Logical Methods in Computer Science
Volume 22, Issue 1, 2026, pp. 11:1–11:32
https://lmcs.episciences.org/

Submitted Jan. 05, 2024
Published Feb. 16, 2026

ASYNCHRONOUS COMPOSITION OF LTL PROPERTIES OVER INFINITE AND FINITE TRACES

ALBERTO BOMBARDELLI [a,b] AND STEFANO TONETTA [a]

[a] Fondazione Bruno Kessler, via Sommarive 18, Trento Italy 380123
e-mail address: abombardelli@fbk.eu, tonettas@fbk.eu

[b] University of Trento, via Sommarive 9, Trento Italy 380123
e-mail address: alberto.bombardell-1@unitn.it

Abstract. The verification of asynchronous software components poses significant challenges due to the way components interleave and exchange input/output data concurrently. Compositional strategies aim to address this by separating the task of verifying individual components on local properties from the task of combining them to achieve global properties. This paper concentrates on employing symbolic model checking techniques to verify properties specified in Linear-time Temporal Logic (LTL) on asynchronous software components that interact through data ports. Unlike event-based composition, local properties can now impose constraints on input from other components, increasing the complexity of their composition. We consider both the standard semantics over infinite traces as well as the truncated semantics over finite traces to allow scheduling components only finitely many times.

We propose a novel LTL rewriting approach, which converts a local property into a global one while considering the interleaving of infinite or finite execution traces of components. We prove the semantic equivalence of local properties and their rewritten version projected on the local symbols. The rewriting is also optimized to reduce formula size and to leave it unchanged when the temporal property is stutter invariant. These methods have been integrated into the OCRA tool, as part of the contract refinement verification suite. Finally, the different composition approaches were compared through an experimental evaluation that covers various types of specifications.

1. Introduction

Model checking asynchronous software poses significant challenges due to the non-deterministic interleaving of components and concurrent access to shared variables. Compositional techniques are often used to tackle scalability issues. The idea of this approach is to decouple the problem of verifying local properties specified over the component interfaces from the problem of composing them to ensure some global property.

For example, in [RBH$^+$01] is described the following compositional reasoning. Given some local component $(\mathcal{M}_1, \ldots, \mathcal{M}_n)$, some local properties of these components $(\varphi_1, \ldots, \varphi_n)$, a global property (φ), a notion of composition for the components (γ_S) and a notion of composition for the properties (γ_P); if the local components satisfy the local properties and the composition of local properties entails the global properties, the local component composition satisfies the global property. While in the synchronous setting, the composition is simply giving by a conjunction, the asynchronous composition of local temporal properties may be tricky when considering software components communicating through data ports.

In this paper, we define the asynchronous composition of LTL [Pnu77] properties local to components; which means that the local properties reason only over the part of the execution of the local component, e.g., when a local property refers to the *next* state, the composition will consider the next state along the local run of the component. The communication between local components is achieved with I/O data ports while their execution is controlled by scheduling constraints. Due to uncontrollable input changes, the composition must take into account whether or not the local component is running even if the formula does not contain "next"[1]. Another important factor to consider is the possibly finite execution of local components. If for instance, the scheduler is not fair, the unlucky local component might be scheduled only for a finite amount of time. This *"possibly finite"* local semantics also allows for a natural way to represent permanent faults in asynchronous systems (e.g. a local component is not scheduled anymore means that it crashed).

To represent the *"possibly finite"* local semantics, we use the truncated weak semantics defined in [EFH+03a] to represent local properties. The idea is that the local formula can be interpreted to both finite and infinite trace, and the formula semantics does not force the system to execute the local component. For completeness, we define the composition in a way that soundly supports the finite execution of the composed system as well; therefore, the compositional reasoning applies to hierarchical systems too.

Our composition approach is based on a syntactical rewriting $\mathcal{R}_c^*$ of the local temporal property to a property of the composite model. The rewriting is then conjoined with a constraint ensuring that output variables do not change when the component is not running (ψ_{cond}). The property composition γ_P is in the following form: $\psi_{cond} \wedge \bigwedge_{1 \leq i \leq n} \mathcal{R}_i^*(\varphi_i)$. We also provide an optimized version of the rewriting that works when all components run infinitely often, thus without possible truncation of the local traces.

The proposed approach has been implemented inside OCRA[CDT13], which allows a rich extension of LTL and uses a state-of-the-art model checking algorithm implemented in nuXmv [CCD+14] as back-ends to check satisfiability.

We evaluated our approach on models representing compositions of pattern formulas and real models from the automotive domains [CCG+23]. We show both the qualitative differences in results between the two semantics and the impact of the optimization when dealing with local infinite executions.

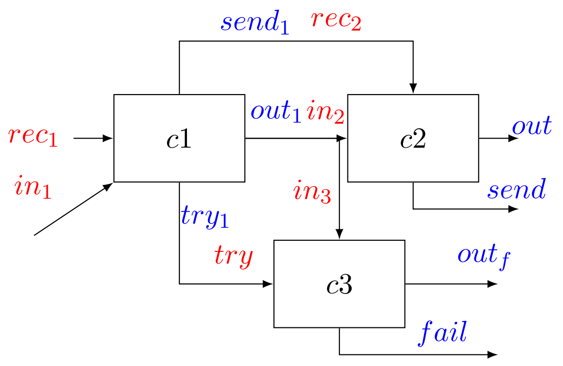
1.1. **Motivating examples.**

We propose two examples to motivate our work. The first model is a toy example representing a system that tries to send a value to a network while the second is a real model coming from the automotive domain. Both models can be naturally represented through possibly finite scheduling of local components.

1.1.1. *Sender.*

We now model a three-component system that represents a system that receives a message and tries to send it through a network. The component either successfully delivers the message or fails to send it and logs the message. The example is represented in Figure 1. This example is composed of three components: Component $c1$ receives a message rec_1 and an input in_1 and tries to send the value to component $c2$. If $c1$ is eventually able

[1]Usually when an LTL formula does not contain next, it is *stutter invariant*. In such cases, the asynchrony would not interfere with the formula.



$$
\begin{aligned}
\varphi_{c1} :=& G(rec_1 \rightarrow out_1' = in_1 \wedge X((try_1 \wedge out_1' = out_1) U send_1)) \\
\varphi_{c2} :=& G(rec_2 \rightarrow out2' = in2 \wedge X send_2) \\
\varphi_{c3} :=& G(try \rightarrow out_f' = in_3 \wedge X fail) \\
\varphi :=& G((rec_1 \wedge in_1 = v) \rightarrow F(send \wedge out = v \vee fail \wedge out_f = v)) \\
\alpha :=& G(in_1 \rightarrow run_1) \wedge G(send_1 \rightarrow run_2) \wedge G(H^{\leq p} try_1 \rightarrow run_3)
\end{aligned}
$$

Figure 1. Figure representing the sender model. The colour red represents input variables while the colour blue represents the output variables.

to send $send_2$ message through the network, then $c2$ will run and will output the original input. The network guarantees that eventually, $c1$ will be able to send the message to $c2$; however, if $c1$ runs only finitely many times, $c3$ at some point will report a failure. The global property states that if an input message is received, it is either eventually delivered as output or an error occurs. If $c1$ runs infinitely often, the global property is satisfied without the need for $c3$.

1.1.2. *Automotive compositional contract.* Another interesting example comes from the automotive domain. In [CCG+23], the EVA framework was proposed for the compositional verification of AUTOSAR components. That work used the rewriting technique we proposed in [BT22] to verify the correct refinement of contracts defined as a pair of LTL properties.

Due to the complexity of the model, we omit a full description of the system, the specifications and the properties. We focus on a specific requirement of the system that states that the system shall brake when the Autonomous Emergency Braking module gets activated. A simplified version of the specification is defined by the following LTL formula:

$$G(X(aeb\_breaking.status \neq 0) \rightarrow F^{\leq 2} Brake\_\_In\_\_BrakeActuator \neq 0)$$

The specification is entailed by a brake actuator component that is composed of the actual actuator (BrakeActuator#BA_Actuator) and a watchdog (BrakeActuator#BA_Watchdog). The actuator is scheduled every time a signal is received in input while the watchdog is scheduled periodically. By reasoning over finite executions of components, it is possible to verify whether or not the global specification is valid even if at some point the Actuator stop working. We omit the detailed structure and specification of the sub-components (actuator and watchdog), for a complete view please refer to [CCG$^+$23] or to the experimental evaluation.

1.2. **Overall contribution.**

The main contribution of this paper is the definition of a rewriting-based technique to verify compositional asynchronous systems in a general way. Furthermore, we provide an additional optimized rewriting technique to cover the case in which local components are assumed to run infinitely often. The main advantages of our compositional approach are the following:

- It supports asynchronous communication between data ports.
- It supports generic scheduling constraints expressible through LTL formulas.
- It supports both finite and infinite executions of local components making it a suitable approach for safety assessment as well.

This work is an extension of the conference paper [BT22]. The work has been extended with the following contributions:

- A weak semantics for the logic with two decision procedures for its verification.
- A new definition of asynchronous composition of Interface Symbolic Transition Systems that deals with possibly finite execution of components.
- A new rewriting that generalizes the previous one for possibly finite systems; the rewriting introduced in [BT22] is then presented as an optimized version of our new rewriting when each component is scheduled infinitely often.
- A new experimental evaluation with new models and a set of benchmarks from the automotive domain.

1.3. **Outline.**

The rest of the paper is organized as follows: in Sec. 2, we compare the proposed solution with related works; in Sec. 3, we define our logic syntax, semantics and verification; in Sec. 4, we formalize the problem; in Sec. 5, we define the rewriting approach, its basic version, its complete version and an optimized variation that is suited for infinite executions only; in Sec. 6, we report on the experimental evaluation; finally, in Sec. 7, we draw the conclusions and some directions for future works.

2. Related works

One of the most important works on temporal logic for asynchronous systems is Temporal Logic of Action (TLA) [Lam94] by Leslie Lamport, later extended with additional operators[Lam97]. TLA has been also used in a component-based manner in [RR09]. Our formalism has similarities with TLA. We use a (quantifier-free) first-order version of LTL [MP92b] with "next" function to specify the succession of actions of a program. TLA natively supports the notion of stuttering for composing asynchronous programs so that the composition is simply obtained by conjoining the specifications. We focus instead on local properties that are specified independently of how the program is composed so that "next" and input/output data refer only to the local execution. Another substantial difference with TLA is that our formalism also supports a finite semantics of LTL, permitting reasoning over finite traces.

As for propositional LTL, the composition of specifications is studied in various papers on assume-guarantee reasoning (see, e.g., [McM99, PDH99, JT96, CT15a]) for both synchronous and asynchronous composition. In the case of asynchronous systems, most works focus on fragments of LTL without the next operator, where formulas are always stutter invariant. Other studies investigated how to tackle down state-space explosion for that scenario usually employing techniques such as partial order reduction [BBC$^+$09]. However, our work covers a more general setting, where the presence of input variables makes formulas non-stutter-invariant.

Similar to our work, [BBC$^+$09] considers a rewriting for LTL with events to map local properties into global ones with stuttering. However, contrary to this paper, it does not consider input variables (nor first-order extension) and assume that every variable does not change during stuttering, resulting in a simpler rewriting. In [EFH$^+$03b], a temporal clock operator is introduced to express properties related to multiple clocks and, in principle, can be used to interpret formulas over the time points in which a component is not stuttering. Its rewriting is indeed similar to the basic version defined in this paper, but is limited to propositional LTL and has not been conceived for asynchronous composition. The optimization that we introduce to exploit the stutter invariance of subformulas results in simpler formulas easy to be analyzed as shown in our experimental evaluation.

The rewriting of asynchronous LTL is similar to the transformation of asynchronous symbolic transition systems into synchronous ones described in [CMT11]. That work considers connections based on events where data are exchanged only upon synchronization (allowing optimizations as in shallow synchronization [BCL$^+$10]). Thus, it does not consider components that read from input variables that may be changed by other components. Moreover, [CMT11] is not able to transform temporal logic local properties in global ones as in this paper.

In [LBA$^+$22], the authors defined a technique to verify asynchronous assume/guarantee systems in which assumptions and guarantees are defined on a safety fragment of LTL with predicates and functions. The main differences between that work and this one are the following. The logic considered by [LBA$^+$22] is limited to a safety fragment of LTL with "globally" and past operators; on the other hand, our work covers full LTL with past operators with event freezing functions and finite semantics. Another key difference is that they consider scheduling in which a component is first dispatched and then, after some steps it completes its action; instead, our framework defines scheduling constraints with LTL formulas and actions are instantaneous.

Another related work is the one proposed in [BCMT14]. Here, the authors propose a compositional reasoning based on assume-guarantee contracts for model based safety assessment (MBSA). They extend the model with possible faults that are modelled as input Boolean parameter to the system. However, contrary to our work, asynchronous composition is not considered and a fault represents the non-satisfaction of a local property.

In [BCB$^+$21], a related rewriting is proposed in the context of Asynchronous Hyper-properties. That work considers an hyper-logic based on LTL that uses a rewriting to align different traces of the same systems on "interesting" points. Although for certain aspects the work is similar, it considers a very different problem.

Finally, our logic semantics is based on the works of Eisner, Fisman et al. on truncated LTL of [EFH$^+$03a]. The main differences are the following. Our logic includes past operators, first-order predicates and functions; we consider only weak/strong semantics for our logic while their work instead also deals with neutral semantics.

In summary, while existing works address various aspects related to our approach, none provide a unified framework for the composition of asynchronous systems with I/O data ports that explicitly accounts for the termination of local components. To the best of our knowledge, this work is the first to bridge this gap, offering a comprehensive solution that integrates these considerations into a general framework.

