Glasgow Automata Illustrated

Alexandros.Koliousis & Joseph.Sventek
@glasgow.ac.uk
Outline

1. What are Glasgow automata?
2. An exemplar automaton for the challenge
3. Demo application
Glasgow automata

Glasgow automata are concise *imperative programs* for complex event processing, executable in the context of a topic-based publish/subscribe cache.
Glasgow automata

Topic-based Publish/subscribe cache

- register
- insert
- insert
- insert

Topic 1 ➔ Auto A ➔ Topic 3

Topic 2 ➔ Auto B ➔ send

Wednesday, 18 July 12
Glasgow automata

- *Compiled* to instructions for a stack machine
- *Bound* to a separate thread
- *Awakened* & executed when a topic (stream) of interest is updated
- *Send* derived events to the registering application, and/or
- *Publish* to another topic
DEBS Challenge

Data generator

TCP socket

Google Protocol Buffers

SRPC channel

Insert Queries

DEBS 2012 Grand Challenge programs

Receiver

Data

.send

Cache

C

.txt

DEBS Challenge

Wednesday, 18 July 12
DEBS Challenge

CDataPoint

Query 1
Query 2

Wednesday, 18 July 12
An example

**Operator 4** records the sensor readings mf01-mf03, and *when the relative variation* (as computed by operators 1-3) of any one of them *exceeds 30%*, the operator must output these records.

The recording starts 20 seconds before, and ends 70 seconds after the threshold violation.

*When multiple violations occur*, the operator must always capture raw data 20 seconds before the first and 70 seconds after the last violation.
An example

- subscriptions to topics (i.e. event streams)
- *associations with relations* – *not used here.*
- declarations of local state (e.g. `int x;`)
- initialization of local state (e.g. `x = 0;`)
- behavior
An example

subscribe e to CDataPoint;
subscribe s1 to S1;
subscribe s2 to S2;
subscribe s3 to S3;

Comments

An automaton can subscribe to one or more topics.

An automaton processes events serially in the temporal order in which they have been published.

Event attributes are accessed via local variables, e.g. e.mf01.
An example

```java
window w;
bool critical;
sequence s;
tstamp last;
real avg1, rng1, ...;

initialization {
    w = Window(sequence, SECS, 20);
critical = false;
    avg1 = 0.0;
    rng1 = 0.0;
    ...
}
```

Comments

Local variables can be used to carry state across executions of the behavior clause.

E.g., variables `avg1` & `rng1` will hold state about the last reported event $S_1$.

The initialization clause of an automaton is executed only once, when it is created.
An example

window \( w \);
bool critical;
sequence \( s \);
tstamp last;
real \( \text{avg1}, \text{rng1}, \ldots \);

initialization {
  \( w = \text{Window}(\text{sequence}, \text{SECS}, 20) \);
  critical = false;
  \text{avg1} = 0.0;
  \text{rng1} = 0.0;
  \ldots
}

Comments
Window \( w \) will hold the last 20 seconds of events before a threshold violation.
Windows *slides to the right* as new events are appended.
An example

window w;
bool critical;
sequence s;
tstamp last;
real avg1, rng1, ...;

initialization {
    w = Window(sequence, SECS, 20);
critical = false;
    avg1 = 0.0;
    rng1 = 0.0;
    ...;
}

Comments

A sequence is a list of heterogeneous basic data types (e.g. int or real). It can be used to construct new events.

For example, sequence s will hold CDataPoint events merged with S1, S2, and S3 events.
An example

behavior {
    if (currentTopic() == 'S1') {
        avg1 = s1.ave;
        rng1 = s1.rng;
        if (s1.rng > 0.3) {
            if (! critical) {
                critical = true;
                send(w);
            }
            last = tstampDelta(s1.t, 70, FALSE);
        }
    } else if (currentTopic() == 'S2') {...
    } else if (currentTopic() == 'S3') {...
}

Comments

The behavior clause of an automaton is executed whenever a topical event is inserted into the cache.
An example

behavior {
    if (currentTopic() == ‘S1’) {
        avg1 = s1.ave;
        rng1 = s1.rng;
        if (s1.rng > 0.3) {
            if (! critical) {
                critical = true;
                send(w);
            }
        }
        last = tstampDelta(s1.t, 70, FALSE);
    } else if (currentTopic() == ‘S2’) {...
    } else if (currentTopic() == ‘S3’) {...

Comments

For multiple subscriptions, an automaton distinguishes different events using the currentTopic() function.
An example

```java
behavior {
    if (currentTopic() == 'S1') {
        avg1 = s1.ave;
        rng1 = s1.rng;
        if (s1.rng > 0.3) {
            if (! critical) {
                critical = true;
                send(w);
            }
        
        last = tstampDelta(s1.t, 70, FALSE);
    }
    else if (currentTopic() == 'S2') {...
} else if (currentTopic() == 'S3') {...
```
An example

behavior {
    if (currentTopic() == 'S1') {
        avg1 = s1.ave;
        rng1 = s1.rng;
        if (s1.rng > 0.3) {
            if (! critical) {
                critical = true;
                send(w);
            }
        }
        last = tstampDelta(s1.t, 70, FALSE);
    }
} else if (currentTopic() == 'S2') {...
} else if (currentTopic() == 'S3') {...

