Semantics of Data Streams and Operators

David Maier, Jin Li, Kristin Tufte, Vassilis Papadimos
Portland State University
Pete Tucker
Whitworth College

What do you say in an invited talk?

- Retrospective
- Tutorial
- All the stuff you've been mulling over but haven't quite figured out yet
  - *Maybe someone here will get it better*

Stream representation, reconstruction functions, monotonicity, punctuation, disorder, windows

What does a stream mean?

- A recent paper define a data streams as
  "bags of append-only of ordered tuples"
  or with the addition of timestamps, as
  "an unbounded appended-only bags of elements"
- Lots of questions
  - an ordered bag?
  - what constitutes a duplicate?
  - can a stream be finite?
  - is it being updated?

Stream Denotation and Representation

- Denotation (what it means)
- Representation (encoding)
- Can have multiple representations for the same denotation

Example: Sequence of finite relation states

```
r1(A B)   r2(A B)   r3(A B) …
  1 2       1 2       1 3
  1 3       1 3       2 4
            2 4       2 5
```

- serial

  <1 2>, <1 3>, EOR, <1 2>, <1 3>, <2 4>, EOR,
  <1 3>, <2 4>, <2 5>, EOR, …

- tagged

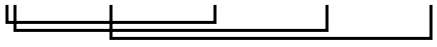
  (<1 2>:1), (<1 3>:2), (<1 3>:1), (<1 2>:2),
  (<1 3>:3), (<2 4>:2), (<2 4>:3), (<2 5>:3),
  …

- delta

  +<1 2>, +<1 3>, EOR, +<2 4>, EOR,
  -<1 2>, +<2 5>, EOR, …

- windowed

  <1 2> <1 3> <2 4> <2 5> …



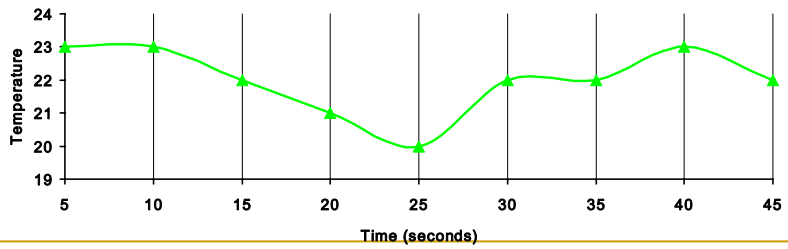
Different characteristics

- Space efficiency
- States can be interleaved with tagged (but hard to know when a state is complete)
- Operator implementation, e.g., $\sigma_{A+1=B}$
  - Can apply elementwise on serial, tagged, delta
  - Not on windowed

Example: Sensor readings

- Reg. intervals: `23, 23, 22, 21, 20, 22, 22, 23, …`
- Change only: `<23 5s>, <22 15s>, <21 20s>, <20 25s>, <22, 30s>, <23, 40s>, …`

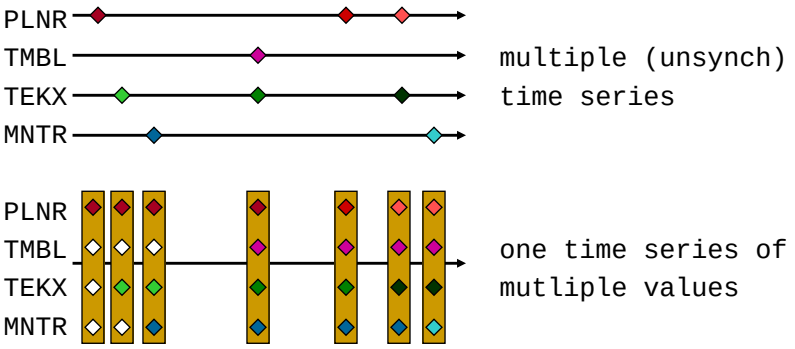
  Hard to tell no data vs. stall



Can interpret same representation different ways

- Stream of stock quotes

  <ticker, time, price>



Reconstitution functions

- Want to treat streams incrementally
- Use a ***reconstitution function*** to give successive approximations of stream denotation from successive prefixes

