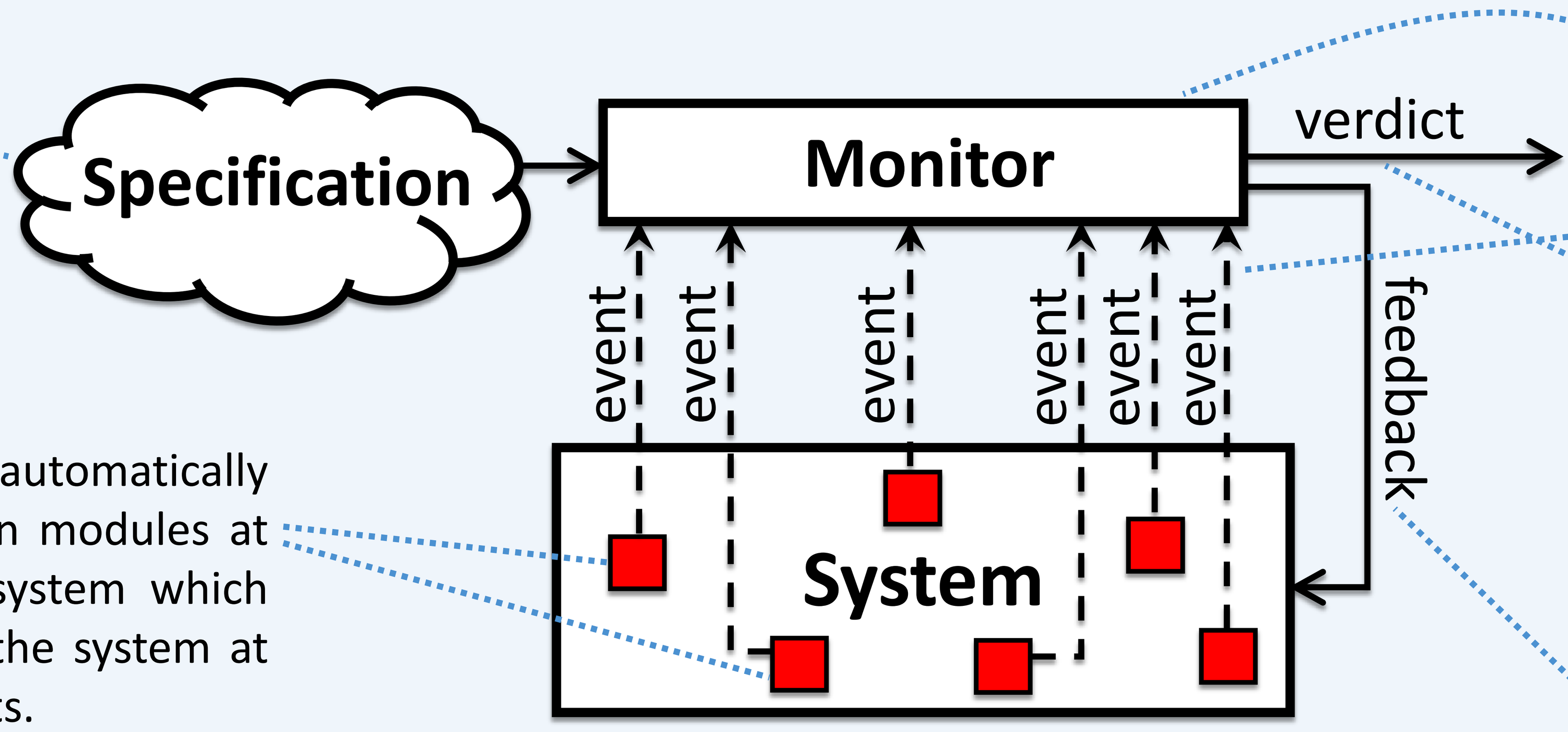
Runtime Verification of Timed Regular Expressions in Larva

Author: Miguel Dingli
Supervisors: Prof. Gordon J. Pace and Dr. Christian Colombo

Runtime Verification

Given a specification describing a system's expected behaviour…

…a runtime verification tool automatically instruments event-extraction modules at various key points in the system which translate the behaviour of the system at runtime into a trace of events.



The RV tool also generates the monitor that will observe the trace of events generated by the system while it is running…

…based on which it decides whether the specification is being satisfied or violated. The monitor may also directly intervene into the system to attempt to solve errors.

Aim

*To explore the potential of the **timed regular expressions** formalism as a specification language using the Larva runtime verification tool and its native automaton-based specification language, DATEs.*

Timed Regular Expressions

Matches Nothing | Matches any event | Matches any non-x event | Concatenation Operator | Union Operator (OR)

$$\mathbb{E} ::= \mathbf{0} \mid \mathbf{1} \mid \mathbf{?} \mid \boldsymbol{x} \mid \boldsymbol{\sim x} \mid \mathbb{E}^{*} \mid \mathbb{E}.\mathbb{E} \mid \mathbb{E}\&\mathbb{E} \mid \mathbb{E}+\mathbb{E} \mid \langle\mathbb{E}\rangle_{[t_1,t_2)}$$

Matches the empty word | Matches event x | Repetition Operator | Intersection Operator (AND) | Time Interval Operator

Example 1: for a system which accepts an input, processes it, and gives back some output, we may require that upon receiving an input, the system must present the output within two seconds.

$$\left(input.\langle process.output\rangle_{[0,2)}\right)^{*}$$

$$\left((\sim write)^{*}.login.(read + write)^{*}.logout\right)^{*}$$

Example 2: for a system which allows users to read or write to files, we may require that the user is only allowed to write to a file after logging into the system. Otherwise, only reading is allowed.

