

# Quantitative Policies over Streaming Data

Rajeev Alur

University of Pennsylvania

# Thanks to Collaborators



Dana Fisman



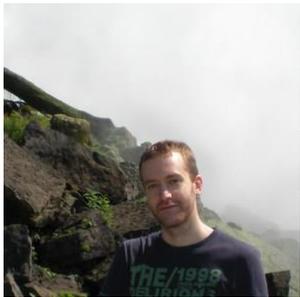
Zack Ives



Sanjeev Khanna



Boon Thau Loo



Kostas Mamouras



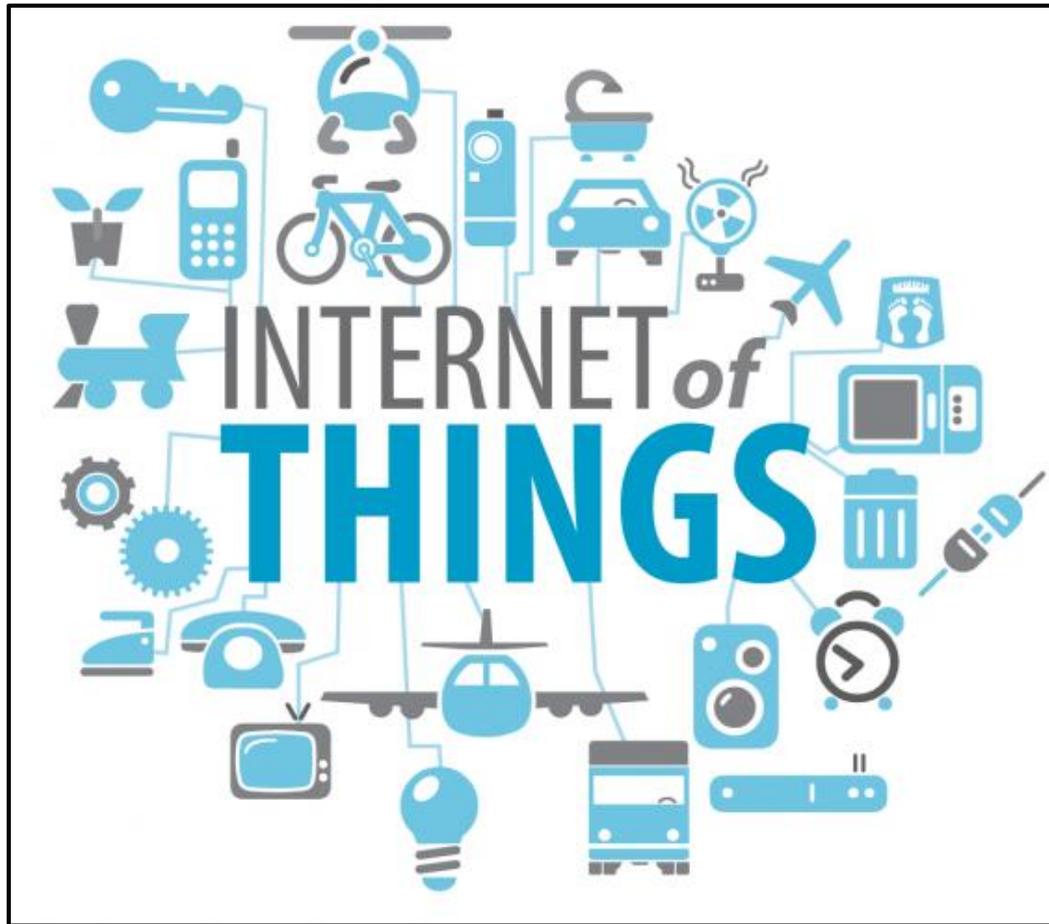
Mukund Raghothaman



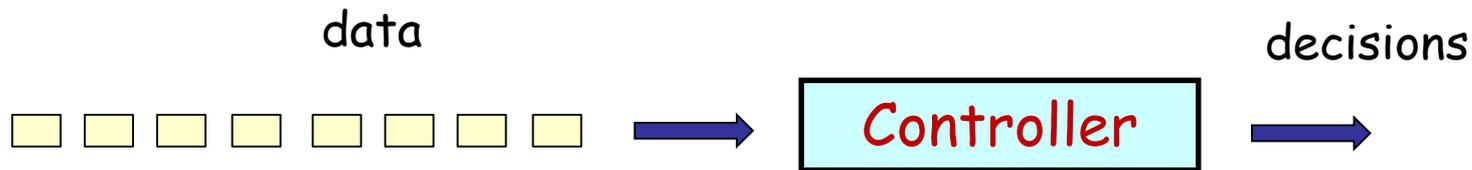
Caleb Stanford



Yifei Yuan

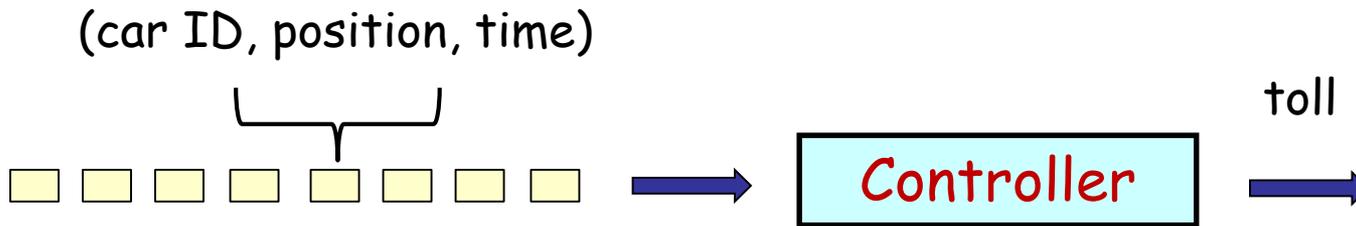


# Real-time Decision Making in IoT Applications



Smart buildings  
Network switches  
Autonomous medical devices  
Smart highways ...

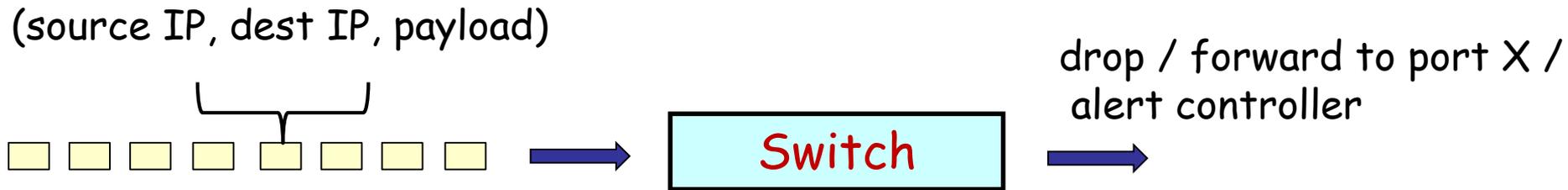
# Variable Tolling



Adjust toll rate at each toll booth dynamically based on time of day and congestion conditions in road segments

Reference: Linear road benchmark for stream management systems

# Network Traffic Engineering

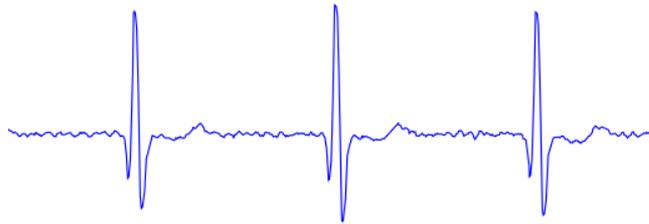


Dynamic network management for traffic engineering  
Real-time response to emerging attacks / security threats

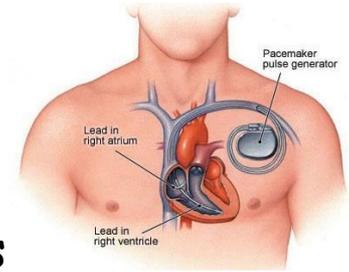
Software Defined Networking (SDN)

Opportunity for increased programmability/functionality

# Safety-critical CPS



pacing stimulus



Medical device software:

Need and opportunity for applying formal verification

Recent success in case studies (pacemaker, infusion pump)

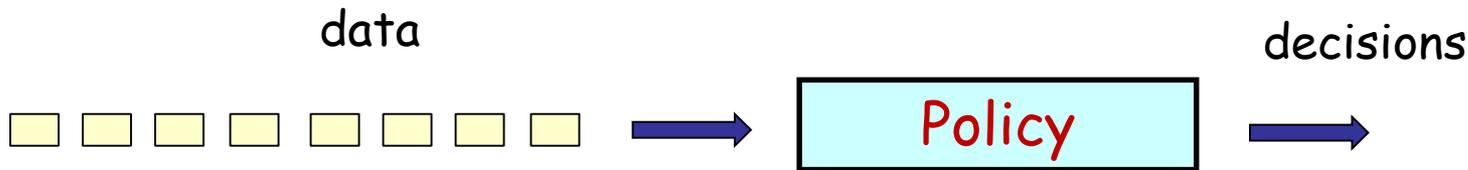
Verifying models much easier than verifying code

Higher-level programming abstractions →

Easier verifiability

Improved programmability

# Quantitative Policy



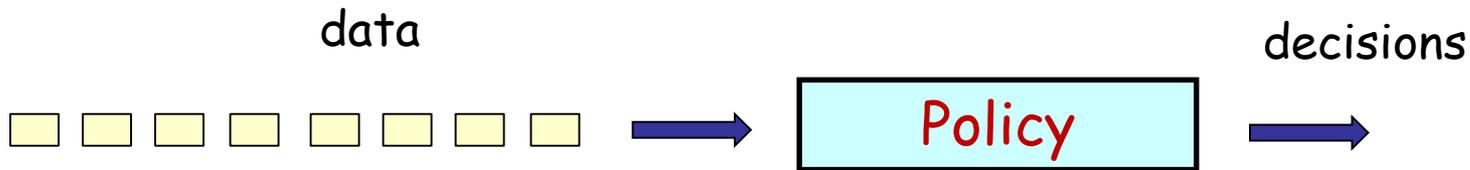
Example network policy:

if number of packets in current VoIP session exceeds the average over past VoIP sessions by a threshold  $T$ , then drop the packet

**Stateful:** Need to maintain state and update it with each item

**Quantitative:** Based on numerical aggregate metrics of past history

# Design and Implementation of Policies



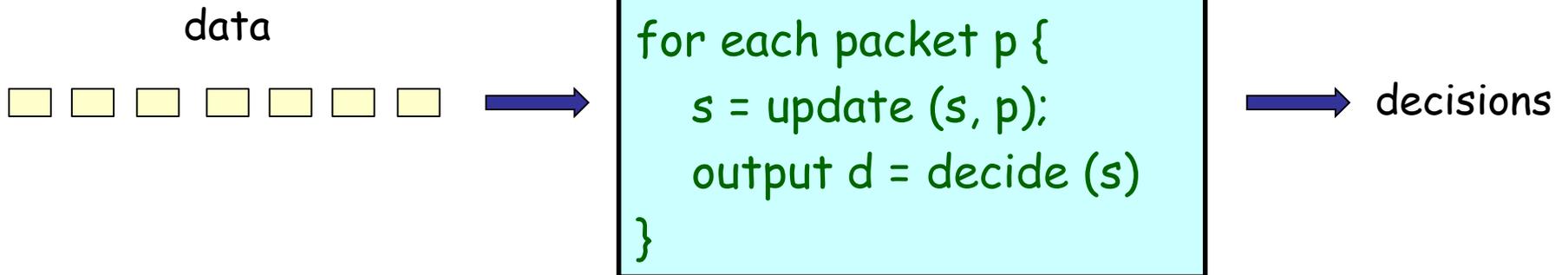
Which policies are effective ?

Based on traffic models and domain specific insights

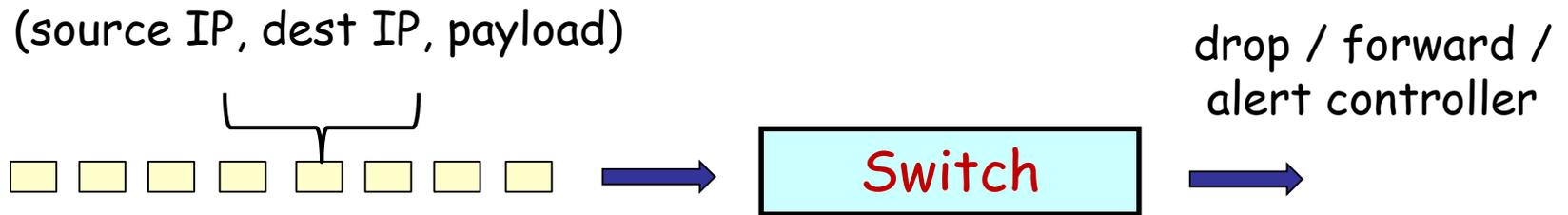
How to specify and evaluate policies ?