3. First Order Past LTL with weak truncated semantics

This section presents the syntax and the semantics of the logic used in this paper and its verification.

The logic is an extension of LTL with event-freezing functions of [Ton17] that is interpreted over both finite and infinite traces. As we mentioned in the introduction, we are considering weak semantics for our logic that is based on the works of Eisner and Fisman [EFH$^+$03a, FK18]. The overall idea behind this logic is to have an expressive language that can reason on both infinite and truncated executions of programs. Finally, we can observe that for each infinite trace, if the trace satisfies a property, then all its finite prefixes satisfy it as well under the weak semantics.

Prior to the logic, we denote the notion of traces, which represent finite or infinite executions of input/output components. Therefore, traces distinguish between input and output symbols. A local component reads the input to decide the next state and output. Thus, the finite traces in our setting do not contain the evaluation of input variables at the end of the trace.

Definition 3.1. We define a trace π as a sequence $s_0, s_1, \dots$ of assignments over a set of input variables V^I and output variables V^O i.e. in which each s_i is defined over $V^I \cup V^O$. A trace π is denoted as finite if the sequence of assignments $s_0, s_1, \dots$ is finite. Moreover, if a trace is finite, its last assignment is defined only over the symbols of V^O. We denote the set of traces finite and infinite over V^I, V^O as $\Pi(V^I, V^O)$.

3.1. Syntax.

In this paper, we consider LTL [MP92a] extended with past operators [LPZ85] (with S as "since" and Y as "yesterday") as well as "if-then-else" (ite) and "at next" ($@\tilde{F}$), and "at last" ($@\tilde{P}$) operators from [Ton17]. For simplicity, we refer to it simply as LTL.

We work in the setting of Satisfiability Modulo Theory (SMT) [BSST09] and LTL Modulo Theory (see, e.g., [CGM$^+$19]). First-order formulas are built as usual by proposition logic connectives, a given set of variables V and a first-order signature Σ, and are interpreted according to a given Σ-theory $\mathcal{T}$. We assume to be given the definition of $M, \mu \models_{\mathcal{T}} \varphi$ where M is a Σ-structure, μ is a value assignment to the variables in V, and φ is a formula. Whenever $\mathcal{T}$ and M are clear from contexts we omit them and simply write $\mu \models \varphi$.

Definition 3.2. Given a signature Σ and a set of variables V, LTL formulas φ are defined by the following syntax:

$$\varphi := \top | \bot | pred(u_1, \dots, u_n) | \neg\varphi_1 | \varphi_1 \vee \varphi_2 | X\varphi_1 | \varphi_1 U \varphi_2 | Y\varphi_1 | \varphi_1 S \varphi_2$$

$$u := c | x | func(u_1, \dots, u_n) | next(u_1) | ite(\varphi, u_1, u_2) | u_1 @\tilde{F}\varphi | u_1 @\tilde{P}\varphi$$

where c, $func$, and $pred$ are respectively a constant, a function, and a predicate of the signature Σ and x is a variable in V.

Apart from $@\tilde{F}$ and $@\tilde{P}$, the operators are standard. $u@\tilde{F}\varphi$ represents the value of u at the next point in time (excluding the current point) in which φ holds. Similarly, $u@\tilde{P}\varphi$ represents the value of u at the last point in time in which φ holds (excluding the current point). Figure 2 provides an intuitive graphical view of the operator's semantics.

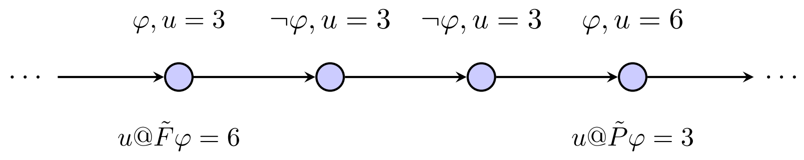


FIGURE 2. Graphical representation of $@\tilde{F}$ and $@\tilde{P}$.

Notation simplification. In the following, we assume to have a background theory such that the symbols in Σ are interpreted by an implicit structure M (e.g., theory of reals, integers, etc.). We therefore omit M to simplify the notation, writing $\pi, i \models_{t^{sgn}} \varphi$ and $\pi(i)(u)$ instead of respectively $\pi, M, i \models_{t^{sgn}} \varphi$ and $\pi^M(i)(u)$.

Finally, we use the following standard abbreviations: $\varphi_1 \wedge \varphi_2 := \neg(\neg\varphi_1 \vee \neg\varphi_2)$, $\varphi_1 R \varphi_2 := \neg(\neg\varphi_1 U \neg\varphi_2)$ (φ_1 releases φ_2), $F\varphi := \top U \phi$ (sometime in the future φ), $G\varphi := \neg F \neg\varphi$ (always in the future φ), $O\varphi := \top S \varphi$ (once in the past φ), $H\varphi := \neg O \neg \varphi$ (historically in the past φ), $Z\varphi := \neg Y \neg \varphi$ (yesterday φ or at initial state), $X^n\varphi := XX^{n-1}\varphi$ with $X^0\varphi := \varphi$, $Y^n\varphi := YY^{n-1}\varphi$ with $Y^0\varphi := \varphi$, $Z^n\varphi := ZZ^{n-1}\varphi$ with $Z^0\varphi := \varphi$, $F^{\leq n}\varphi := \varphi \vee X\varphi \vee \cdots \vee X^n\varphi$, $G^{\leq n}\varphi := \varphi \wedge X\varphi \wedge \cdots \wedge X^n\varphi$, $O^{\leq n}\varphi := \varphi \vee Y\varphi \vee \cdots \vee Y^n\phi$, $H^{\leq n}\varphi := \varphi \wedge Z\varphi \wedge \cdots \wedge Z^n\varphi$.

Example 3.3. *We provide an example of our logic from* [Ton17] *representing a sensor. In this example, the sensor has an input real variable y, an output real variable x and a Boolean flag correct that represents whether or not the value reported by the sensor is correct. We specify that x is always equal to the last correct input value with* $G(x = y@\tilde{P}(correct))$*. We assume that a failure is permanent with* $G(\neg correct \rightarrow G\neg correct)$*. Additionally, we consider also a Boolean variable read representing the event of reading x. In our example, reading occurs periodically with period* 5 *(*$read \wedge G(read \rightarrow X(G^{\leq 3}\neg read) \wedge X^5 read))$*: initially read is true; at each step i, if read is true, then in the states ranging from* $i+1$ *to* $i+4$ *(*$XG^{\leq 3}$*) read is false; and, finally read is true at position* $i+5$*.*

Finally, let us say that an alarm a is true if and only if the last two read values are the same: $G(a \leftrightarrow x@\tilde{P}(read) = (x@\tilde{P}(read))@\tilde{P}(read))$*.*

We can prove that, given the behaviour defined above, every point at which the sensor is not correct, is followed by an alarm in at most 10 *steps.*

$$
\begin{aligned}
&(G(x = y@\tilde{P}(correct)) \wedge G(\neg correct \rightarrow G\neg correct) \wedge \\
&read \wedge G(read \rightarrow X(G^{\leq 3}\neg read) \wedge X^5 read) \wedge \\
&G(a \leftrightarrow x@\tilde{P}(read) = (x@\tilde{P}(read))@\tilde{P}(read))) \\
&\rightarrow G(\neg correct \rightarrow F^{\leq 10} a)
\end{aligned}
$$

3.2. **Semantics.** We propose an alternative semantics for LTL with past operators and event freezing function based on the truncated semantics of LTL defined in [EFH+03a]. This semantics is tailored for both finite and infinite traces. For what regards infinite traces, the semantics is identical to the standard one. When we reason over finite traces, the predicates are interpreted "weakly" i.e. all the predicates are evaluated as true at the end of the trace. Moreover, the semantics differentiate between output and input predicates by treating the latter as *next* operators i.e. they are evaluated as true in the last state.

Another important difference between standard LTL and other finite semantics (e.g. LTL$_f$[DGV13]) lies in the negation. When a formula is negated the interpretation passes from *weak* to *strong* and vice versa. Therefore, a negated formula $\neg\phi$ is satisfied with weak (strong) semantics if and only if ϕ is not satisfied with strong (weak) semantics. This means for instance that a proposition P is evaluated as *true* at the end of a trace independent of the polarity of its variables. These different semantics are denoted as $\models_{t^-}$ for weak semantics and as $\models_{t^+}$ for strong semantics; moreover, we use $\models_{t^{sgn}}$ to refer to both interpretations. For what regards the difference with LTL$_f$, LTL$_f$ requires finite traces to satisfies eventualities i.e. $\pi \models Fp$ iff at some point in the trace p is true. However, in our scenario that semantics is not desirable because we would like to consider as "good" traces also traces that are prefixes of traces satisfying Fp (a further discussion on the choice of semantics is given in Section 4.1.1).

Contrary to the original truncated semantics of [EFH$^+$03a], we consider also past operators, if-then-else, event freezing operators ($@\tilde{F}, @\tilde{P}$), functions and terms.

For what regards past operators, the semantics is similar to standard LTL. The only alteration of the past semantics is that Y is evaluated as *true* at the end of the trace. This is done to keep the value of each formula trivially true when evaluated outside of the trace bounds.

In this paper, terms are not evaluated with a polarity because they are interpreted inside predicates. Besides $ite, @\tilde{F}, @\tilde{P}$, these operators are interpreted identically to standard LTL.

For if-then-else, instead of considering 2 possible term interpretations (u_1, u_2), the new semantics consider a third default value. The third value is chosen when both the if condition and its negation are satisfied weakly (which is possible with finite traces). The idea is that, when both φ and $\neg\varphi$ are weakly satisfied, we avoid committing to either value. As a result, with this three-valued if-then-else, it is still true that $ite(\varphi, u_1, u_2) = ite(\neg\varphi, u_2, u_1)$ which would be not true with a two value semantics.

Regarding $\varphi@\tilde{F}u$ and $\varphi@\tilde{P}u$, the new semantics considers strong satisfaction of the formula to get the value of the term. This occurs because strong semantics ensures that variables are well-defined in the trace. When the φ is not strongly satisfied, a default value $def_{\varphi@\tilde{F}u}$/ $def_{\varphi@\tilde{P}u}$ is considered.

From an higher level perspective, the motivation on using the "weak" semantics is briefly described in Section 4.1.1.

The semantics is defined as follows:

- $\pi, M, i \models_{t^-} pred^O(u_1, \ldots, u_n)$ iff $|\pi| \leq i$ or $pred^M(\pi^M(i)(u_1), \ldots, \pi^M(i)(u_n))$
- $\pi, M, i \models_{t^+} pred^O(u_1, \ldots, u_n)$ iff $|\pi| > i$ and $pred^M(\pi^M(i)(u_1), \ldots, \pi^M(i)(u_n))$
- $\pi, M, i \models_{t^-} pred^I(u_1, \ldots, u_n)$ iff $|\pi| \leq i - 1$ or $pred^M(\pi^M(i)(u_1), \ldots, \pi^M(i)(u_n))$
- $\pi, M, i \models_{t^+} pred^I(u_1, \ldots, u_n)$ iff $|\pi| > i + 1$ and $pred^M(\pi^M(i)(u_1), \ldots, \pi^M(i)(u_n))$
- $\pi, M, i \models_{t^{sgn}} \varphi_1 \wedge \varphi_2$ iff $\pi, M, i \models_{t^{sgn}} \varphi_1$ and $\pi, M, i \models_{t^{sgn}} \varphi_2$
- $\pi, M, i \models_{t^-} \neg\varphi$ iff $\pi, M, i \not\models_{t^+} \varphi$
- $\pi, M, i \models_{t^+} \neg\varphi$ iff $\pi, M, i \not\models_{t^-} \varphi$
- $\pi, M, i \models_{t^{sgn}} \varphi_1 U \varphi_2$ iff there exists $k \geq i, \pi, M, k \models_{t^{sgn}} \varphi_2$ and for all $l, i \leq l < k, \pi, M, l \models_{t^{sgn}} \varphi_1$
- $\pi, M, i \models_{t^-} \varphi_1 S \varphi_2$ iff $i \geq |\pi|$ or there exists $k \leq i, \pi, M, k \models_{t^-} \varphi_2$ and for all $l, k < l \leq i, \pi, M, l \models_{t^{sgn}} \varphi_1$
- $\pi, M, i \models_{t^+} \varphi_1 S \varphi_2$ iff $i < |\pi|$ and there exists $k \leq i, \pi, M, k \models_{t^-} \varphi_2$ and for all $l, k < l \leq i, \pi, M, l \models_{t^{sgn}} \varphi_1$

- $\pi, M, i \models_{t^{sgn}} X\varphi$ iff $\pi, M, i+1 \models_{t^{sgn}} \varphi$
- $\pi, M, i \models_{t^-} Y\varphi$ iff $i \geq |\pi|$ or $i > 0$ and $\pi, M, i-1 \models_{t^-} \varphi$
- $\pi, M, i \models_{t^+} Y\varphi$ iff $0 < i < |\pi|$ and $\pi, M, i-1 \models_{t^+} \varphi$

where $pred^I$ denotes a predicate containing input variables or $@\tilde{F}$ terms, $pred^O$ denotes a predicate that contains only output variables without at next, $sgn \in \{-, +\}$ and $pred^M$ is the interpretation in M of the predicate in Σ.

