TEMPORAL CORRELATION PATTERNS

Intersecting Joins, Streams, Events and Reactive Programming

Oliver Bračevac
Software Technology Group (Mira Mezini)
TU Darmstadt, Germany
bracevac@cs.tu-darmstadt.de

REBLS’ 15
10/27/15 - Pittsburgh, PA, USA
Reactive Programming?

- React
- akka
- FrTime
- java.util.stream
- Rx
- Flapjax
- Bacon.js
CORRELATE |ˈkɒrəlɪt, -ɪt|
Have/establish a mutual relationship or connection, in which one thing affects or depends on another.
Correlate |ˈkɒrəleɪt, -rɪ-|

- Have/establish a mutual relationship or connection, in which one thing affects or depends on another.

- “We should correlate general trends in public opinion with trends in the content of television news.”

(Oxford Dictionary of English)
Correlate |ˈkɔrəleɪt, -rɪ-|

- Have/establish a mutual relationship or connection, in which one thing affects or depends on another.
- “We should correlate general trends in public opinion with trends in the content of television news.”

Reactive languages correlate data!

(Oxford Dictionary of English)
Data Correlation in RP

http://reactivex.io
Data Correlation in RP

http://reactivex.io
Data Correlation in RP

filter { }

groupBy

http://reactivex.io

Count

$\text{Count}$

$X_1, X_2, X_3, X_4, ..., X_n$
Data Correlation in RP

filter {  }

groupBy

http://reactivex.io
WHERE RP FAILS FAST:

Timing

Pattern

Observations

10ms

✓

✗
WHERE RP FAILS FAST:

Timing

Partial orders
Where RP Fails Fast:

### Timing

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Pattern" /></td>
<td><img src="image2" alt="Observations" /></td>
</tr>
</tbody>
</table>

### Absence/negation

<table>
<thead>
<tr>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image3" alt="Observations" /></td>
</tr>
</tbody>
</table>

### Partial orders

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image4" alt="Pattern" /></td>
<td><img src="image5" alt="Observations" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image6" alt="Observations" /></td>
</tr>
<tr>
<td><img src="image7" alt="Observations" /></td>
</tr>
</tbody>
</table>
Where RP Fails Fast:

**Timing**

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Diagram" /></td>
<td><img src="image2.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>

**Absence/negation**

<table>
<thead>
<tr>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image3.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>

**Partial orders**

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image4.png" alt="Diagram" /></td>
<td><img src="image5.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>

**“Criss-crossing”**

<table>
<thead>
<tr>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image6.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>
Where RP Fails Fast:

- No exposure of arrival orders or timings!
- We need a pattern language!
CEP to the Rescue!

- **Complex Event Processing**
  - Derives high-level events from combinations of input events
- **Features expressive pattern languages for ordering and timing constraints**
CEP to the Rescue!

Why not just combine RP with CEP?
Variability in CEP

Selection

Pattern

Observations

which ones to select?
Variability in CEP

Selection

Pattern

Observations

which ones to select?

Contiguity

Pattern

Observation

does it match?
Variability in CEP

Selection
Pattern
Observations
which ones to select?

Contiguity
Pattern
Observation
does it match?

Consumption
Pattern
Observations
which pattern matches?

Selection
Pattern
Observations
which ones to select?
Variability in CEP

Selection

Pattern
Observations
which ones to select?

Contiguity

Pattern
Observation
does it match?

Consumption

Pattern
Observations
which pattern matches?

Time Model

\[ t, [t_1, t_2), [t_1, t_2], \langle t_1, \ldots, t_n \rangle \]

what is the occurrence “time” of complex events?
**Variability in CEP**

- Options are hard-wired in CEP languages
- Unclear semantics

**Time Model**

\[ t, [t_1, t_2), [t_1, t_2] \]

what is the occurrence “time” of complex events?
SUMMARY

- Reactive Programming (RP)
  - Weak support for correlations by timing and order
  - Strong semantic foundations

- CEP-style patterns complement RP
  - Huge variability of features
  - Ambiguous and diverse semantics
CorrL
The Correlation Language
CorrL
The Correlation Language

- CEP-style patterns + Reactive Programming
  - Embedded DSL (Scala, WIP)