Focus of these lectures !

# Streaming Algorithm



# High-level Abstractions over Data Streams ??



Example network policy:

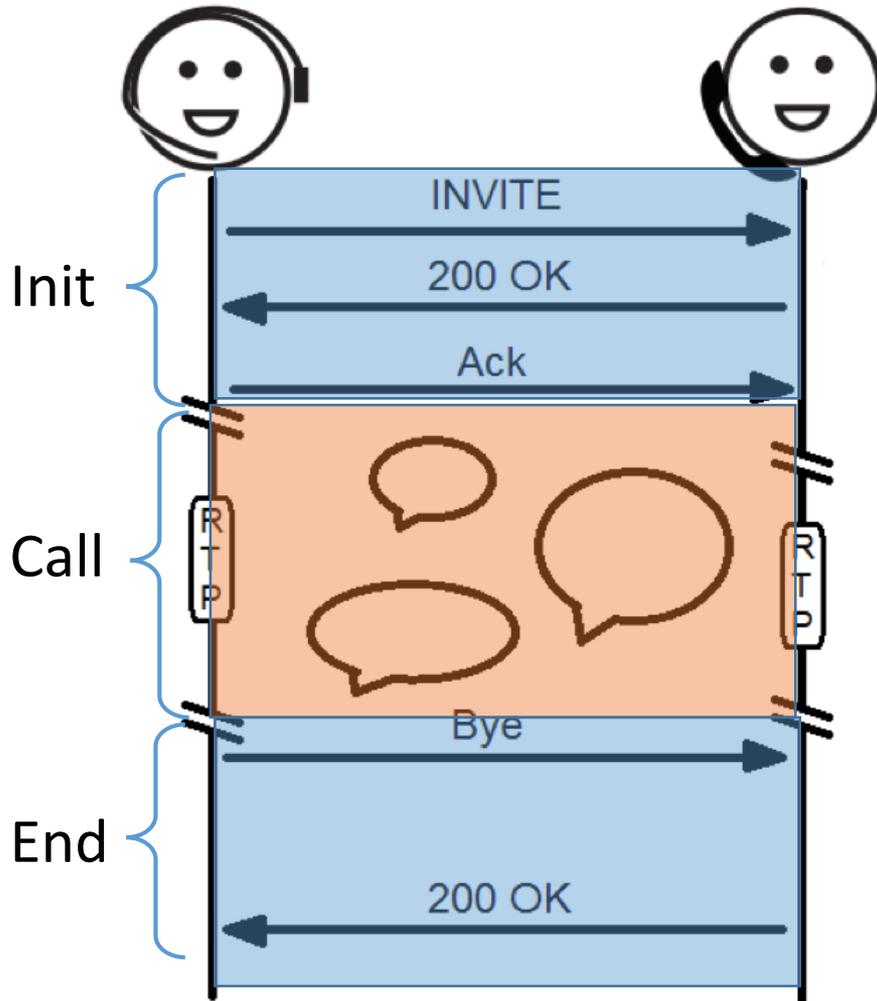
if number of packets in current VoIP session exceeds the average over past VoIP sessions by a threshold  $T$  then drop the packet

Low-level programming:

What state to maintain? How to update it?

Desired high-level abstraction: Beyond packet sequence

# Modular Specification of VoIP Session Monitor



Session Initiation Protocol

1. Focus on traffic between a specific source and destination
2. View data stream as a sequence of VoIP sessions
3. View a VoIP session as a sequence of three phases
4. Aggregate cost over call phase during a session, and aggregate cost across sessions

# Design Goals for Policy Language

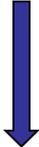


Programming abstractions for processing data stream ??

Policy spec

Theoretical foundations  
Expressiveness  
Optimization

Policy compiler



data



Policy code

decisions

Efficiency critical: Key parameters

1. Time to process each packet
2. State that needs to maintained

Ideally both should be constant or logarithmic in length of data stream

# Do We Need A New Policy Language ?

## State-based Languages

- Regular expressions
- Temporal logics
- Dataflow/synchronous languages

Application: Runtime monitoring

Quantitative extension:  
Weighted automata

## Relational languages

- SQL + Continuous queries
- Regular expressions +  
time windows to select events

Industrial-strength implementations

IBM Streams Processing Language

MSR StreamInsight / CEDR

# Lectures Outline

- ✓ Motivation
- ⇒ Quantitative Regular Expressions (QRE)
- QRE Compilation
- Experimental Evaluation
- Theory of Regular Functions
- Conclusions and Research Opportunities

# Illustrative Example: Patient Monitoring

Data items:

Begin episode 

Measurement 

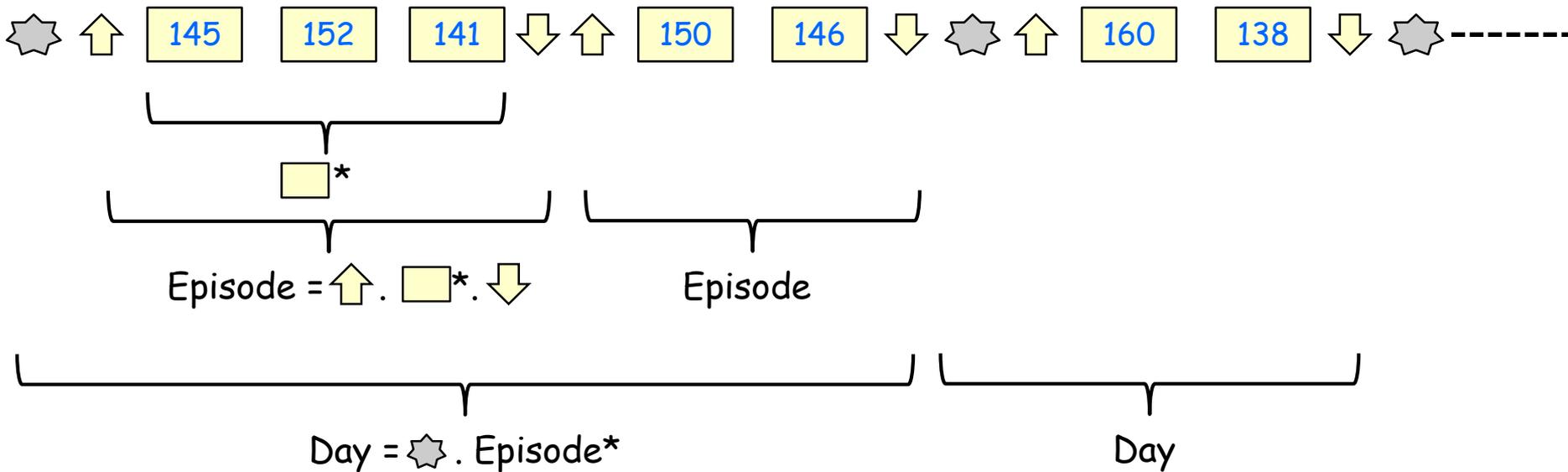
End episode 

End of day 



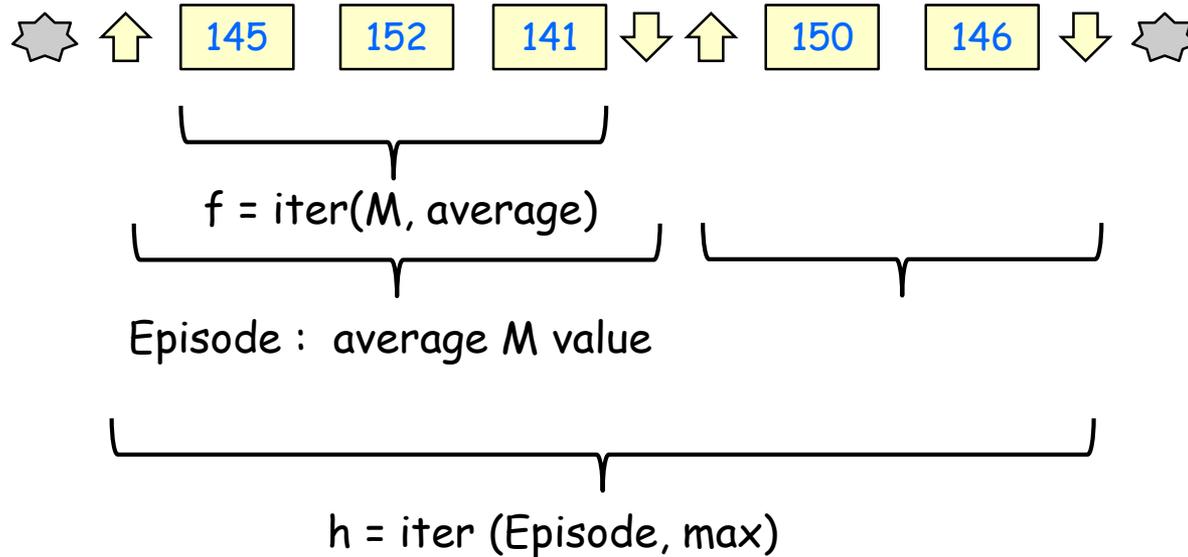
Output every day, maximum over episodes during that day,  
average measurement during the episode

# Regular Hierarchical Structure



Regular expressions is a natural match  
But need a quantitative extension !

# Quantitative Iteration



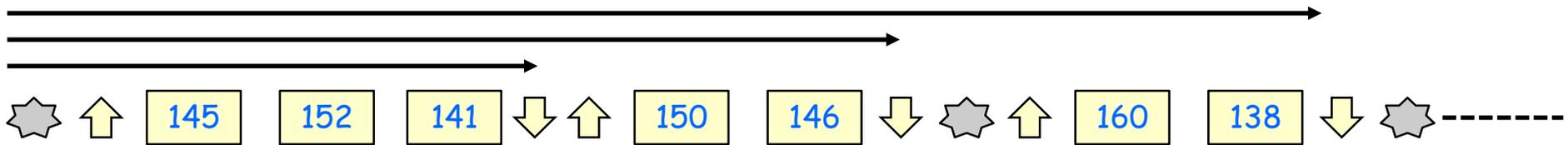
- Atomic function  $M$  maps an item, if it is a measurement, to its value
- Function  $f$  maps a sequence of measurements to its average
- Function  $\text{Episode}$  maps an episode to average measurement within it
- Function  $h$  maps a sequence of episodes to the maximum episode value

# Quantitative Regular Expressions

- Each QRE  $f$  maps a sequence of data items to a cost value  
 $f$  is a partial function from  $D^*$  to  $C$
- Sets  $D$  and  $C$  can be of arbitrary types with basic operations
- Example  $D$ : {  ,  ,  , v: N }
- Example  $C$ : Set of integers with constants, min, max, sum, average

# QRE Rate

- A QRE  $f$  is a partial function from  $D^*$  to  $C$
- $\text{Rate}(f)$  = Subset of  $D^*$  for which  $f$  is defined
- QRE produces output whenever input stream so far matches its Rate



- Rate = Data streams that end with a well-formed episode
- $\text{Rate}(f)$  captured by "symbolic" regular expression

$$D^*. (\uparrow. \square^*. \downarrow)$$

# Atomic QRE

➤ Each data domain  $D$  is equipped with a set of unary predicates

1. Satisfiability is decidable (supported by SMT-solver)

2. Set of predicates closed under Boolean operations

Ref: Symbolic automata and symbolic transducers (Veanes et al)

➤ QRE  $f : p(d) \rightarrow f(d)$  where  $p$  is unary predicate,  $f$  is data operation

If input data stream consists of a single item  $d$  satisfying  $p$ ,  
then return  $f(d)$

$$\text{Rate}(f) = p(d)$$

# Atomic QRE Examples

➤ Example D: {  ,  ,  , v: N }

➤ Example basic predicates:

d equals 

d equals v with  $v > 150$

➤ Example operations from D to C

$$f(\text{star}) = 0$$

$$f(\text{span style="border: 1px solid black; padding: 2px;">v}) = \min(80, v)$$

# Quantitative Concatenation: $\text{split}(f, g, \text{op})$

$f$  and  $g$  are QREs and  $\text{op}$  is a binary operation over costs (e.g.  $+$ ,  $\max$ )

Divide input data stream  $s$  into two parts  $s_1$  and  $s_2$  such that

$s_1$  matches  $\text{Rate}(f)$  and  $s_2$  matches  $\text{Rate}(g)$  and return  $\text{op}(f(s_1), g(s_2))$

