First-Order Linear Temporal Logic for Testing Distributed Protocols

José João Ferreira[1], Nuno Policarpo[1], José Fragoso Santos[1], Alcino Cunha[2] and Alessandro Gianola[1]

[1]INESC-ID / Instituto Superior Técnico, Universidade de Lisboa, Portugal
[2]INESC TEC / Universidade do Minho, Portugal

Abstract

Distributed systems depend on correct protocol implementations, but testing them remains complex due to concurrency and data-dependent behaviors. We explore the use of First-Order Linear Temporal Logic (FOLTL) for both offline and online monitoring, enabling precise specification of system properties over execution traces. While, in general, FOLTL is undecidable, recent advances in automated reasoning tools make reasoning in practical fragments feasible, opening new directions for validating real-world distributed systems. We present example specifications of real-world distributed protocols and outline research avenues that could facilitate the use of FOLTL fragments in testing such systems.

Keywords

First-Order Linear Temporal Logic, Distributed Protocols, Online Monitoring, Offline Monitoring

1. Introduction

Distributed systems are crucial in the modern world, as they underpin essential services such as social networks, banking infrastructures, and cloud applications. Consequently, they must remain available at all times. For these systems to operate correctly, protocols are required to ensure communication and coordination among nodes. However, these distributed protocols are error-prone and inherently challenging to test. Ideally, we would like to formally prove that they are error-free. However, such proofs typically target the theoretical protocol rather than real-world implementations, as verifying production code with existing formal tools is a labour-intensive, complex task beyond the skillset of the standard developer. Hence, these proofs often ensure algorithmic correctness, but do not guarantee that implementations are free of bugs.

Despite this, implementations of widely known consensus algorithms, such as FaB Paxos [1] and Raft [2], have been put to the test and found to contain severe flaws [3, 4]. More recently, Tendermint [5], used in the Cosmos blockchain, and Gasper [6], employed by Ethereum, are examples of proof-of-stake consensus protocols whose implementations were also discovered to exhibit critical errors [7, 8].

Specification-based testing approaches for distributed protocols have long been used to identify violations of desired properties. These approaches rely on formal specification languages to model expected behaviors, such as Alloy [9]—used to uncover counterexamples in the Chord protocol [10] by generating traces through the Alloy Analyzer model checker [11]. Similarly, TLA$^+$ [12] has been employed to prevent bugs in AWS protocols from reaching production [13], and to detect violations in protocol implementations through *offline monitoring*—that is, checking whether execution traces comply with a specification after system execution has completed [14]. Offline monitoring techniques have also been explored using specifications written in fragments of interval temporal logics [15].

Recent work has investigated *online monitoring* of execution traces—checking properties at runtime as the system executes—using specifications expressed in fragments of First-Order Linear Temporal Logic (FOLTL) [16, 17, 18], which extends classic Linear Temporal Logic (LTL) [19] with first-order theories.

7th International Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis (OVERLAY 2025), October 26, 2025, Bologna, Italy

josejoaoferreira@tecnico.ulisboa.pt (J. J. Ferreira); nuno.policarpo@tecnico.ulisboa.pt (N. Policarpo); jose.fragoso@inesc-id.pt (J. Fragoso Santos); alcino@di.uminho.pt (A. Cunha); alessandro.gianola@inesc-id.pt (A. Gianola)

For example, Decker et al. [20] monitor temporal logic over structures within some theory; Felli et al. [21] introduce ALTL$_f$, which supports arithmetic constraints over finite traces; and Faella et al. [22] propose monitoring using LTL$_f^{MT}$ [23], another fragment with first-order theories. These approaches typically rely on tightly integrated reasoning engines, such as Satisfiability Modulo Theories (SMT) solvers [24], tableaux procedures, or automata-based methods.