The interpretation of terms $\pi^M(i)$ is defined as follows:

- $\pi^M(i)(c) = c^M$
- $\pi^M(i)(x) = s_i(x)$ if $x \in V$
- $\pi^M(i)(func(u_1, \ldots, u_n)) = func^M(\pi^M(i)(u_1), \ldots, \pi^M(i)(u_n))$
- $\pi^M(i)(next(u)) = \pi^M(i+1)(u)$ if $|\pi| > i+1$
  $\pi^M(i)(next(u)) = def_{next(u)}$ otherwise.
- $\pi^M(i)(u@\tilde{F}(\varphi)) = \pi^M(k)(u)$if there exists $k > i$ such that, for all $l, i < l < k, \pi, M, l \models_{t^+} \neg\varphi$ and $\pi, M, k \models_{t^+} \varphi$;
  $\pi^M(i)(u@\tilde{F}(\varphi)) = def_{u@\tilde{F}\varphi}$ otherwise.
- $\pi^M(i)(u@\tilde{P}(\varphi)) = \pi^M(k)(u)$if $i < |\pi|$ and there exists $k < i$ such that, for all $l, i > l > k, \pi, M, l \models_{t^+} \neg\varphi$ and $\pi, M, k \models_{t^+} \varphi$;
  $\pi^M(i)(u@\tilde{P}(\varphi)) = def_{u@\tilde{P}\varphi}$ otherwise.
- $\pi^M(i)(ite(\varphi, u_1, u_2)) = \begin{cases} \pi^M(i)(u_1) & \text{if } \pi, M, i \models_{t^+} \varphi \\ \pi^M(i)(u_2) & \text{if } \pi, M, i \models_{t^+} \neg\varphi \\ def_{ite(\varphi, u_1, u_2)} & \text{otherwise} \end{cases}$

where $func^M, c^M$ are the interpretation M of the symbols in Σ, and $def_{u@\tilde{F}\varphi}$, $def_{u@\tilde{P}\varphi}$ and $def_{ite(\varphi,u_1,u_2)}$ are extra variables used to represent default values.

Finally, we have that $\pi, M \models_t \varphi$ iff $\pi, M, 0 \models_{t^-} \varphi$.

Example 3.4. *Here we provide a short example to clarify the semantics. Suppose that a local component is represented by a temporal property $\varphi_i := G(i \to Xo)$; the property expresses that, if the input i is received, at the next step in the local trace the output o is provided. In our semantics, this means that the execution $\pi_f = \{i\}, \{o\}, \{o, i\}, \{o\}$ is satisfied by the property, but since the semantics is weak also $\pi'_f = \{o, i\}\{o\}, \{i\}$ satisfies the property according to the semantics. It should be noted that when we reason over infinite executions, the truncated weak semantics is identical to the standard semantics; therefore, this semantics can be interpreted as a generalisation of the standard one.*

3.3. **Verification.**

We present two techniques to verify LTL formulae with truncated semantics. The first technique is a variation of the rewriting proposed in [DGV13]. It reduces the verification under finite-trace semantics to standard infinite-trace LTL semantics by introducing a fresh Boolean variable $Tail$, which is true only at the last position of the original finite trace and remains true in all subsequent positions. This effectively marks the end of the trace. The rewriting consider both traces in which $Tail$ becomes true and traces in which $Tail$ is never true. In our setting, the extended trace may assign arbitrary values to the input variables from the last position onward. The key intuition is that, under our semantics, the values of input variables matter only at positions where $Tail$ evaluates to false i.e. before position $|\pi| - 1$.

Definition 3.5. We define $Tr(\varphi)$ as follows:

$$Tr^-(Pred^I) := Tail \vee Pred^I \quad Tr^+(Pred^I) := \neg Tail \wedge Pred^I$$
$$Tr^{sgn}(Pred^O) := Pred^O \qquad Tr^{sgn}(\varphi_1 \vee \varphi_2) := Tr^{sgn}(\varphi_1) \vee Tr^{sgn}(\varphi_2)$$
$$Tr^-(\neg\varphi) = \neg Tr^+(\varphi) \quad Tr^+(\neg\varphi) = \neg Tr^-(\varphi)$$
$$Tr^-(X\varphi) := Tail \vee X(Tr^-(\varphi)) \quad Tr^+(X\varphi) := \neg Tail \wedge X(Tr^+(\varphi))$$
$$Tr^{sgn}(\varphi_1 U \varphi_2) := Tr^{sgn}(\varphi_1)U(Tr^{sgn}(\varphi_2))$$

where $sgn \in \{-,+\}$ and $Tr(\varphi) := \neg Tail \wedge G(Tail \rightarrow XTail \wedge \bigwedge_{v_o \in V^O} v_o = next(v_o)) \wedge \rightarrow Tr^-(\varphi)$ is the top-level rewriting that defines the behaviour of $Tail$ [2]

The following theorem relates the truncated semantics to the rewriting. For each finite or infinite trace π we show that the given a formula φ, π satisfies φ if and only if a

Theorem 3.6. *Let π be a trace over V^I, V^O and π' be the infinite trace over $V^I, V^O \cup \{Tail\}$ constructed from π by extending the original trace with $Tail$ as follows.*

- *For all $0 \leq i < |\pi| - 1 : \pi'(i)(V^I, V^O) = \pi(i)(V^I, V^O), \pi'(i)(Tail) = \bot$;*
- *$\pi(|\pi| - 1)(V^O) = \pi'(|\pi| - 1)(V^O)$;*
- *for all $j \geq |\pi| - 1$: $\pi'(j)(Tail) = \top$, $\pi(j)(V^O) = \pi(j+1)(V^O)$.*
- *for all $j \geq |\pi| - 1$, values of V^I are unconstrained i.e. each input variable can be assigned to any value.*

It is true that:

$$\pi \models_{t^{sgn}} \varphi \Leftrightarrow \pi' \models_{LTL} Tr^{sgn}(\varphi)$$

Proof. We can prove the theorem inductively over the structure of the formula for each $i \geq 1$. Formally the statement is: for all $i \geq 0$: $\pi, i' \models_{t^{sgn}} \varphi \Leftrightarrow \pi, i \models_{LTL} Tr^+(\varphi)$ where $i' = \min(|\pi| - 1, i)$.

The base cases on predicates are very simple. If $i' \geq |\pi| - 1$ then $Tr^-(Pred^I)$ is true, $Tr^+(Pred^I)$ is false and $Tr^{sgn}(Pred^O) = Pred^O$ which has the value of $\pi(|\pi| - 1)(Pred^O)$ when $i \geq |\pi| - 1$.

If $i' < |\pi| - 1$ then output variables are immediate to prove and input variables follows since $Tail$ is false.

For the inductive cases, negation, disjunction and Until follow from induction. We need to prove only X. If $i' = |\pi| - 1$, then $\pi, i' \models_{t^-} X\varphi$ and $\pi, i' \not\models_{t^+} X\varphi$; since $\pi, i' \models_{LTL} Tail$, then $\pi', i \models_{LTL} Tr^-(X\varphi)$ and $\pi', i \not\models_{LTL} Tr^+(X\varphi)$. If $i' < |\pi| - 1$, then the case holds by induction ($Tail$ is false). □

The second technique reduces the problem to the verification of safetyLTL fragment [Sis85]. The idea is that the property is valid on the truncated semantics iff the property is valid with standard semantics and its "safety part" is valid with truncated semantics. The verification of the safety fragment is carried out reducing the problem to invariant checking. Contrary to the standard semantics, in our setting it is not necessary to

[2]To verify the rewritten formula over a transition system, the system must be extended with a sink state in which the predicate $Tail$ holds. This ensures that when $Tail$ becomes true, the system remains in that state indefinitely (i.e., the transition relation becomes $(Tail \rightarrow V^{O'} = V^O \wedge Tail') \wedge (\neg Tail \rightarrow \mathcal{T})$). Note that the rewritten formula implicitly refers to the transition relation of this modified system.

check that the finite trace is extendible as done instead in [BCTZ23]. The reduction from safetyLTL to invariant is based on [CGH94, Lat03, KV01] and have been detailed practically in [BCTZ23, BCGT25]. In the following, we propose the rewriting to transform a formula from LTL to safetyLTL. The transformation is performed by replacing Until with Release on the formula in negative normal form (see Definition 3.7 below). The intuition is that with weak semantics and finite traces Until are interpretable as "Weak" Until $(G\varphi_1 \vee (\varphi_1 U \varphi_2))$ which is easily rewritten as Release $(\varphi_2 R(\varphi_1 \vee \varphi_2))$.

Definition 3.7. Given a formula φ in negative normal form, we define the "Weak-to-Safety" translation function $\mathcal{W}2\mathcal{S}(\varphi)$, which maps formulas interpreted under weak semantics into equivalent formulas interpreted within the SafetyLTL fragment, as follows:

$$\mathcal{W}2\mathcal{S}(Pred) := Pred \quad \mathcal{W}2\mathcal{S}(\neg Pred) := \neg Pred$$
$$\mathcal{W}2\mathcal{S}(\varphi_1 \vee \varphi_2) := \mathcal{W}2\mathcal{S}(\varphi_1) \vee \mathcal{W}2\mathcal{S}(\varphi_2) \quad \mathcal{W}2\mathcal{S}(\varphi_1 \wedge \varphi_2) := \mathcal{W}2\mathcal{S}(\varphi_1) \wedge \mathcal{W}2\mathcal{S}(\varphi_2)$$
$$\mathcal{W}2\mathcal{S}(X\varphi) := X\mathcal{W}2\mathcal{S}(\varphi) \quad \mathcal{W}2\mathcal{S}(Y\varphi) := Y\mathcal{W}2\mathcal{S}(\varphi)$$
$$\mathcal{W}2\mathcal{S}(\varphi_1 U \varphi_2) := \mathcal{W}2\mathcal{S}(\varphi_2) R(\mathcal{W}2\mathcal{S}(\varphi_1) \vee \mathcal{W}2\mathcal{S}(\varphi_2))$$
$$\mathcal{W}2\mathcal{S}(\varphi_1 R \varphi_2) := \mathcal{W}2\mathcal{S}(\varphi_1) R(\mathcal{W}2\mathcal{S}(\varphi_2))$$

Theorem 3.8. *For each trace π:*

- *If π is an infinite trace: $\pi \models_{t^-} \varphi \Leftrightarrow \pi \models_{LTL} \varphi$.*
- *If π is a finite trace: $\pi \models_{t^-} \varphi \Leftrightarrow \pi, \models_{t^-} \mathcal{W}2\mathcal{S}(\varphi)$.*

Proof. In the case of infinite traces, the theorem follows from the fact that the truncated semantics is equivalent to LTL (with event-freezing functions).

We prove the finite trace case by induction on the structure of the formula considering each possible index $0 \leq i < n$ with $n = |\pi|$.

We only need to prove the case of the operator U since $\mathcal{W}2\mathcal{S}$ is transparent for the other operators. Moreover, we need to prove only weak semantics because the top level formula is in negative normal form.

$\pi, i \models_{t^-} \varphi_1 U \varphi_2 \Leftrightarrow \exists k \geq i$ s.t. $\pi, k \models_{t^-} \varphi_2$ and $\forall i \leq j < k : \pi, j \models_{t^-} \varphi_1$. By induction we obtain that $\cdots \Leftrightarrow \exists k > i$ s.t. $\pi, k \models_{t^-} \mathcal{W}2\mathcal{S}(\varphi_2)$ and $\forall i \leq j < k : \pi, j \models_{t^-} \mathcal{W}2\mathcal{S}(\varphi_1)$.

If $k \geq |\pi|$ then $\pi, k \models_{t^-} \varphi_2$ since all formulae are satisfied weakly at the end of the trace; therefore, $\pi, i \models_{t^-} G\varphi_1 \vee \varphi_1 U \varphi_2$ which is equivalent to $\varphi_2 R(\varphi_1 \vee \varphi_2)$. □

4. Compositional reasoning

4.1. Formal problem.

Compositional verification proves the properties of a system by proving the local properties on components and by checking that the composition of the local properties satisfies the global one (see [RBH+01] for a generic overview). This reasoning is expressed formally by inference 4.1, which is parametrized by a function γ_S that combines the component's implementations and a related function γ_P that combines the local properties.

Inference 4.1. *Let $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n$ be a set of n components, $\varphi_1, \varphi_2, \ldots, \varphi_n$ be local properties on each component, γ_S is a function that defines the composition of $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n$, γ_P combines the properties depending on the composition of γ_S and φ a property. The* *following inference is true:*

$$\frac{\dfrac{\mathcal{M}_1 \models \varphi_1, \mathcal{M}_2 \models \varphi_2, \ldots, \mathcal{M}_n \models \varphi_n}{\gamma_S(\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n) \models \gamma_P(\varphi_1, \varphi_2, \ldots, \varphi_n)} \quad \gamma_P(\varphi_1, \varphi_2, \ldots, \varphi_n) \models \varphi}{\gamma_S(\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n) \models \varphi}$$

The problem we address in this paper is to define proper γ_S, γ_P representing the composition of possibly terminating components and temporal properties such that the inference rule holds.

4.1.1. *Motivation for the Semantic choice.* In this work, we adopt the weak semantics of temporal operators described in Section 3. This choice is motivated by the characteristic of our setting. Specifically, during composition, local traces may be truncated by the scheduler or due to a component failure, and these truncated traces remain relevant and may contribute to the satisfaction of the system-level property. In a sense, we need to pertain the safety part of the property ensured by the finite trace, while disregarding the liveness part of it. The weak semantics proposed by [EFH$^+$03a] allows us to account for this, ensuring that truncated traces are not disregarded in the logical reasoning process.