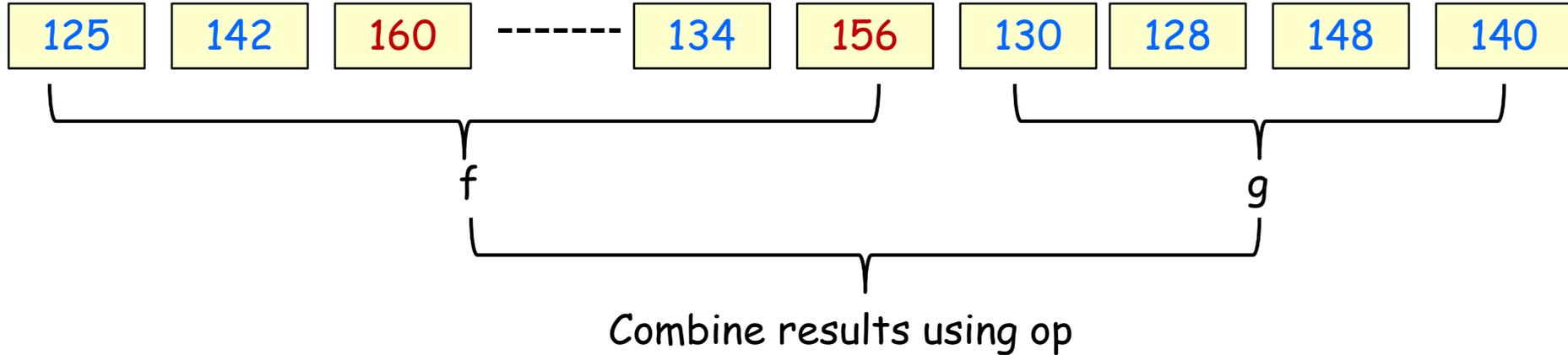
$\text{Rate}(\text{split}(f, g, \text{op})) = \text{Rate}(f) \cdot \text{Rate}(g)$

Key requirement: split must be unique (unambiguous)

Type checking requirement:

$\text{split}(f, g, \text{op})$  allowed only when if a stream matches  $\text{Rate}(f) \cdot \text{Rate}(g)$   
then there is exactly one way to split it

# Split Illustration

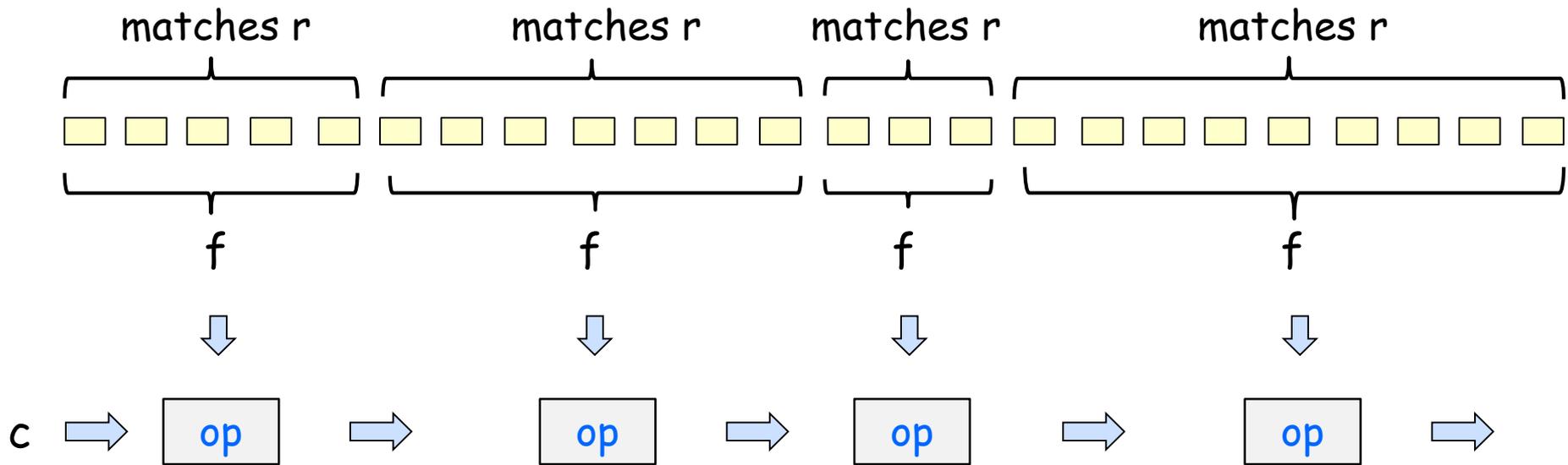


Rate(f) : Streams ending with a high-risk measurement (value > 150)

Rate(g) : Stream without high-risk measurements

# Quantitative Iteration: $\text{iter}(f, c, \text{op})$

$f$  is a QRE with rate  $r$ ,  $c$  is a constant, and  $\text{op}$  is a binary operation

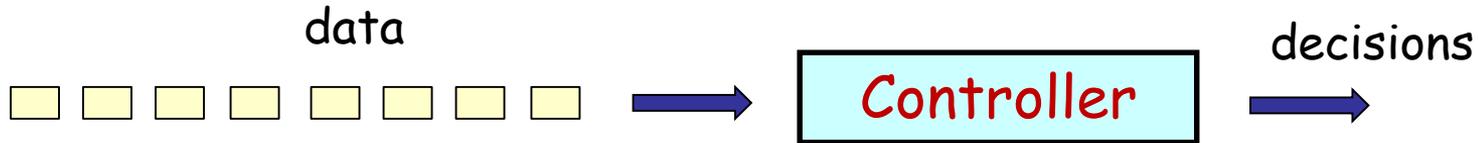


# Quantitative Iteration: $\text{iter}(f, c, \text{op})$

- $f$  is a QRE with rate  $r$ ,  $c$  is a constant, and  $\text{op}$  is a binary operation
- Divide input data stream  $s$  into multiple parts  $s_1, s_2, \dots, s_k$  such that each  $s_i$  matches  $r$ , apply  $f$  to each part, and return
$$\text{op}(\text{op}(\dots \text{op}(\text{op}(c, f(s_1)), f(s_2)), \dots \dots, f(s_k)))$$
- $\text{Rate}(\text{iter}(f, c, \text{op})) = \text{Rate}(f)^*$
- Allowed when the split is guaranteed to be unique
- Special case:  $\text{op}$  is set-aggregator (apply  $\text{op}$  to "set" of returned values)  
max, min, sum, average, median, standard deviation ...
- Order dependent: Linear interpolation, Discounted sum

# Choice: f else g

Given a stream  $s$ , if  $f(s)$  is defined, return it, else return  $g(s)$



Example:  $f$  makes decisions for a stream that does not contain high-risk measurements (e.g. with value  $> 150$ ), and  $g$  makes decisions for streams that do contain such measurements

Benefit: Test based on a **global** property of stream

Strong typing restriction:

Allowed only when  $\text{Rate}(f)$  and  $\text{Rate}(g)$  are disjoint

$\text{Rate}(f \text{ else } g) = \text{Rate}(f) \cup \text{Rate}(g)$

# Key-based Partitioning

Suppose stream contains events for both Alice and Bob



Suppose we want to compute for each patient, whether the daily summary (max over episodes, average measurement during episode) exceeds a threshold value

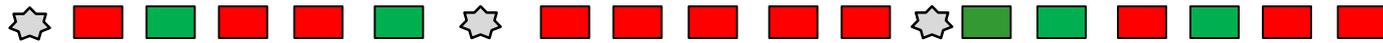
QRE  $f$  maps stream of single-patient events to daily summary

Modular programming: Partition input stream into multiple streams, one for each patient identifier, and apply  $f$  to each

Challenges: How to synchronize outputs of different partitions?

What is the type of combined outputs?

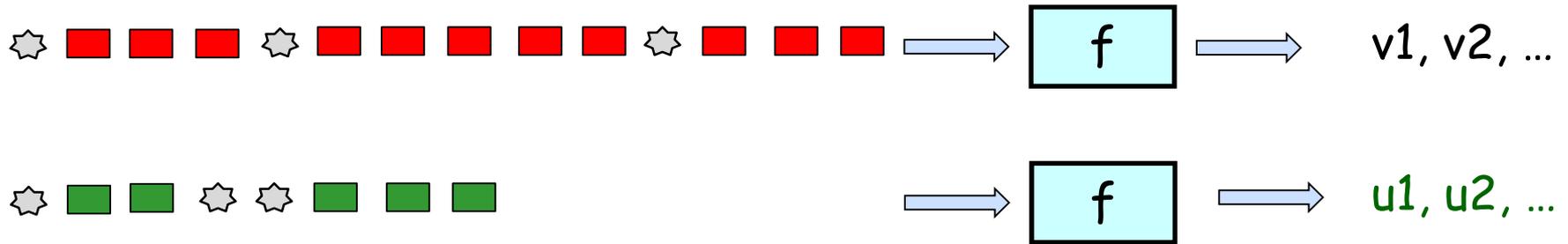
# Map-collect illustration



QRE  $f$  computes daily summary for single-patient input streams

Synchronization item: end-of-day  $\star$

$g = \text{map-collect}(f, \star^*)$  i.e. produce joint output at end of each day



Output of  $g$ :  $\{v_1, u_1\}, \{v_2, u_2\}, \dots$

Type of output: set of values produced by each thread tagged with key

# Key-based Partitioning: map-collect

Type D of data items =  $D_s \cup [D_k \times D_v]$

Each item is a synchronization item or of the form (key, value)

QRE  $f$  maps streams over  $D_v$  to output values  $C$

QRE  $g = \text{map-collect} ( f, r )$ ,  $r$  is a symbolic reg-exp over  $D_s$

QRE  $g$  processes streams over  $D$ :

if item is in  $D_s$  then send it to all threads/partitions

if item =  $(k,v)$ , send it to the thread/partition for key  $k$

whenever  $r$  holds, collect outputs of all threads

Output type = Relation (multi-set) over  $D_k \times C$

# Output Composition

Suppose  $g$  outputs each day set of tuples (patient-id, daily summary)

Want to output set of patients for which daily summary  $\geq 160$

Select : Relation (PID  $\times$  V)  $\rightarrow$  Relation (PID)

Select ( I ) = { p | there is v such that (p,v) is in I and v  $\geq 160$  }

Then desired QRE h is Select(g)

Count(h) outputs number of high-risk patients each day

If  $f_j : D^* \rightarrow C_j$  are QREs with **equivalent** rates  $r$ , and  $op: C_1 \times \dots \times C_n \rightarrow C$

Then  $op(f_1, \dots, f_n) : D^* \rightarrow C$  with rate  $r$

# Streaming Composition

Suppose  $h$  outputs each day number of high-risk patients

Want to output the daily average number of high-risk patients so far

$h'$  maps sequence of numbers to average

$h' = \text{iter}(\text{id}, \text{average})$ , where  $\text{id}$  is the identity function

Then desired QRE is  $h \gg h'$

If  $f : D^* \rightarrow C$  and  $g : C^* \rightarrow B$  are QREs, then  $f \gg g : D^* \rightarrow B$

Stream sequence of outputs of  $f$  as input stream to  $g$

Note: rate  $(f \gg g)$  is a subset of  $D^*$ , and may not be regular

Current solution: allow  $\gg$  only at top-level

# Quantitative Regular Expressions Summary

- Each QRE  $f$  maps a sequence of data items to a cost value  
rate( $f$ ) specifies when  $f$  produces outputs  
given by symbolic regular expression
- Core combinators:
  - Atomic QRE:  $p(d) \rightarrow f(d)$
  - Quantitative concatenation:  $\text{split}(f, g, \text{op})$
  - Quantitative iteration:  $\text{iter}(f, c, \text{op})$
  - Choice:  $f \text{ else } g$
  - Key-based partitioning:  $\text{map-collect}(f, r)$
  - Output composition:  $\text{op}(f_1, \dots, f_n)$
  - Streaming composition:  $f \gg g$
- Type checking rules check compatibility of rates (decidable!)

# Type Checking and Compilation

# Symbolic Regular Expressions

- Similar to traditional regular expressions.

$e ::= a \mid$  [letter from alphabet]

$e \cup e \mid$  [choice]

$e.e \mid$  [concatenation]

$e^*$  [iteration]

- What if alphabet large or unbounded?

- "Symbolic" regular expressions:

unary predicates instead of letters

- Examples of symbolic REs:

$\text{even}(n)^*.\text{odd}(n)$

$(n = 0).(n > 0)^*$

# Symbolic Regular Expressions

- Symbolic regular expressions:

$e ::= p \mid$  [unary predicate]

$e \cup e \mid$  [choice]

$e.e \mid$  [concatenation]

$e^*$  [iteration]

- Predicates:

Closed under Boolean operations

Decidable satisfiability

- E.g., alphabet  $\mathbb{N}$  (the set of natural numbers).

- Possible sets of predicates:

Presburger arithmetic

linear integer arithmetic

- Can be decided with SMT solver.