Performing online monitoring with FOLTL is challenging, as it requires solving satisfiability problems, and satisfiability in FOLTL is highly undecidable, since the set of valid formulas in the full FOLTL formalism is not even recursively enumerable [25]. However, several (semi-)decidable fragments have been identified for satisfiability and model checking tasks, and are already used in state-of-the-art tools. For instance, a notable line of research in verification has studied linear-time properties enriched with first-order conditions for specific datatypes, such as arithmetic constraints: in these settings, variables can store numeric values subject to (linear) arithmetic operations [26, 27, 28]. Other fragments involving arithmetics have been studied in [29], where an SMT-based decision procedure has been implemented. The aforementioned LTL$_f^{MT}$, supported by the general-purpose BLACK satisfiability checker [30] and introduced by Geatti et al. [23], led to decidable fragments studied further [31], which are tractable thanks to the efficient SMT solvers used as backend (e.g., Z3 [32] and cvc5 [33]). Another SMT solver based on decidable fragments of FOLTL is the model checker LinDMT [34], which significantly extends [29] to support general first-order theories and richer datatypes. In contrast, the offline monitoring of distributed protocols using FOLTL remains largely unexplored, despite not facing the same undecidability barriers.

Beyond its role as a formalism, FOLTL is significant for motivating the development of reasoning techniques for expressive fragments in the context of satisfiability, reactive synthesis and realizability, as demonstrated by the approaches by Rodríguez et al. [35, 36, 37].

The choice of FOLTL over LTL alone is motivated by expressiveness. While LTL suffices for reasoning about propositional temporal properties, it lacks the ability to express constraints involving structured data and quantification over variables [38], capabilities required to specify the correctness of distributed protocols. For instance, verifying a key-value store's behavior requires reasoning about relationships between keys and values, which necessitates first-order constructs and theories. When extended with past temporal operators, something not supported by TLA$^+$, despite its widespread use, FOLTL becomes more readable and allows properties to be expressed more naturally, making it a strong candidate for general-purpose specification and monitoring of complex systems.

This paper investigates how specifications written in FOLTL can be used to express correctness properties for testing a wide range of distributed protocol implementations. The proposed approach involves instrumenting the system under test to produce logs that capture operations and relevant state changes during execution. These logs form execution traces, which are then checked against FOLTL specifications defining the desired system behavior.

A central challenge lies in designing a monitor that, given a trace and a formula from a general-purpose fragment of FOLTL, can determine whether the trace satisfies the specified property. While offline monitoring is typically more tractable than online monitoring from a theoretical standpoint, practical scalability remains an open question. Execution traces and domains over which quantifiers range may be large.

To demonstrate the feasibility of this approach, we provide example specifications for real-world distributed protocols and highlight research directions that could support the use of first-order temporal logic fragments in testing complex systems.

2. First-Order Linear Temporal Logic

FOLTL [16, 17, 18] extends LTL [19] with first-order quantification, allowing precise specification of time-dependent, data-rich, dynamic behaviors of complex systems that evolve over time. The following syntax is based on that presented by Geatti et al. [39].

Let $\Sigma = \mathcal{C} \cup \mathcal{V} \cup \mathcal{F} \cup \mathcal{P}$ be a first-order signature over mutually disjoint, finite sets of constants,

variables, functions, and predicate symbols, respectively. The syntax of Σ-terms is defined as follows:

$$t := c \mid x \mid f(t_1, \ldots, t_n)$$

where $c \in \mathcal{C}$, $x \in \mathcal{V}$, $f \in \mathcal{F}$, and $t_1, \ldots, t_n$ are Σ-terms. The syntax of Σ-formulas is defined as follows:

$$\phi := p(t_1, \ldots, t_n) \mid t_1 = t_2 \mid \neg\phi \mid \phi \vee \phi \mid \exists x \,.\, \phi \mid \mathsf{X}\,\phi \mid \mathsf{Y}\,\phi \mid \phi\ \mathsf{U}\ \phi \mid \phi\ \mathsf{S}\ \phi$$

where $p \in \mathcal{P}$, $t_1, \ldots, t_n$ are Σ-terms, $x \in \mathcal{V}$, and the temporal operators X, Y, U, and S are called *next*, *yesterday*, *until* and *since*. Derived constructs, such as $\wedge, \Rightarrow, \top, \forall, \mathsf{F}, \mathsf{G}, \mathsf{O}, \mathsf{H}$, are defined as usual.

