ΔΗΜΟΚΡΙΤΟΣ
ΕΘΝΙΚΟ ΚΕΝΤΡΟ ΕΡΕΥΝΑΣ ΦΥΣΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

Applied Data Mining

ALEVIZOPOULOU SOFIA 2022201704002

AVGEROS GIANNIS 2022201704003

TSIATSIOS GEORGE 2022201704024

MSC DATA SCIENCE
ATHENS 2018

Lazy Evaluation Methods for Detecting Complex Events

ILYA KOLCHINSKY, TECHNION, ISRAEL INSTITUTE OF TECHNOLOGY

IZCHAK SHARFMAN, TECHNION, ISRAEL INSTITUTE OF TECHNOLOGY

ASSAF SCHUSTER, TECHNION, ISRAEL INSTITUTE OF TECHNOLOGY

Complex Event Processing

Complex Event Processing (CEP)

- Real time system
- Efficiently detection of complex patterns over a sequence
  - Sequence SEQ(a,b,c,…) : stream of primitive events in a predefined order
- Non-deterministic Finite Automata (NFA) used to evaluate CEP queries
  - Eager strategy
  - Construct partial matches according to the order of the events in the sequence
  - Every incoming event in the sequence extends previously observed prefixes or starts a new prefix
  - Match prefixes grow exponentially
  - Inefficient in terms of memory and computational resources

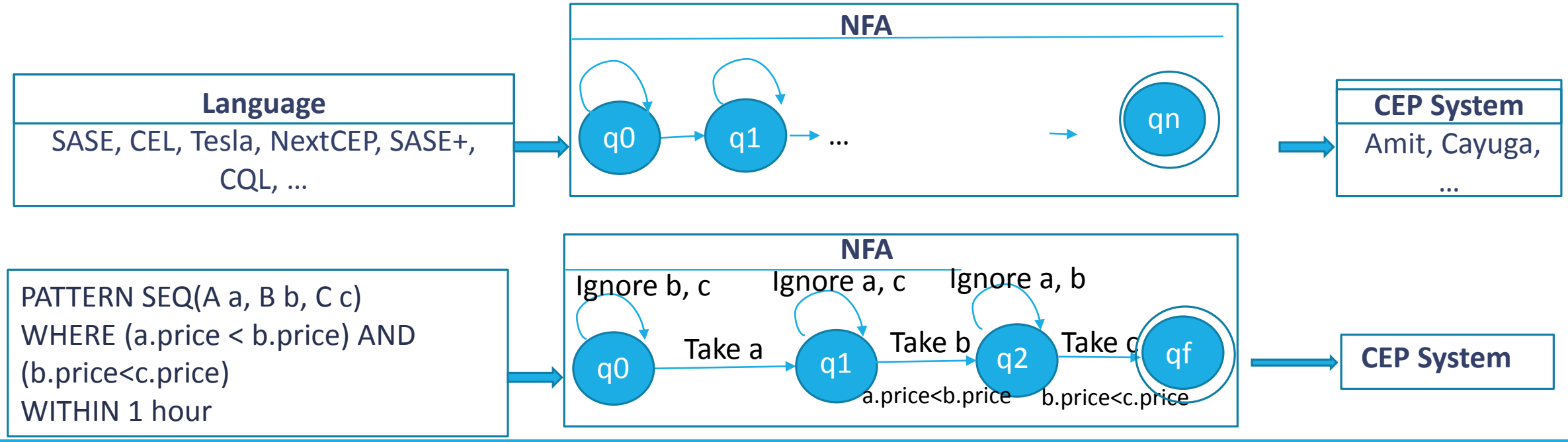
➢ Propose a **lazy evaluation**, NFA based: process events in descending order of selectivity

Complex Event Processing Systems

Complex Event (CE) patterns expressed in declarative languages→

Declarative languages turn into NFA's →

NFA's at the end are used from the event processing machine



Event Specification Language

Eager NFA matching mechanism

SASE language is used for patterns definition

- SQL-like syntax
- High degree of expressiveness
- Supports logic operators, iterations , aggregates, sequences and time windows
- Composed of three building blocks:
  - PATTERN: specifies simple events would like to detect (sequence patterns)
  - WHERE: specifies constraints on the values of data attributes
  - WITHIN: specifies a time window over the entire pattern
- Each primitive event has an arrival timestamp, a type, and a set of attributes

The Eager Evaluation Mechanism

NFA definition:

$$A = (Q, E, q_1, F)$$

Q: set of states, q1: initial state, F: final accepting state,
E: set of directed edges (where edge is e = (qs, qd, action, name, condition), qs: source state, qd: destination state))