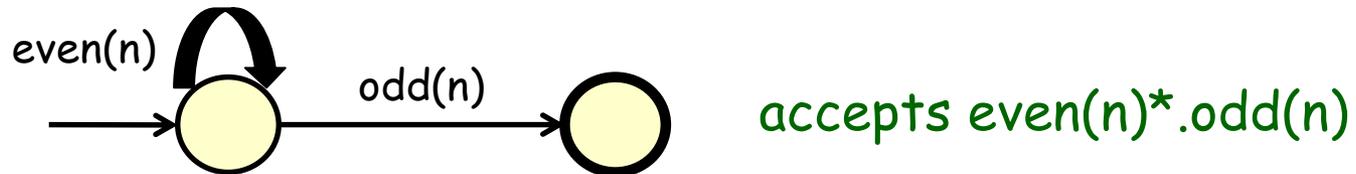
- Cannot handle: full arithmetic with multiplication (**UNDECIDABLE**)

# Symbolic Automata

- Traditional automata: transitions annotated with letters.



- Symbolic automata: transitions annotated with unary predicates.



- Translation from expressions to automata is the same.

# Product Construction

- Traditional automata  $A = (Q, \Delta, I, F)$  and  $A' = (Q', \Delta', I', F')$ .
- The product  $A \times A'$  has states  $Q \times Q'$ , initial states  $I \times I'$ , final states  $F \times F'$ , and transitions

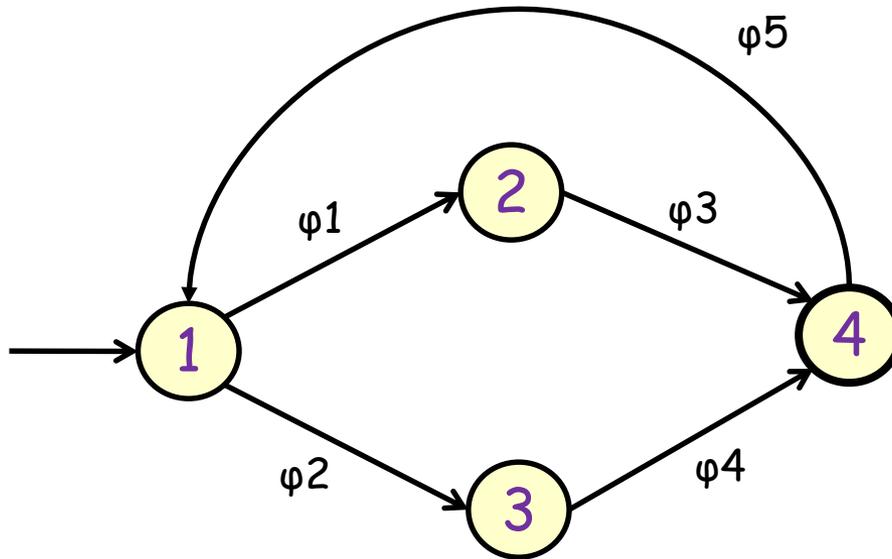
$$(p,p') \rightarrow^a (q,q'),$$

when  $p \rightarrow^a p'$  is transition of  $A$  and  $q \rightarrow^a q'$  is transition of  $A'$ .

- Suppose now that  $A$  and  $A'$  are symbolic.
- If  $p \rightarrow^\varphi p'$  is transition of  $A$  and  $q \rightarrow^\psi q'$  is transition of  $A'$ , then
$$(p,p') \rightarrow^{\varphi \ \& \ \psi} (q,q')$$
is a transition of  $A \times A'$ .
- **BUT:** If  $(\varphi \ \& \ \psi)$  **unsatisfiable**, the transition can be eliminated.

# Symbolic Automata: Reachability

- Essentially the same as graph reachability.



- Satisfiability check to see if an edge can be traversed.
- Reachability solves non-emptiness.

# Type Checking: Atomic QRE

- Input type  $D$ , output type  $C$ .
- $p(d)$  is unary predicate on  $D$ .
- Operation  $op: D \rightarrow C$ .
- Atomic QRE:

$$p(d) \rightarrow op(d): D^* \rightarrow C.$$

- Type checking:

“ $p(d)$  is satisfiable”

HOW: one invocation of SMT solver

# Type Checking: Quantitative Concatenation

➤ QREs  $f: D^* \rightarrow A$  and  $g: D^* \rightarrow B$ .

➤ Operation  $op: A \times B \rightarrow C$ .

➤ Quantitative concatenation:

$split(f, g, op): D^* \rightarrow C$ .

➤ Type checking:

"Rate( $f$ ) and Rate( $g$ ) are unambiguously concatenable"

HOW: unambiguity check for Rate( $f$ ).Rate( $g$ )

# Type Checking: Quantitative Iteration

- QRE  $f: D^* \rightarrow A$ .
- Constant  $c$  of type  $C$ .
- Binary operation  $op: C \times A \rightarrow C$ .
- Quantitative iteration:  
$$\text{iter}(f, c, op): D^* \rightarrow C.$$
- Type checking:  
"Rate( $f$ ) is unambiguously iterable"  
HOW: unambiguity check for Rate( $f$ )\*

# Type Checking: Global Choice

➤ QREs  $f: D^* \rightarrow C$  and  $g: D^* \rightarrow C$ .

➤ Global choice:

$f \text{ else } g: D^* \rightarrow C$ .

➤ Type checking:

"Rate( $f$ ) and Rate( $g$ ) are disjoint"

HOW: intersection of Rate( $f$ ) and Rate( $g$ ) empty

# Type Checking: Output Composition

➤ QREs  $f: D^* \rightarrow A$  and  $g: D^* \rightarrow B$ .

➤ Binary operation  $op: A \times B \rightarrow C$ .

➤ Output composition:

$$op(f, g): D^* \rightarrow C.$$

➤ Type checking:

“Rate( $f$ ) and Rate( $g$ ) are equivalent”

HOW: equivalence algorithm of Stearns and Hunt (FOCS '81)

See also:

Minimization of Symbolic Automata by D'Antoni & Veanes (POPL '14)

# Type Checking: Map-Collect

- Input type  $D = D_S \cup [ D_K \times D_V ]$
- $D_S$  = Synchronization elements
- $D_K$  = Keys, and  $D_V$  = values.
- QRE  $f: D^* \rightarrow C$ , symbolic RE  $R$  over  $D_S$ .
- Map-collect QRE:

$$\text{map-collect}(f, R): D^* \rightarrow \text{Rel}(D_K \times C).$$

- Type checking:

"Rate(f) is contained in expansion of R to D"

HOW: inclusion algorithm of Stearns and Hunt (FOCS '81)

# Type Checking: Summary

- Atomic:  $p(d)$  is satisfiable.
- Split:  $\text{Rate}(f)$  and  $\text{Rate}(g)$  are unambiguously concatenable.
- Iter:  $\text{Rate}(f)$  is unambiguously iterable.
- Else:  $\text{Rate}(f)$  and  $\text{Rate}(g)$  are disjoint.
- Op():  $\text{Rate}(f)$  and  $\text{Rate}(g)$  are equivalent.
- Map-collect:  $\text{Rate}(f)$  is contained in  $R$ .
  - All problems can be decided in time that is **polynomial** in sizes of expressions and number of minterms over predicates.  
(assuming satisfiability checks take unit time)
- Automaton for  $\text{Rate}(f)$  is nondeterministic but **unambiguous**.
- No need for determinization (no exponential blowup).
- RE equivalence: PSPACE.
- Unambiguous RE equivalence: P.

# Goals for Compiler



QRE

QRE compiler



data



```
state s = initialize;  
for each packet p {  
  s = update (s, p);  
  output d = decide (s)  
}
```

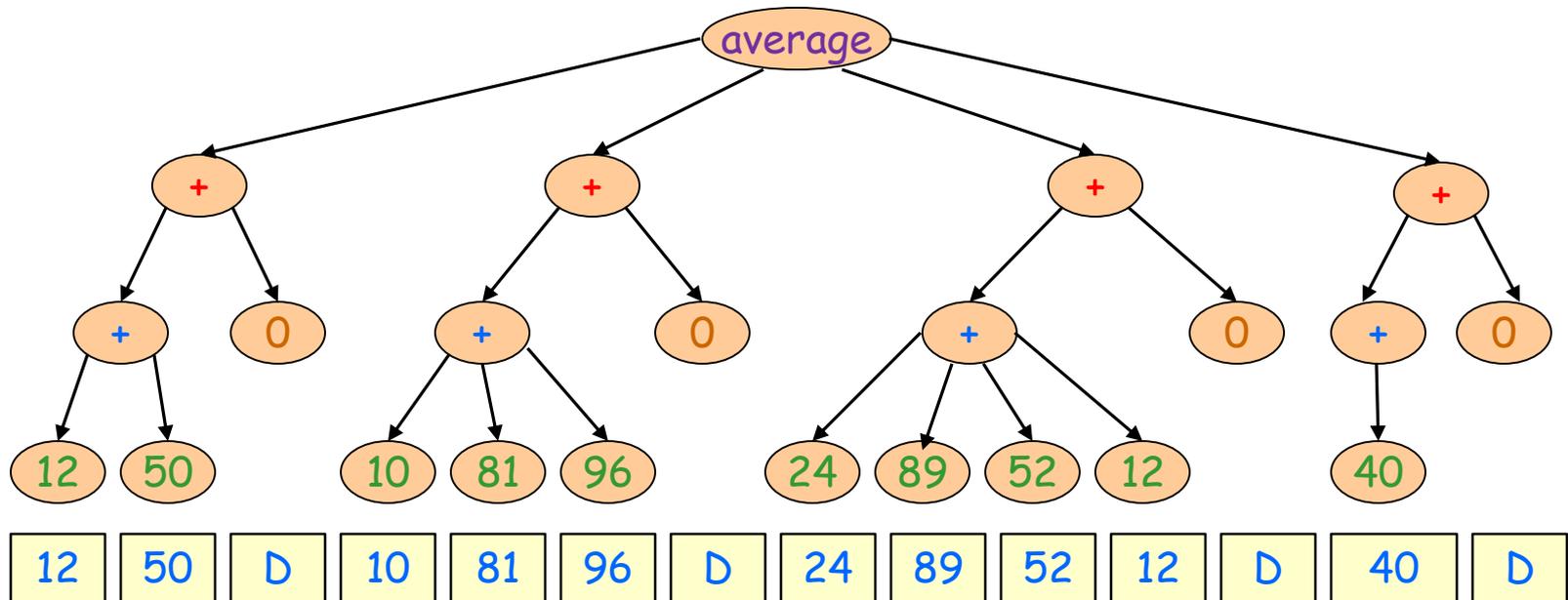


decisions

Optimize bits needed to store state and time for update  
Ideally independent of length of data stream

# QRE Evaluation → Hierarchical Expression

Average measurement per day: iter (split (iter (M, +) , D, +) , average)



Computing  $f(s)$ , where  $f$  is a QRE and  $s$  is input stream, amounts to evaluating an expression tree of size linear in length of  $s$

# Stack-based Evaluation

- Incremental evaluation of expression:

Maintain state as a stack

Perform intermediate computations as soon as possible

Stack elements correspond to nodes of the expression tree

Evaluating + : sum of values seen so far

Evaluating average : sum and count of values seen so far

- Resources (total space / per-item processing time):

[ Depth of expression tree (dependent only on QRE size) ]

Times [ resources needed at each node of expression tree ]

# Approximation



Suppose we want to compute average of numbers in a streaming fashion  
Need to remember total sum (73) and count of items (5) so far

Suppose we want to compute median of numbers

To ensure exact answer, must remember all numbers seen so far

Exact algorithm for median:

Maintain the multiset of items seen so far.

Implementation 1: Extensible array of counts.