As a motivating example, consider the local property φ_{c1} in Figure 1. This property states that whenever component $c1$ receives an input in_1 via the rec_1 signal, it will attempt to propagate the message to component $c2$ until it succeeds. Suppose that $c1$ receives the value 3 at time 0 and begins this propagation. If a failure occurs or the scheduler stops activating $c1$, its execution may be prematurely truncated, and only a finite prefix of its local trace is available. Still, this partial trace captures the component's behaviour up to the end of its execution– specifically, in this case, that c_1 initiated the process of sending the input to c_2. Weak semantics ensures that such truncated traces are preserved in system-level reasoning, without requiring that liveness properties be fully realized. If (on the composition) stronger guarantees are required–e.g., to ensure infinite progress or rule out failures—this can be encoded explicitly in the environment assumption, such as $\alpha' := \alpha \wedge GFrun_{c1}$, ensuring fair scheduling and infinite execution.

At the global level, traces may also be finite to support hierarchical composition. In such cases, weak semantics provides a conservative and practical interpretation: a property satisfied on a finite trace indicates that no counterexample has been found yet, though one may still arise on an extension. Consider the system-level property $\varphi := G((rec_1 \wedge in_1 = v) \rightarrow F(send \wedge out = v \vee fail \wedge out_f = v))$ from Figure 1. In this case, since φ is a liveness property, every finite trace satisfy weakly φ. In general, the only problematic cases would involve finite traces that actually constitute counterexamples, for instance due to reaching a deadlock state that violates liveness. However, removing deadlocks is computationally costly, especially in a compositional setting. Thus, adopting weak semantics is a conscious compromise: it avoids penalizing valid partial behaviours, and in practice, systems are typically defined to avoid such pathological deadlock cases.

4.2. **Interface Transition Systems.** In this paper, we represent I/O components as Interface Transition Systems, a symbolic version of interface automata [dAH01] that considers I/O variables instead of I/O actions.

Definition 4.2. An Interface Transition System (ITS) $\mathcal{M}$ is a tuple
$\mathcal{M} = \langle V^I, V^O, \mathcal{I}, \mathcal{T}, \mathcal{SF} \rangle$ where:

- V^I is the set of input variables while V^O is the set of output variables where $V^I \cap V^O = \emptyset$.
- $V := V^I \cup V^O$ denotes the set of the variables of $\mathcal{M}$
- $\mathcal{I}$ is the initial condition, a formula over V^O,
- $\mathcal{T}$ is the transition condition, a formula over $V \cup V^{O'}$ where $V^{O'}$ is the primed versions of V^O
- $\mathcal{SF}$ is the set of strong fairness constraints, a set of pairs of formulas over V.

A symbolic transition system with strong fairness $\mathcal{M} = \langle V, \mathcal{I}, \mathcal{T}, \mathcal{SF}\rangle$ is an interface transition system without input variables (i.e., $\langle \emptyset, V, \mathcal{I}, \mathcal{T}, \mathcal{SF}\rangle$).

Definition 4.3. A trace π of an ITS $\mathcal{M}$ is a trace $\pi = s_0 s_1 \cdots \in \Pi(V^I, V^O)$ s.t.

- $s_0 \models \mathcal{I}$
- For all $i < |\pi| - 1$, $s_i \cup s'_{i+1} \models \mathcal{T}$, and
- If π is infinite: for all $\langle f_A, f_G\rangle \in \mathcal{SF}$, If for all i there exists $j \geq i$, $s_j \models f_A$ then for all i there exists $j \geq i : s_j \models f_G$.

The finite and infinite languages $\mathcal{L}^{<\omega}(\mathcal{M})$ and $\mathcal{L}^{\omega}(\mathcal{M})$ of an ITS $\mathcal{M}$ is the set of all finite and infinite traces of $\mathcal{M}$, respectively. Finally, we denote $\mathcal{L}(\mathcal{M}) := \mathcal{L}^{<\omega}(\mathcal{M}) \cup \mathcal{L}^{\omega}(\mathcal{M})$ as the language of $\mathcal{M}$.

Definition 4.4. Let $\pi = s_0 s_1 \ldots$ be a trace of an ITS $\mathcal{M}$ and $V' \subseteq V$ a set of symbols of $\mathcal{M}$. We denote $s_i(V')$ as the restriction of the assignment s_i to the symbols of V'; moreover, we denote $\pi_{|V'} := s_0(V')s_1(V')\ldots$ as the restriction of all the state assignments of π to the symbols of V'. Furthermore, we denote $\mathcal{L}(\mathcal{M})_{|V'} = \{\pi_{|V'} | \pi \in \mathcal{L}(\mathcal{M})\}$ as the restriction of all the traces of the language of an ITS $\mathcal{M}$ to a set of symbols $V' \subseteq V$.

Definition 4.5. Let $\mathcal{M}_1, \ldots, \mathcal{M}_n$ be n ITS, they are said compatible iff they share respectively only input with output (i.e. $\forall i \leq n, j \leq n$ s.t. $i \neq j : V_i \cap V_j = (V_i^O \cap V_j^I) \cup (V_i^I \cap V_j^O)$)

4.3. **Asynchronous composition of ITS.** We now provide the notion of asynchronous composition ($\otimes$) that will be used as the composition function (γ_S) of Inference 4.1.

Definition 4.6. Let $\mathcal{M}_1, \ldots, \mathcal{M}_n$ be n compatible interface transition systems, let $run_1, \ldots,$ run_n be n Boolean variables not occurring in $\mathcal{M}_1, \ldots, \mathcal{M}_n$ (i.e. $run_1, \ldots run_n \notin \bigcup_{1 \leq i \leq n}(V_i^I \cup V_i^O)$) and $end_1, \ldots, end_n$ be n Boolean variables not occurring in $\mathcal{M}_1, \ldots, \mathcal{M}_n$ i.e. end_1, $\ldots end_n \notin \bigcup_{1 \leq i \leq n}(V_i^I \cup V_i^O)$. $\mathcal{M}_1 \otimes \cdots \otimes \mathcal{M}_n = \langle V^I, V^O, \bigwedge_{1 \leq i \leq n} \mathcal{I}_i, \mathcal{T}, \mathcal{SF}\rangle$ where:

$$V^I = (\bigcup_{1 \leq i \leq n} V_i^I \cup \{run_i\}) \setminus (\bigcup_{1 \leq i < n} \bigcup_{i < j \leq n} V_i \cap V_j)$$

$$V^O = (\bigcup_{1 \leq i \leq n} V_i^O \cup \{end_i\}) \cup (\bigcup_{1 \leq i < n} \bigcup_{i < j \leq n} V_i \cap V_j)$$

$$\mathcal{T} = \bigwedge_{1 \leq i \leq n} ((run_i \rightarrow \mathcal{T}_i) \wedge \psi_{cond}^{\mathcal{M}_i} \wedge (end_i \leftrightarrow end'_i \wedge \neg run_i))$$

$$\mathcal{SF} = \bigcup_{1 \leq i \leq n} (\{\langle \top, run_i \vee end_i\rangle\} \cup \{\langle run_i \wedge \varphi_a, run_i \wedge \varphi_g\rangle | \langle \varphi_a, \varphi_g\rangle \in \mathcal{SF}_i\})$$

where $\psi_{cond}^{\mathcal{M}_i} := \neg run_i \rightarrow \bigwedge_{v \in V_i^O} v = v'$.

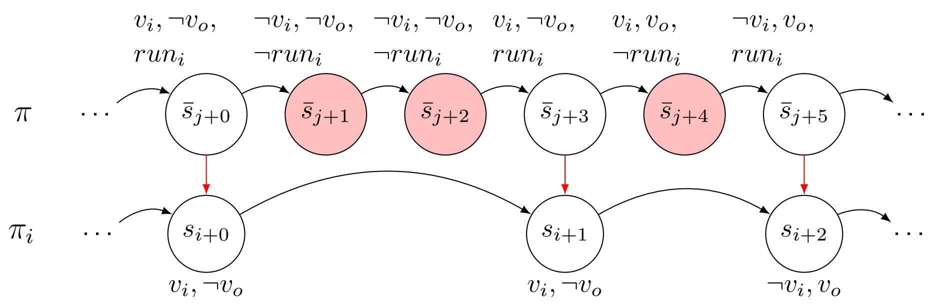


Figure 3. Graphical view of trace projection. White states of π represent the states of the sequence map_i, pink states represent states in which the local component stutters, and red arrows represent the link between the states of π and the states of π_i formally represented by map_k.

The operator $\otimes$ provides a notion of asynchronous composition of Interface Transition Systems based on interleaving. Each component can either run (execute a transition) or stutter (freeze output variables). Contrary to our previous work[BT22], this new definition allows for finite executions of local components. For each i, we introduce the variable run_i representing the execution of a transition of $\mathcal{M}_i$; furthermore, we introduce the prophecy variables end_i that monitor whether or not a component will execute new transitions in the future i.e. end_i is equivalent to $G\neg run_i$.

Definition 4.7. Let π be a trace of $\mathcal{M} = \mathcal{M}_1 \otimes \cdots \otimes \mathcal{M}_n$ with $i \leq n$. We define the projection function $Pr_{\mathcal{M}_i} : \Pi(V) \to \Pi(V_i)$ as follows:

$$Pr_{\mathcal{M}_i}(\pi) = \begin{cases} s_{map_0}(V_i), \dots & \text{If } \pi \text{ is infinite and } \pi \models GFrun_i \\ s_{map_0}(V_i), \dots, s_{map_{n-1}}(V_i), s_{map_n}(V_i^O) & \text{Otherwise} \end{cases}$$

where map_k is the sequence mapping each state of π_i into a state of π as follows:

$$map_k := \begin{cases} k & \text{If } k < 0 \\ map_{k-1} + 1 & \text{If } k \geq |\pi| - 1 \\ k' \mid k' > map_{k-1}, \pi, k' \models run_i \text{ and } \forall_{map_{k-1} < j' < k'} \pi, j' \not\models run_i & \text{Otherwise} \end{cases}$$

Moreover, we define the inverse operator of Pr, denoted by Pr^{-1}:

$$Pr^{-1}_{\mathcal{M}_i}(\pi) = \{\pi' | Pr_{\mathcal{M}_i}(\pi') = \pi\}$$

Definition 4.7 provides a mapping between the composed ITS and the local transition system. The function map_k maps indexes of the local trace to the indexes of the global trace. For each k in the range of the trace excluding the last point (i.e. $[0, |\pi| - 1)$) map_k represent the k-th occurrence of run counting from 0. It should be noted that map_0 is not 0 but the first point of the trace in which run_i is true ($k' | map_{-1} < k', \pi, k' \models run_i$ and $\forall map_{-1} < k'' < k'\ \pi \not\models run_i$). Finally, the projection of a global trace to a local trace is defined using map. A graphical representation of projection is shown in Figure 3. We now show that the language of each local ITS $\mathcal{M}_i$ contains the language of the composition projected to the local ITS. From this result we can derive a condition for γ_P such that Inference 4.1 holds. We do that by considering a mapping between local trace and global traces that follows the same mapping map_i.

Theorem 4.8. *Let $\mathcal{M} = \mathcal{M}_1 \otimes \cdots \otimes \mathcal{M}_n$:*

$$\text{For all } 1 \leq i \leq n : \mathcal{L}(\mathcal{M}_i) \supseteq \{Pr_{\mathcal{M}_i}(\pi) | \pi \in \mathcal{L}(\mathcal{M})\}$$

Proof. Given a trace $\pi \in \mathcal{L}(\mathcal{M}), \pi_i := Pr_{\mathcal{M}_i}(\pi)$. We prove the theorem by induction on the length of π_i. The inductive hypothesis states that if $|\pi_i| > k+1$ and $\pi_i^{0\dots k} \in \mathcal{L}^{<\omega}(\mathcal{M}_i)$, then $\pi_i^{0\dots k+1} \in \mathcal{L}^{<\omega}(\mathcal{M}_i)$.

- Base case:
  By definition $\mathcal{I} := \bigwedge_{1 \leq i \leq n} \mathcal{I}_i$ and $\pi_i(0) = \pi(map_0)(V_i)$. We derive that $\pi_i, 0 \models \mathcal{I}_i \Rightarrow \pi_i^{0\dots 0} \in \mathcal{L}^{<\omega}(\mathcal{M}_i)$. It should be noted that this holds also if $|\pi_i| = 1$ and $\pi, |\pi| - 1 \not\models run_i$ because $\mathcal{I}_i$ is a proposition on symbols of V_i^O.
- Inductive case:
  By definition $\pi_i(k) = \pi(map_k)(V_i)$ where map_k is the kth position s.t. run_i holds. Therefore, $\pi(map_k)\pi(map_k+1) \models run_i \rightarrow \mathcal{T}_i \Rightarrow \pi(map_k)\pi(map_k+1) \models \mathcal{T}_i$. By $\psi_{cond}^{\mathcal{M}_i}$ we obtain that $\pi_{map_{k+1}}(V_i^O) = \pi_{map_k+1}(V_i^O) = \pi_i(k+1)(V_i^O)$. Since $\mathcal{T}_i$ reasons over symbols of $V \cup V^{O\prime}$, then $\pi(map_k)\pi(map_k+1) \models \mathcal{T}_i \Leftrightarrow \pi_i(k)\pi_i(k+1) \models \mathcal{T}_i$. Therefore, $\pi_i, k \models \mathcal{T}_i$ which guarantees that $\pi_i^{0\dots k+1} \in \mathcal{L}^{<\omega}(\mathcal{M}_i)$.