  S ∈ strm(**T**) [finite or infinite]

  P a prefix of S has type seq(**T**)

  D is domain of interpretation

  reconst(P) = d ∈ **D**

[1, 5, 6, 5, 7, 2, …]

Example: Insert (bag)

- `ins([]) = {{}}`
- `ins(P:i) = insert(i, ins(P))`

```
ins([1, 5, 6, 5]) = {{1, 5, 5, 6}}
```


Example: Insert unique

- `ins-u([]) = ∅`
- `ins-u(P:i) = if i ∈ ins-u(P)`
  `then ins-u(P)`
  `else`
  `insert(i,ins-u(P))`

```
ins-u([1, 5, 6, 5]) = {1, 5, 6}
```

Example: Insert-collect

- `ins-c([]) = []`
- `ins-c(P:i) = ins-c(P):ins-u(P:i)`

```
ins([1, 5, 6, 5]) =
[{1}, {1,5}, {1,5,6}, {1,5,6}]
```


Remarks

- These examples are all incremental, but reconst function need not be
- Element types of input and output structures need not be the same

  T vs. **±T**
- if `reconst(P) = d`, then write

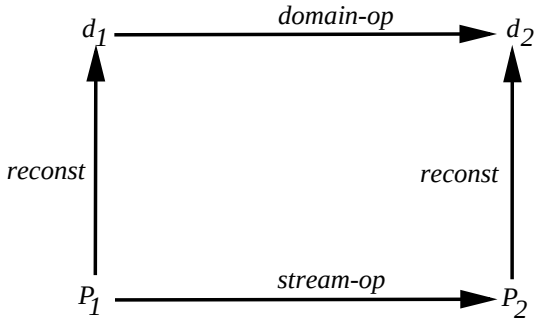
  `const(d) = P`

  (not really a function)

Faithfulness of stream versions of operators

- When is a stream operator a reasonable analogue of its domain counterpart?

  For every prefix P_1 of input stream S

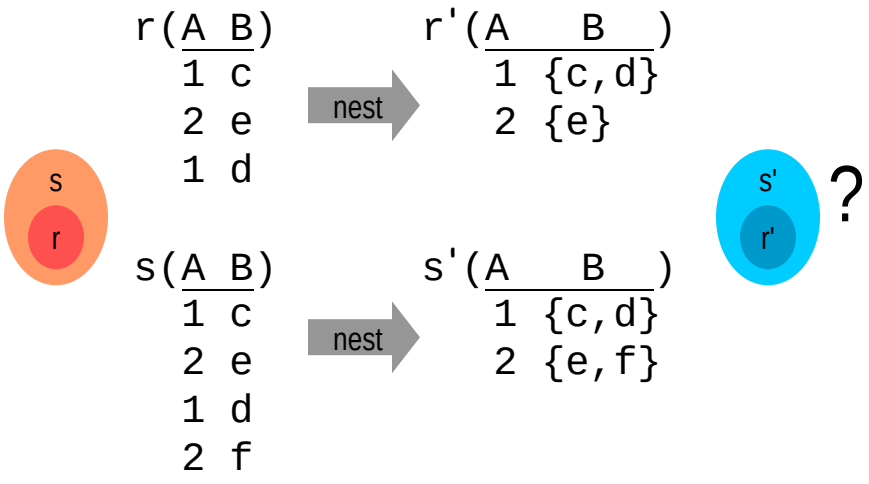


Monotone/Non-blocking/ Reconstitution

Monotone relational ops carry over easily to stream counterparts (under ins)

cr = const(r)

What about hierarchical data?