Let $\sigma : (\mathcal{C}, \mathcal{V}, \mathcal{F} \to D)$ be an *environment* that maps Σ-terms t to values in their domains. The *evaluation* of t under σ, denoted t^σ, is defined as: $c^\sigma = c, x^\sigma = \sigma(x)$, and $f(t_1, \ldots, t_n)^\sigma = f({t_1}^\sigma, \ldots, {t_n}^\sigma)$.

Let $\varepsilon = p(v_1, ..., v_n)$ be an event, where $p \in \mathcal{P}$ is an n-ary predicate symbol and $v_1, \ldots, v_n$ are domain values. An event states that a certain predicate with arguments is true.

Let τ be a finite sequence of finite sets of events, henceforth referred to as a *trace*, where each set corresponds to a discrete time point $i \in \mathbb{N}$. A trace emulates an execution of a system, representing the sequence of events that occur over time in a specific run: $\tau = [\{\varepsilon_1, \ldots, \varepsilon_p\}, \ldots, \{\varepsilon_q, \ldots, \varepsilon_r\}]$.

The *satisfaction* of a FOLTL formula ϕ with respect to an environment σ and a trace τ at time point $i \in \mathbb{N}$ is denoted by $\sigma, \tau, i \models \phi$. The inductive semantics for boolean connectives, quantifiers, and temporal operators follow standard definitions (e.g., the one presented by Geatti et al. [39]), except for the following adaptation to our custom trace: $\sigma, \tau, i \models p(t_1, \ldots, t_n)$ iff $\tau[i] \ni p({t_1}^\sigma, \ldots, {t_n}^\sigma)$.

3. FOLTL for Testing

The general-purpose FOLTL fragment provides a framework for validating execution traces. In order for us to get traces, the system under test must be instrumented. In distributed systems, this includes correlating events across nodes. These traces are then checked against specifications by a monitor.

Formally, given an execution trace τ, a property specification ϕ is *verifiable* if and only if there exists an environment σ such that $\sigma, \tau, 0 \models \phi$, i.e., if ϕ holds at the initial time point of the trace.

Using two distributed protocol examples, we illustrate how to express both *safety*—"nothing bad ever happens"—and *liveness*—"something good eventually happens"—properties using FOLTL that can be later tested against concrete implementations. The modeled specifications are mostly event-based and applicable to both online and offline trace monitoring.

3.1. Two Phase Commit (2PC)

2PCs [40, 41] are fundamental to distributed systems, enabling atomic commitment across multiple nodes, i.e., all participants in a transaction either commit or abort together. Unlike consensus algorithms, which rely on majority voting, 2PC requires unanimous agreement. Though conceptually simple, the protocol must handle failures and asynchronous communication.

Our model includes a single coordinator and multiple uniquely identified participants. Upon receiving a prepare request, a participant writes the transaction data to disk, checks database consistency and constraints, and, if valid, votes to commit, locking all the necessary resources. Example properties, modeled as FOLTL formulas, use predicates parameterized by nodes n, transactions t, and data d.

- **prepare**(n, t, d): the coordinator requests node n to prepare for the transaction t with data d.
- **v-commit**(n, t): node n votes to commit transaction t and sends the response to the coordinator.
- **v-abort**(n, t): node n votes not to commit transaction t and sends the response to the coordinator.
- **i-commit**(t): the coordinator instructs all participant nodes to commit transaction t.
- **i-abort**(t): the coordinator instructs all participant nodes not to commit transaction t.
- **commit**(n, t): node n commits transaction t.

