


# A Theory of (Linear-Time) Timed Monitors (Artifact)

Mouloud Amara ✉


Université Paris Cité, CNRS, IRIF, F-75013 Paris, France

Giovanni Bernardi ✉ 

Université Paris Cité, CNRS, IRIF, F-75013 Paris, France

Mohammed Aristide Foughali ✉ 

Université Paris Cité, CNRS, IRIF, F-75013 Paris, France

Adrian Francalanza ✉ 

Dept. of Computer Science, ICT, University of Malta, Msida, Malta

## Abstract

We provide an OCaml implementation of the logics MTL and  $T^{lin}$ , as well as monitors. Our artefact includes a compiler that translates  $T^{lin}$  formulae into monitors. The generation of a monitor  $M$  from some formula  $\phi$  is decorated with a warning whenever  $\phi$  is not in the syntax of the maximally-expressive monitorable fragment. The resulting monitors being reactive and deterministic, we also

implement their semantics, and provide further a pseudo-monitoring prototype where the monitor incrementally consumes an infinite timed word and reaches a verdict whenever possible. For convenience, for users that prefer to use MTL (bearing in mind the loss of expressivity), we also provide a compiler from MTL to  $T^{lin}$ .

**2012 ACM Subject Classification** Theory of computation → Logic and verification

**Keywords and phrases** Timed logics, Runtime verification, Monitorability

**Digital Object Identifier** 10.4230/DARTS.11.2.8

**Funding** This work received support under the program “Investissement d’Avenir” launched by the French Government and implemented by ANR, with the reference “ANR-18-IdEx-000” as part of its program “Émergence”. Other fundings include the ANR-JST Grant CyPhAI as well as the ANR project MAVeRiQ.

**Related Article** Mouloud Amara, Giovanni Bernardi, Mohammed Aristide Foughali, and Adrian Francalanza, “A Theory of (Linear-Time) Timed Monitors”, in 39th European Conference on Object-Oriented Programming (ECOOP 2025), LIPIcs, Vol. 333, pp. 1:1–1:30, 2025.

<https://doi.org/10.4230/LIPIcs.ECOOP.2025.1>

**Related Conference** 39th European Conference on Object-Oriented Programming (ECOOP 2025), June 30–July 2, 2025, Bergen, Norway

**Evaluation Policy** The artifact has been evaluated as described in the ECOOP 2025 Call for Artifacts and the ACM Artifact Review and Badging Policy.

## 1 Scope

This artefact relates to a theoretical paper where we naturally did not claim any implementational counterpart. The goal of this artefact is rather to provide a prototype implementation (proof of concept) going beyond the compilers from MTL to  $T^{lin}$  and from  $T^{lin}$  to monitors.

A global view relating the implementation with the ECOOP paper is depicted in Fig. 1; the following references to figures and sections concern thus to the ECOOP paper. The function `mtl_to_tlin` implements the compiler from MTL to  $T^{lin}$  shown in Fig. 5. It also adds constraints for recursion variables (Sect. 5.3) to bring the output formula into  $CT^{lin}$  whenever all intervals in the input formula are bounded. The function `tlin_to_mon` implements the compiler from  $T^{lin}$  to monitors (Fig. 10 in the ECOOP paper). The semantics in Fig. 7 + the MPAR rule



© Mouloud Amara, Giovanni Bernardi, Mohammed A. Foughali and Adrian Francalanza;  
licensed under Creative Commons License CC-BY 4.0

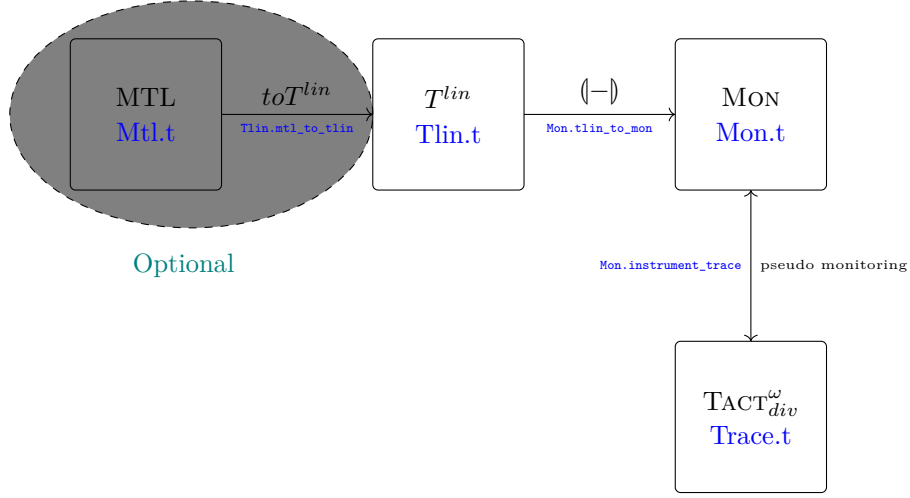
Dagstuhl Artifacts Series, Vol. 11, Issue 2, Artifact No. 8, pp. 8:1–8:3



