An Efficient Algorithm for Learning Event-Recording Automata

Shang-Wei LIN, Étienne André, Jin Song DONG, Jun SUN, and Yang LIU

Department of Computer Science
School of Computing
National University of Singapore

October 14, 2011
Motivation

Why is learning of models important?

- Automatic inference or construction of abstract models
Outline

The L* Algorithm

Timed Language and Event-Recording Automata

The TL* Algorithm

Conclusion and Future Work
Outline

The L* Algorithm

Timed Language and Event-Recording Automata

The TL* Algorithm

Conclusion and Future Work
The $L^*$ Algorithm

The $L^*$ algorithm is a formal method to learn a minimal DFA that accepts an unknown language $U$ over an alphabet $\Sigma$.


The $L^*$ algorithm interacts with a Minimal Adequate *Teacher*

- *membership query*
  - Is a string in the unknown language $U$?

- *candidate query*
  - Does a DFA accept the unknown language $U$?
The unknown language $U = (a \mid b \mid c) \cdot a^*$

<table>
<thead>
<tr>
<th></th>
<th>$\lambda$</th>
<th>$c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$a$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$b$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$c$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$a \cdot a$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$a \cdot b$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$a \cdot c$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$ab \cdot a$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$ab \cdot b$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$ab \cdot c$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Outline

The L* Algorithm

Timed Language and Event-Recording Automata

The TL* Algorithm

Conclusion and Future Work
Timed Language

Let $\Sigma$ be a finite alphabet.

For every symbol (event) $a \in \Sigma$, we use $x_a$ to denote the event-recording clock of the symbol $a$

- $x_a$ records the time elapsed since the last occurrence of the symbol $a$
- We use $C_\Sigma$ to denote the set of event-recording clocks over $\Sigma$

An atomic clock guard $\tau$ is an inequation of the form $x_a \sim n$ for $x_a \in C_\Sigma$, $\sim \in \{<, \leq, >, \geq\}$, and $n \in N$.

- A clock guard $g$ is a conjunction of atomic clock guards.
- We use $G_\Sigma$ to denote the set of clock guards over $C_\Sigma$

A guarded word is a sequence $w_g = (a_1, g_1)(a_2, g_2) \cdots (a_n, g_n)$ where $a_i \in \Sigma$ for $i \in \{1, 2, \ldots, n\}$ and $g_i \in G_\Sigma$ is a clock guard.
Event-Recording Automata

An \textit{event-recording automaton} (ERA) $D = (\Sigma, L, l_0, \delta, L^f)$ consists of

- a finite input \textit{alphabet} $\Sigma$
- a finite set of \textit{locations} $L$
- an \textit{initial location} $l_0 \in L$
- a \textit{transition function} $\delta : L \times \Sigma \times G_\Sigma \mapsto 2^L$
- a set of \textit{accepting locations} $L^f \subseteq L$

Each event-recording clock $x_a \in C_\Sigma$ is implicitly and automatically \textit{reset} when a transition with event $a$ is taken.
Event-Recording Automata (cont.)

An event-recording automaton $D = (\Sigma, L, l_0, \delta, L^f)$ is deterministic if

- $\delta(l, a, g)$ is a singleton set when it is defined
- if both $\delta(l, a, g_1)$ and $\delta(l, a, g_2)$ are both defined, then $[[g_1]] \cap [[g_2]] = \emptyset$

A guarded word $w_g = (a_1, g_1)(a_2, g_2) \cdots (a_n, g_n)$ is accepted by an ERA $D = (\Sigma, L, l_0, \delta, L^f)$ if

- $l_i = \delta(l_{i-1}, a_i, g_i)$ is defined for all $i \in \{1, 2, \ldots, n\}$
- $l_n \in L^f$

The timed language accepted by $D$, denoted by $\mathcal{L}(D)$, is the set of guarded words accepted by $D$. 
The following ERA $\mathcal{A}_1$ accepts the timed language
$U_T = ((a, x_a = 1)(a, x_a = 3))^*$
Outline

The L\textsuperscript{*} Algorithm

Timed Language and Event-Recording Automata

The TL\textsuperscript{*} Algorithm

Conclusion and Future Work
The TL* Algorithm

The TL* algorithm is a timed extension of the L* algorithm.