(LIVENESS) **Eventual decision**: if the coordinator asks some participant node n to prepare for transaction t, that transaction will eventually be either committed or aborted.

$$\forall\, t \,.\, \mathsf{G}\Big(\big(\exists\, n \,.\, \textbf{prepare}(n, t, -)\big) \Rightarrow \mathsf{F}\big(\textbf{i-commit}(t) \vee \textbf{i-abort}(t)\big)\Big)$$

(SAFETY) **Atomic commitment**: if the coordinator instructs to commit transaction t, then all participants n must have voted to commit it; if it instructs to abort t, then at least one node has voted to abort it.

$$\forall\, t \,.\, \mathsf{G}\Big(\big(\textbf{i-commit}(t) \Rightarrow \neg\exists\, n \,.\, \mathsf{O}\, \textbf{v-abort}(n, t)\big) \wedge \big(\textbf{i-abort}(t) \Rightarrow \exists\, n \,.\, \mathsf{O}\, \textbf{v-abort}(n, t)\big)\Big)$$

(SAFETY) **Commit justification**: if a node commits a transaction, then the coordinator must have instructed the commit, which must have been preceded by a vote to commit from that node, and in turn, a prepare.

$$\forall\, t, n. \,\mathsf{G}\Big(\textbf{commit}(n, t) \Rightarrow \mathsf{O}\big(\textbf{i-commit}(t) \wedge \mathsf{O}(\textbf{v-commit}(n, t) \wedge \mathsf{O}\, \textbf{prepare}(n, t, -))\big)\Big)$$

3.2. Distributed Hash Table (DHT)

DHTs [10, 41] are central to peer-to-peer systems, offering scalable, decentralized key–value storage. However, their dynamic topology and inherent concurrency make correctness testing particularly challenging.

In our model, the network consists of uniquely identified nodes that store key–value pairs. While network states span multiple time points, operations take inputs and yield outputs only upon completion. Thus, for this DHT scenario, we model them as instantaneous, with explicit FOLTL predicates marking their beginning and end. Effects of operations are defined to occur between these events. Predicates are parameterized by nodes n, keys k, values v, and unique identifiers z that link related predicates. For readability, these identifiers appear as subscripts but are semantically just additional arguments. Examples of operations, network states, and properties include:

- **b-store**$_z(n, k, v)$: the client requests node n to store the key-value pair (k, v) in the DHT.
- **e-store**$_z(n')$: the operation completes at node n', which notifies the client upon completion.
- **b-lookup**$_z(n, k)$: the client requests the value for key k from node n in the DHT.
- **e-lookup**$_z(n', v)$: the operation completes at the node n' storing (k, v) and returns v to the client.
- **stable**: the network remains unchanged, with no node joins, departures, or failures.

(SAFETY) **Lookup consistency**: if a lookup z obtains a value v for a given key k, then the (k, v) pair was previously written by a store operation and there are no fabricated values.

$$\forall\, k, v \,.\, \mathsf{G}\Big(\forall\, z \,.\, \big(\textbf{b-lookup}_z(-, k) \wedge \mathsf{F}\, \textbf{e-lookup}_z(-, v)\big) \Rightarrow \mathsf{O}\, \textbf{b-store}(-, k, v)\Big)$$

(SAFETY) **Value freshness**: if a lookup z obtains a value v for a given key k, then v is the latest stored value for key k, that is, any past store of a different value v' for k must have been followed by a store of v.

$$\forall\, k, v \,.\, \mathsf{G}\Big(\forall\, z \,.\, \big(\,\textbf{b-lookup}_z(-, k) \wedge \mathsf{F}\, \textbf{e-lookup}_z(-, v)\big) \Rightarrow \big(\,\forall\, v' \neq v \,.\, \mathsf{O}\big(\textbf{b-store}(-, k, v') \Rightarrow \mathsf{F}\, \textbf{b-store}(-, k, v)\big)\big)\Big)$$

(LIVENESS) **Operation termination**: if the network is stable, then every store or lookup operation eventually completes, that is, for any operation z, once it starts, it will eventually end.

$$\forall\, z \,.\, \mathsf{G}\Big(\textbf{stable} \Rightarrow \big(\big(\textbf{b-store}_z(\ldots) \Rightarrow \mathsf{F}\, \textbf{e-store}_z(-)\big) \wedge \big(\textbf{b-lookup}_z(\ldots) \Rightarrow \mathsf{F}\, \textbf{e-lookup}_z(\ldots)\big)\big)\Big)$$

3.3. Online and Offline Monitoring

Online monitoring is the process of checking a system's execution at runtime to determine whether its ongoing behavior satisfies a specification formula. At each step, only a finite prefix of the trace is available, requiring the monitor to evaluate compliance with respect to all possible future extensions. This setting demands incremental reasoning under uncertainty and involves decision procedures such as (FOLTL) SAT solving, making the problem challenging and at least as hard as satisfiability checking.