Event arrivals trigger transition between edges

Multiple instances of the NFA -one for each partial match- run in parallel

Each instance is associated with a match buffer

Match buffer used to store the primitive events constituting a partial match

Final's state match buffer, returned as a successful match for the pattern

At startup match buffer is empty

Two kind of edge actions:

- Take: add it to the match buffer
- Ignore: don't add it to the match buffer

Runtime Behavior of Eager Evaluation

A sequence pattern of n events will be compiled into a chain of n+1 states

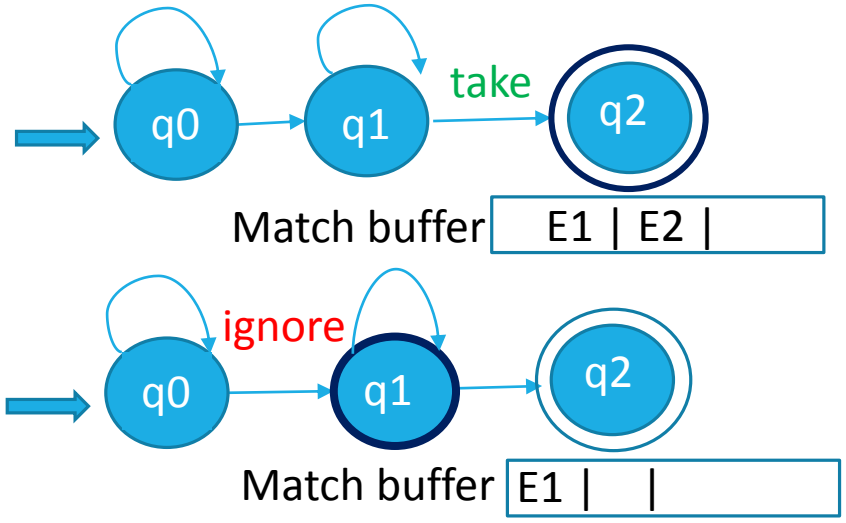
n first states

- self-loops edges: edge leading to themselves with an **ignore** action
  - Don’t add it to the match buffer
  - allow detection of all possible combinations of events
- connecting edges: edge leading to the next state with a **take** action
  - Add it to the match buffer
  - the instance will be duplicated, and both possible traversals
    will be executed on different copies

n+1 state is the final state F

Predicate: defines conditions on the edge at which

- instance and match buffer will be discarded
- Instance duplicated in 2 different copies one for each traversal: ignore edge, take edge



Current state:q1, received event:E2,
take action: add E2 at match buffer and proceed to q3,
Ignore action: traverse a self loop

Lazy evaluation [1/2]

Lazy evaluation

Simple selectivity

- focus on the rarest events first of the sequence
- minimize computational and memory requirements

Evaluation of an expression is delayed until its value is needed

SASE language based

Performing lazy evaluation on CEP

- input buffer: events inserted in it, in chronological order
- Selective ordering: sorted items based on descending selectivity (inverse frequency)
- Store action: the event is inserted into the input buffer in chronological order
- Modified take action: Take(x event)
  - x event name will be associated with the event it inserts in the match buffer
  - searches in the input buffer to combine the results with the input stream and spawn additional NFA instances

Lazy evaluation [2/2]

Scope parameters

Utilize prior knowledge for the range of events

Use them to choose the source of events for an edge

- the input buffer
- the input stream

Formalized as *event ((s)tart of scope,(f)inish of scope)→e(s,f)*

Defines the time range on which the events are valid for the edge

Reserved keywords: start, finish or an event name

There are four different cases for s and f:

- Event(start,x): Event is taken from the beginning input stream
- Event(eventName,x):Only events chronologically follow event name in the sequence order from the match buffer are read
- Event(x,eventName):Only event chronologically preceding the event name in the sequence order from the match buffer are read
- Event(x,finish): Events are derived from the input stream

Chain NFA [1/2]

Chain NFA

Implementation of lazy evaluation in a NFA

Selectivity order is predefined

Consists of n+1 states (n for the events plus one final accepting state)

Each state will have outgoing edges that are formalized as:

- $Event_{(n)}^{ignore}$: Ignore the event that is before the n_{th} element in selective order because this new event is already taken.
- $Event_{(n)}^{store}$: Any event that is after the n_{th} element in selective order is stored to be used later for a take action
  - lazy evaluation principle , do not process at the time and store items that may be used later
- $Event_{(n)}^{take}$: Any event that satisfy the condition in the initial pattern and is between the scope parameters triggers a state change