The TL* algorithm is a formal method to learn a minimal event-recording automaton (ERA) that accepts an unknown timed language $U_T$ over an alphabet $\Sigma$.

- We use $U$ to denote the untimed language of $U_T$.
The TL* Algorithm (cont.)

The TL* algorithm has to interact with a Minimal Adequate Teacher

- **untimed membership query** $Q_m$
  - Is an untimed word in the unknown untimed language $U$?

- **untimed candidate query** $Q_c$
  - Does a DFA accept the unknown untimed language $U$?

- **timed membership query** $Q^T_m$
  - Is a guarded word in the unknown timed language $U_T$?

- **timed candidate query** $Q^T_c$
  - Does an ERA accept the unknown timed language $U_T$?
The TL* Algorithm (cont.)

The TL* algorithm consists of two phases

▶ *Untimed Learning* Phase
  ▶ The L* algorithm is used to learn a DFA $M$ accepting the untimed language $U$

▶ *Timed Refinement* Phase
  ▶ The DFA $M$ is refined into an event-recording automaton (ERA) by adding time constraints
The TL* Algorithm (cont.)

input: \( \Sigma \): alphabet, \( C_\Sigma \): the set of event-recording clocks
output: a deterministic ERA accepting the unknown timed language \( U_T \)

Use \( L^* \) to learn a DFA \( M \) accepting \( \text{Untime}(U_T) \);
Let \((S, E, T)\) be the observation table during the \( L^* \) learning process;
Replace \( \alpha \) by \((\alpha, \text{true})\), \( s \) by \((s, \text{true})\), and \( e \) by \((e, \text{true})\) for each \( \alpha \in \Sigma, s \in S \) and \( e \in E \);
while \( true \) do
  if \( Q^T_c(M) = 1 \) then return \( M \);
  else
    Let \((a_1, g_1)(a_2, g_2) \cdots (a_n, g_n)\) be the counterexample given by Teacher;
    foreach \((a_i, g_i)\), \( i \in \{1, 2, \ldots, n\} \) do
      if \((a_i, g)\) is a substring of \( p \) or \( e \) for some \( p \in S \cup (S \cdot \Sigma) \) and \( e \in E \) such that \([g_i] \subset [g] \) then
        Let \( G = \{\hat{g}_1, \hat{g}_2, \ldots, \hat{g}_m\} \) obtained by \([g] - [g_i] \);
        \( \Sigma = \Sigma \setminus \{(a_i, g)\} \cup \{(a_i, g_i), (a_i, \hat{g}_1), (a_i, \hat{g}_2), \ldots, (a_i, \hat{g}_m)\} \);
        Split \( p \) into \( \hat{p}_0, \hat{p}_1, \hat{p}_2, \ldots, \hat{p}_m \) where \((a_i, g_i)\) is a substring of \( \hat{p}_0 \) and \((a_i, \hat{g}_j)\) is a substring of \( \hat{p}_j \) for all \( j \in \{1, 2, \ldots, m\} \);
        Split \( e \) into \( \hat{e}_0, \hat{e}_1, \hat{e}_2, \ldots, \hat{e}_m \) where \((a_i, g_i)\) is a substring of \( \hat{e}_0 \) and \((a_i, \hat{g}_j)\) is a substring of \( \hat{e}_j \) for all \( j \in \{1, 2, \ldots, m\} \);
        Update \( T \) by \( \varphi_mT(\hat{p}_j \cdot \hat{e}_j) \) for all \( j \in \{0, 1, 2, \ldots, m\} \);
    while \( \text{there exists } (s \cdot \alpha) \text{ such that } s \cdot \alpha \not\equiv s' \text{ for all } s' \in S \) do
      \( S \leftarrow S \cup \{s \cdot \alpha\} \);
      Update \( T \) by \( \varphi_mT((s \cdot \alpha) \cdot \beta) \) for all \( \beta \in \Sigma \);
      \( v \leftarrow \text{WS}((a_1, g_1)(a_2, g_2) \cdots (a_n, g_n)) \);
      if \( |v| > 0 \) then
        \( E \leftarrow E \cup \{v\} \);
        Update \( T \) by \( \varphi_mT(s \cdot v) \) and \( \varphi_mT(s \cdot \alpha \cdot v) \) for all \( s \in S \) and \( \alpha \in \Sigma \);
    Construct candidate \( M \) from \((S, E, T)\);
An Example

Suppose \( U_T = ((a, x_a = 1)(a, x_a = 3))^* \) is the timed language to be learned.