Implementation 2: Map as balanced binary search tree

(key: item, value: count)

# Approximation



Approximation algorithm for median:

Map each number  $n$  to bucket  $k$  such that  $(1+\epsilon)^k \leq n < (1+\epsilon)^{k+1}$

Maintain for each bucket, count of numbers mapped to that bucket

Number $n$	Bucket $k$	Number $(1+\epsilon)^k$	Error
50	393	49.923	0.154%
100	462	99.192	0.808%
80,000	1134	79,512.950	0.609%
1,200,000	1406	1,190,834.857	0.764%

Space needed:  $\log_{1+\epsilon}(U) \approx \epsilon^{-1} \cdot \log(U)$ ,

where  $U$  is the range of numerical values

# Approximation



Approximation algorithm for median:

Map each number  $n$  to bucket  $k$  such that  $(1+\epsilon)^k \leq n < (1+\epsilon)^{k+1}$

Maintain for each bucket, count of numbers mapped to that bucket

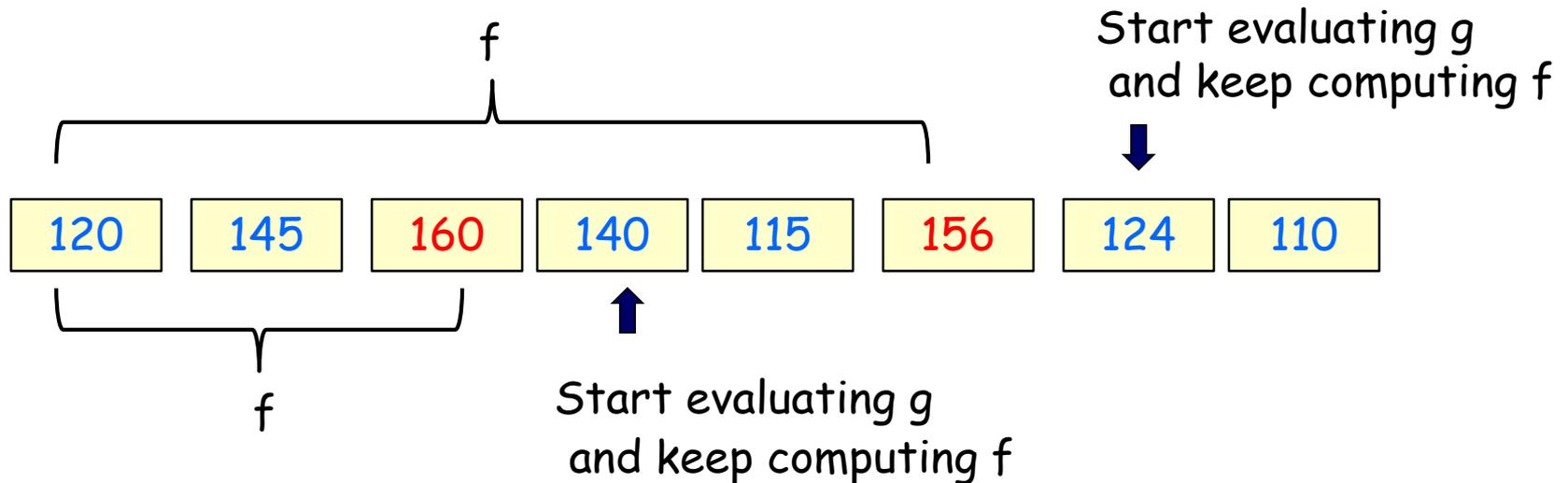
Number $n$	Bucket $k$	Number $(1+\epsilon)^k$	Error
50	393	49.923	0.154%
100	462	99.192	0.808%
80,000	1134	79,512.950	0.609%
1,200,000	1406	1,190,834.857	0.764%

Approximation error: Multiplicative factor of  $\epsilon$ .

$$n' \leq n < (1+\epsilon) n' \Rightarrow 0 \leq n - n' < \epsilon n' \Rightarrow 0 \leq (n - n')/n < \epsilon$$

# Online Computation of Split Points

To process  $\text{split}(f,g,+)$ , find the position where  $f$  ends and  $g$  starts  
Domain of  $f$  : Streams ending with high-risk measurements ( $\text{val} > 150$ )

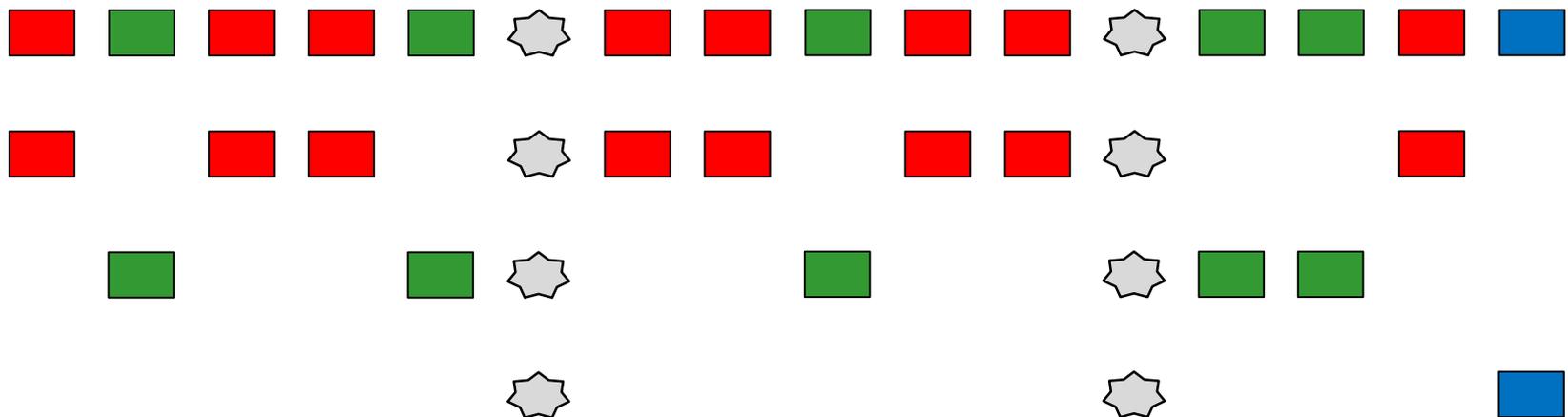


Need to maintain multiple parallel computations of same subexpression initialized at different positions in input stream

Insight: number of parallel copies is bounded (bound depends on query)

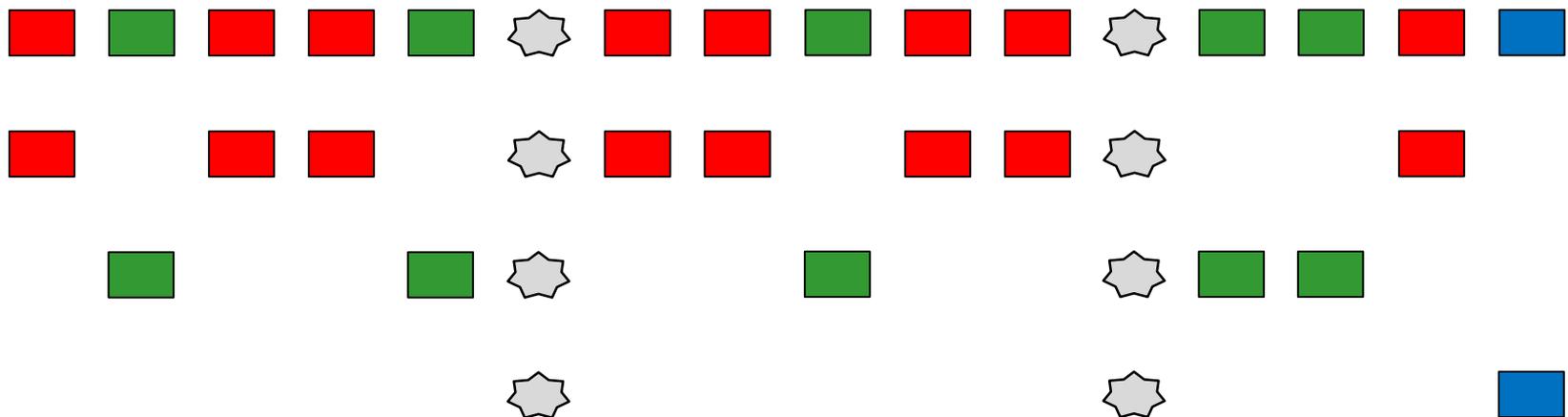
# Map-collect Evaluation

- To evaluate  $\text{map-collect}(f, r)$ , for each new key encountered, a new "thread" evaluating  $f$  must be initialized
- Synchronization items must be input to all threads  
Even to those whose keys have not appeared yet
- IDEA: Maintain a special thread receiving only synchronization items  
Fork that thread when a new key appears



# Map-collect Evaluation

- Collecting outputs of all threads when input matches rate  $r$  requires careful implementation
- Resources needed: (Resource for  $f$ )  $\times$  (Number of active copies)
- Amenable to high-performance distributed implementation (STORM)



# QRE Compiler Summary

- Given a QRE, compiler first checks all typing rules are met  
(e.g. when split is applied, the splitting must be unambiguous)
- Then it compiles it into an executable streaming algorithm
- General case: Memory used is linear in length of stream
  
- If numerical operators are min, max, sum, average, and no map-collect,  
then constant memory and constant per item processing time
  
- If, in addition, median is also used, then
  - $\log U$  memory, where  $U$  is (dynamically updated) range of values
  - constant time to process each item
  - user specified multiplicative factor of approximation error

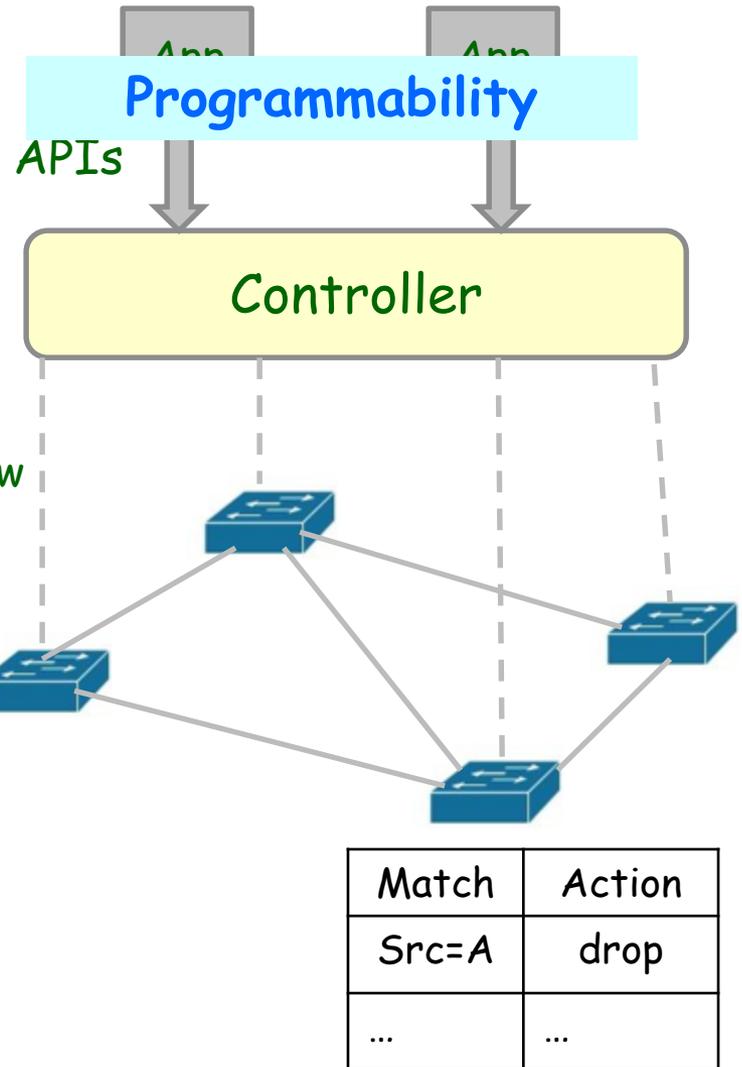
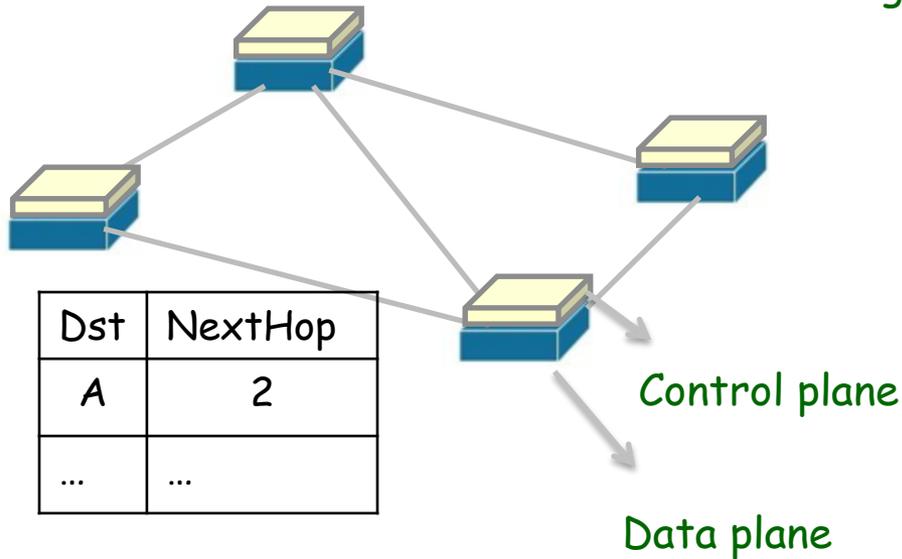
# Implementation and Experimental Evaluation

- StreamQRE Java Library (PLDI 2017)
- NetQRE for network traffic engineering (SIGCOMM 2017)

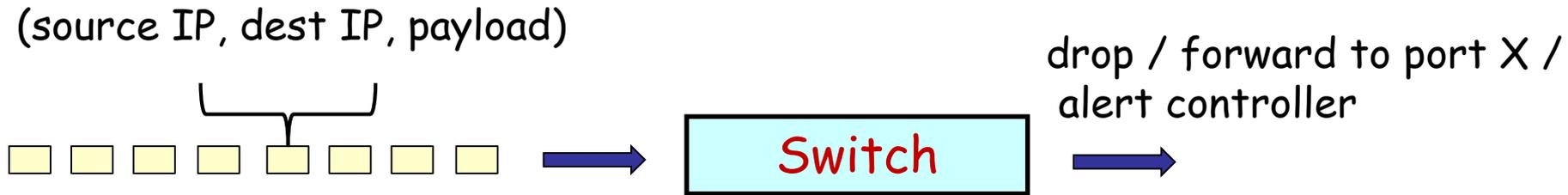
# Software Defined Networking

Distributed  
Protocols

e.g. POX, NOX,  
Floodlight



# NetQRE Language



Domain-specific extension/adaptation of core QRE

Basic types: ports, IP addresses, tests of packet fields

Actions on packets: drop, flood, forward, augmentation with fields...