Comments

A threshold violation has been detected.
behavior { 
    if (currentTopic() == 'S1') {
        avg1 = s1.ave;
        rng1 = s1.rng;
        if (s1.rng > 0.3) {
            if (! critical) {
                critical = true;
                send(w);
            }
        
        last = tstampDelta(s1.t, 70, FALSE);
    } 
} else if (currentTopic() == 'S2') {...
} else if (currentTopic() == 'S3') {...

Comments
Send the 20-second window to the process that has registered the automaton, if we enter the critical window for the first time.
behavior {
    if (currentTopic() == 'S1') {
        avg1 = s1.ave;
        rng1 = s1.rng;
        if (s1.rng > 0.3) {
            if (! critical) {
                critical = true;
                send(w);
            }
            last = tstampDelta(s1.t, 70, FALSE);
        }
    } else if (currentTopic() == 'S2') {...
    } else if (currentTopic() == 'S3') {...

Comments
Timestamp last points 70sec ahead of the most recent (i.e. this) violation.
(behavior continued)

} else {

    s = Sequence(e.time, e.mf01, e.mf02,
               e.mf03, avg1, rng2, ...);
    if (critical) {
        if (tstampDiff(e.time, last) < 0)
            send(s);
        else {
            critical = false;
            append(w, s, e.time);
        }
    } else append(w, s, e.time);
}

Comments

Handle CDataPoint events.
(behavior continued)

} else {

    s = Sequence(e.time, e.mf01, e.mf02, e.mf03, avg1, rng2, ...);

    if (critical) {
        if (tstampDiff(e.time, last) < 0)
            send(s);
        else {
            critical = false;
            append(w, s, e.time);
        }
    } else {
        append(w, s, e.time);
    }
}

Comments

Create a new type of event by merging attributes of CDataPoint, S1, S2, and S3 events.

Populate window \textbf{w} accordingly. The window slides based on \textbf{e.time} timestamps.
Send events that have occurred within 70 seconds since the most recent threshold violation.

(behavior continued)

} else {
    s = Sequence(e.time, e.mf01, e.mf02,
                 e.mf03, avg1, rng2, ...);
    if (critical) {
        if (tstampDiff(e.time, last) < 0)
            send(s);
        else {
            critical = false;
            append(w, s, e.time);
        }
    } else append(w, s, e.time);
}

Comments

Send events that have occurred within 70 seconds since the most recent threshold violation.
Glasgow automata allow for multiple, efficient implementations of logical query operators.
Our solutions

(a) 19 automata

(b) 2 automata
Results from processing the first 5,000,000 events on a 2.4GHz Intel Core i7 with 8GB of RAM, running Mac OS X Lion 10.7.3.
\[ s = \text{Sequence}(e.\text{time}, e.\text{mf01}, e.\text{mf02}, e.\text{mf03}, \text{avg1}, \text{rng1}, \ldots, \text{avg3}, \text{rng3}); \]

\text{send}(s);
Execution time

The diagram shows a graph with two axes: Execution time (ms) on the x-axis ranging from $10^{-2}$ to $10^2$ and Frequency (10μs bins) on the y-axis ranging from $10^0$ to $10^6$. The graph highlights a 70-sec critical period and a critical event. The data points are distributed across the graph, indicating the frequency of execution times within the specified range.
Demo application

- Performs real-time throughput measurements;
- Speed up simulation 100x;
- With 19 automata, the application reports ~3500 events/sec (35x);
- with 2 automata, ~5500 events/sec (55x).
Demo application
Thank you.

Alexandros.Koliousis & Joseph.Sventek

@glasgow.ac.uk
Queuing delay

Wednesday, 18 July 12
Merging automata

subscribe e to CDataPoint;
int bm05;
initialization { bm05 = -1; }
behavior {
    if (bm05 == 0 && e.bm05 == 1) {
        publish('S5', e.time, 1); // rising edge
    } else
    if (bm05 == 1 && e.bm05 == 0) {
        publish('S5', e.time, 0); // falling edge
    }
    bm05 = e.bm05; // store variable
}
subscribe e to CDataPoint;
int bm05; sequence s5;
initialization { bm05 = -1; ...}
behavior {
    if (bm05 == 0 && e.bm05 == 1) {
        s5 = Sequence(1, e.time);
    } else
    if (bm05 == 1 && e.bm05 == 0) {
        s5 = Sequence(1, e.time);
    }
    bm05 = e.bm05; # store variable
}
Experiments run on two AMD Athlon 64 dual core 2.7GHz processors with 4GB of RAM.
vs. Cayuga

112,635 (x10) events.

Wednesday, 18 July 12
Cost of built-ins

Execution time (µs)

nothing  seqElement hourInDay  insert  hasEntry  lookup  Identifier  publish  send
Throughput

![Graph showing throughput versus buffer size (bytes) for 1-way and 2-way systems.](#)