We proved the theorem for finite traces, we now extend the proof to consider infinite traces. To do so, it is sufficient to prove that each infinite trace π_i satisfies the strong fairness conditions of $\mathcal{SF}_i$ since it already satisfies $\mathcal{I}_i$ and $\mathcal{T}_i$.

Since $\pi_i = Pr_{\mathcal{M}_i}(\pi)$ and $\pi \in \mathcal{L}(\mathcal{M})$, $\pi \models \bigwedge_{\langle f_a, f_g \rangle \in \mathcal{SF}} (GFf_a \rightarrow GFf_g)$. In particular, due to the composition, $\pi \models \bigwedge_{\langle f_a, f_g \rangle \in \mathcal{SF}_i} (GF(run_i \wedge f_a) \rightarrow GF(run_i \wedge f_g))$ The projection defines the sequence $map_0, \dots$ representing the points in which run_i holds; therefore, for all $\langle f_a, f_g \rangle \in$if for all k there exists $j \geq k$ s.t. $\pi, map_k \models f_a$ then for all k' there exists $j' \geq k'$ s.t. $\pi, map_k' \models f_g$. From the projection definition then $\pi_i \models GFf_a \rightarrow GFf_g$ for each $\langle f_a, f_g \rangle \in \mathcal{SF}_i$. □

Definition 4.9. Let $\mathcal{M}_1, \dots, \mathcal{M}_n$ be n ITS, we define γ_S as follows:

$$\gamma_S(\mathcal{M}_1, \dots, \mathcal{M}_n) := \mathcal{M}_1 \otimes \cdots \otimes \mathcal{M}_n$$

From Theorem 4.8, we obtain a composition γ_S for which each projected global trace is an actual behaviour of a local trace. Therefore, when we consider the global traces, we do not introduce new local traces which cannot be witnessed locally.

To complete our compositional reasoning, we need to define the function γ_P such that Inference 4.1 holds. To do so, Section 5 provides a rewriting technique that maps each local trace satisfying φ_i to a global trace satisfying γ_P.

5. Rewriting

In this section, we introduce a rewriting-based approach for the composition of local properties. In section 5.1, we present the rewriting of the logic of Section 3 that maps local properties to global properties with proofs and complexity results. In section 5.2, we propose an optimized version of the rewriting. Finally, in section 5.3, we propose a variation of the optimized rewriting tailored for infinite executions of local properties i.e. in which the semantics is the one of standard LTL because we assume fairness of run_i.

To simplify the notation, we assume to be given n interface transition systems $\mathcal{M}_1, \dots,$ $\mathcal{M}_n$, a composed ITS $\mathcal{C} = \mathcal{M}_1 \otimes \cdots \otimes \mathcal{M}_n$, a trace π of $\mathcal{M}_i$ with $i < n$, a local property φ and a local term u. For brevity, we refer to $\mathcal{R}_{\mathcal{M}_i}$, $\mathcal{R}^*_{\mathcal{M}_i}$, $\mathcal{R}^\theta{}_{\mathcal{M}_i}$, $Pr^{-1}_{\mathcal{M}_i}$, $Pr_{\mathcal{M}_i}$, end_i

and run_i as respectively $\mathcal{R}, \mathcal{R}^*, \mathcal{R}^\theta, \mathcal{R}^{\theta*}, Pr^{-1}, Pr, end$ and run. Moreover, we will refer to $map_0, map_1, \dots$ as the sequence of definition 4.7. As in section 3.2, we denote input predicates with apex I: $Pred^I$ and the output predicates and terms with apex O: $Pred^O$. Finally, we denote $sgn \in \{-, +\}$.

5.1. **Trucanted LTL with Event Freezing Function Compositional Rewriting.**

In this section, we propose a rewriting to asynchronously compose propositional truncated LTL properties over Interface Transition Systems symbols.

The idea is based on the notions of projection (introduced in Definition 4.7) and asynchronous composition (introduced in Definition 4.6). We produce a rewriting $\mathcal{R}^*$ for φ such that each trace π of $\mathcal{M}_i$ satisfies the rewritten formula iff the projected trace satisfies the original property.

Definition 5.1. We define $\mathcal{R}$ as the following rewriting function:

$$\mathcal{R}^-(Pred^I(u_1, \dots, u_n)) := \neg run \vee Pred^I(\mathcal{R}^-(u_1), \dots, \mathcal{R}^-(u_n))$$
$$\mathcal{R}^+(Pred^I(u_1, \dots, u_n)) := run \wedge Pred^I(\mathcal{R}^+(u_1), \dots, \mathcal{R}^+(u_n))$$
$$\mathcal{R}^{sgn}(Pred^O(u_1, \dots, u_n)) := Pred^O(\mathcal{R}^{sgn}(u_1), \dots, \mathcal{R}^{sgn}(u_n))$$
$$\mathcal{R}^{sgn}(\varphi_1 \vee \varphi_2) := \mathcal{R}^{sgn}(\varphi_1) \vee \mathcal{R}^{sgn}(\varphi_2)$$
$$\mathcal{R}^-(\neg\varphi) := \neg\mathcal{R}^+(\varphi), \mathcal{R}^+(\neg\varphi) := \neg\mathcal{R}^-(\varphi)$$
$$\mathcal{R}^-(X\varphi) := X(state R(\neg state \vee \mathcal{R}^-(\varphi)))$$
$$\mathcal{R}^+(X\varphi) := X(\neg state U(state \wedge \mathcal{R}^+(\varphi)))$$
$$\mathcal{R}^-(\varphi_1 U \varphi_2) := (\neg state \vee \mathcal{R}^-(\varphi_1))U((state \wedge \mathcal{R}^-(\varphi_2)) \vee Y end)$$
$$\mathcal{R}^+(\varphi_1 U \varphi_2) := (\neg state \vee \mathcal{R}^+(\varphi_1))U(state \wedge \mathcal{R}^+(\varphi_2))$$
$$\mathcal{R}^{sgn}(Y\varphi) := Y(\neg run S(run \wedge \mathcal{R}^{sgn}(\varphi)))$$
$$\mathcal{R}^{sgn}(\varphi_1 S \varphi_2) := (\neg state \vee \mathcal{R}^{sgn}(\varphi_1))S(state \wedge \mathcal{R}^{sgn}(\varphi_2))$$
$$\mathcal{R}^{sgn}(func(u_1, ..., u_n)) := func(\mathcal{R}^{sgn}(u_1), ..., \mathcal{R}^{sgn}(u_n))$$
$$\mathcal{R}^{sgn}(x) := x, \mathcal{R}^{sgn}(c) := c$$
$$\mathcal{R}^{sgn}(ite(\varphi, u_1, u_2)) := ite(\mathcal{R}^+(\varphi), \mathcal{R}^-(u_1), ite(\mathcal{R}^+(\neg\varphi), \mathcal{R}^-(u_2), def_{ite(\varphi,u_1,u_2)}))$$
$$\mathcal{R}^{sgn}(next(u)) := \mathcal{R}^{sgn}(u)@\tilde{F}(state)$$
$$\mathcal{R}^{sgn}(u@\tilde{F}\varphi) := \mathcal{R}^{sgn}(u)@\tilde{F}(state \wedge \mathcal{R}^+(\varphi))$$
$$\mathcal{R}^{sgn}(u@\tilde{P}\varphi) := \mathcal{R}^{sgn}(u)@\tilde{P}(state \wedge \mathcal{R}^+(\varphi))$$

where $state := run \vee (Zrun \wedge end)$ and $\mathcal{R}(\varphi) = \mathcal{R}^-(\varphi)$.

$\mathcal{R}$ transforms the formula applying run and $state$ to φ. The general intuition is that when run is true, the local component triggers a transition. Moreover, $state$ represents a local state of π in the global trace. It should be noted that the rewriting has to deal with the last state. Therefore, $state$ is represented by a state that either satisfies run or is the successor of the last state that satisfies run.

For output predicates, the rewriting is transparent because the projection definition guarantees that each state i of π has the same evaluation of each map_i state of π^{ST}. Although this holds for input predicates as well, there is a semantic technicality that should be taken

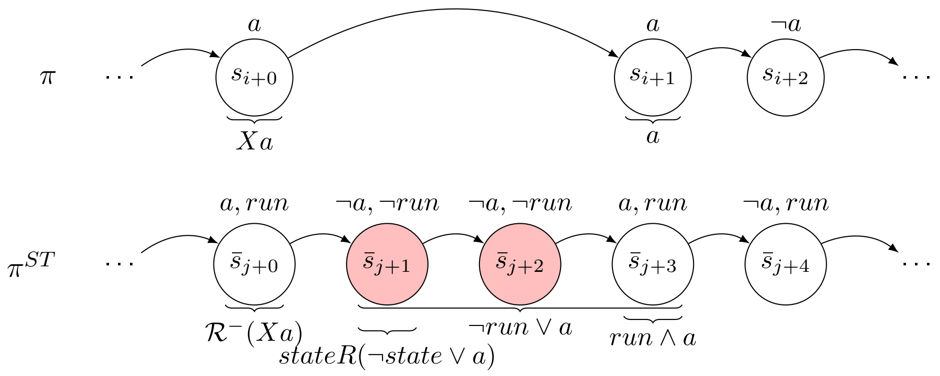


FIGURE 4. Graphical representation of rewriting of Xa. π represents the local trace while π^{ST} represents the trace of the composition. White states are states of local trace while pink states are states in which the local component is not running. In this example, a is an input variable which happens to be true in state s_i and s_{i+1} of local trace π and state $\bar{s}_j$. To show the intuition of the rewriting, we show that Release operator permits to skip the pink states (which are not relevant w.r.t. the local trace). Finally, at state $\bar{s}_{j+3}$ a is evaluated since run is true.

into account. Input predicates are evaluated differently on the last state: with *strong* semantics (+) they do not hold while on *weak* semantics (−) they hold. Since (i) the last state of a local trace might not be the last state of the global trace, and (ii) a variable that locally is an input variable might be an output variable of the composed system, we have to take input predicates into account inside the rewriting. Therefore, with *strong* semantics, the rewriting forces the predicate to hold in a transition i.e. when *run* holds ($\pi, i \models_{t^{sgn}} Pred^I \Leftrightarrow i < |\pi| - 1$ and $Pred^I(\pi(i)(u_1), \dots, \pi(i)(u_n))$). The *weak* semantics is managed specularly.

Regarding X, $\mathcal{R}^-$ needs to pass from point map_i to map_{i+1} and to verify that the sub-formula is verified in that state. The first X passes from map_i to $map_i + 1$. Then, the rewriting skips all states that are not map_{i+1} and "stops" in position map_{i+1}. Figure 4 shows an intuitive representation of the rewriting of X when the trace is not in its last state.

For Until formulae, the rewriting needs to "skip" all points that either do not belong to the local trace or satisfy the left part of the formula while it must "stop" to a point that satisfies the right part and is a state in the local trace. To do so, it introduces a disjunction with *state* on the left side of the formula and a conjunction with *state* on the right side of the formula. When the rewriting and the formula are interpreted weakly, reaching the end of the local trace makes the formula true; therefore, $Yend$ is also put in disjunction with the right side of the formula. Figure 5 shows an intuitive representation of the rewriting of U when the trace does not terminate before reaching b.

Lemma 5.2. *For all $\pi^{ST} \in Pr^{-1}(\pi)$, for all $i < |\pi|$:*

$$\pi, i \models_{t^{sgn}} \varphi \Leftrightarrow \pi^{ST}, map_i \models_{t^{sgn}} \mathcal{R}^{sgn}(\varphi)$$

Proof. We prove Lemma 5.2 by induction on the formula. The high level intuition is that, assuming that we are in a point map_i of the trace, we can "reach" the corresponding map_k

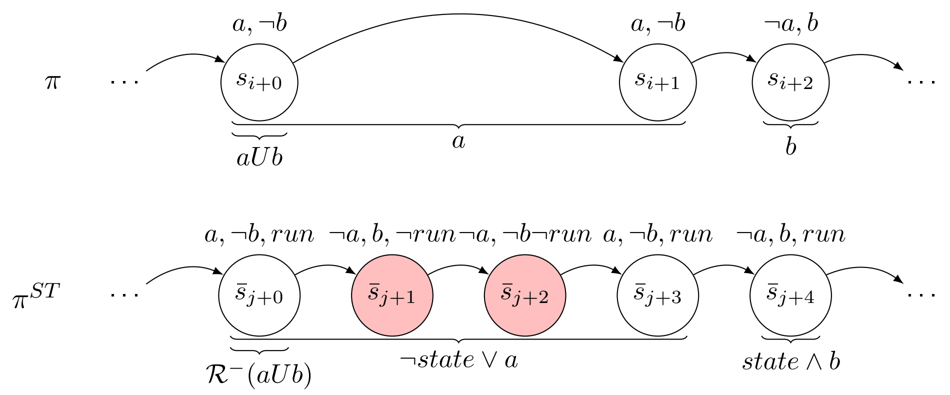


Figure 5. Graphical representation of rewriting of U. In this example both a and b variables are input variables. Local trace π satisfies aUb at position i since a is true at state s_i, s_{i+1} while b is true at state s_{i+2}. The corresponding global trace contains 2 additional states $(\bar{s}_{j+1}, \bar{s}_{j+2})$ which are not considered in the rewriting since $\neg state$ holds these 2 states. Finally, the state $\bar{s}_{j+4}$ of the global trace satisfies both $state$ and b.

using the syntactically rewriting for each temporal operator. Two graphical examples of that are given in Figure 4 and Figure 5 for respectively X and U.