Depends on the ordering

- s' *superset* r': not monotone
- s' *subsumes* r': monotone

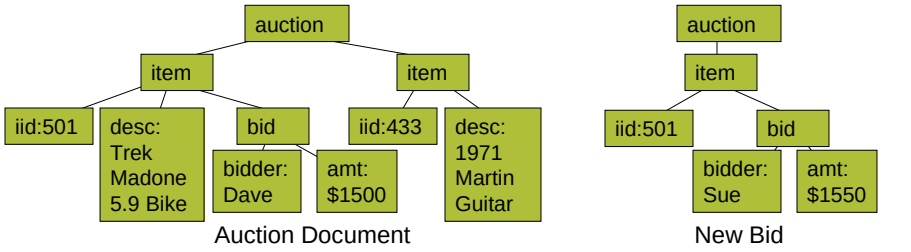
Can we get

Not if ins is the reconstitution function

What works?

- insert-replace – treat A as key
  `ins-r(r', <2, {e,f}>) = s'`
- merge (recursive union on XML)

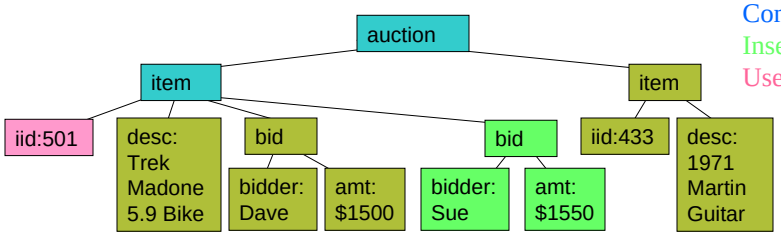
Merge is like Deep Union (Penn)



Auction Document

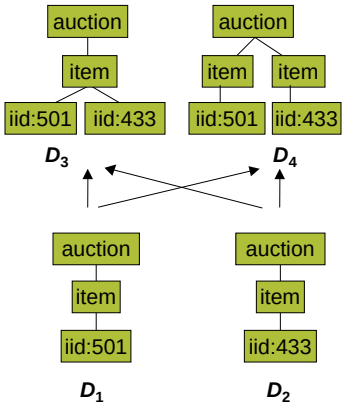
New Bid

Combined
Inserted
Used in Match



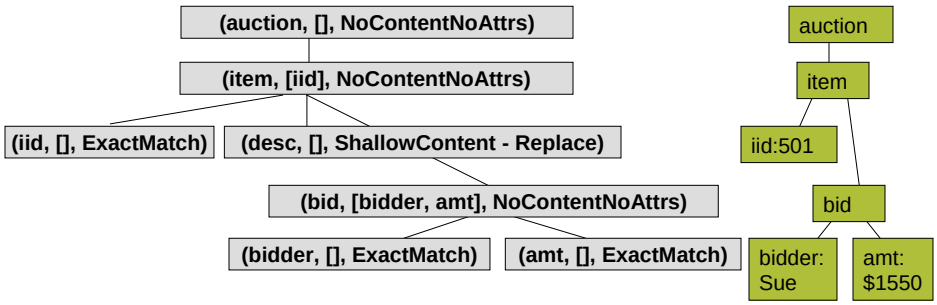
Merged Document

Issue: Non-unique upper bounds



- Would like Merge to be LUB in subsumption order
- Keys in Merge Template eliminate ambiguity
- Know D_4 is correct result if we know iid is a key for item

