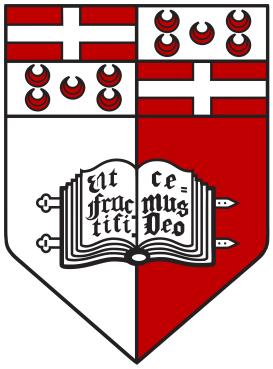


Runtime Verification of Timed Regular Expressions in Larva

Progress Report

Miguel Dingli (49997M)
Supervisor: Prof. Gordon Pace
Co-Supervisor: Dr. Christian Colombo

Contents

1 Introduction **1**
1.1 Aims and Objectives 1

2 Background **1**
2.1 Timed Regular Expressions 1
2.1.1 Syntax of TREs 2
2.1.2 Semantics of TREs 2
2.2 Timed Derivatives 3
2.3 Timed Automata 4

3 Work Done **4**
3.1 The Tool 4
3.2 The Use Case 4

4 Related Work **4**

5 Evaluation **5**

6 Time Plan **5**

7 Appendix **6**
7.1 Some Definitions 6
7.2 Representations 6

1 Introduction

Runtime verification (RV) [1] is a sector of verification techniques dedicated to the monitoring of the behaviour of a system without altering its underlying code. This allows the concerns of normal behaviour of the system to be kept distinct from those of verification, thus providing the ability to easily turn on or off the verification. The expected behaviour of a system is described by a formal specification written in a specification language. Given such a specification and the system itself, an RV tool automatically introduces checks into the system which enable any illegal behaviour defined by the properties in the specification to be identified during runtime.

A possible specification language is that of Timed Regular Expressions (TREs) [2], which are an extension over traditional regular expressions (REs) with the additional support for expressing timed elements. Regular expressions in general can be used as a means to specify the properties of a system which are then monitored during runtime. On top of this, TREs provide the ability to specify *timed* properties. In comparison, the monitoring of TREs is significantly more challenging than monitoring classical REs, due to the aspect of time.

Runtime verification tools come with their own native property language for writing specifications, and so for the properties to be expressed using a logic such as TREs, these properties have to be translated into the tool's native language. In this project, the runtime verification tool Larva [3] will be used. As its native property language, Larva uses an automaton-based formalism called Dynamic Automata with Timers and Events (DATEs).

1.1 Aims and Objectives

The aim of this project is to use the runtime verification tool Larva to investigate different means by which TREs can be monitored. These means will take the form of translations from TREs to Larva's native property language, DATEs. The following objectives will be followed:

Objective 1. Develop three ways of monitoring TREs, which will be encapsulated in a tool that translates TREs into DATEs. The following are the proposed approaches:

1. As the first approach, the notion of the derivative of a RE [4] will be extended to TREs and will be used by a DATE to obtain the derivative when an event occurs.
2. The next approach would be to convert the TRE into a timed automaton (TA) [5] which is then easily converted into a DATE.
3. The third approach is to lazily unfold the TRE on-the-fly using derivatives.

Objective 2. The performance of the three approaches will be compared to find the most efficient one by applying them on a use case for which a number of properties will be written.

2 Background

2.1 Timed Regular Expressions

Timed Regular Expressions (TREs) are an extension of the classical regular expressions which introduce the concept of time and the time interval operator, $\langle E \rangle_{[t,t']}$, to traditional regular expressions, meaning that as opposed to traces of only symbols, timed regular expressions are concerned with sequences of events with explicit time duration information. These can take the form of time-event sequences or piecewise-continuous signals. Asarin et al. [2] defined TREs with the aim of being able to specify discrete behaviour augmented with timing information. The syntax introduced by Asarin et al. includes the mentioned interval operator which, applied to a sequence of events, restricts the time of occurrence of the events to a specific time bound.

Additional definitions and representations used in this seciton were included in the appendix.

2.1.1 Syntax of TREs

Definition 1 (Syntax of TREs)**.** *Based on a modified version of the syntax of TREs described by Asarin et al. [2], timed regular expressions E over an alphabet Σ are defined as follows:*

$$Base ::= a \mid ? \mid 0 \mid 1$$

$$E ::= Base \mid \sim E \mid E^* \mid E \,\&\, E \mid E + E \mid EE \mid \langle E \rangle_{[t,t']}$$

...where $t, t' \in \mathbb{R}_{\geq 0}$, $t < t'$, $a \in \Sigma$, $?$ matches any symbol, 0 matches nothing, 1 matches only the empty string λ, and $[t, t']$ represents an actual interval $[t, t')$, i.e. the upper limit is excluded.

