Target Application: Arrhythmia Detection

Heart signal

Boolean signal

Arrhythmia detection algorithm

Deliver shock

No action

“Real-time decision-making”
Target Application: Arrhythmia Detection

Heart signal

Boolean signal

Streaming Algorithm:
- item-by-item
- constant time per-element
- constant space

Arrhythmia detection algorithm

Deliver shock

No action

“No real-time decision-making”
Target Application: Runtime Monitoring

- Online system
- Instrumentation
- Diagnostic event stream
- Declarative Specification
  - Monitor
    - Report violation
    - No action

Streaming Algorithm:
- item-by-item
- constant time per-element
- constant space
Target Application: Runtime Monitoring

Online system
Instrumentation
Diagnostic event stream

Declarative Specification
Monitor
Report violation
No action

Streaming Algorithm:
- item-by-item
- constant time per-element
- constant space
Existing Declarative Specification Languages

Runtime monitoring:
- past-LTL (Havelund and Roșu, 2004)
- Eagle (Barringer et al., 2004)
- LOLA (d’Angelo et al., 2005)
- ...

Real-time decision making:
- StreamQRE (Mamouras et al., 2017)
- ...

Differences: expressiveness; succinctness and evaluation cost of compiled monitor; modularity of compilation
Modular Compilation

Classical regular expression evaluation: use NFAs

\[ \text{NFA}(R \cdot S) := \text{NFA}(R) \cdot \varepsilon \cdot \text{NFA}(S) \]

NFAs = succinct \textbf{intermediate representation}

\textbf{Benefits:}

- Compilation for free!
- Extensibility
- Guaranteed cost of evaluation
Main Contribution: General-Purpose Streaming IR

We propose a general-purpose intermediate representation (IR) for data-stream monitoring applications.

Why another model?
- Expressiveness \((quantitative + temporal properties)\)
- Modular compilation \((composition operations on DTs)\)
- Efficient streaming evaluation algorithm
- Maximally succinct
Main Contribution: General-Purpose Streaming IR

We propose a general-purpose intermediate representation (IR) for data-stream monitoring applications.

Why another model?
- Expressiveness (quantitative + temporal properties)
- Modular compilation (composition operations on DTs)
- Efficient streaming evaluation algorithm
- Maximally succinct
Outline

The Model: Quantitative States + Transitions

Composition Operations

Expressiveness (vs. all Streaming Computations)

QRE-Past Illustration
The Model: Quantitative States + Transitions

Quantitative states: $q_0$, $q_1$, $q_2$

- Active with a value or Inactive

Input stream: $(a, 6)(a, 7)(b, 9)(a, 11)$

Transitions update quantitative states
“What was the delay between the last two a-messages?”
The Model: Quantitative States + Transitions

“What was the delay between the last two a-messages?”

Transitions

- $q_0$: Active (initial state)
- $q_1$: Active with last a-value OR Inactive ($\bot$)
- $q_2$: Active with difference of last two a-values OR Inactive ($\bot$)
The Model: Quantitative States + Transitions

“What was the delay between the last two a-messages?”

Transitions

Input stream: (a, 6)

0 q₀: Active (initial state)
6 q₁: Active with last a-value OR Inactive (⊥)
⊥ q₂: Active with difference of last two a-values OR Inactive (⊥)
The Model: Quantitative States + Transitions

“What was the delay between the last two $a$-messages?”

Transitions

Input stream: $(a, 6)(a, 7)$

0. $q_0$: Active (initial state)

7. $q_1$: Active with last $a$-value OR Inactive ($\perp$)

1. $q_2$: Active with difference of last two $a$-values OR Inactive ($\perp$)
The Model: Quantitative States + Transitions

“What was the delay between the last two a-messages?”

Transitions

Input stream: \((a, 6)(a, 7)(b, 9)\)

0. \(q_0\): Active (initial state)

7. \(q_1\): Active with last a-value OR Inactive (⊥)

1. \(q_2\): Active with difference of last two a-values OR Inactive (⊥)
The Model: Quantitative States + Transitions

“What was the delay between the last two a-messages?”

Transitions

Input stream: \((a, 6)(a, 7)(b, 9)(a, 11)\)

\begin{itemize}
  \item \(q_0\) : Active (initial state)
  \item \(q_1\) : Active with last a-value OR Inactive (⊥)
  \item \(q_2\) : Active with difference of last two a-values OR Inactive (⊥)
\end{itemize}
The Model: Quantitative States + Transitions

“What was the delay between the last two a-messages?”