Merge Template removes ambiguity



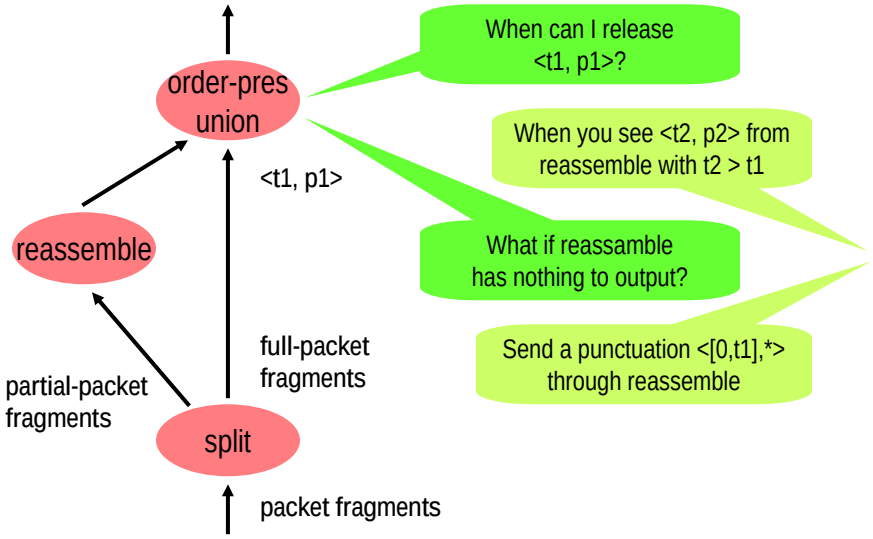
- Merge Template is an XML document consisting of a tree of Element Merge Templates (EMT)
- EMT is a triplet containing: <name, local key, content-combine function>

Additional stream semantics

May have useful prior knowledge about stream content

- |reconst(P)| < k
- B-sorted on attribute A with slack k [Aurora]
- Limits on remainder of stream (punctuation)
  Everything matching pattern p has already been seen

Example from Gigascope



Disorder

Many reasons for physical stream to be disordered

- different data paths
- combining sources
- more than one order

  start, end time of a netflow record

Example of disorder

- flow start, ordered by 8th packet observation time

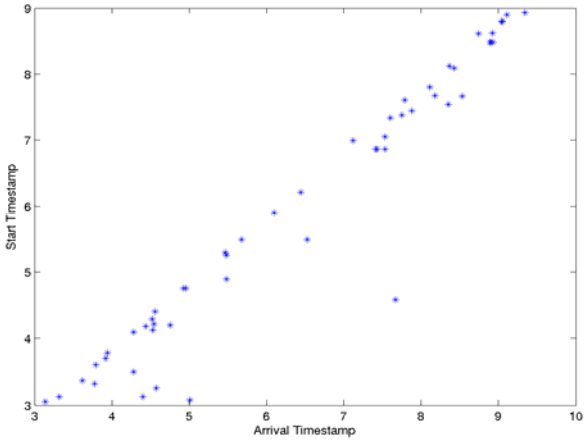


Figure 6: Band Disorder

How to deal with disorder?

Allow for a maximum displacement: *slack*

Slack of k2 catches all points; slack of k1 introduces a little inaccuracy, but improves latency.

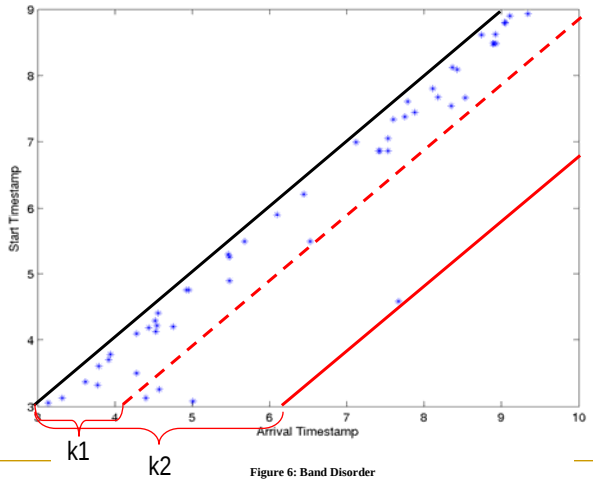
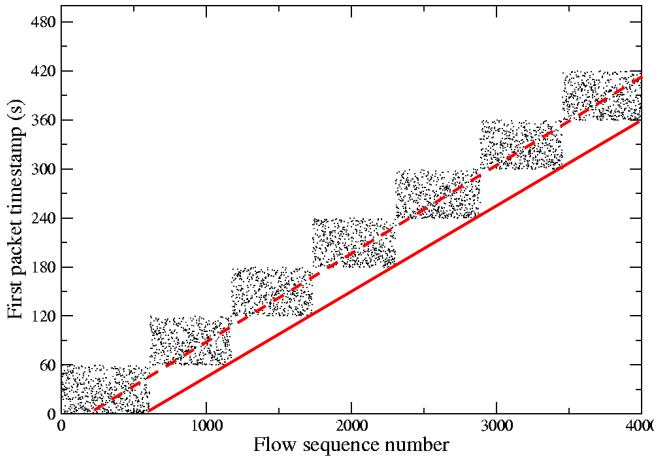