For example, for a login system, the property that every three consecutive bad logins 'b' must be followed by a good login 'g' can be written as $\left((\sim b)^* \, bbbg\right)^*$, which includes a prefix of zero or more non-'b' events until the first bad login occurs. On the other hand, the property to detect three bad logins within 10 seconds is $(?)^* \, \langle bbb \rangle_{[0,10]}$. The former property specifies good behaviour while, for the latter, an accepted time-event sequence indicates bad behaviour.

2.1.2 Semantics of TREs

Besides the time interval operator, TREs also differ from their untimed counterparts in that a symbol $a \in \Sigma$ has an associated implicit arbitrary time value $t \in \mathbb{R}_{\geq 0}$, due to the fact that TREs deal with time-event sequences. These will collectively be denoted by ${}_t a$. The time value represents the time elapsed between the previous symbol and the occurrence of a. If no time restriction is placed on t, the event associated with symbol a can occur after any length of time.

In general, the time interval operator applied on an expression E, i.e. $\langle E \rangle_{[t,t']}$, requires the timed words that satisfy E to also satisfy two timing criteria. Firstly, that events start after at least t time units have passed from when the expression started being evaluated, and secondly, when t' time units have passed, the timed word must have led the expression to a state of acceptance. If further events are expected after the time-bound expression $\langle E \rangle_{[t,t']}$, the events may start occurring as soon as and if and only if the expression E is in a state of acceptance.

Definition 2 (Semantics of TREs)**.** *Based on the semantics presented by Asarin et al. [2], a sample of the formal semantics $[\![]\!] : E \to Seq$ of timed regular expressions, where the function $len : Seq \to \mathbb{R}_{\geq 0}$ denotes the time duration of a timed word, is presented below:*

$$[\![0]\!] = \varnothing \qquad [\![a]\!] = \{{}_t a \mid t \in \mathbb{R}_{\geq 0}\}$$

$$[\![1]\!] = \{\lambda\} \qquad [\![\langle E \rangle_{[t,t']}]\!] = \{s \in [\![E]\!] \mid len(s) \in [t, t')\}$$

As an example, consider the property that after an 'a' is received, 'b' and 'c' must occur after at least 5 seconds, but before 10 seconds, given by $a\langle bc \rangle_{[5,10]}$. The semantics evaluate as follows:

$$[\![a\langle bc \rangle_{[5,10]}]\!] = \{{}_{t_1} a \mid t_1 \in \mathbb{R}_{\geq 0}\} \cdot \{s \in [\![bc]\!] \mid len(s) \in [5, 10)\}$$

The first part of this result, $\{{}_{t_1} a \mid t_1 \in \mathbb{R}_{\geq 0}\}$, indicates that a valid timed word must start with 'a', after an arbitrary t_1 time units have elapsed. The second part, $\{s \in [\![bc]\!] \mid len(s) \in [5, 10)\}$, indicates that the timed word must be an element of the timed words accepted by 'bc', $[\![bc]\!]$ (which can be evaluated further), but is also restricted to the time interval $[5, 10)$. This implies that after the 'a' has occurred, 'b' must occur after at least 5 seconds have elapsed, and must be followed by 'c', which must occur before 10 seconds have elapsed.

The above result can be evaluated further and simplified to $\{{}_{t_1} a \, {}_{t_2} b \, {}_{t_3} c \mid (t_2 + t_3) \in [5, 10)\}$, which makes it clear that the sum of the time periods before the 'b' and 'c', which is equivalent to the time units that 'b' and 'c' collectively took to occur, has to be in the specified time interval.

2.2 Timed Derivatives

The concept of the derivative of a RE was introduced by Brzozowski [4] with the aim of being able to easily construct a state diagram from a RE containing any number of logical operators. Derivatives will be extended to TREs defined by Asarin et al. [2] in the form of two types of derivatives to be used in the first and third approaches proposed in section 1.1.