For a given trace τ and a FOLTL formula ϕ, the online monitoring problem is the task of determining the monitoring state $s \in \{CS, PS, CV, PV\}$ that satisfies one of the following conditions [22]:

- $s = CS$ iff $\tau \models \phi$ and $\tau\tau' \not\models \phi$ for some trace τ' — *current satisfaction*
- $s = PS$ iff $\tau \models \phi$ and $\tau\tau' \models \phi$ for every trace τ' — *permanent satisfaction*
- $s = CV$ iff $\tau \not\models \phi$ and $\tau\tau' \models \phi$ for some trace τ' — *current violation*
- $s = PV$ iff $\tau \not\models \phi$ and $\tau\tau' \not\models \phi$ for every trace τ' — *permanent violation*

While satisfiability for general-purpose FOLTL is undecidable, several fragments are known to be tractable or (semi-)decidable, with dedicated reasoning techniques [20, 21, 22]. Tools like BLACK [23], based on SAT/SMT solving, and analyzers like Alloy [9], have been developed to target fragments and are good candidates to be adapted for online monitoring by encoding specifications into their languages.

Offline monitoring, on the other hand, refers to the post-execution analysis of a system, where a complete trace τ is checked against a specification formula ϕ. Since the entire finite trace is available beforehand, the monitor can evaluate ϕ with full access to past and future events.

This setting often avoids SAT solving, remaining decidable, as variable domains are bounded by the trace. Verification becomes concrete: variables are instantiated with observed values, reducing the task to checking ϕ over known data. Efficient monitoring can possibly leverage pattern matching, automata, or symbolic rewriting, aided by preprocessing and static analysis. Despite offline monitoring not requiring SAT tools, FOLTL fragments remain valuable for their expressiveness, as they support quantification, temporal reasoning, and predicates for specifying rich properties.

4. Discussion and Future Work

Because this work focuses on testing distributed protocols, it is important to acknowledge that obtaining suitable execution traces is a non-trivial challenge. Instrumenting distributed systems requires correlating and ordering events across independent nodes, often in the presence of delays, clock skew, or failures. Although this is a separate concern and beyond the scope of this work, reliably constructing such traces is a necessary prerequisite and it directly impacts the feasibility of monitoring.

The protocols in Section 3 are simplified case studies meant to illustrate the expressive power of FOLTL in capturing correctness properties. They use a limited set of operations and deliberately abstract away system-level complexities. For example, our 2PC model omits fault tolerance, but aspects like crashes or retries could be incorporated by adding operations and node states, as the FOLTL formalism is expressive enough to accommodate such extensions.

For the DHT protocol, operations were modeled using paired begin and end events rather than single predicates spanning multiple time points. While this increases the verbosity of specifications, it better reflects the client-server nature of the operations, where effects occur between invocation and response, and makes it more intuitive to distinguish input from output arguments. It also provides a useful contrast with the 2PC example, highlighting FOLTL flexibility in supporting different modeling styles.

Although FOLTL, as a general approach, allows specifying any property, fragments with syntactic variations may be particularly tailored for the DHT case or other protocols where operations are continuous and there is a clear distinction between inputs and outputs, so that complex properties can have simpler specifications.

While the theoretical study of online monitoring with fragments of FOLTL is not novel, despite inherent undecidability, semi-decidable fragments and tools such as BLACK remain useful for this task.

Key challenges include adapting existing solvers for online monitoring and identifying fragments that balance expressiveness with decidability.