Two Approaches

1. *The cumulative computation of **timed derivatives** of the original timed regular expression.*
2. *The state exploration of a **timed automaton** obtained from the timed regular expression.*

Testing and Real-World Use Case

Testing: The approaches were applied to a mock transaction system (FiTS), using which multiple aspects of the approaches were tested. **Real-World Use Case**: The approaches were also applied to an open-source real-world FTP server called MinimalFTP. Properties, such as:

"For any command processed from an FTP client, a response should be sent back within 1 second."

$$\left((\sim \boldsymbol{procComm})^{*}.\boldsymbol{procComm}.\langle \boldsymbol{sendResp}\rangle_{[0,1)}\right)^{*}$$

What must be obeyed

What must not happen

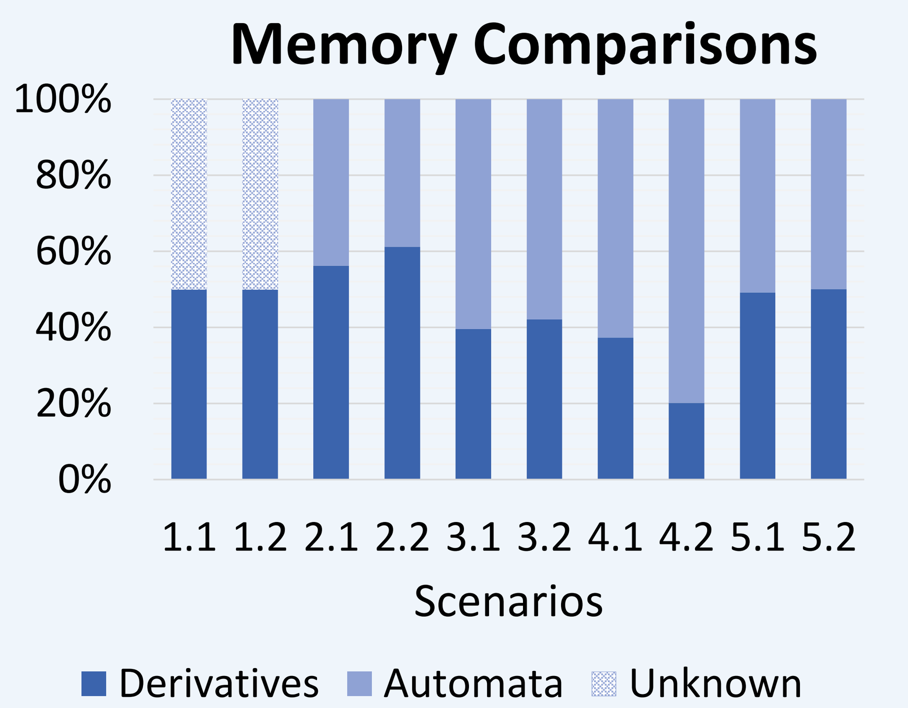
"A user cannot initiate more than four file transfers within a timespan of 10 seconds."

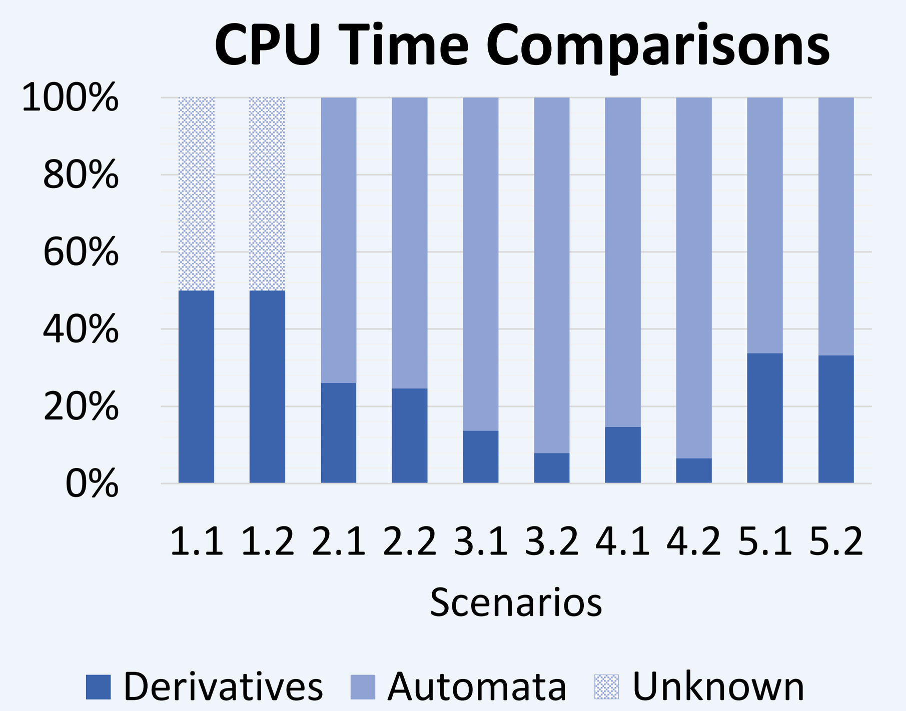
$$?^{*}.\boldsymbol{trans}.\langle \boldsymbol{trans}.\boldsymbol{trans}.\boldsymbol{trans}.\boldsymbol{trans}\rangle_{[0,10)}$$

were defined and, by expressing them as timed regular expressions, the server was monitored using both approaches. The memory and CPU overheads were measured for each of the two approaches.

Results and Conclusions

The timed derivatives approach significantly outperformed the timed automata approach both in terms of the generally lower overhead but also due to an overall simpler implementation.





Future Work

1. *Make a wider use of the features available in Larva.*
2. *Explore and implement silent eventless time periods.*
3. *Apply optimizations to the timed automata approach.*