Before starting the proof we observe the following facts:

(1) For all $i : i < |\pi| - 1 \Leftrightarrow \pi^{ST}, map_i \models_{t^+} run$.
(2) For all $i : i < |\pi| \Leftrightarrow \pi^{ST}, map_i \models_{t^+} state$.
(3) $\forall i < |\pi|, \forall map_{i-1} < j \leq map_i : \pi^{ST}, map_i \models_{t^-} \psi \Leftrightarrow \pi^{ST}, j \models_{t^-} stateR(\neg state \vee \psi)$.
(4) $\forall i < |\pi|, \forall map_{i-1} < j \leq map_i : \pi^{ST}, map_i \models_{t^+} \psi \Leftrightarrow \pi^{ST}, j \models_{t^+} \neg stateU(state \wedge \psi)$.
(5) $\pi^{ST}, map_{|\pi|-1} + 1 \models_{t^{sgn}} Yend$.

Base cases:

- $Pred^I$: $\pi, i \models_{t^-} Pred^I \Leftrightarrow i \geq |\pi| - 1 \vee Pred^I \overset{Pr}{\Leftrightarrow} i \geq |\pi| - 1$ or $\pi^{ST}, map_i \models_{t^-} Pred^I \overset{1}{\Leftrightarrow} \pi^{ST}, map_i \models_{t^-} \neg run \vee Pred^I$. The strong semantics case is identical.
- $Pred^O, x, c$: Trivial.

Inductive cases:

- $\neg$: Trivial in both cases. It follows the semantics definition
- $\vee, Pred^I, Pred^O, func$: Trivial.
- $X\varphi$: $\pi^{ST}, map_i \models_{t^-} \mathcal{R}(X\varphi) \Leftrightarrow \pi^{ST}, map_i + 1 \models_{t^-} stateR(\neg state \vee \mathcal{R}(\varphi)) \overset{3}{\Leftrightarrow} \pi^{ST}, map_{i+1} \models_{t^-} \mathcal{R}(\varphi)$ if $i < |\pi| - 1$ (By ind. holds). Otherwise, $i = |\pi| - 1$ and thus $\pi^{ST}, map_i + 1 \models_{t^-} G\neg run$ which implies $\pi^{ST}, map_i \models \mathcal{R}(X\varphi)$ as expected by the weak semantics. We skip the strong case because it is similar.
- $\varphi_1 U \varphi_2$: $\pi^{ST}, map_i \models_{t^-} \mathcal{R}^-(\varphi_1 U \varphi_2) \Leftrightarrow \pi^{ST}, map_i \models_{t^-} (state \vee \mathcal{R}^-(\varphi_1))U(\neg state \wedge \mathcal{R}^-(\varphi_2) \vee Yend) \Leftrightarrow \exists k' \geq map_i$ s.t. $(\pi^{ST}, k' \models_{t^-} \mathcal{R}^-(\varphi_2)$ and $\pi^{ST}, k' \models_{t^-} state$ or $\pi^{ST}, k' \models_{t^-} Yend)$ and $\forall map_i \leq j' < k' : \pi^{ST}, j' \models_{t^-} \mathcal{R}(\varphi_1)$ or $\pi^{ST}, j' \models_{t^-} state \overset{2,5}{\Leftrightarrow} \exists k \geq i$ s.t. $\pi^{ST}, map_k \models_{t^-} \mathcal{R}(\varphi_2)$ and $k < |\pi|$ or $k = |\pi|$ and $\forall i \leq j < k \pi^{ST}, map_j \models_{t^-} \mathcal{R}(\varphi_1) \overset{Ind.(k, j<|\pi|), \forall \phi:\pi, |\pi| \models_{t^-} \phi}{\Leftrightarrow} \exists k \geq i$ s.t. $\pi, k \models_{t^-} \varphi_2$ and $\forall i \leq j < k : \pi, j \models_{t^-} \varphi_1 \Leftrightarrow \pi, i \models_{t^-} \varphi_1 U \varphi_2$. The proof of the strong case is the same.

- $Y\varphi$: The case is specular to the case of X. In this specific case, we can use *run* instead of *state* because we are assuming that $i < |\pi|$; therefore $i - 1 < |\pi| - 1 \overset{1}{\Rightarrow} \pi^{ST}, map_{i-1} \models_{t^-} run$ (if $i = 0$, then property is false).
- $\varphi_1 S \varphi_2$: The case is specular to U. It is sufficient to observe that 4 can be applied for the past as well.
- $ite(\varphi, u_1, u_2)$: $\pi^{ST}(map_i)(\mathcal{R}^{sgn}(ite(\varphi, u_1, u_2))) = \pi^{ST}(map_i)(\mathcal{R}^-(u_1))$ if $\pi^{ST}, map_i \models_{t^+} \varphi; \dots = \pi^{ST}(map_i)(\mathcal{R}^-(u_2))$ if $\pi^{ST}, map_i \models_{t^+} \neg\varphi; \dots = \pi^{ST}(def_{ite(\varphi,u_1,u_2)})$ otherwise. By induction the case holds.
- $u@\tilde{F}\varphi$: $\pi^{ST}(map_i)(\mathcal{R}^{sgn}(u@\tilde{F}\varphi)) = \pi^{ST}(map_i)(\mathcal{R}^-(u)@\tilde{F}(state \wedge \mathcal{R}^+(\varphi)))$. By the semantics of the at next operator operator, we obtain the following.

  $\pi^{ST}(map_i)(\mathcal{R}^{sgn}(u@\tilde{F}\varphi)) = \pi^{ST}(j)(\mathcal{R}^-(u))$ if there exists $j > map_i$ s.t. $\pi^{ST}, j \models_{t^+} state$ and $\pi^{ST}, j \models_{t^+} \mathcal{R}^+(\varphi)$; $\pi^{ST}(map_i)(\dots) = def_{u@\tilde{F}\varphi}$ otherwise. $\pi^{ST}(map_i)(\dots) \overset{2}{=} \pi^{ST}(map_j)(\mathcal{R}^-(u))$ if $\exists j \geq i$ s.t. $\pi^{ST}, map_j \models_{t^+} \mathcal{R}^+(\varphi); \pi^{ST}(map_i)(\dots) = def_{\dots}$ otherwise. Finally, by induction hypothesis, the case is proved.
- $u@\tilde{P}\varphi$: Identical to $@\tilde{F}$.
- $next(u)$: Identical to case $u@\tilde{F}\top$. □

Lemma 5.2 states that each point in the sequence $map_0, \dots$ of π^{ST} satisfies $\mathcal{R}(\varphi)$ iff the local property satisfies in that point φ; therefore, providing a mapping between the two properties.

Definition 5.3. We define $\mathcal{R}^*$ as $\mathcal{R}^*(\varphi) := stateR(\neg state \vee \mathcal{R}(\varphi))$.

Lemma 5.4. *For all* $\pi^{ST} \in Pr^{-1}(\pi) : \pi^{ST}, map_0 \models_t \mathcal{R}(\varphi) \Leftrightarrow \pi^{ST}, 0 \models_t \mathcal{R}^*(\varphi)$

Proof. Proofs follows simply applying observation 3 of the proof of Lemma 5.2. □

Lemma 5.2 shows that $\mathcal{R}$ guarantees that satisfiability is preserved in the active transitions of the global traces. However, map_0 is not always granted to be equal to 0 (see definition 4.7), and thus, the rewriting must guarantee that satisfiability is preserved in the first transition as well. From lemma 5.2 and lemma 5.4, we infer the rewriting theorem (Theorem 5.5) which shows that $\mathcal{R}^*$ maps the local trace to the global trace as follows:

Theorem 5.5. *For all* $\pi^{ST} \in Pr^{-1}(\pi) : \pi \models_t \varphi \Leftrightarrow \pi^{ST} \models_t \mathcal{R}^*(\varphi)$

Proof. We prove Theorem 5.5 through Lemma 5.2 and Lemma 5.4:

By Lemma 5.2, $\forall i : \pi, i \models_{t^{sgn}} \varphi \Leftrightarrow \pi^{ST}, map_i \models_{t^{sgn}} \mathcal{R}^{sgn}(\varphi)$. Therefore, $\pi \models_{t^-} \varphi \Leftrightarrow \pi^{ST}, map_0 \models_{t^-} \mathcal{R}^-(\varphi)$. By Lemma 5.4, $\pi^{ST}, map_0 \models_{t^{sgn}} \mathcal{R}^-(\varphi) \Leftrightarrow \pi^{ST} \models_{t^{sgn}} \mathcal{R}^*(\varphi)$. Therefore, $\pi \models_{t^-} \varphi \Leftrightarrow \pi^{ST} \models_{t^-} \mathcal{R}^*(\varphi)$. □

Theorem 5.6. *Let φ be a truncated LTL formula:*

(1) *If φ does not contain ite, the size of the rewritten formula with $\mathcal{R}^*$ is linear w.r.t. φ i.e. $|\mathcal{R}^*(\varphi)| = O(|\varphi|)$.*
(2) *If φ contains ite, the size of the rewritten formula is in the worst-case exponentially larger than φ.*

Proof. We prove Theorem 5.6 by first showing that $|\mathcal{R}(\varphi)| = |\varphi| + c$ where c is a constant inductively on the structure of the formula. The base case trivially holds since the $\mathcal{R}^{sgn}(x) = x$ and $\mathcal{R}^{sgn}(c) = c$. We now show the other cases assuming that the theorem holds on the sub-formulae. For brevity, we prove only the weak part of the rewriting; the prove of the strong part is identical since the sizes of the generated formulae are the same.

- $(Pred)$ $|\mathcal{R}^-(Pred^I(u_1,\ldots,u_n))| = |\neg run \vee Pred^I(\mathcal{R}^-(u_1),\ldots,\mathcal{R}^-(u_n))| = 3+\sum_{1\le i\le n}|u_i| + c_i = 2+\sum_{1\le i\le n} c_i + 1 + \sum_{1\le i\le n}|u_i| = 2+\sum_{1\le i\le n} c_i + |Pred^I|(u_1,\ldots,Pred_n)$. Since c_i are constants the rewriting is linear in this case. The proof for output predicate is almost identical.
- $(\vee,\neg)$ Trivial.
- (X) $|\mathcal{R}^-(X\varphi)| = |X(stateR(\neg state \vee \mathcal{R}^-(\varphi)))| = 1+2|state|+2+|\mathcal{R}^-(\varphi)| = 3+|state|+|\varphi|+c$. $|state|$ and c are constants; therefore, the rewriting is still linear.
- (U) $|\mathcal{R}^-(\varphi_1 U \varphi_2)| = |(\neg state \vee \mathcal{R}^-(\varphi_1))U(state \wedge \mathcal{R}^-(\varphi_2))| = 3+2|state|+1+|\varphi_1|+c_1+|\varphi_2|+c_2 = |\varphi_1 U\varphi_2| + c_1 + c_2 + 3 + 2|state|$. $|state|$, c_1 and c_2 are constants; therefore, the rewriting is still linear.
- (S,Y) The proof is respectively as U and X.
- $(func)$ Trivial.
- $(@\tilde{F})$ $|\mathcal{R}^{sgn}(u@\tilde{F}\varphi)| = |\mathcal{R}^{sgn}(u)@\tilde{F}(state \wedge \mathcal{R}^+(\varphi))| = |u|+c_1+1+|state|+1+|\varphi|+c_2 = |u@\tilde{F}\varphi| + c_1 + c_2 + |state| + 1$. Since c_1 (constant of $\mathcal{R}^{sgn}(u)$), c_2 (constant of $\mathcal{R}^{sgn}(\varphi)$) and $state$ are constants; then the rewriting is linear.
- $(@\tilde{P}$, $next)$ Identical to $@\tilde{F}$.

Finally, $|\mathcal{R}^*(\varphi)| = |stateR(\neg state \vee \mathcal{R}^-(\varphi))| = |\mathcal{R}^-(\varphi)| + 2|state| + 3$; since $|\mathcal{R}^-(\varphi)|$ is linear w.r.t $|\varphi|$ we deduce that $|\mathcal{R}(\varphi)|$ is linear as well.

Finally, the size of the rewriting is worst-case exponential when considering ite because $\mathcal{R}^{sgn}(ite(\varphi,u_1,u_2))$ contains, both weakly and strongly $\mathcal{R}(\varphi)$, the rewriting of φ. □

Example 5.7. *Consider the formula $\varphi_{c2} := G(rec_2 \to out_2' = in_2 \wedge X send_2)$ from Figure 1. The rewriting $\mathcal{R}^*(\varphi_{c2})$ is defined as the following formula.*

$$G(\neg state \vee (\underbrace{run \wedge rec_2}_{\mathcal{R}^+(rec_2)} \to \overbrace{(out_2@\tilde{F}state = in_2)}^{\mathcal{R}^-(next(out_2))} \wedge \underbrace{X(runR(\neg run \vee send_2))}_{\mathcal{R}^-(Xsend_2)}))$$

Finally, $\mathcal{R}^*(\varphi_{c2})$ is defined as $stateR(\neg state \vee \mathcal{R}^-(\varphi_{c2})$.

Recall that G is an abbreviation of Until: $G\psi := \neg(\top U \neg\psi)$. Therefore, the rewriting of G is equal to $\neg\mathcal{R}^+(\top U\neg\psi) := \neg((\neg state \vee \top)U(state \wedge \mathcal{R}^+(\neg\psi)))$; we can simplify the left side of until and we obtain $\neg(\top U(state \wedge \mathcal{R}^+(\neg\psi)))$; and, with further simplifications we obtain $\neg F(state \wedge \neg\mathcal{R}^-(\psi)) \equiv G(\neg state \vee \mathcal{R}^-(\psi))$. Intuitively, with the always modality, we are interested in evaluating the states that are local by evaluating as true any state in which the component stutters.