Transitions

Input stream: \((a, 6)(a, 7)(b, 9)(a, 11)\)

0 \(q_0\): Active (initial state)

11 \(q_1\): Active with last a-value OR Inactive (⊥)

4 \(q_2\): Active with difference of last two a-values OR Inactive (⊥)

Matches (produces output on) a regular language over \(\{a, b\}^*\)

Active/Inactive \(\equiv\) finite control
Active with value \(\equiv\) data register
Transitions, More Generally

Transition with multiple source states:

\[ q_1 \xrightarrow{a} q_3' := q_1 / q_2 \]

\[ q_2 \]

Average Delay = \( \frac{\text{Total Delay}}{\text{Count}} \)

- \( \bot / \bot = \bot \)
- \( \bot / 3 = \bot \)
- \( 9 / \bot = \bot \)
- \( 9 / 3 = 3 \)
Transitions, More Generally

Transition with multiple source states:

arbitrary expression over \( q_1, q_2 \)

![Diagram showing transitions and states](image)

Average Delay = \( \frac{\text{Total Delay}}{\text{Count}} \)

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \bot / \bot )</td>
<td>( \bot )</td>
</tr>
<tr>
<td>( \bot / 3 )</td>
<td>( \bot )</td>
</tr>
<tr>
<td>( 9 / \bot )</td>
<td>( \bot )</td>
</tr>
<tr>
<td>( 9 / 3 )</td>
<td>3</td>
</tr>
</tbody>
</table>
Transitions, More Generally

Transition with multiple source states:

\[ q_1 \xrightarrow{a} q_3 ; q'_3 := q_1 / q_2 \]

Average Delay = \( \frac{\text{Total Delay}}{\text{Count}} \)

\[
\begin{align*}
\bot / \bot &= \bot \\
\bot / 3 &= \bot \\
9 / \bot &= \bot \\
9 / 3 &= 3
\end{align*}
\]

Multiple transitions to a state:

\[ q_3 \xrightarrow{a} q_3 \]

\[
\begin{align*}
\bot, \bot &\mapsto \bot \\
\bot, x &\mapsto x \\
x, \bot &\mapsto x \\
x, y &\mapsto ?
\end{align*}
\]

Special state mode: \textbf{conflict} \( \top \)

Behaves like: “multiset of two or more values”

\[
\begin{align*}
\top / x &= \top \\
\top / \bot &= \bot
\end{align*}
\]
Data Transducers: Final Definition

1. Finite set of **quantitative states**
   - Inactive \( \perp \)
   - or Active with a value \( \times \)
   - or Conflict \( \top \)

2. Finite set of **transitions**
   - Label \( e \): new value of a state in terms of other states
   - Epsilon transitions \( \varepsilon \)

3. **Initial states** and **final states**
Composition Operations: Maximum

**Compute:** \( \max(f_1(w), f_2(w)) \) (whenever both match)
Composition Operations: Split

Compute: $op(f_1(w_1), f_2(w_2))$ for unique split $w = w_1 w_2$ ($f_1$ matches $w_1$, $f_2$ matches $w_2$)
Expressiveness (vs. all Streaming Computations)

$f$ computable by DTs $\subseteq$ Streamable $f$

- initialize : () $\rightarrow$ state
- update : state $\times$ $X$ $\rightarrow$ state
- output : state $\rightarrow$ option $Y$

Why not arbitrary loop-free code?

- Lacks composition under $\text{split}(f, g, op)$.
- Arbitrary loop-free code with “regularity restriction” $\rightarrow$ DTs
  - matching complexity (see paper)
QRE-Past Illustration

- Quantitative constructs \((\text{max}, \text{split}, \ldots)\)
- Temporal operators \((\text{always in the past}, \ldots)\)

Monitoring property: delay between adjacent a-messages is always at most 5

\[
\begin{align*}
\square \quad (\text{split}(f_{\text{prev}}, f_{\text{lasttwo}}, \lambda x y \rightarrow y) \leq 5)
\end{align*}
\]

always in the past
Theorem

For every well-typed query $\alpha$, the compilation via the constructions on Data Transducers produces a DT $A_\alpha$ of quadratic size in the length of $\alpha$ which implements the semantics.

- Cost of DT evaluation: $O(|A|)$ time per element of the input stream
- Cost of QRE-Past evaluation ($A_\alpha$): $O(|\alpha|^2)$ time per element

Assumptions: data values are stored in constant space; operations on data values take constant time
Conclusion

Declarative monitoring language

IR: Data Transducers

Composition operations

Input data stream  Monitor  Output

• Expressiveness (quantitative + temporal properties)
• Modular compilation (composition operations on DTs)
  ▶ QRE-Past
• Efficient streaming evaluation algorithm
• Maximally succinct
Future Work

- **Implementation** of DTs for domain-specific languages
  - wearable medical monitors
  - network traffic monitoring
  - streaming IoT devices

- **Query optimization** using decidable properties of DTs
• **Data Transducers**: a general-purpose IR for monitoring data streams

• **QRE-Past**: DT composition operations enable modular compilation with guaranteed cost of evaluation.
Succinctness

(Alur et al., 2013)

Unambiguous CRA

exp, lower bound

Restartable DT

exp

DFA

exp, uniform l.b.

NFA

exp, l.b.

DT

reversed AFA

(Chandra, Kozen, and Stockmeyer, 1981)

(Salomaa, Wu, and Yu, 2000)