Although online monitoring is widely used, offline monitoring with FOLTL has received little attention. In scenarios where complete executions are available, treating the problem offline can enable more efficient violation detection. This is a promising and relatively unexplored direction. Future challenges involve identifying useful fragments of FOLTL for both online and offline monitoring and developing scalable decision procedures and optimizations for evaluating quantifier-heavy specifications over large complete traces. We leave for future work to determine whether the presented example specifications of distributed protocols fall into some already known decidable fragment of FOLTL.

We believe that many problems over distributed protocols can be effectively expressed using FOLTL, and expect specification-based testing with first-order temporal logics to gain broader attention and adoption. We are also interested in investigating the use of FOLTL for expressing properties over protocols used in public administration processes, which we believe would largely benefit from this work. This work positions FOLTL as a viable and flexible formalism for validating distributed protocols in both online and offline settings, with expressive fragments capable of capturing meaningful correctness properties. It opens the way for practical monitoring systems that bridge the gap between formal logic and implementation-level testing.

Acknowledgments

This work was partially supported by the 'OptiGov' project, with ref. n. 2024.07385.IACDC (DOI: 10.54499/2024.07385.IACDC), fully funded by the 'Plano de Recuperação e Resiliência' (PRR) under the investment 'RE-C05-i08 - Ciência Mais Digital' (measure 'RE-C05-i08.m04'), framed within the financing agreement signed between the 'Estrutura de Missão Recuperar Portugal' (EMRP) and Fundação para a Ciência e a Tecnologia, I.P. (FCT) as an intermediary beneficiary. This work was also partly supported by Portuguese national funds through Fundação para a Ciência e a Tecnologia, I.P. (FCT), under projects UID/50021/2025 and UID/PRR/50021/2025.

Declaration on Generative AI

During the preparation of this work, the author(s) used Grammarly and ChatGPT for grammar and spelling checks, as well as to identify other writing mistakes. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content

References

[1] J.-P. Martin, L. Alvisi, Fast byzantine consensus, IEEE Transactions on Dependable and Secure Computing 3 (2006) 202–215. doi:10.1109/TDSC.2006.35.

[2] D. Ongaro, J. K. Ousterhout, In search of an understandable consensus algorithm, in: Proceedings of the 2014 USENIX Annual Technical Conference, 2014, pp. 305–319. doi:10.5555/2643634.2643666.

[3] I. Abraham, G. Gueta, D. Malkhi, L. Alvisi, R. Kotla, J.-P. Martin, Revisiting fast practical byzantine fault tolerance, CoRR abs/1712.01367 (2017). arXiv:1712.01367.

[4] C. Jensen, H. Howard, R. Mortier, Examining Raft's behaviour during partial network failures, in: Proceedings of the 1st Workshop on High Availability and Observability of Cloud Systems, 2021, pp. 11–17. doi:10.1145/3447851.3458739.

[5] E. Buchman, Tendermint: Byzantine fault tolerance in the age of blockchains, Ph.D. thesis, University of Guelph, 2016. URL: http://hdl.handle.net/10214/9769.

[6] V. Buterin, D. Hernandez, T. Kamphefner, K. Pham, Z. Qiao, D. Ryan, J. Sin, Y. Wang, Y. X. Zhang, Combining GHOST and Casper, CoRR abs/2003.03052 (2020). arXiv:2003.03052.

[7] C. Cachin, M. Vukolic, Blockchain consensus protocols in the wild, CoRR abs/1707.01873 (2017). arXiv:1707.01873.

[8] J. Neu, E. N. Tas, D. Tse, Ebb-and-flow protocols: A resolution of the availability-finality dilemma, in: Proceedings of the 42nd IEEE Symposium on Security and Privacy, 2021, pp. 446–465. doi:10.1109/SP40001.2021.00045.

[9] D. Jackson, Alloy: A language and tool for exploring software designs, Communications of the ACM 62 (2019) 66–76. doi:10.1145/3338843.

[10] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan, Chord: A scalable peer-to-peer lookup service for internet applications, SIGCOMM Computer Communication Review 31 (2001) 149–160. doi:10.1145/964723.383071.

[11] P. Zave, Using lightweight modeling to understand chord, Computer Communication Review 42 (2012) 49–57. doi:10.1145/2185376.2185383.