Ulus et al. [6] defined derivatives for TREs based on piecewise-continuous signals so that these may be used for online timed pattern-matching, where the input signal becomes available as time passes. In this project, the derivatives will be based on TREs that are instead concerned with time-event sequences. These are more suitable when dealing with discrete events where the focus is on the time interval between adjacent events rather than the duration of the event.

The notion of a derivative [4] of a regular expression E with respect to a symbol 'a' of the alphabet gives a new regular expression E' such that $[\![E']\!] = \{s \mid as \in [\![E]\!]\}$. Each operator defined on REs applies the derivative to its operand(s) in different ways, with many of them simply applying the derivative to their operands homomorphically. The derivatives defined by Brzozowski [4] were extended so that a derivative can be applied on the time interval operator.

For timed regular expressions, two types of timed derivatives are defined:

1. The first type is a derivative *with respect to* just the time elapsed δt, meaning a derivative after δt time units have passed with no event observed within this time period.
2. The second type is *with respect to* the time elapsed δt and a symbol from the alphabet 'a', which is similar to the first type but with event 'a' observed after δt time units.

Definition 3 (Type 1 Timed Derivatives)**.** *Given (i) a timed regular expression $e \in E$ and (ii) a time difference $\delta t \in \mathbb{R}_{\geq 0}$, the derivative e' of e with respect to δt is denoted by $e \overset{\delta t}{\rightsquigarrow} e'$, where $\rightsquigarrow \subseteq E \times \mathbb{R}_{\geq 0} \times E$ and $\rightsquigarrow$ is functional on $E \times \mathbb{R}_{\geq 0}$, and is defined as follows:*

$$\frac{}{u \overset{\delta t}{\rightsquigarrow} u} \quad (1) \qquad \frac{e \overset{\delta t}{\rightsquigarrow} e' \qquad f \overset{\delta t}{\rightsquigarrow} f' \qquad \delta(e)}{ef \overset{\delta t}{\rightsquigarrow} e'f + f'} \quad (3)$$

$$\frac{e \overset{\delta t}{\rightsquigarrow} e'}{e^* \overset{\delta t}{\rightsquigarrow} e'e^* + 1} \quad (2) \qquad \frac{e \overset{\delta t}{\rightsquigarrow} e' \qquad \delta t < t'}{\langle e \rangle_{[t,t']} \overset{\delta t}{\rightsquigarrow} \langle e' \rangle_{[t \ominus \delta t, t' - \delta t]}} \quad (4)$$

Definition 4 (Type 2 Timed Derivatives)**.** *Given (i) a timed regular expression $e \in E$, (ii) a symbol from the alphabet $a \in \Sigma$, and (iii) a time difference value $\delta t \in \mathbb{R}_{\geq 0}$, the derivative e' of e with respect to a and δt is denoted by $e \xrightarrow{a}_{\delta t} e'$, where $\longrightarrow \subseteq E \times \Sigma \times \mathbb{R}_{\geq 0} \times E$ and $\longrightarrow$ is functional on $E \times \Sigma \times \mathbb{R}_{\geq 0}$, and is defined as follows:*

$$\frac{}{a \xrightarrow{a}_{\delta t} 1} \quad (5) \qquad \frac{e \xrightarrow{a}_{\delta t} e' \qquad f \xrightarrow{a}_{\delta t} f' \qquad \delta(e)}{ef \xrightarrow{a}_{\delta t} e'f + f'} \quad (7)$$

$$\frac{e \xrightarrow{a}_{\delta t} e'}{e^* \xrightarrow{a}_{\delta t} e'e^*} \quad (6) \qquad \frac{e \xrightarrow{a}_{\delta t} e' \qquad \delta t \in [t, t')}{\langle e \rangle_{[t,t']} \xrightarrow{a}_{\delta t} \langle e' \rangle_{[t \ominus \delta t, t' - \delta t]}} \quad (8)$$

(In the above definitions, only a sample of the inference rules are presented).