Reference to time windows (e.g. stream of packets in last 5 sec)

Basic functions on packets (written in C) +

QRE combinators (else, split, iter, max, min, sum, average) +

Keys: IP addresses

# Implementation and Evaluation

## NetQRE Compiler

+ NetQRE Runtime system (to process packets and update state)

1. Can network policies be expressed in concise and intuitive manner ?
2. Is compiled code efficient for throughput and memory footprint ?
3. Can our system be used for real-time monitoring and alerting ?

## Flow-level traffic measurements

e.g. detection of heavy hitters, super spreaders

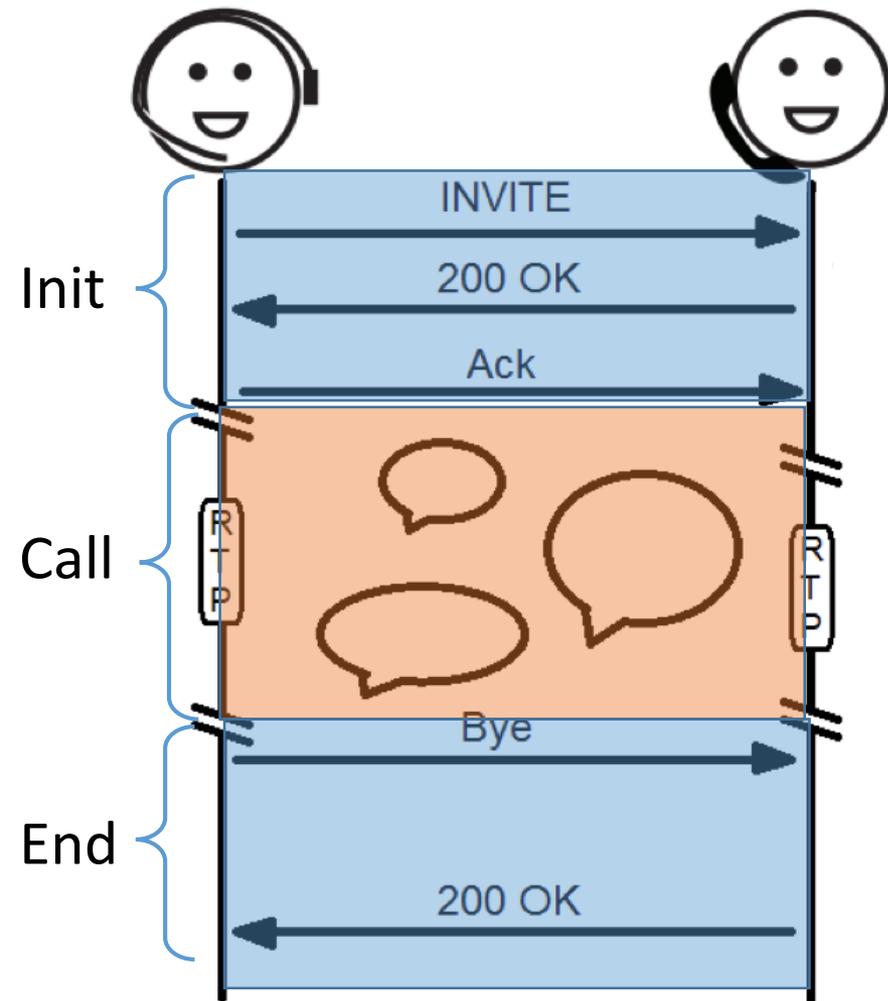
## TCP state monitoring

e.g. aggregate statistics of TCP connections, detect SYN flood attack

## Application level monitoring

e.g. collect statistics about VoIP sessions

# Monitoring of VoIP Sessions



Session Initiation Protocol

Detect if current VoIP session is using excessive bandwidth compared of past average

Modular specification using  
Map-collect on IP-addresses  
Split and Iter constructs  
Aggregation across users  
Aggregation across sessions

18 lines of NetQRE code  
(vs 100s of lines C++ code)

# Throughput and Memory Footprint

How does NetQRE generated code compare with hand-crafted code?

Example: Detection of heavy hitters

(a source IP address has consumed  $> K$  bandwidth in past  $T$  sec)

Workload: CAIDA traffic trace of  $\sim 50$  million packets

Throughput (million packets per second)

Manual: 18.5 vs NetQRE: 18.3

Upto 10x faster than systems such as Bro and Opensketch

Memory: Manual: 14 MB vs NetQRE: 15.1 MB

Summary for other queries (measured for 20 queries)

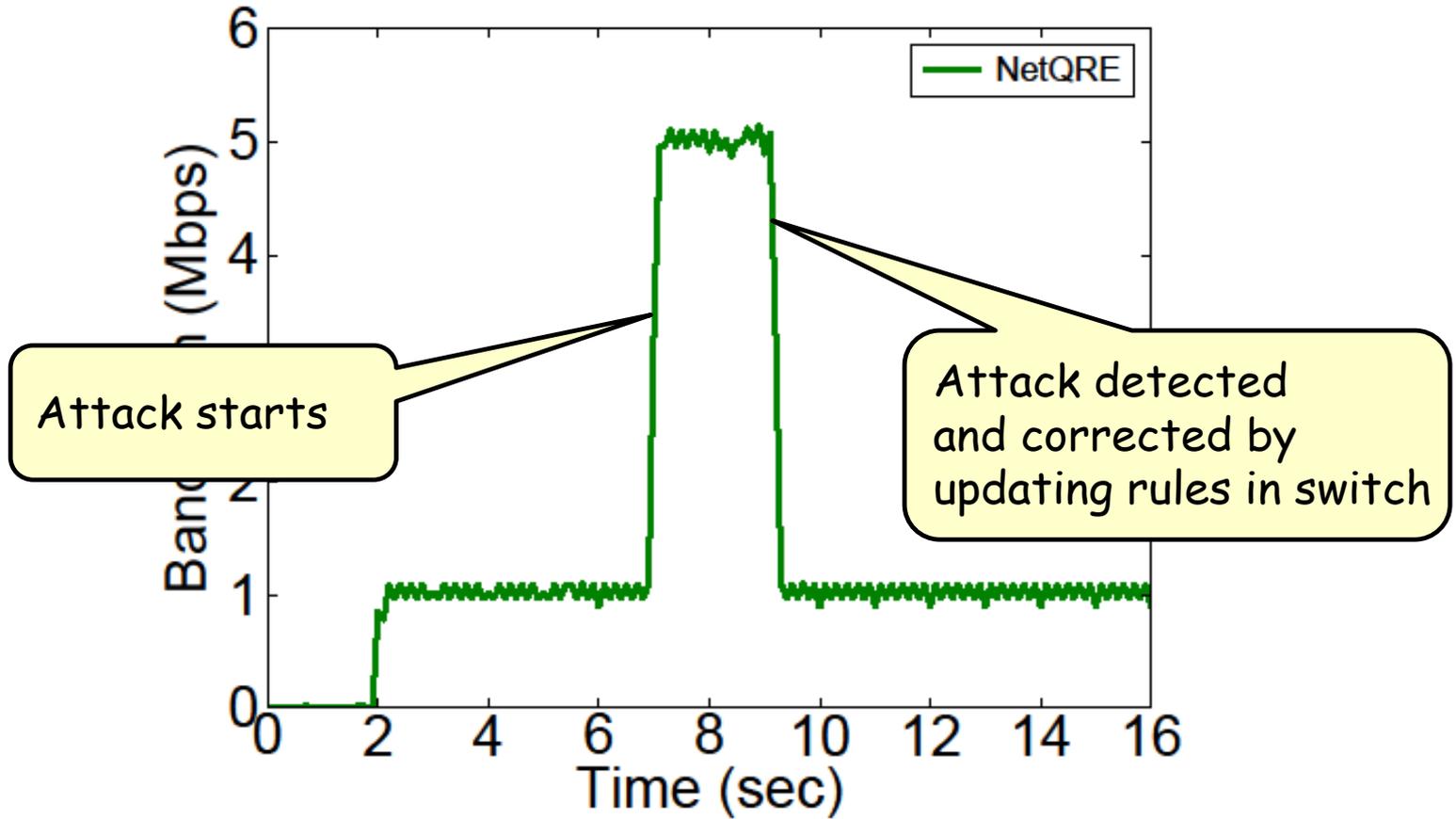
Throughput within 4% overhead

SYN flood attack: NetQRE uses twice as much memory

# Real-Time Response

- Experimental setup:
  - Network of two clients and one SDN switch
  - SDN Controller based on POX
  - Network emulated by Mininet with link bandwidth 100 Mbps
- How long does it take to detect an attack and block traffic ?
  - Note: correction requires SDN controller to update rules on switch
- Incomplete TCP handshake:
  - SYN packet, followed by matching SYNACK, but no subsequent ACK
- SYN flood attack: Too many incomplete TCP handshakes

# SYN Flood Attack



# StreamQRE Java Library

- StreamQRE: Strong theoretical efficiency guarantees.
- Performance for practical workloads?
- Implementation of StreamQRE as a library in Java

## **Yahoo Streaming Benchmark (2015)**

Interaction of web users with advertisements

**Event**(userId, pageId, adId, eventType, eventTime)

## **NEXMark Benchmark (2002)**

Monitoring of an online auction system (e.g., eBay)

**NewPerson**(personId, name, timestamp)

**Auction**(itemId, sellerId, initPrice, timestamp, duration, category)

**EndAuction**(itemId, timestamp)

**Bid**(itemId, bidderId, bidIncrement, timestamp)

# Experimental Evaluation

- StreamQRE: Strong theoretical efficiency guarantees.
- Performance for practical workloads?

## **StreamQRE**

Streaming extension of  
Quantitative Regular  
Expressions

## **RxJava**

**(ReactiveX for Java)**

API for observable  
streams

## **Esper for Java**

SQL-like language with  
Complex Event  
Processing features

## **Flink**

Distributed Stream  
Processing Framework

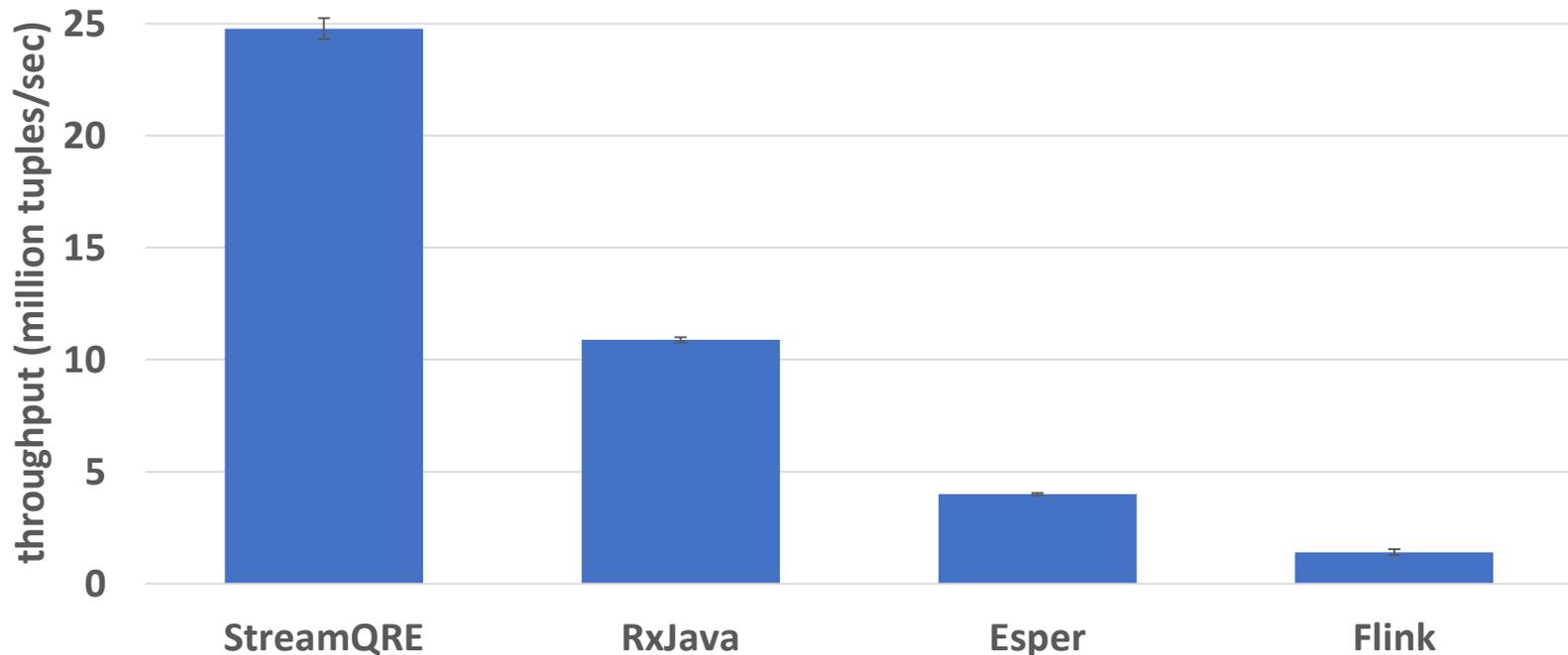
Popular and actively maintained engines with Java implementation.