[12] L. Lamport, Specifying Systems: the TLA$^+$ language and tools for hardware and software engineers, volume 388, Addison-Wesley, 2002. URL: http://research.microsoft.com/users/lamport/tla/book.html.

[13] C. Newcombe, T. Rath, F. Zhang, B. Munteanu, M. Brooker, M. Deardeuff, How Amazon Web Services uses formal methods, Communications of the ACM 58 (2015) 66–73. doi:10.1145/2699417.

[14] H. Cirstea, M. A. Kuppe, B. Loillier, S. Merz, Validating traces of distributed programs against TLA$^+$ specifications, in: Proceedings of the 22nd International Conference on Software Engineering and Formal Methods, 2024, pp. 126–143. doi:10.1007/978-3-031-77382-2_8.

[15] N. Policarpo, J. F. Santos, A. Cunha, J. Leitão, P. Á. Costa, Specifying distributed hash tables with Allen Temporal Logic, in: Proceedings of the 13th IEEE/ACM International Conference on Formal Methods in Software Engineering, 2025, pp. 63–73. doi:10.1109/FORMALISE66629.2025.00013.

[16] I. M. Hodkinson, F. Wolter, M. Zakharyaschev, Decidable fragments of first-order temporal logics, Annals of Pure and Applied Logic 106 (2000) 85–134. doi:10.1016/S0168-0072(00)00018-X.

[17] T. Braüner, S. Ghilardi, First-order modal logic, in: Handbook of Modal Logic, Studies in logic and practical reasoning, North-Holland, 2007, pp. 549–620. doi:10.1016/S1570-2464(07)80012-7.

[18] A. Artale, A. Mazzullo, A. Ozaki, First-order temporal logic on finite traces: Semantic properties, decidable fragments, and applications, ACM Transactions on Computational Logic 25 (2024) 13:1–13:43. doi:10.1145/3651161.

[19] A. Pnueli, The temporal logic of programs, in: Proceedings of the 18th Annual Symposium on Foundations of Computer Science, 1977, pp. 46–57. doi:10.1109/SFCS.1977.32.

[20] N. Decker, M. Leucker, D. Thoma, Monitoring modulo theories, International Journal on Software Tools for Technology Transfer 18 (2016) 205–225. doi:10.1007/S10009-015-0380-3.

[21] P. Felli, M. Montali, F. Patrizi, S. Winkler, Monitoring arithmetic temporal properties on finite traces, in: Proceedings of the 37th AAAI Conference on Artificial Intelligence, 2023, pp. 6346–6354. doi:10.1609/AAAI.V37I5.25781.

[22] M. Faella, G. Parlato, A unified automata-theoretic approach to LTL$_f$ modulo theories, in: Proceedings of the 27th European Conference on Artificial Intelligence, 2024, pp. 1254–1261. doi:10.3233/FAIA240622.

[23] L. Geatti, A. Gianola, N. Gigante, Linear temporal logic modulo theories over finite traces, in: Proceedings of the 31st International Joint Conference on Artificial Intelligence, 2022, pp. 2641–2647. doi:10.24963/IJCAI.2022/366.

[24] C. W. Barrett, R. Sebastiani, S. A. Seshia, C. Tinelli, Satisfiability modulo theories, in: Handbook of Satisfiability - Second Edition, volume 336 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2021, pp. 1267–1329. doi:10.3233/FAIA201017.

[25] D. Gabbay, A. Kurucz, F. Wolter, M. Zakharyaschev, Fragments of first-order temporal logics, in: Many-Dimensional Modal Logics: Theory and Applications, volume 148 of *Studies in Logic and the Foundations of Mathematics*, Elsevier, 2003, pp. 465–545. doi:https://doi.org/10.1016/S0049-237X(03)80012-5.

[26] S. Demri, LTL over integer periodicity constraints, Theor. Comput. Sci. 360 (2006) 96–123. URL: https://doi.org/10.1016/j.tcs.2006.02.019. doi:10.1016/J.TCS.2006.02.019.