Figure 6: Band Disorder

What about this?

Netflow records from a router; start time by output order

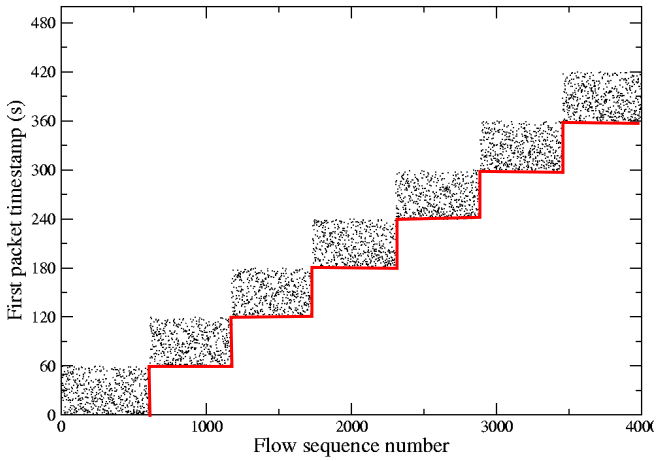
Much inaccuracy with anything other than maximal slack.



Problem: constant slack

Punctuation can give "variable slack"

Data source likely knows block boundaries.



Shield your eyes: Performance graph

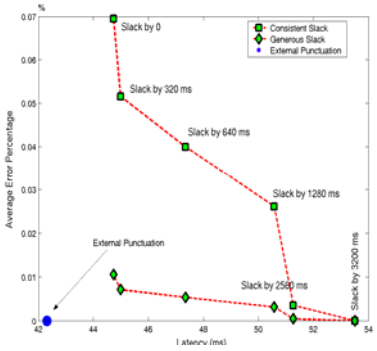


Figure 8: Latency vs. Accuracy Band-Disorder
(average error percentage)

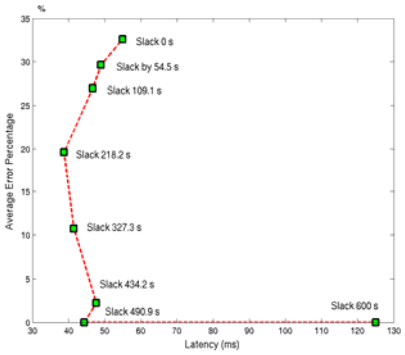
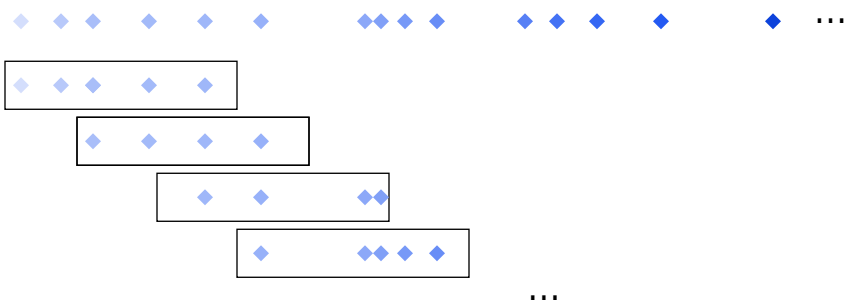


Figure 9: Latency vs. Accuracy Block-Sorted-Disorder
(average error percentage)