Rich high-level APIs for stream processing.

# Experimental Evaluation

Time-based window with nested key-based partitioning:  
“Compute every second the number of views associated with each ad campaign”.

Yahoo Benchmark - Query 1



# Experimental Evaluation

## Slowdown compared to StreamQRE

	<b>RxJava</b>	<b>Esper</b>	<b>Flink</b>
<b>Yahoo 1</b>	2.3	6.2	18
<b>Yahoo 2</b>	3.6	6.7	9.8
<b>NEXMark 1</b>	4.3	76	141
<b>NEXMark 2</b>	2.1	22	42
<b>NEXMark 3</b>	2.1	21	42
<b>NEXMark 4</b>	2.0	27	35
<b>NEXMark 5</b>	2.6	18	33

The StreamQRE engine has good performance.

- Consistently faster than RxJava (about 2-4 times).
- Much faster than Esper (6-70 times) and Flink (10-140 times).

# Theory of Regular Functions

# Language Classes in Complexity Theory

--- Recursive  
--- NP  
--- P  
--- Linear-time  
--- Regular

What if we consider functions?

From strings to natural numbers

From strings to strings

No essential change for

Recursive, NP, P, linear-time...

# Expressiveness of QREs

Do we have enough operators?

Is expressiveness of QREs robust?

## Regular languages

- Regular expressions
- Deterministic finite automata
- Monadic second-order logic MSO

Beautiful well-understood theory

## Regular functions

parameterized by cost operations

- Quantitative regular expressions
- Cost register automata (CRA)
- MSO-definable string to term transformations

Emerging theory (open problems...)

# Mapping Strings to Costs

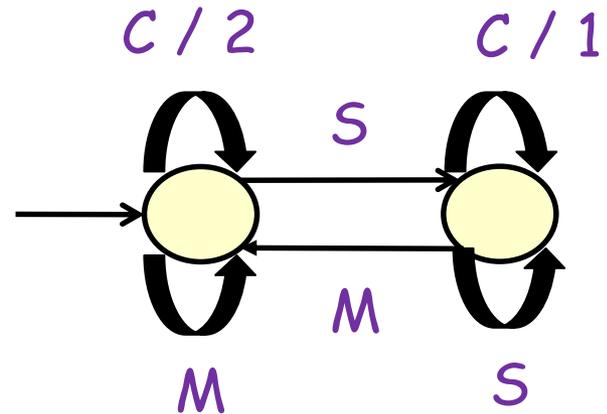
- Each QRE  $f$  maps  $\Sigma^*$  to  $D$
- Cost domain  $D$  has a basic set of operations
- Combinators:
  - Atomic QRE:  $a \rightarrow c$
  - Quantitative concatenation:  $\text{split}(f, g, \text{op})$
  - Quantitative iteration:  $\text{iter}(f, c, \text{op})$
  - Choice:  $f \text{ else } g$
  - ~~Key-based partitioning:  $\text{map-collect}(f, r)$~~
  - Output composition:  $\text{op}(f_1, \dots, f_n)$
  - ~~Streaming composition:  $f \gg g$~~

# Finite Automata with Cost Labels

C: Buy Coffee

S: Fill out a survey

M: End-of-month



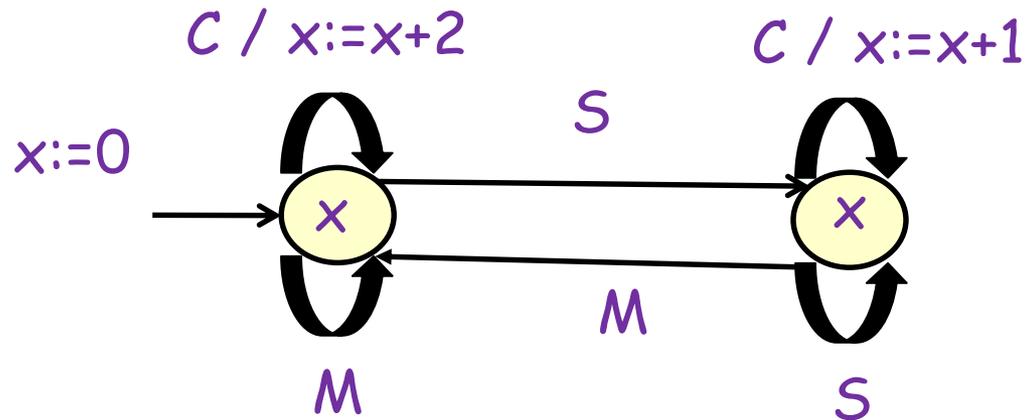
Maps a string over  $\{C,S,M\}$  to a cost value:

Cost of a coffee is 2, but reduces to 1 after filling out a survey until the end of the month

Output is computed by implicitly adding up transition costs

How to define automata with richer set of operations?

# Finite Automata with Cost Registers



## Cost Register Automata:

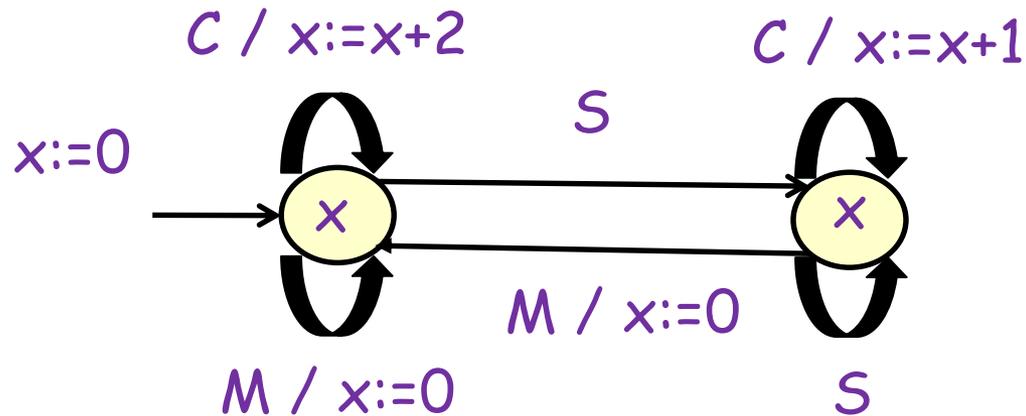
Finite control + Finite number of registers

Registers updated explicitly on transitions

Registers are write-only (no tests allowed)

Each (final) state associated with output register

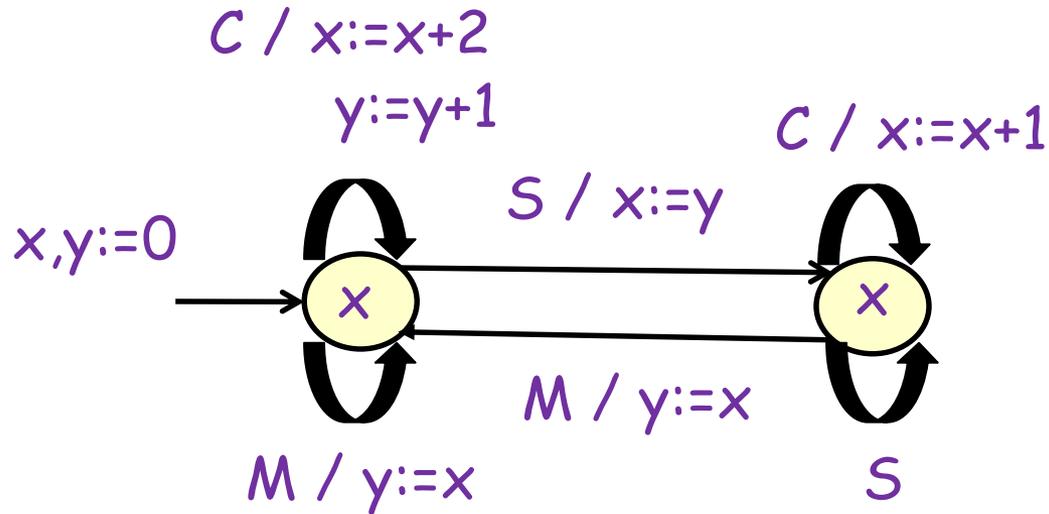
# CRA Example



At any time,  $x$  = cost of coffees during the current month

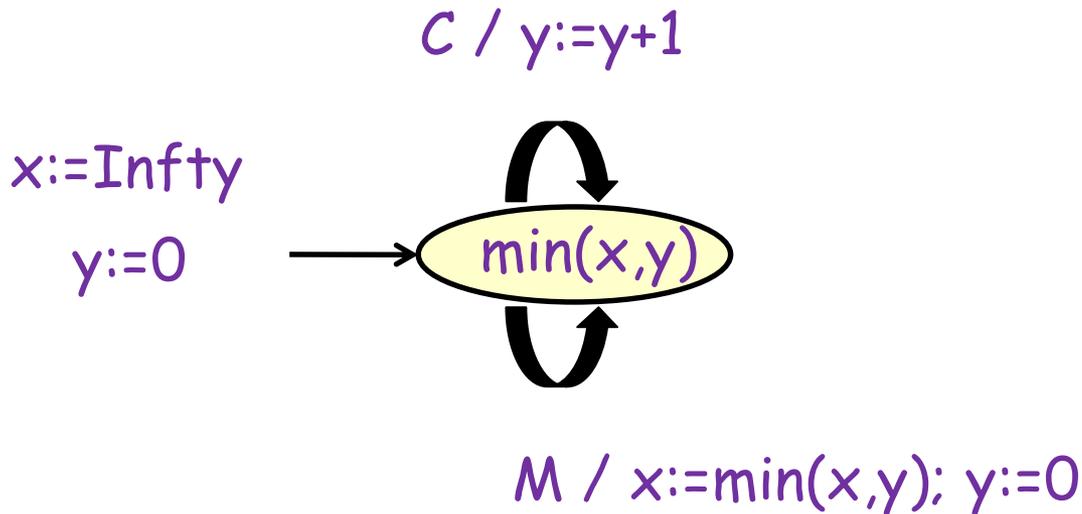
Cost register  $x$  reset to 0 at each end-of-month

# CRA Example



Filling out a survey gives discount for all coffees during that month

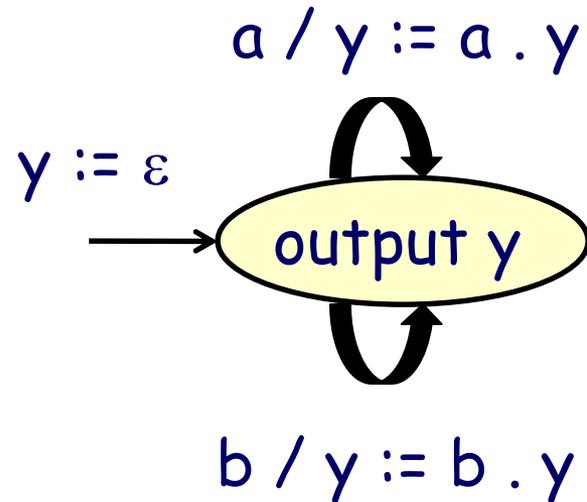
# CRA Example



Output = minimum number of coffees consumed during a month  
Updates use two operations: increment and min

# String Transformation Example

Rev(w) = String w in reverse



String variables updated at each step as in a program

Key restriction: No tests ! Write-only variables !