For what regards rec_2, since it is in the left side of an implication, we rewrite it with strong semantics by asking for run to be true. Then, $next(out_2)$ is rewritten via at-next operator; the intuition is that at-next provides the "next" value of the variable out_2 if $state$ will be true, otherwise a default value is given. For what regards next, the intuition is given by Figure 4.

5.2. **Optimized LTL compositional rewriting.**

The main weakness of the rewriting proposed in previous sections is the size of the resulting formula. However, there are several cases in which it is possible to apply a simpler rewriting. For instance, Gv_o is rewritten by $\mathcal{R}^{sgn}$ into $G(\neg state \vee v_o)$ while by ψ_{cond} (see Definition 4.6) it does not need to be rewritten since output variables do not change when run is false. Similarly, Xv_o can be rewritten in the weak semantics to $end \vee Xv_o$.

To do so, we apply the concept of *stutter-tolerance* introduced in [BT22] tailored for possibly finite traces. Informally, a formula is said *stutter-tolerant* if it keeps the same value when rewritten with $\mathcal{R}^{sgn}$ in all adjacent stuttering transitions.

Definition 5.8. An LTL formula φ and a term u are respectively said *stutter-tolerant* w.r.t. $\mathcal{R}^{sgn}$ iff:
For all $\pi,$ for all $\pi^{ST} \in Pr^{-1}(\pi),$ for all $0 \leq i < |\pi|$: for all $map_{i-1} < j < map_i$:

$$\begin{aligned}\pi^{ST}, j \models_{t^{sgn}} \mathcal{R}^{sgn}(\varphi) \Leftrightarrow & \pi^{ST}, map_i \models_{t^{sgn}} \mathcal{R}^{sgn}(\varphi) \text{ and}\\ \pi^{ST}(j)(\mathcal{R}^{sgn}(u)) = & \pi^{ST}(map_i)(\mathcal{R}^{sgn}(u))\end{aligned}$$

Definition 5.9. An LTL formula φ_{st} is *syntactically stutter-tolerant*—abbreviated as synt.st.tol.—iff it has the following grammar:

$$\begin{aligned}\varphi_{st} :=& \varphi_{st} \vee \varphi_{st} \mid \neg\varphi_{st} \mid Pred^O(u, \dots, u) \mid \varphi U \varphi \mid Y\varphi\\ u_{st} :=& func(u_{st}, \dots, u_{st}) \mid s \mid c \mid ite(\varphi_{st}, u_{st}, u_{st}) \mid u@\tilde{P}\varphi\end{aligned}$$

where φ is an LTL formula, u is a term from LTL syntax, s is an output variable and c is a constant.

Lemma 5.10. *Syntactically stutter-tolerant formulas are stutter-tolerant w.r.t.* $\mathcal{R}^{sgn}$

Proof. We prove the Lemma by induction on the size of the formula.

The base case is trivial since output variables remain unchanged during stuttering. The inductive case is proved as follows:

- $\vee, Pred^O$ and $\neg$: Trivial.
- U: We can prove the correctness by induction on j, with base case $j = map_i - 1$. $\pi^{ST}, j \models_{t^-} (\neg state \vee \mathcal{R}^-(\varphi_1))U(state \wedge \mathcal{R}^-(\varphi_2) \vee end) \Leftrightarrow \pi^{ST}, j \models_{t^-} (\mathcal{R}^-(\varphi_2) \wedge state \vee end) \vee (\neg state \vee \mathcal{R}^-(\varphi_1)) \wedge X((\neg state \vee \mathcal{R}^-(\varphi_1)U(state \wedge \mathcal{R}^-(\varphi_2))) \overset{\pi^{ST}, j \not\models_t state}{\Leftrightarrow} \pi^{ST}, j \models_{t^-} X((\mathcal{R}^-(\varphi_1) \vee \neg state)U(state \wedge \mathcal{R}^-(\varphi_2) \vee end) \overset{j<|\pi^{ST}|-1}{\Leftrightarrow} \pi^{ST}, j+1 \models_{t^-} \mathcal{R}^-(\varphi_1 U \varphi_2)$ which is map_i for the base case. The inductive case follows trivially.
- Y: The case of Y can be prove in the same way of U by expanding S instead of U.
- $@\tilde{P}$: The semantics of at last evaluates $\mathcal{R}^+(\varphi)$ at the first occurrence in the past of *state*. The proof is the same as Y. □

From Lemma 5.10, we derive a syntactical way to determine identify a relevant fragment of stutter tolerant formulae. Since this definition is purely syntactical, it is very simple for an algorithm to determine whether or not a formula is syntactically stutter tolerant; it is sufficient to traverse the structure of the formula and look at the variables and operators.

From the notion of syntactically stutter tolerant formula, we provide an optimized rewriting that is semantically equivalent to $\mathcal{R}^{sgn}$. If the sub-formulas of φ are syntactically stutter tolerant, we can simplify the rewriting for φ.

Definition 5.11. We define $\mathcal{R}^\theta$ as follows. We omit the cases that are identical to $\mathcal{R}$.

$\mathcal{R}^{\theta^-}(X\varphi) := end \vee X\mathcal{R}^{\theta^-}(\varphi)$ If φ is syntactically stutter-tolerant

$\mathcal{R}^{\theta^+}(X\varphi) := \neg end \wedge X\mathcal{R}^{\theta^+}(\varphi)$ If φ is syntactically stutter-tolerant

$\mathcal{R}^{\theta^-}(\varphi_1 U \varphi_2) := \mathcal{R}^{\theta^-}(\varphi_1)U(Yend \vee \mathcal{R}^{\theta^-}(\varphi_2))$ If φ_1, φ_2 are syntactically stutter-tolerant

$\mathcal{R}^{\theta^+}(\varphi_1 U \varphi_2) := \mathcal{R}^{\theta^+}(\varphi_1)U(\neg Yend \wedge \mathcal{R}^{\theta^+}(\varphi_2))$ If φ_1, φ_2 are syntactically stutter-tolerant

$\mathcal{R}^{\theta sgn}(u@\tilde{F}\varphi) := \mathcal{R}^{\theta sgn}(u)@\tilde{F}(\mathcal{R}^{\theta sgn}(\varphi) \wedge \neg end)$ If φ is syntactically stutter-tolerant

where $\mathcal{R}^\theta(\varphi) := \mathcal{R}^{\theta^-}(\varphi)$.

Lemma 5.12. *For all π, for all $\pi^{ST} \in Pr^{-1}(\pi)$, for all $i < |\pi|$:*

$$\pi, i \models_{t^{sgn}} \varphi \Leftrightarrow \pi^{ST}, map_i \models_{t^{sgn}} \mathcal{R}^\theta(\varphi) \qquad \pi(i)(u) = \pi^{ST}(map_i)(\mathcal{R}^\theta(u))$$

Proof. From Lemma 5.2, we deduce that to prove the Lemma it suffices to prove that $\forall_{0\leq i<|\pi|}\pi^{ST}, map_i \models_{t^{sgn}} \mathcal{R}(\varphi) \Leftrightarrow \pi^{ST}, map_i \models_{t^{sgn}} \mathcal{R}^\theta(\varphi)$ and $\pi^{ST}(map_i)(\mathcal{R}^{sgn}(u)) = \pi^{ST}(map_i)(\mathcal{R}^{\theta sgn}(u))$. To prove that, we also prove inductively that if φ is syntactically stutter-tolerant, then $\forall_{0\leq i<|\pi|}\forall_{map_{i-1}<j<map_i}\pi^{ST}, j \models_{t^{sgn}} \mathcal{R}^{\theta sgn}(\varphi) \Leftrightarrow \pi^{ST}, map_i\mathcal{R}^{sgn}(\varphi)$.

We need to prove only the cases in which the sub-formulae are stutter tolerant w.r.t $\mathcal{R}^*$.

- X: If end is true, then $i >= |\pi| - 1$. Therefore, with weak semantics, the formula shall be true while with strong semantics the formula shall be false.

  If end is false, then $\pi^{ST}, map_i \models_{t^{sgn}} \mathcal{R}^{\theta sgn}(X\varphi) \Leftrightarrow \pi^{ST}, map_i \models_{t^{sgn}} X\mathcal{R}^{\theta sgn}(\varphi) \Leftrightarrow \pi^{ST}, map_i + 1 \models_{t^{sgn}} \mathcal{R}^{\theta sgn}(\varphi)$. By induction hypothesis $\cdots \Leftrightarrow \pi^{ST}, map_i + 1 \models_{t^{sgn}} \mathcal{R}(\varphi) \Leftrightarrow \pi^{ST}, map_{i+1} \models_{t^{sgn}} \mathcal{R}(\varphi)$.
- U: $\pi^{ST}, map_i \models_{t^-} \mathcal{R}^{\theta^-}(\varphi_1)U(Yend \vee \mathcal{R}^{\theta^-}(\varphi_2)) \Leftrightarrow \exists k' \geq map_i$ s.t. $\pi^{ST}, k' \models_{t^-} Yend$ or $\pi^{ST}, k' \models_{t^-} \mathcal{R}^{\theta^-}(\varphi_2)$ and $\forall_{map_i\leq j'<k'}\pi^{ST}, j' \models_{t^-} \mathcal{R}^{\theta^-}(\varphi_1)$. By induction hypothesis $\exists map_k \geq map_i$ s.t. $\pi^{ST}, map_k \models_{t^-} \mathcal{R}(\varphi_2)$ and $\forall_{map_i\leq map_j<map_k}\pi^{ST}, map_j \models_{t^-} \mathcal{R}(\varphi_1)$.

  We now prove the additional part of the Lemma. If φ_1 or φ_2 are not syntactically stutter tolerant, then the rewriting is identical and thus, the lemma holds since φ is stutter tolerant.

  If φ_1 and φ_2 are syntactic stutter tolerant, $\pi^{ST}, j \models_{t^-} \mathcal{R}^{\theta^-}(\varphi_1)U(Yend \vee \mathcal{R}^{\theta^-}(\varphi_2)) \Leftrightarrow \exists k' \geq map_i$ s.t. $\pi^{ST}, k' \models_{t^-} Yend$ or $\pi^{ST}, k' \models_{t^-} \mathcal{R}^{\theta^-}(\varphi_2)$ and $\forall_{map_i\leq j'<k'}\pi^{ST}, j' \models_{t^-} \mathcal{R}^{\theta^-}(\varphi_1)$. We observe that $\pi^{ST}, k' \models_{t^-} Yend \Leftrightarrow k' > map_{|\pi|-1}$. Moreover, by induction hypothesis, each j' and each k' can be replaced with respectively map_j and map_k s.t. $i \leq k \leq |\pi|$ and $i \leq j < k$. Therefore, $\cdots \Leftrightarrow \exists k \geq i$ s.t. $\pi^{ST}, map_k \models_{t^-} \mathcal{R}(\varphi_2)$ or $k = |\pi|$ and $\forall i \leq j < k : \pi^{ST}, map_j \models_{t^-} \mathcal{R}(\varphi_1) \Leftrightarrow \pi, i \models_{t^-} \varphi_1 U \varphi_2$.
- $@\tilde{F}$: It follows from induction hypothesis and by Lemma 5.2 □

Definition 5.13. We define $\mathcal{R}^{\theta^*}$ as follows:

$$\mathcal{R}^{\theta^*}(\varphi) := \begin{cases} \mathcal{R}^{\theta^-}(\varphi) & \text{If } \varphi \text{ is syntactically stutter-tolerant} \\ stateR(\neg state \vee \mathcal{R}^{\theta^-}(\varphi)) & \text{Otherwise} \end{cases}$$

Lemma 5.14. *For all $\pi^{ST} \in Pr^{-1}(\pi) : \pi^{ST}, map_0 \models_t \mathcal{R}^\theta(\varphi) \Leftrightarrow \pi^{ST}, 0 \models_t \mathcal{R}^{\theta^*}(\varphi)$*

Proof. If φ is not syntactically stutter tolerant, then the proof is the same one of lemma 5.4. If φ is syntactically stutter tolerant, then $\forall_{map_{-1}<j<map_i}\pi^{ST}, j \models_{t^-} \mathcal{R}^{\theta^-}(\varphi) \Leftrightarrow \pi^{ST}, map_0 \models_{t^-}$

$\mathcal{R}^{\theta^-}(\varphi)$. Since map_{-1} is -1, then either $map_0 = 0$ or j gets the value 0 in the for all; thus, proving the lemma. □

Theorem 5.15. *For all $\pi^{ST} \in Pr^{-1}(\pi) : \pi \models_t \varphi \Leftrightarrow \pi^{ST} \models_t \mathcal{R}^{\theta^*}(\varphi)$*

Proof. The proof is identical to the proof of Theorem 5.5 using Lemma 5.12 and Lemma 5.14. □

Theorem 5.5 and Theorem 5.15 show that respectively $\mathcal{R}^*$ and $\mathcal{R}^{\theta^*}$ are able to translate a local LTL property into a global property without changing its semantics in terms of traces. Therefore, we can use the two rewritings to prove Inference 4.1.