Acceptance example: $(a\ \langle b \rangle_{[3,5]}\ (c + d)) \xrightarrow{a}_{10} (\langle b \rangle_{[3,5]}\ (c + d)) \xrightarrow{b}_{4} (c + d) \xrightarrow{d}_{7} 1$ (acceptance)

Rejection example: $(a\ \langle b \rangle_{[3,5]}\ (c + d)) \xrightarrow{a}_{10} (\langle b \rangle_{[3,5]}\ (c + d)) \overset{5}{\rightsquigarrow} 0$ (time interval limit exceeded)

2.3 Timed Automata

For the second TREs-to-DATEs translation approach proposed in section 1.1, timed automata (TAs) were explored. TAs were originally defined by Alur et al. [5] to be used in the modelling of the behaviour of real-time systems. In a TA, transitions are not only based on the event that occurs but also on conditions placed on a finite number of real-time clocks, which can be reset during state transitions. These clocks give TAs the ability to accept timed words from a corresponding timed language, just like TREs. Besides introducing TREs, Asarin et al. [2] used a minor modification of the definition of TAs of Alur et al. [5] to present a method to convert a TRE to a TA which accepts finite timed words. This is precisely the conversion that will be used for the second approach, where a DATE can then be easily derived from the resultant TA.

3 Work Done

3.1 The Tool

The tool mentioned in section 1.1 will contain three forms of translations from TREs to DATEs. The implementation of the tool so far tackles the first translation approach using the defined timed derivatives. The tool parses TREs and outputs a Larva script containing a property for each TRE. The alphabet Σ is inferred from the input. The input expression also indicates whether the expression describes good or bad behaviour, which determines the condition that leads a DATE to a bad state. This adds flexibility to how a property is specified.

Derivative calculation is implemented as a TREs library and a TRE holder. The library provides a way to construct TREs and to obtain their derivatives, whereas the main task of the holder is to keep track of the time that elapses between the occurrence of events. The TRE holder also performs automatic timeout derivatives after a time window has been exceeded, so that if a property is expected to fail due to the exceeded time window, this is detected.

3.2 The Use Case

The main use case selected for this project is an open source file transfer client called JFtp [7]. This client uses various protocols such as FTP, HTTP,and file I/O to enable the user to transfer files between offline locations or between networks.

As proposed in section 1.1, using the tool being implemented, the three approaches will be applied to this system, for which a number of properties expressed using timed regular expressions will be written. Examples of such properties are the following:

- There should never be more than three bad logins within a ten-second time window.
- If no data is sent to a server for five minutes, a timeout should logout the user.
- For every data packet sent, an acknowledgement should be received within one second.

4 Related Work

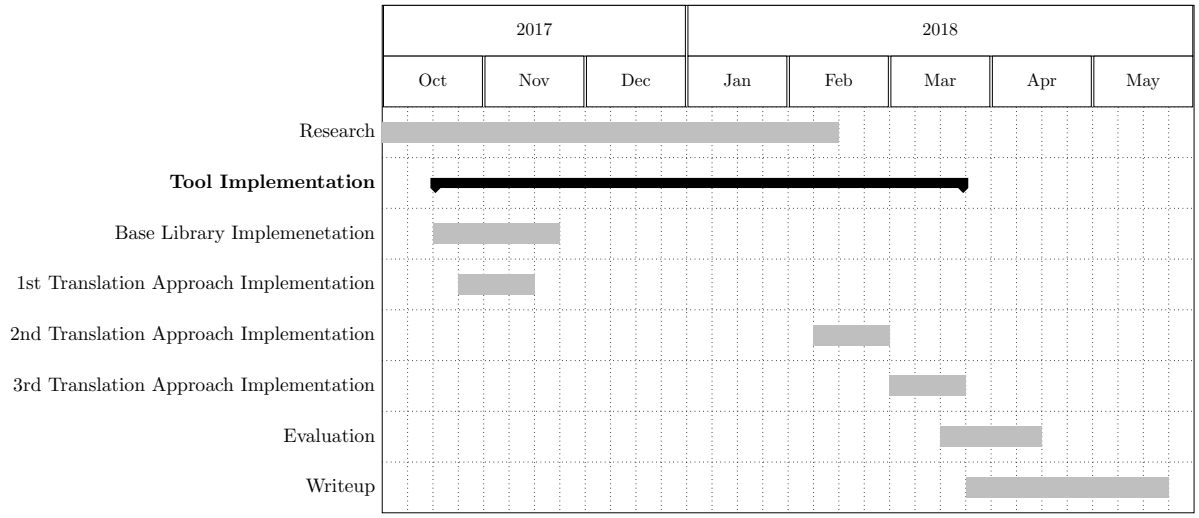
Based his theory of online timed pattern-matching [6], Dogan Ulus presented a tool for the online and offline monitoring of TREs called, MONTRE [8]. In contrast with Larva, TREs are the native property language in MONTRE. Due to the inclusion of both online and offline monitoring algorithms, this tool is able to match patterns, specified by TREs, both in streamed behaviour during runtime and in logged execution traces. Beyond verification, MONTRE can also be used for specification mining.