- Formally specified semantics
  - Aim for maximal expressivity
  - Support the semantic variability
  - State-machine-based
await
  down:MouseDown
  on down ->
    move:MouseMove
  unless evt:MouseUp
  yield Drag(move.x, move.y)

await
  down:MouseDown
  & key:KeyDown.
    key.code == SHIFT
  on down & key ->
    move:MouseMove
  unless evt:MouseUp | evt:KeyUp
  yield Drag(move.x, move.y)

await
  down:MouseDown
  on down ->
    last dr:Drag.
    on dr -> move:MouseMove
  unless evt:MouseUp
    | time(move)-time(dr) > 0.2s
  yield Drag(move.x, move.y)

await
  down:MouseDown
  on down ->
    move:MouseMove.
      move.y == max(MouseMove.y, 1ms)
  unless evt:MouseUp
  yield Spike(move.x, move.y)
await
  x:T. P_1(x) & y:U. P_2(y)
on x & y ->
    z:V. P_3(x,y,z) unless K_3(x,y,z)
unless K_0 | K_1(x) | K_2(y)
yield W(e(x,y,z))
Handling Variability

Pattern

Observations
Handling Variability

Pattern

Observations

Matches

{ red, pink, red, orange, red, yellow, green, orange, green, yellow }
Handling Variability

Pattern → Observations

Matches

Cut (e.g. first-received semantics)
CorrL
Summary and Outlook

- Reactive language with expressive correlation patterns
  - Transformations, Aggregations, Windows, Timing, Partial Orders

- Which cut functions are efficiently computable?
  - “No regrets” vs. retraction

- Explore connection to coordination/synchronization
Automata Network
Automata Network

Dispatcher

Timestamps & periodic clocks

T1

T2

T3

T4

T5

T7

T9

T8
Automata Network

Complex events:
$C_1(x) \& C_2(y) \Rightarrow C_4(x+y)$
Automata Network

Dispatcher

T1 → T2

T3 → T4

T5 → T7

T9 → T8

Recursion/Kleene
Automata Network

Consistent observations at joins

<table>
<thead>
<tr>
<th>$C5(t_1)$</th>
<th>$C7(t_2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_1$</td>
<td>$e'_1$</td>
</tr>
<tr>
<td>$e_2$</td>
<td>$e'_2$</td>
</tr>
<tr>
<td>$e_3$</td>
<td></td>
</tr>
</tbody>
</table>


```plaintext
await
  x: T. P_1(x) & y: U. P_2(y)
on x & y ->
  z: V. P_3(x, y, z) unless K_3(x, y, z)
unless K_0 | K_1(x) | K_2(y)
yield W(e(x, y, z))
```
Semantics

```
await
  x:T. P_1(x) & y:U. P_2(y)
  on x & y ->
    z:V. P_3(x,y,z) unless K_3(x,y,z)
unless K_0 | K_1(x) | K_2(y)
yield W(e(x,y,z))
```

```
await
  y:U. P_2(y)
  on y ->
    z:V. P_3(v_1,y,z) unless K_3(v_1,y,z)
unless K_0 | K_1(v_1) | K_2(y)
yield W(e(v_1,y,z))
```

\( v_1 = T(\ldots) \)

fork!
**Semantics**

```
await
  x: T. P_1(x) & y: U. P_2(y)
on x & y ->
    z: V. P_3(x,y,z) unless K_3(x,y,z)
unless K_0 | K_1(x) | K_2(y)
yield W(e(x,y,z))
```

```
await
  y: U. P_2(y)
on y ->
    z: V. P_3(v_1,y,z) unless K_3(v_1,y,z)
unless K_0 | K_1(v_1) | K_2(y)
yield W(e(v_1,y,z))
```

```
await
  z: V. P_3(v_1,v_2,z)
unless K_0 | K_1(v_1) | K_2(v_2) | K_3(v_1,v_2,z)
yield W(e(v_1,v_2,z))
```

```
fork!
v_1 = T(...)
```

```
fork!
v_2 = U(...)
```

```
```
await
  x:T. P_1(x) & y:U. P_2(y)
on x & y ->
  z:V. P_3(x,y,z) unless K_3(x,y,z)
unless K_0 | K_1(x) | K_2(y)
yield W(e(x,y,z))

\[ v_1 = \mathbf{T}(\ldots) \]

fork!

\[ v_2 = \mathbf{U}(\ldots) \]

fork!

\[ v_3 = \mathbf{V}(\ldots) \]

fork!

\[ W(v) \] output

\[ \text{output} \]

\[ \text{output} \]

\[ \text{output} \]
Fork on every reduction step, maximal selectivity
JOIN PATTERNS

//initialize buffer with the first frame
srcs<frame> if frame.id == 0 ▷
    buffer<frame :: Nil> || last<0>

//append to buffer in the order of the id
srcs<frame> & buffer<xs> & last<n>
    if frame.id == n + 1 ▷
        buffer<xs.frame> || last<frame.id>

//output buffer in chunks
buffer<fs> if fs.length ≥ N ▷
    buffer<fs.drop(N)> || out<fs.take(N)>

Synchronization = Correlation!