# Regular Function

Definition parameterized by cost domain  $D$  with a set of operations

Terms over  $D$ : Trees whose nodes are labeled with given operations

A (partial) function  $f: \Sigma^* \rightarrow D$  is regular if there exists a function  $g$  mapping strings to terms over  $D$  such that

- (1) for all strings  $w$ ,  $f(w) = \text{Evaluation of } g(w)$
- (2)  $g$  is a regular string-to-tree transformation



# Regular String-to-tree Transformations

- Definition based on *MSO* (Monadic Second Order Logic) - definable graph-to-graph transformations (Courcelle)
- Studied in context of syntax-directed program transformations, attribute grammars, and XML transformations
- Operational models:
  - Macro Tree Transducers (Engelfriet et al)
  - Streaming tree transducers (ICALP 2012, JACM 2017)

**Thm:** QREs mapping  $\Sigma^*$  to costs  $D$  with given set of operations define exactly regular functions

# MSO-definable String-to-tree Transformations

## □ MSO over strings

$\Phi := a(x) \mid X(x) \mid x=y+1 \mid \sim \Phi \mid \Phi \ \& \ \Phi \mid \text{Exists } x. \Phi \mid \text{Exists } X. \Phi$

## □ MSO-transduction from strings to trees:

### 1. Number $k$ of copies

For each position  $x$  in input, output-tree has nodes  $x_1, \dots, x_k$

### 2. For each symbol $a$ and copy $c$ , MSO-formula $\Phi_{a,c}(x)$

Output-node  $x_c$  is labeled with  $a$  if  $\Phi_{a,c}(x)$  holds for unique  $a$

### 3. For copies $c$ and $d$ , MSO-formula $\Phi_{c,d}(x,y)$

Output-tree has edge from node  $x_c$  to node  $x_d$  if  $\Phi_{c,d}(x,y)$  holds

# Properties of Regular Functions

Known properties of regular string-to-tree transformations imply:

- If  $f$  and  $g$  are regular w.r.t. a cost model  $D$ , and  $L$  is a regular language, then “if  $L$  then  $f$  else  $g$ ” is regular w.r.t.  $D$
- Reversal: define  $\text{Rev}(f)(w) = f(\text{reverse}(w))$ .  
If  $f$  is regular w.r.t. a cost model  $D$ , then so is  $\text{Rev}(f)$
- Costs grow linearly with the size of the input string:  
Term corresponding to a string  $w$  is  $O(|w|)$
- What about decision problems (e.g. are two QREs equivalent?)  
Need to focus on specific cost models

# Regular Functions over Commutative Monoid

Cost model:  $D$  with binary function  $+$

Interpretation for  $+$  is commutative, associative, with identity  $0$

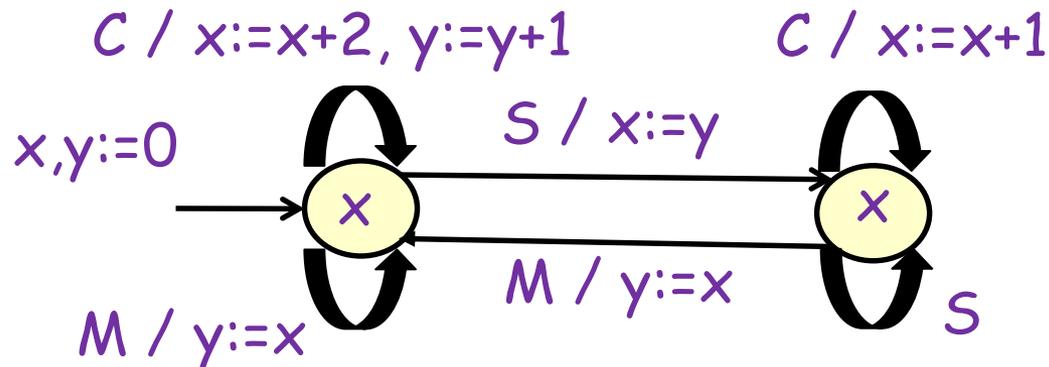
Cost model  $D(+)$ : No restriction on use of  $+$

Cost model  $D(+c)$ : Only addition by constant allowed

Thm: Regularity w.r.t.  $D(+)$  coincides with regularity w.r.t.  $D(+c)$

Proof intuition: Show that rewriting terms such as  $(2+3)+(1+5)$  to  $((2+3)+1)+5$  is a regular tree-to-tree transformation, and use closure properties of tree transducers

# Additive Cost Register Automata



## Additive Cost Register Automata:

DFA + Finite number of registers

Each register is initially 0

Registers updated using assignments  $x := y + c$

Each final state labeled with output term  $x + c$

Given commutative monoid  $(D, +, 0)$ , an ACRA defines a partial function from  $\Sigma^*$  to  $D$

# Regular Functions and ACRAs

- Thm: Given a commutative monoid  $(D, +, 0)$ , a function  $f: \Sigma^* \rightarrow D$  is definable using an ACRA iff it is regular w.r.t. cost model  $D(+)$ .
- Establishes ACRA as an intuitive, deterministic operational model to define this class of regular functions
- Proof relies on the model of SSTT (Streaming string-to-tree transducers) that can define all regular string-to-tree transformations

# Decision Problems for ACRAs

- **Min-Cost:** Given an ACRA  $M$ , find  $\min \{M(w) \mid w \text{ in } \Sigma^*\}$   
Solvable in Polynomial-time  
Shortest path in a graph with vertices (state, register)

- **Equivalence:** Do two ACRAs define the same function  
Solvable in Polynomial-time

**Exercise:** Design algorithm for equivalence checking !

- **Register Minimization:** Given an ACRA  $M$  with  $k$  registers, is there an equivalent ACRA with  $< k$  registers?  
Algorithm polynomial in states, and exponential in  $k$

# ACRA Equivalent QREs

## Additive QREs:

- Base functions:  $a \rightarrow c$  and  $\varepsilon \rightarrow c$
- Concatenation:  $\text{split}(f, g, +)$
- Iteration:  $\text{iter}(f, +)$
- Choice:  $f \text{ else } g$

Unambiguity requirements for all above constructors

Note: Output composition not included

**Thm:** Additive QREs are equivalent to ACRA (i.e. regular functions over commutative monoid)

# Emerging Theory of Regular Functions

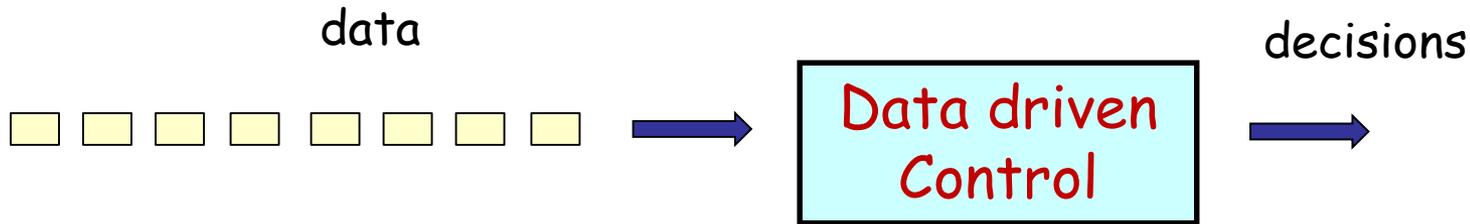
- A few classes that have been (partially) studied
  - ▶ Finite strings to finite strings (DReX: specialized QREs)
  - ▶ Infinite strings to infinite strings
  - ▶ Finite strings to semiring  $(\mathbb{N}, +, \min)$
  - ▶ Finite strings to discounted costs
  
- Many open problems
  - ▶ Decidability of equivalence of functions from  $\Sigma^*$  to  $(\mathbb{N}, +, \min)$
  - ▶ Theory of congruences
  - ▶ Learning algorithms...
  
- Unexplored classes (e.g. mapping trees to numerical costs)

# Back to QRE Evaluation Algorithm

- ❑ QREs and CRAs are expressively equivalent
- ❑ Can compiling a QRE into a CRA give an optimal streaming algorithm for evaluating QREs?
  - Recall connection between regular expressions and NFA/DFA
- ❑ No! Translation from QRE to CRA causes exponential blow-up
  - Deterministic simulation of unambiguous choice
  - Intersection (due to output composition)
- ❑ Research challenge: what's a suitable model for "automata-based stream processing"?
  - ICALP'17: Unambiguous weighted automata + nesting + parallelism
  - Ongoing work: Data transducers

# Conclusions and Research Directions

# Real-time Decision Making in IoT Applications

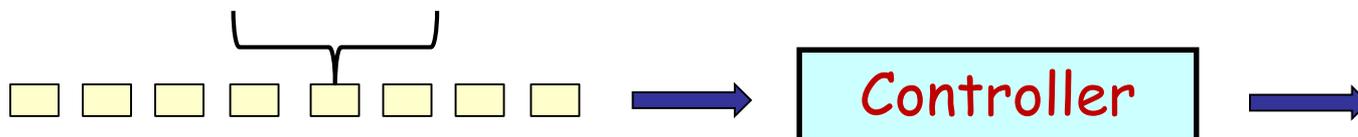


- One research question:  
How to specify quantitative policies over data streams ?
- One solution: Quantitative Regular Expressions (QRE)
  - Modular high-level specifications
  - Theoretically robust expressiveness
  - Guaranteed space/time requirements of generated code
  - Evaluation for network traffic engineering

# Privacy ??



(car ID, position, time)



Query: Max over CarID { Average speed of CarID in past month }

How much information about a specific car does answer to this query leak ?

Research opportunity:

Anonymity / privacy guarantees for queries over streaming data

# Learning ??

(source IP, dest IP, payload)



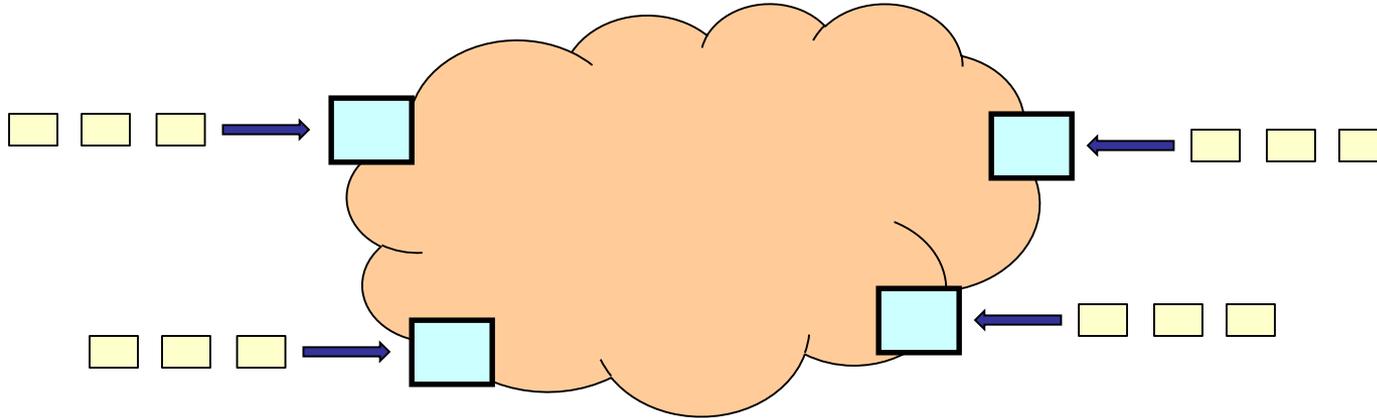
What traffic constitutes an attack ?

Known patterns can be captured by, say, QREs, but can the switch dynamically learn the attack pattern?

Research opportunity:

Learning high-level declarative patterns, say QREs, more plausible than learning low-level code

# Distributed Processing ??

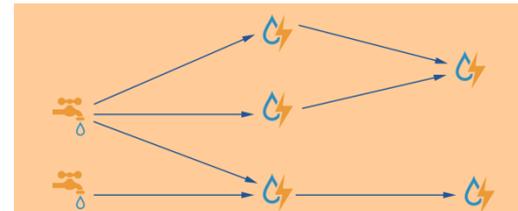


Logical query on a single stream of data

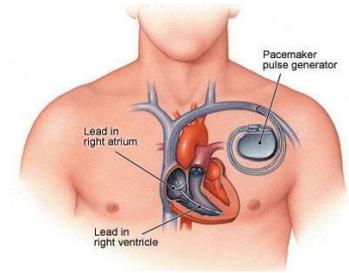
Physical implementation: distributed system

How to ensure consistency ? High performance ? Resilience to errors ?

Emerging architectures: Apache STORM



# Safety-critical Applications ??



Clinical diagnosis  
→ pacing stimulus



Specification: logical query over analog signal

→ Implementation: discrete control software

Predictable response time critical

Key resource constraint: battery life, so need optimized code

Goal: design more effective diagnosis strategies