A theory of stream queries

Yuri Gurevich[1] Dirk Leinders[2] Jan Van den Bussche[2]

[1]Microsoft Research

[2]Hasselt University

DBPL 2007

Motivation

- research on querying streaming data
- main focus: query language, query processor
- general theory of stream queries not yet well developed

Outline

Outline

1. General theory
   - Abstract computability
   - Continuity
   - The finite case
   - Time synchronization

2. streaming ASMs
   - Universality of streaming ASMs
   - Bounded-memory restrictions

What is a stream (query)?

universe of data elements $\mathbb{U}$

- predicates and functions
- structure (logic)

stream

- = a (possibly infinite) sequence of data elements from $\mathbb{U}$
- *Stream* = set of all streams
- *finStream* = set of all finite streams

stream query

= a function $\mathit{Stream} \rightarrow \mathit{Stream}$

What is a stream (query)?

example: unreasonable stream query ($\textsc{check}$)

- Input: stream **s** over $\{a, b\}$
- Output:

$$\textsc{check}(\mathbf{s}) = \begin{cases} () & \text{if } a \in \mathbf{s} \\ b & \text{otherwise} \end{cases}$$

example: reasonable stream query (filter $\pi_A \sigma_{B>10} \mathbf{s}$)

- Input: stream **s** of tuples with attributes A and B
- Output: stream of A-values of tuples in **s** with B-value higher than 10.

What is a computable stream query?

Repeat(K)

Let $K: \mathit{finStream} \to \mathit{finStream}$. Define $\mathit{Repeat}(K): \mathit{Stream} \to \mathit{Stream}$ as follows

$$\mathbf{s} \mapsto \bigodot_{k=0}^{\mathit{size}(\mathbf{s})} K(\mathbf{s}^{\leq k})$$

Definition

$\mathcal{Q}$ is abstract computable if $\mathcal{Q} = \mathit{Repeat}(K)$ for some K. K is called a kernel for $\mathcal{Q}$.

example: kernel for filter $\pi_A \sigma_{B>10} \mathbf{s}$

$$K(s_1 \ldots s_k) = \begin{cases} s_k.A & \text{if } s_k.B > 10, \\ () & \text{otherwise} \end{cases}$$

What is a computable stream query?

Repeat(K)

Let K: *finStream* $\to$ *finStream*. Define *Repeat*(K): *Stream* $\to$ *Stream* as follows

$$\mathbf{s} \mapsto \bigodot_{k=0}^{size(\mathbf{s})} K(\mathbf{s}^{\leq k})$$

Definition

$\mathcal{Q}$ is abstract computable if $\mathcal{Q} = \mathit{Repeat}(K)$ for some K. K is called a kernel for $\mathcal{Q}$.

remarks

- result on infinite input stream can be finite
- result on finite input stream must be finite

Outline

1. General theory
   - Abstract computability
   - Continuity
   - The finite case
   - Time synchronization
2. streaming ASMs
   - Universality of streaming ASMs
   - Bounded-memory restrictions

What is a continuous stream query?

recall

A real function $f\colon \mathbb{R} \to \mathbb{R}$ is called continuous if for all $x \in \mathbb{R}$, for every neighborhood around $f(x)$, there exists a neighborhood around x that is completely mapped into the neighborhood of $f(x)$.

Definition

An open ball is a set of the form

$$\mathbf{B}(\mathbf{p}) := \{\mathbf{s} \in \mathit{Stream} \mid \mathbf{p} \text{ is a prefix of } \mathbf{s}\},$$

for some $\mathbf{p} \in \mathit{finStream}$. Elements are called continuations of $\mathbf{p}$.

Definition

$\mathcal{Q}\colon \mathit{Stream} \to \mathit{Stream}$ is continuous if for all streams $\mathbf{s}$, and all open balls $\mathbf{B}(\mathbf{q})$ with $\mathbf{q}$ a prefix of $\mathcal{Q}(\mathbf{s})$, there exists a prefix $\mathbf{p}$ of $\mathbf{s}$ such that $\mathcal{Q}(\mathbf{B}(\mathbf{p})) \subseteq \mathbf{B}(\mathbf{q})$.

Computability and continuity

Theorem

Let $\mathcal{Q}$ be a stream query mapping finite inputs to finite outputs. Then $\mathcal{Q}$ is abstract computable if and only if $\mathcal{Q}$ is continuous.

application

- difference is not abstract computable
  - Input: interleaving of two streams **r** and **s**
  - Output: all elements in **r** that do not occur in **s**
- CHECK is not abstract computable

remark

Qualification that $\mathcal{Q}$ maps finite inputs to finite outputs is important.

Outline

1. General theory
   - Abstract computability
   - Continuity
   - The finite case
   - Time synchronization
2. streaming ASMs
   - Universality of streaming ASMs
   - Bounded-memory restrictions

Computability and continuity of finite stream queries

considering only finite streams . . .

- finite stream query = a function *finStream* $\rightarrow$ *finStream*
- open balls are finite continuations of finite streams

Theorem

A finite stream query is abstract computable if and only if it is continuous.