Windowed operators

- General idea easy to get; details are dicey
- Useful for
  - Blocking operators (e.g., aggregate)
  - Unbounded stateful operators (e.g., join)
- Divide stream into finite substreams, execute operation over each, say max

Can overlap: Sliding window



- Often described operationally on physical order
- Causes problems when stream is disordered

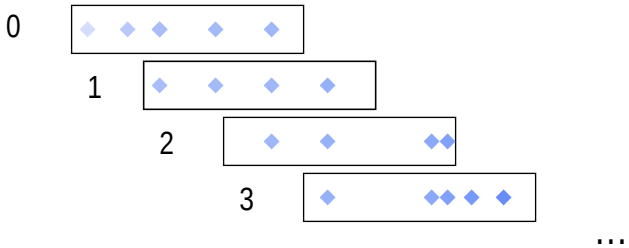
Specify windows independent of physical presentation

Consider a window specification

`[Range 30, Slide 10] on A`

Use explicit window IDs

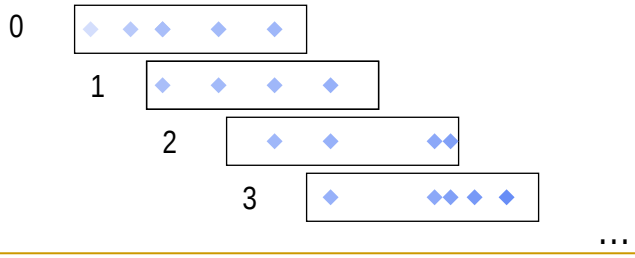
`{0, 1, 2, … }`



Define extent of each window

Gives stream items for each window ID w

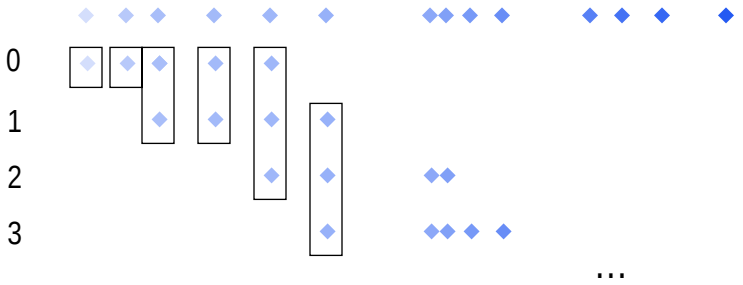
```
extent(w) =
  {i|w·10 ≤ i.A ≤ w·10+30}
```



Useful to have dual function

Gives window IDs for each item i

```
wids(i) =
  {w|⌈i.A/10⌉-1 < w < ⌈(i.A+30)/10⌉-1}
```



Can be basis for implementation

Tag data items with window IDs; treat ID as a regular attribute

Need a means, such as punctuation, to know when a window is complete

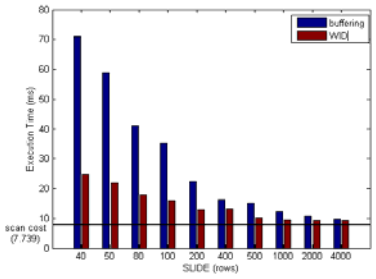


Figure 8 (b): Execution Time: WID versus Buffering – Zoom-in

Conclusion: What would make me like you

- Explain to me what I'm doing with reconstitution functions – trying to reason about the infinite with finite fragments?
- Understand the monotone/non-blocking/ reconstitution connection better.
- Automate the use of punctuation in stream query processing – for example, when can we prove space bounds?

- Find an effective way to describe disorder; effective =
  - can determine for a stream
  - can sythesize
  - can calculate latency vs. accuracy trade-offs
- Use properties of window specifications to determine properties of windowed operator implementaions.

Questions

Niagara West:

http://www.cse.ogi.edu/dot/niagara

Acknowledgements:

- Niagara Midwest (Wisconsin)
- Tim Sheard, Leo Fegaras, Ted Johnson
- Abilene Observatory
- NSF grant IIS 00-86002