Chain NFA [2/2]

Scope parameters

Define the scope in which an event is accepted

Formalized as *event ((s)tart of scope,(f)inish of scope)→e(s,f)*

- s: The latest event of the union between the previous events of $event_{(n)}$'s
  - It the union is not equal to null set: take action in selective and sequential ordering, according to selective ordering
  - If the union is equal to null set: Start
- f:The earliest event of the union between the previous events of $event_{(n)}$'s
  - It the union is not equal to null set: take action in selective ordering and the successors of the event(n) in sequential ordering, according to selective ordering
  - If the union is equal to null set: Finish

Use Case of Chain NFA

Inverted event frequency selective ordering : c, a, b

Sequence: SEQ (a,b,c)

Initial pattern is a.price<b.price<c.price

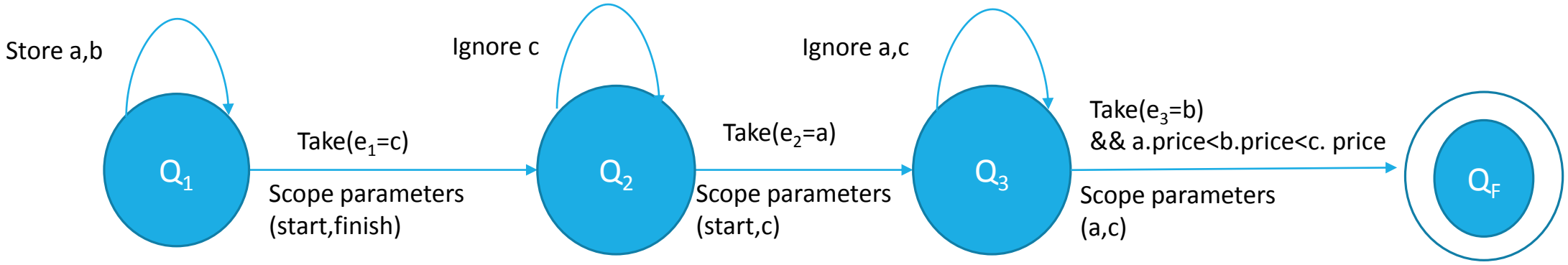
Q1 state: store the a,b that follow c in selective order

- event c will also be in the scope of the parameters (start,finish)

Q2 state: ignore c as it is already taken and accept event

Qf state: final state where the initial pattern accept event b along with the condition stated in initial pattern

PATTERN SEQ(A a, B b, C c)
WHERE (a.price < b.price) AND (b.price<c.price)
WITHIN 1 hour



Tree NFA [1/2]

Tree NFA

- Selectivity order is not predefined (ad hoc manner)
  - Each state have knowledge regarding the current selectivity of each event
- Contains all the possible Chain NFAs for a given pattern
- The input buffer contains all events arrived in a specified time window
- There is a counter for counting the times of each event occurred in the input buffer
- Evaluation order will be determined only when all events occurred more than once
- Derive of the selectivity order by sorting the counters
- Each state selects the next state based on the current contents of the input buffer

Tree NFA [2/2]

Advantages over Eager NFA

- performance gain of two orders of magnitude in comparison with eager NFA

Advantages over Chain NFA

- On-the-fly selectivity , chain evaluation according to the order reflecting the current frequency events
- Route the events dynamically, guaranteeing that any partial match will be evaluated using the most efficient order possible

Structure of Tree NFA

Depth n-1

Root is the initial state

Leaves are connected to the accepting state

Nodes located at layer k where 0 <= k <= n-1 are all states responsible for all orderings of k event names out of the n event names of the pattern

Each node has n-k outgoing edges for each event that is responsible for

The number of states might be exponential in n

- Implementation issues:
  - only those states reached by at least a single active instance will be instantiated and will actually occupy memory space
  - All Instances reaching a particular state are terminated and removed from the NFA

Edges are designed in such a way that the least frequent event is chosen at each evaluation step

Tree NFA Edge Description

Scope parameters

Predicates

$l_n Scope_{ord}$ (e):an event e is located within the corresponding scope (s (q_{ord}, e), f (q_{ord} ,e))

$P_n(q_{ord})$: condition that checks if the input buffer contain at least a single instance of each primitive event not appearing in order prefix ($_{ord}$ =order prefix)

$P_{se}(q_{ord})$: condition that checks if an event denote the most infrequent event in the input buffer during the evaluation step ($_{se}(q_{ord})$)

The set of outgoing edges e_{ord} contain the following edges:

- Ignore edge: Ignores any event whose type corresponds to one of the already taken events
- Store edge: The incoming event is stored in the input buffer if Pse and Pne predicates are not satisfied
- Take edge: The incoming event is stored in the match buffer and the NFA advances to the next layer If the incoming event satisfies the predicates P_{se}, P_{ne}, $l_n Scope_{ord}$ (e)) and the initial pattern ($cond_e$)

Experimental Evaluation [1/2]

Evaluation of chain and tree NFA in comparison to the eager model measuring:

Runtime complexity: How many times a condition on an Edge is evaluated

Memory consumption

- The peak number of simultaneously active NFA instances
- The peak number of buffered events waiting to be processed

Experiments with real-world stock price histories to evaluate chain and tree NFA

Detection Pattern:

- Sequence: SEQ(a, b, c) where a, b, c should be highly correlated based on the Pearson correlation coefficient
- price histories of correlated stocks should be above Threshold T
- Time window has been set to the length of the price history

Evaluated patterns:

- Pattern 1: c less frequent than a, b
- Pattern 2: c as the most selective event and sequence produced c, b, a and a, c, b
- Frequency of event c with respect to a and b affects the overall efficiency of the evaluation models
- As the ratio of c events to all events approaches 1 the performance will not improve because no selectivity is optimal

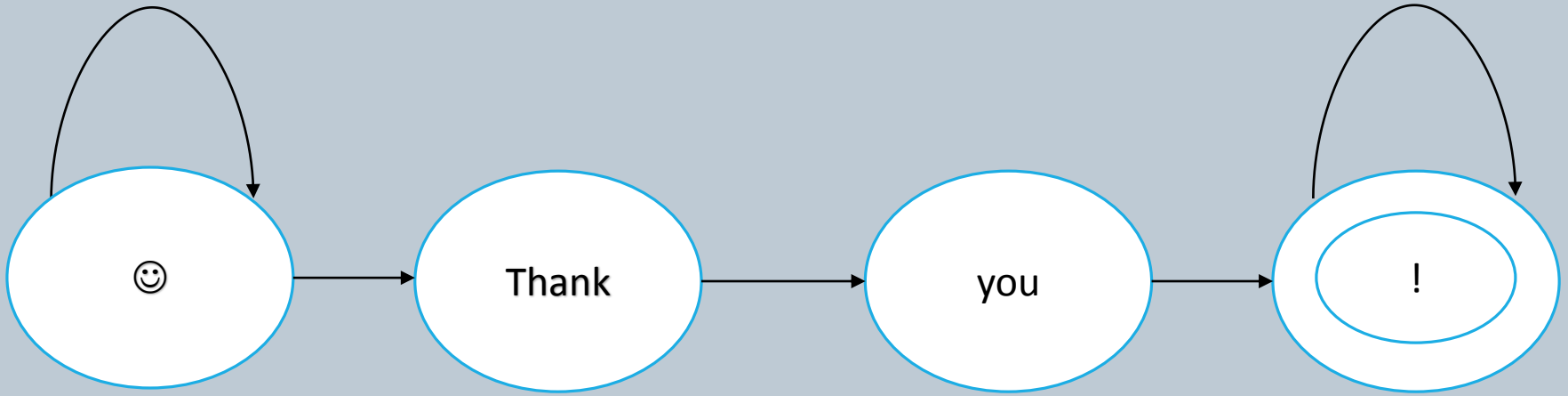
Experimental Evaluation [2/2]

```
PATTERN SEQ(Stock a, Stock b, Stock c)
WHERE (a.ticker=Finance) AND (b.ticker=HiTech) AND (c.ticker
= GOOG) AND
(Corr (a.history; b.history)>T) AND
(Corr (b.history; c.history)>T)
WITHIN h
```

Runtime complexity	• Eager NFA: very poor performance in Pattern 1 , improved in Pattern 2
	• Lazy chain NFA
	  • With c as second or third event: poor performance
	  • With c as first event: evaluation starts only when the rarest event arrives reducing the number of calculation
	  • Very poor regarding the pattern and the selectivity order
	• Tree NFA better performance due to dynamic structure for any pattern due to dynamic structure

Memory consumption	• Lazy chain NFAs:
	  • With c as first event: consumes less memory as there is smaller number of instances
	  • With a as first event : consumes significantly more memory
	  • With b as first event: better but not optimal due to selectivity of mutual conditions between a and b
	• Tree NFA is the most efficient NFA

Adaptive selectivity	• Tree NFA is high adaptive to changes in event selectivity • Tree NFA shows consistent improvement over all chain NFAs regardless of the input selectivity


☺
Thank
you
!