Untimed Learning Phase

\[
\begin{array}{c|cc}
\lambda & \lambda & 1 (s_0) \\
\hline
\lambda & 1 & 1 \\
a & 1 & 1 \\
\end{array}
\]

(a) \( T_1 \)

(b) \( M_1 \)

\[
\begin{array}{c|cc}
\lambda & \lambda & 1 (s_0) \\
\hline
\lambda & 1 & 1 \\
(a, \text{true}) & 1 & 1 \\
\end{array}
\]

(c) \( T_2 \)

\[\mathcal{L}(M_1) = U = a^*\]
$Q^T_c(M_1) = 0$ with a negative counterexample $(a, x_a < 1)$

Timed Refinement 1

(a) $T_3$

<table>
<thead>
<tr>
<th></th>
<th>$\lambda$</th>
<th>$\lambda$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda$</td>
<td>$1 (s_0)$</td>
<td>$1 (s_0)$</td>
</tr>
<tr>
<td>$(a, x_a &lt; 1)$</td>
<td>0</td>
<td>$(a, x_a &lt; 1)$</td>
</tr>
<tr>
<td>$(a, x_a \geq 1)$</td>
<td>0</td>
<td>$(a, x_a \geq 1)$</td>
</tr>
</tbody>
</table>

(b) $T_4$

<table>
<thead>
<tr>
<th></th>
<th>$\lambda$</th>
<th>$\lambda$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda$</td>
<td>$1 (s_0)$</td>
<td>$1 (s_0)$</td>
</tr>
<tr>
<td>$(a, x_a &lt; 1)$</td>
<td>0</td>
<td>$(a, x_a &lt; 1)$</td>
</tr>
<tr>
<td>$(a, x_a \geq 1)$</td>
<td>0</td>
<td>$(a, x_a \geq 1)$</td>
</tr>
</tbody>
</table>

(c) $M_2$
An Example (cont.)

\[ Q_c^T(M_2) = 0 \] with a positive counterexample \((a, x_a = 1)\)

Timed Refinement 2

\[
\begin{array}{ccc|c}
\lambda & \lambda \\
(a, x_a < 1) & 1 (s_0) \\
(a, x_a = 1) & 0 (s_1) \\
(a, x_a > 1) & 1 \\
(a, x_a < 1)(a, x_a < 1) & 0 \\
(a, x_a < 1)(a, x_a = 1) & 0 \\
(a, x_a < 1)(a, x_a > 1) & 0 \\
\end{array}
\]

(a) \( T_5 \)

(b) \( M_3 \)
An Example (cont.)

\[ Q_c^T(M_3) = 0 \] with a negative counterexample \((a, x_a = 1)(a, x_a = 1)\)

A suffix \((a, x_a = 1)\) shows that \(\lambda\) and \((a, x_a = 1)\) should not be in the same class

Timed Refinement 3

<table>
<thead>
<tr>
<th>(\lambda)</th>
<th>(\lambda)</th>
<th>(a, x_a = 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>((a, x_a &lt; 1))</td>
<td>1</td>
<td>1 ((s_0))</td>
</tr>
<tr>
<td>((a, x_a = 1))</td>
<td>0</td>
<td>0 ((s_1))</td>
</tr>
<tr>
<td>((a, x_a &gt; 1))</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>((a, x_a &lt; 1)(a, x_a &lt; 1))</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>((a, x_a &lt; 1)(a, x_a = 1))</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>((a, x_a &lt; 1)(a, x_a &gt; 1))</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(a) \(T_6\)