5 Evaluation

The approaches proposed in section 1.1 will be evaluated on the basis of efficiency in terms of memory and temporal overheads induced to find the best approach. The efficiency metrics will be computed for each of the properties specified for the system, since different circumstances are expected to result in different efficiency values. For each approach, the average of these results can then be calculated. The first approach is expected to be less efficient in terms of overheads since it involves computing the derivative of the TRE for each event that occurs during runtime.

6 Time Plan

The following is a Gantt chart representing the time plan:



References

[1] Martin Leucker and Christian Schallhart. A brief account of runtime verification. *J. Log. Algebr. Program.*, 78(5):293–303, 2009.

[2] Eugene Asarin, Paul Caspi, and Oded Maler. Timed regular expressions. *J. ACM*, 49(2):172–206, 2002.

[3] Christian Colombo, Gordon J. Pace, and Gerardo Schneider. Dynamic event-based runtime monitoring of real-time and contextual properties. In *Formal Methods for Industrial Critical Systems*, pages 135–149, 2008.

[4] Janusz A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, 1964.

[5] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[6] Dogan Ulus, Thomas Ferrère, Eugene Asarin, and Oded Maler. Online timed pattern matching using derivatives. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 736–751, 2016.

[7] JFtp - The Java Network Browser. `j-ftp.sourceforge.net/`. Accessed: 2017-12-02.

[8] Dogan Ulus. Montre: A tool for monitoring timed regular expressions. In *Computer Aided Verification*, pages 329–335, 2017.

7 Appendix

7.1 Some Definitions

The following definitions are used throughout the document:

$$\delta(e) \stackrel{\text{def}}{=} \lambda \in [\![e]\!] \tag{9}$$

$$x \ominus y \stackrel{\text{def}}{=} max(0, x - y) \tag{10}$$

$$\infty - x \stackrel{\text{def}}{=} \infty \tag{11}$$

$$\mathbb{R}_{\geq 0} \stackrel{\text{def}}{=} \mathbb{R}_{+} \cup \{0\} \tag{12}$$

$$\Sigma^* \boxplus \mathbb{R}_{\geq 0} \text{ is the set of all time-event sequences.} \tag{13}$$

In other words, $\Sigma^* \boxplus \mathbb{R}_{\geq 0}$ is the set of all sequences made up of an alternation between symbols from $\mathbb{R}_{\geq 0}$ and Σ. For $\Sigma = \{a, b, c\}$, an example would be the sequence ${}_4a\,{}_{10}c\,{}_1b$.

7.2 Representations

Unless otherwise indicated, the following representations are used in the document:

1. Σ represents the alphabet of a timed language.
2. Seq represents the set of all time-event sequences, $\Sigma^* \boxplus \mathbb{R}_{\geq 0}$.
3. r and s represent time-event sequences (i.e. elements of $\Sigma^* \boxplus \mathbb{R}_{\geq 0}$).
4. u and v represent basic regular expressions (i.e. elements of $Base$).
5. e and f represent any general regular expression (i.e. elements of E).
6. e' and f' represent the corresponding derivatives (also elements of E).
7. t and t' represent time values (i.e. elements of $\mathbb{R}_{\geq 0}$).
8. λ represents the empty string, such that $[\![1]\!] = \{\lambda\}$.
9. $\varnothing$ represents the empty set, such that $[\![0]\!] = \varnothing$.
10. I represents the universal set, such that $I = \Sigma^*$
11. $[t, t']$ represents an interval $[t, t')$, i.e. t' is not included.