[27] S. Demri, Linear-time temporal logics with presburger constraints: an overview, J. Appl. Non Class. Logics 16 (2006) 311–348. URL: https://doi.org/10.3166/jancl.16.311-347. doi:10.3166/JANCL.16.311-347.

[28] S. Demri, D. D'Souza, An automata-theoretic approach to constraint LTL, Inf. Comput. 205 (2007) 380–415. URL: https://doi.org/10.1016/j.ic.2006.09.006. doi:10.1016/J.IC.2006.09.006.

[29] P. Felli, M. Montali, S. Winkler, Linear-time verification of data-aware dynamic systems with arithmetic, in: Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022, AAAI Press, 2022, pp. 5642–5650. URL: https://doi.org/10.1609/aaai.v36i5.20505. doi:10.1609/AAAI.V36I5.20505.

[30] L. Geatti, N. Gigante, A. Montanari, BLACK: A fast, flexible and reliable LTL satisfiability checker, in: Proceedings of the 3rd Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis, 2021, pp. 7–12. URL: https://ceur-ws.org/Vol-2987/paper2.pdf.

[31] L. Geatti, A. Gianola, N. Gigante, S. Winkler, Decidable fragments of LTL_f modulo theories, in: Proceedings of the 26th European Conference on Artificial Intelligence, 2023, pp. 811–818. doi:10.3233/FAIA230348.

[32] L. M. de Moura, N. S. Bjørner, Z3: an efficient SMT solver, in: Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, 2008, pp. 337–340. doi:10.1007/978-3-540-78800-3_24.

[33] H. Barbosa, C. W. Barrett, M. Brain, G. Kremer, H. Lachnitt, M. Mann, A. Mohamed, M. Mohamed, A. Niemetz, A. Nötzli, A. Ozdemir, M. Preiner, A. Reynolds, Y. Sheng, C. Tinelli, Y. Zohar, cvc5: A versatile and industrial-strength SMT solver, in: Proceedings of the 28th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, 2022, pp. 415–442. doi:10.1007/978-3-030-99524-9_24.

[34] A. Gianola, M. Montali, S. Winkler, Linear-time verification of data-aware processes modulo theories via covers and automata, in: Proceedings of 38th AAAI Conference on Artificial Intelligence, 2024, pp. 10525–10534. doi:10.1609/AAAI.V38I9.28922.

[35] A. Rodríguez, C. Sánchez, Boolean abstractions for realizability modulo theories, in: Proceedings of the 35th International Conference on Computer Aided Verification, 2023, pp. 305–328. doi:10.1007/978-3-031-37709-9_15.

[36] A. Rodríguez, C. Sánchez, Realizability modulo theories, Journal of Logical and Algebraic Methods in Programming 140 (2024) 100971. doi:10.1016/J.JLAMP.2024.100971.

[37] A. Rodríguez, C. Sánchez, Adaptive reactive synthesis for LTL and LTL_f modulo theories, in: Proceedings of the 38th AAAI Conference on Artificial Intelligence, 2024, pp. 10679–10686. doi:10.1609/AAAI.V38I9.28939.

[38] L. Geatti, A. Gianola, N. Gigante, Torwards infinite-state verification and planning with linear temporal logic modulo theories (extended abstract), in: Proceedings of the 30th International Symposium on Temporal Representation and Reasoning, 2023, pp. 21:1–21:3. doi:10.4230/LIPICS.TIME.2023.21.

[39] L. Geatti, A. Gianola, N. Gigante, First-order automata, in: Proceedings of the 39th AAAI Conference on Artificial Intelligence, 2025, pp. 14940–14948. doi:10.1609/AAAI.V39I14.33638.

[40] P. A. Bernstein, V. Hadzilacos, N. Goodman, Concurrency Control and Recovery in Database Systems, Addison-Wesley, 1987. URL: http://research.microsoft.com/en-us/people/philbe/ccontrol.aspx.

[41] G. Coulouris, J. Dollimore, T. Kindberg, G. Blair, Distributed Systems: Concepts and Design, 5th ed., Addison-Wesley, 2011.