2004.

[ABW03] A. Arasu, S. Babu, and J. Widom. The CQL Continuous Query Language: Semantic Foundations and Query Execution. Technical report, Stanford University, Oct. 2003. `http://dbpubs.stanford.edu/pub/2003-67`.

[BBD+02] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proc. of the 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*, pages 1–16, June 2002.

[CCC+02] D. Carney, U. Centintemel, M. Cherniak, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. B. Zdonik. Monitoring streams - a new class of data management applications. In *Proc. of the 28th Intl. Conf. on Very Large Data Bases*, pages 215–226, Aug. 2002.

[CCD+03] S. Chandrasekharan, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. A. Shah. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *Proc. of the 1st Conf. on Innovative Data Systems Research*, pages 269–280, Jan. 2003.

[Geh03] J. Gehrke. Data stream processing. *IEEE Computer Society Bulletin of the Technical Comm. on Data Engg.*, 26(1), Mar. 2003.

[GO03] L. Golab and M. T. Özsu. Issues in data stream management. *SIGMOD Record*, 32(1):5–14, Mar. 2003.

[Pie97] B. C. Pierce. *The Computer Science and Engineering Handbook*, chapter Foundational Calculi for Programming Languages, pages 2190–2207. CRC Press, 1997.

[Sch97] D. A. Schmidt. *The Computer Science and Engineering Handbook*, chapter Programming Language Semantics, pages 2237–2254. CRC Press, 1997.

[Sto77] J. E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*. MIT Press, Cambridge, Massachusetts, 1977.

[SW04] U. Srivastava and J. Widom. Flexible time management in data stream systems. In *Proc. of the 23rd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*, June 2004.

[WZL03] H. Wang, C. Zaniolo, and C. R. Luo. ATLAS: A small but complete SQL extension for data mining and data streams. In *Proc. of the 29th Intl. Conf. on Very Large Data Bases*, pages 1113–1116, Sept. 2003. Demonstration description.