Continuity and monotonicity

A finite stream query $\mathcal{Q}$ is called monotonic if for all finite streams $\mathbf{s}$ and $\mathbf{s}'$ with $\mathbf{s} \sqsubseteq \mathbf{s}'$ we have $\mathcal{Q}(\mathbf{s}) \sqsubseteq \mathcal{Q}(\mathbf{s}')$

Theorem

A finite stream query is continuous if and only if it is monotonic.

Outline

Synchronous abstract computability

abstract computability does not imply synchrony

example: filter query

$$\begin{array}{rcccccccc} \mathbf{s} = & 3 & 5 & 12 & 14 & 7 & 8 & \ldots \\ \mathcal{Q}(\mathbf{s}) = & 12 & 14 & \ldots \end{array}$$

Synchronous abstract computability

abstract computability does not imply synchrony

example: filter query

$$\begin{array}{rcccccccc} \mathbf{s} = & 3 & 5 & 12 & 14 & 7 & 8 & \dots \\ \mathcal{Q}(\mathbf{s}) = & 12 & 14 & \dots \end{array} \qquad K(3) = K(3 \quad 5) = ()$$

Synchronous abstract computability

abstract computability does not imply synchrony

example: filter query

$$\begin{array}{rccccccc} \mathbf{s} = & 3 & 5 & 12 & 14 & 7 & 8 & \ldots \\ \mathcal{Q}(\mathbf{s}) = & 12 & 14 & \ldots & & & & \end{array}$$

$K(3) = K(3 \quad 5) = ()$

Definition

$\mathcal{Q}$ is synchronous abstract computable (SAC) if $\mathcal{Q} = \mathit{Repeat}(K)$ for some K such that $K(()) = ()$ and every other $K(\mathbf{s})$ is of length one.

example: SAC filter query

$$\begin{array}{rccccccc} \mathbf{s} = & 3 & 5 & 12 & 14 & 7 & 8 & \ldots \\ \mathcal{Q}'(\mathbf{s}) = & \mathrm{N} & \mathrm{N} & 12 & 14 & \mathrm{N} & \mathrm{N} & \ldots \end{array}$$

$K'(3) = K'(3 \quad 5) = \mathrm{N}$

Outline

Complexity limitations

- kernel function not restricted
- possible restrictions:
  - computable by TM
  - belonging to a certain complexity class
- computable by streaming ASM

Background on Abstract State Machines

- Transition system with structures as states
- interpretations of function and relation symbols change; dynamic/static vocabulary
- program = update rule
  - basic update rule $f(t_1, \ldots, t_n) := t_0$
  - if-then-else
  - parallel application of update rules

Gurevich has shown that every sequential algorithm can be modelled as an ASM.

Outline

1. General theory
   - Abstract computability
   - Continuity
   - The finite case
   - Time synchronization
2. streaming ASMs
   - Universality of streaming ASMs
   - Bounded-memory restrictions

Streaming ASM

example: sASM for filter query $\pi_A \sigma_{B>10} \mathbf{s}$

if $in.B > 10$ then
  $out = in.A$
else
  $out = \perp$
endif

Theorem (Universality)

If $\mathcal{Q}$ is abstract computable by a bounded-length kernel, then $\mathcal{Q}$ is computable by an sASM.

Corollary

Every SAC query is computable by an sASM.

Outline

Bounded-memory computability

Definition

A bounded-memory sASM (bm sASM) is an sASM where

- dynamic functions are nullary;
- non-nullary (static) functions only in $out := t_0$ rules;
- *out* is not used as an argument to a function.

All CQL-queries where a finite window is applied to the input streams are computable by a bounded-memory sASM.

[CQL = Continuous Query Language [Arasu, Babu, Widom]]

INTERSECT not bounded-memory computable

INTERSECT (boolean, synchronous version)

- Input: two interleaved streams **r** and **s**
- Output: do the portions of **r** and **s**, seen so far, intersect?

Theorem

INTERSECT *is not computable by a bm sASM.*

Proof.

Let M compute INTERSECT. Assume total order $<$ on elements.
Ramsey: there is set of elements V s.t. truth of predicates in M's program only depends on $<$.
Choose $e_1 < e_1' < \cdots < e_n < e_n'$ in V.
Run of M on $\langle \mathtt{r} : e_1 \rangle \ldots \langle \mathtt{r} : e_n \rangle \langle \mathtt{s} : e_\ell \rangle$ same as run on $\langle \mathtt{r} : e_1 \rangle \ldots \langle \mathtt{r} : e_n \rangle \langle \mathtt{s} : e_\ell' \rangle$. □

bounded-memory sASM: too restrictive

running average

- intuitively computable with little amount of memory
- can not be modeled by bounded-memory sASM

$o(n)$-sASM

- relax condition on non-nullary (static) functions
- $o(n)$-sASM can compute running average (under reasonable assumption)
- $o(n)$-sASM cannot compute INTERSECT (reduction from theorem on Finite Cursor Machines)

Open problem

- relax bounded-memory sASMs in other ways than with $o(n)$-length bitstrings.