<table>
<thead>
<tr>
<th>(\lambda)</th>
<th>(\lambda)</th>
<th>(a, x_a = 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>((a, x_a &lt; 1))</td>
<td>1</td>
<td>1 ((s_0))</td>
</tr>
<tr>
<td>((a, x_a = 1))</td>
<td>0</td>
<td>0 ((s_1))</td>
</tr>
<tr>
<td>((a, x_a &gt; 1))</td>
<td>0</td>
<td>0 ((s_2))</td>
</tr>
<tr>
<td>((a, x_a &lt; 1)(a, x_a &lt; 1))</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>((a, x_a &lt; 1)(a, x_a = 1))</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>((a, x_a &lt; 1)(a, x_a &gt; 1))</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>((a, x_a = 1)(a, x_a &lt; 1))</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>((a, x_a = 1)(a, x_a = 1))</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>((a, x_a = 1)(a, x_a &gt; 1))</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(b) \(T_7\)

(c) \(M_4\)
An Example (cont.)

\[ Q_c^T(M_4) = 0 \] with a positive counterexample \((a, x_a = 1)(a, x_a = 3)\)

Timed Refinement 4

\[
\begin{array}{c|cc}
\lambda & \lambda & (a, x_a = 1) \\
\hline
(a, x_a < 1) & 1 & 1 (s_0) \\
(a, x_a = 1) & 0 & 0 (s_1) \\
(a, 1 < x_a < 3) & 0 & 0 \\
(a, x_a = 3) & 0 & 0 \\
(a, x_a > 3) & 0 & 0 \\
(a, x_a < 1)(a, x_a < 1) & 0 & 0 \\
(a, x_a < 1)(a, x_a = 1) & 0 & 0 \\
(a, x_a < 1)(a, 1 < x_a < 3) & 0 & 0 \\
(a, x_a < 1)(a, x_a = 3) & 0 & 0 \\
(a, x_a < 1)(a, x_a > 3) & 0 & 0 \\
(a, x_a = 1)(a, x_a < 1) & 0 & 0 \\
(a, x_a = 1)(a, x_a = 1) & 0 & 0 \\
(a, x_a = 1)(a, 1 < x_a < 3) & 0 & 0 \\
(a, x_a = 1)(a, x_a = 3) & 0 & 0 \\
(a, x_a = 1)(a, x_a > 3) & 0 & 0 \\
\end{array}
\]
An Example (cont.)

\[ Q_c^T(M_5) = 1, \text{ i.e., } \mathcal{L}(M_5) = U_T \]

The learning process of TL* is finished
Analysis of TL*

Given a timed language $U_T$ accepted by an ERA $\mathcal{A} = (\Sigma, L, l_0, \delta, L_f)$, the TL* algorithm needs to perform

$\mathcal{O}(|\Sigma| \cdot |G_\mathcal{A}| \cdot |L|^2 + |L| \log |\pi|)$ timed membership queries

$\mathcal{O}(|L| + |\Sigma| \cdot |G_\mathcal{A}|)$ timed candidate queries

Grinchtein’s TL$^\star_{sg}$ needs $\mathcal{O}(|\Sigma \times G_\Sigma| \cdot n^2 |\pi| \cdot |w| (\frac{|\Sigma| + K}{|\Sigma|})$ timed membership queries

$\mathcal{n}$ is the number of locations of the learned ERA

$w$ is the longest guarded word queried

$K$ is the largest constant appearing in the clock guards
Analysis of TL* (cont.)

Theorem
The TL* algorithm is correct.

Theorem
The TL* algorithm terminates.

Theorem
Assume the observation table \((S, E, T)\) is closed and consistent and \(M = (\Sigma, L, l_0, \delta, L_f)\) is the ERA constructed from the observation table \((S, E, T)\). If \(M' = (\Sigma, L', l_0', \delta', L'_f)\) is any other ERA consistent with \(T\), then \(M'\) has at least \(|L|\) locations.
Outline

The L* Algorithm

Timed Language and Event-Recording Automata

The TL* Algorithm

Conclusion and Future Work
Conclusion and Future Work

Conclusion

▶ We proposed an efficient polynomial time algorithm, TL*, for learning event-recording automata (ERA).
▶ The TL* algorithm has been implemented in the PAT model checker.

Future Work

▶ To extend the TL* algorithm to learn other subclasses of timed automata
▶ To automate assume-guarantee reasoning (AGR) for timed systems, we plan to use TL* to automatically generate timed assumptions needed for AGR.