Dagstuhl Artifacts Series

Schloss Dagstuhl – Leibniz-Zentrum für Informatik,  
Dagstuhl Publishing, Germany





■ **Figure 1** Workflow. In blue, the corresponding Ocaml function/type.

(Sect. 4.1) are implemented within the function `one_step` (this straightforward implementation is possible because our compiler in Fig. 10 outputs reactive deterministic monitors). A prototype with “pseudo-monitoring” is provided within the function `instrument_trace` where the monitor asks repeatedly the monitored program to provide the next timed event (action-timestamp pair) until it reaches a verdict. Note that reaching a verdict is not guaranteed since not all formulae are complete monitorable. For instance, no verdict is reached when monitoring the satisfaction-complete monitorable formula in `formulae/enventually.txt`, namely  $F_{[3,\infty)} b$ , meaning “action  $b$  will be seen in the future within the interval  $[3, \infty)$ ”, with the trace `Trace.t2` =  $(b, 0)(b, 1)(a, 2)(a, 3)(a, 4)(c, 5)(a, 6)(a, 7) \dots$ . This is because `t2` does not satisfy the formula, but there is no way for a sound monitor to say `no` based on a finite prefix not containing  $b$  (see Example 5.18. in the paper).

## 2 Content

The artefact is an OCaml project managed with Dune. The source code is located in the `~/artefact` directory within the provided virtual machine (VM). We also offer a docker image, as well as the directory itself compressed for local use. The code is organised as follows:

- `bin/main.ml`: entry point to the program.
- `lib/lexer.mll` and `lib/parser.mly`: lexer and parser for  $T^{lin}$ .
- `lib/lexer_mtl.mll` and `lib/parser_mtl.mly`: lexer and parser for MTL.
- `alphabet.ml` : type of actions `ACT`.
- `variable.ml` : type of variables.
- `trace.ml` : definition of infinite timed words (Def. 2.2 in the paper).
- `guard.ml` : definition of guards and intervals (Def. 3.1 and Def. 2.4 in the paper).
- `mtl.ml`: definition of MTL (Fig. 1 in the paper).
- `tlin.ml`: definition of  $T^{lin}$  (Fig. 3 in the paper) and the compiler from MTL to  $T^{lin}$  (Fig. 5 in the paper).
- `mon.ml`: definition of monitors (Fig. 8 in the paper) and, compiler from  $T^{lin}$  to monitors (Fig. 10 in the paper) and the pseudo-monitoring procedure.

### 3 Getting the artefact

The artefact is open source and available on Zenodo (<https://zenodo.org/records/15362670>). In Oracle VirtualBox, select **Import Appliance**, choose the `artefact-vm.ova` file, and start the virtual machine. In the directory `~/artefact`, running `./run <file_name_formula>` where `<file_name_formula>` is a path to a file that contains an MTL or a  $T^{lin}$  formula, will provide the user with three options (i) compile the given MTL formula into an equivalent  $T^{lin}$  formula (ii) compile the given  $T^{lin}$  formula into a monitor (iii) generate a monitor that processes the traces in `trace.ml` or a trace given via standard input, and if possible, determines whether they satisfy the formula. In addition, information on whether the formula is complete-, satisfaction-, violation-monitorable or not in the syntax of the maximally-expressive monitorable fragment ( $MT^{lin}$ ) is provided. Additional traces can be added by following the examples in the `lib/trace.ml` file. The syntax of the formulae supported by the parsers (covering all  $T^{lin}$  and MTL), as well as how to use the provided trace examples and the expected results are detailed in the `README.md` file.

### 4 Tested platforms

The artefact is known to work on Unix-based systems.

### 5 License

Creative Commons Attribution 4.0 International (CC-BY)

### 6 MD5 sum of the artefact

8b410afac22ae50e4912b9614e168d25 (for `artefact-vm.zip`)

### 7 Size of the artefact

5.5 GB (VM), 728.7 MB (Docker), 19.7 KB (code)