Definition 5.16. Let $\mathcal{M}_1, \dots, \mathcal{M}_n$ be n ITS and $\varphi_1, \dots, \varphi_n$ be LTL formulas on the language of each $\mathcal{M}_i$. We define γ_P as follows

$$\gamma_P(\varphi_1, \dots, \varphi_n) := \mathcal{R}^{\theta^*}_{\mathcal{M}_1}(\varphi_1) \wedge \dots \wedge \mathcal{R}^{\theta^*}_{\mathcal{M}_n}(\varphi_n) \wedge \psi_{cond}$$

where $\psi_{cond} := \psi^{\mathcal{M}_1}_{cond} \wedge \dots \wedge \psi^{\mathcal{M}_n}_{cond}$

Corollary 5.17. *Using γ_P from Definition 5.16, γ_S from Section 4.1, for all compatible ITS $\mathcal{M}_1, \dots, \mathcal{M}_n$, for all local properties $\varphi_1, \dots, \varphi_n$ over the language of respectively $\mathcal{M}_1, \dots, \mathcal{M}_n$, for all global properties φ: Inference 4.1 holds.*

Example 5.18. *Consider the formula $\varphi_{c2} := G(rec_2 \rightarrow out'_2 = in_2 \wedge X send_2)$ from Figure 1. The rewriting $\mathcal{R}^\theta(\varphi_{c2})$ is defined as the following formula.*

$$G(\neg state \vee (\underbrace{run \wedge rec_2}_{\mathcal{R}^{\theta^+}(rec_2)} \rightarrow \underbrace{(end \vee \overbrace{out_2@\tilde{F}\neg end}^{\mathcal{R}^\theta(out_2@\tilde{F}\top)} = in_2)}_{\mathcal{R}^\theta(out'_2=in_2)} \wedge \underbrace{end \vee X send_2}_{\mathcal{R}^{\theta^-}(X send_2)}))$$

Finally, $\mathcal{R}^{\theta^*}(\varphi_{c2})$ is defined as $\mathcal{R}^\theta(\varphi_{c2})$.

The simplification optimizes the formula in various parts. $X send_2$ is rewritten into a simpler formula, in which we only need to check whether we are at the end of the local trace; the same occurs for primed output variable (next). On the contrary, G must be rewritten as for $\mathcal{R}$ because its sub-formula is not stutter-tolerant. On the other hand, the top-level rewriting is simplified because G is syntactically stutter tolerant.

5.3. **Rewriting under fairness assumption.** This section defines a variation of the previous rewriting that assumes infinite execution of local components. The rewriting is presented as a variation of the optimized rewriting; it is meant to exploit the fairness assumption to be more concise and efficient. The general idea is that since the local components run infinitely often, we can consider the semantics of LTL with event-freezing functions instead of the finite semantics considered up to now.

Definition 5.19. We define $\mathcal{R}^{\mathcal{F}}$ as follows:

$\mathcal{R}^{\mathcal{F}}(v) := v$ for $v \in V$

$\mathcal{R}^{\mathcal{F}}(Pred(u_1, \ldots, u_n)) := Pred(\mathcal{R}^{\mathcal{F}}(u_1), \ldots, \mathcal{R}^{\mathcal{F}}(u_n))$

$\mathcal{R}^{\mathcal{F}}(\varphi \vee \psi) := \mathcal{R}^{\mathcal{F}}(\varphi) \vee \mathcal{R}^{\mathcal{F}}(\psi)$

$\mathcal{R}^{\mathcal{F}}(\neg\varphi) = \neg\mathcal{R}^{\mathcal{F}}(\varphi)$

$$\mathcal{R}^{\mathcal{F}}(X\psi) := \begin{cases} X(\mathcal{R}^{\mathcal{F}}(\psi)) & \text{if } \psi \text{ is synt.st.tol.} \\ X(runR(\neg run \vee \mathcal{R}^{\mathcal{F}}(\psi))) & \text{otherwise} \end{cases}$$

$$\mathcal{R}^{\mathcal{F}}(\varphi_1 U \varphi_2) := \begin{cases} \mathcal{R}^{\mathcal{F}}(\varphi_1) U \mathcal{R}^{\mathcal{F}}(\varphi_2) & \text{if } \varphi_1 \text{ and } \varphi_2 \text{ are synt.st.tol.} \\ (\neg run \vee \mathcal{R}^{\mathcal{F}}(\varphi_1)) U (run \wedge \mathcal{R}^{\mathcal{F}}(\varphi_2)) & \text{otherwise} \end{cases}$$

$\mathcal{R}^{\mathcal{F}}(Y\varphi) := Y(\neg run S(run \wedge \mathcal{R}^{\mathcal{F}}(\varphi)))$

$$\mathcal{R}^{\mathcal{F}}(\varphi_1 S \varphi_2) := \begin{cases} \mathcal{R}^{\mathcal{F}}(\varphi_1) S \mathcal{R}^{\mathcal{F}}(\varphi_2) & \text{if } \varphi_1 \text{ and } \varphi_2 \text{ are synt.st.tol.} \\ (\neg run \vee \mathcal{R}^{\mathcal{F}}(\varphi_1)) S (run \wedge \mathcal{R}^{\mathcal{F}}(\varphi_2)) & \text{otherwise} \end{cases}$$

$\mathcal{R}^{\mathcal{F}}(ite(\psi, u_1, u_2)) := ite(\mathcal{R}^{\mathcal{F}}(\psi), \mathcal{R}^{\mathcal{F}}(u_1), \mathcal{R}^{\mathcal{F}}(u_2))$

$$\mathcal{R}^{\mathcal{F}}(u@\tilde{F}\varphi) := \begin{cases} \mathcal{R}^{\mathcal{F}}(u)@\tilde{F}\mathcal{R}^{\mathcal{F}}(\varphi) & \text{if } \psi \text{ is synt.st.tol.} \\ \mathcal{R}^{\mathcal{F}}(u)@\tilde{F}(run \wedge \mathcal{R}^{\mathcal{F}}(\varphi)) & \text{otherwise} \end{cases}$$

$\mathcal{R}^{\mathcal{F}}(u@\tilde{P}\varphi) := \mathcal{R}^{\mathcal{F}}(u)@\tilde{P}(run \wedge \mathcal{R}^{\mathcal{F}}(\varphi))$

Moreover, we define $\mathcal{R}^{\mathcal{F}^*}$ as

$$\mathcal{R}^{\mathcal{F}^*}(\varphi) := \begin{cases} \mathcal{R}^{\mathcal{F}}(\varphi) & \text{If } \varphi \text{ is syntactically stutter-tolerant} \\ runR(\neg run \vee \mathcal{R}^{\mathcal{F}}(\varphi)) & \text{Otherwise} \end{cases}$$

$\mathcal{R}^{\mathcal{F}^*}$ simplifies $\mathcal{R}^{\theta^*}$ in various ways. It does not need to distinguish between input and output variables, it does not distinguish between weak/strong semantics and it does not distinguish between *state* and *run*. Intuitively, these distinctions make sense only with finite semantics; since the rewriting assumes infinite local executions, these technicalities become superfluous.

Theorem 5.20. *Let π be an infinite trace, for all $\pi^{ST} \in Pr^{-1}(\pi) : \pi \models \varphi \Leftrightarrow \pi^{ST} \models \mathcal{R}^{\mathcal{F}^*}(\varphi)$*

Proof. (Sketch) Since π is infinite, *end* is always false, *state* $\Leftrightarrow$ *run*. We can substitute *end* with $\perp$ and *state* with *run* in $\mathcal{R}^{\theta}$ and we obtain this rewriting. □

Example 5.21. *Consider the formula $\varphi_{c2} := G(rec_2 \rightarrow out'_2 = in_2 \wedge Xsend_2)$ from Figure 1. The rewriting $\mathcal{R}^{\mathcal{F}}(\varphi_{c2})$ is defined as the following formula.*

$$G(\neg run \vee (\underbrace{rec_2}_{\mathcal{R}^{\mathcal{F}}(rec_2)} \rightarrow \underbrace{(\overbrace{out'_2}^{\mathcal{R}^{\mathcal{F}}(out'_2)} = in_2)}_{\mathcal{R}^{\mathcal{F}}(out'_2=in_2)} \wedge \underbrace{Xsend_2}_{\mathcal{R}^{\mathcal{F}}(Xsend_2)}))$$

Finally, $\mathcal{R}^{\mathcal{F}^*}(\varphi_{c2})$ is defined as $\mathcal{R}^{\mathcal{F}}(\varphi_{c2})$.

By assuming infinite executions of local traces it is possible to drastically simplify the rewriting. Next outputs are left unchanged because they are stutter tolerant and with infinite execution end is not needed. Input variables are also transparent to the rewriting because *the infinite semantics do not distinguish between input and output variables. As before, G must be rewritten because its sub-formula is not stutter-tolerant; however, in this case, the rewriting can use run instead of state because the two expressions are equivalent with infinite semantics. Finally, as before, the top-level rewriting is simplified because G is syntactically stutter tolerant.*

6. Experimental evaluation

We implemented the compositional techniques proposed in this paper inside the contract-based design tool OCRA [CDT13]. Our extension covers the contract refinement check, in which a contract (defined as a couple $\langle A, G\rangle$ of LTL formulae) is considered correct if its sub-component contracts refine it. For simplicity, we consider contracts without assumptions i.e. in which the assumption is $\top$; in this scenario, the refinement of contracts is given by the compositional reasoning described in this paper. For a detailed description of the assume-guarantee contract proof system employed in the tool refer to [CT15b].

The objective of this experimental evaluation is to do both a quantitative and a qualitative evaluation of our compositional approach differentiating possibly finite and infinite semantics. Qualitatively, we compare the verification results to assess how the finiteness impacts on the result, observing also the required scheduling assumptions for possibly finite systems. Quantitatively, we analyse the overhead of reasoning over a mix of finite and infinite executions to assess whether or not our rewriting scales on real models. Due to the absence of equally expressible formalism to define asynchronous composition, our comparison with related work is limited to another rewriting suited only for local infinite systems with event-based asynchronous composition[BBC+09].

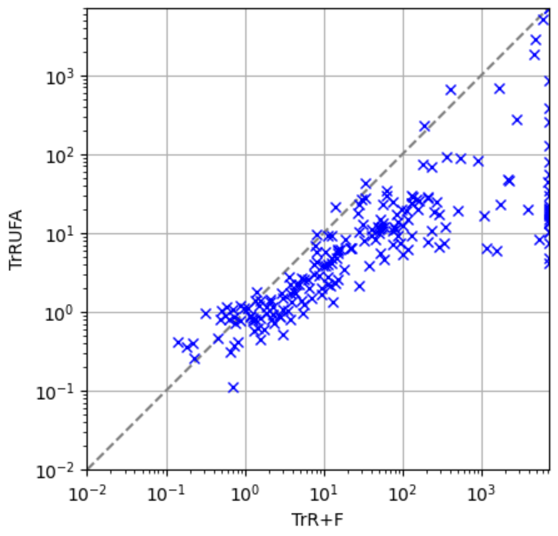
Although we defined compositional reasoning with truncated semantics, in our experiments we reason about global infinite executions and possibly finite local executions. The experiments [BT26] were run in parallel on a cluster with nodes with Intel Xeon CPU 6226R running at 2.9GHz with 32CPU, 12GB. The timeout for each run was two hours and the memory cap was set to 2GB.

In this section, we denote the composition based on $\mathcal{R}^{\theta^*}$ as TrR(Truncated Rewriting); we denote the composition based on $\mathcal{R}^{\theta^*}$ with additional constraints to ensure infinite local executions as TrR+F (Truncated Rewriting + Fairness); we denote the composition based on $\mathcal{R}^{\mathcal{F}^*}$ assuming local infinite execution as TrRuFA (Truncated Rewriting under Fairness Assumption).

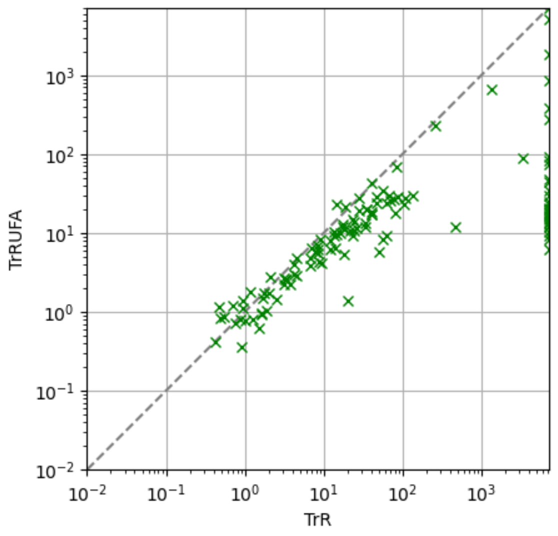
6.1. Benchmarks.

We have considered benchmarks of various kinds:

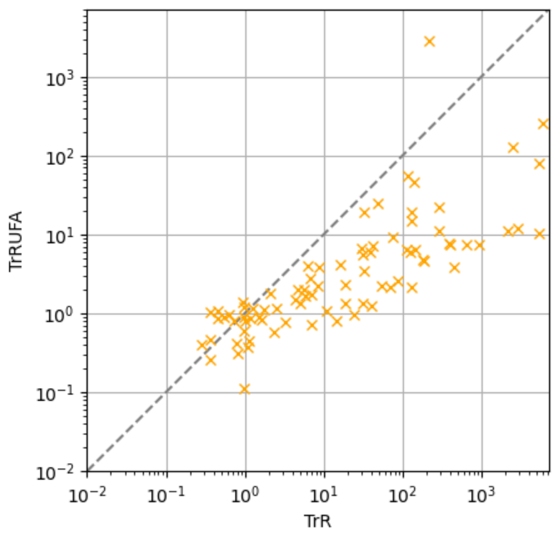
1. Simple asynchronous models from OCRA comprehending the example depicted in Figure 1.
2. Pattern formula compositions from [BT22] experimental evaluation
3. Contracts from the experimental evaluation of [CCG+23] on AUTOSAR models adapted for truncated semantics.



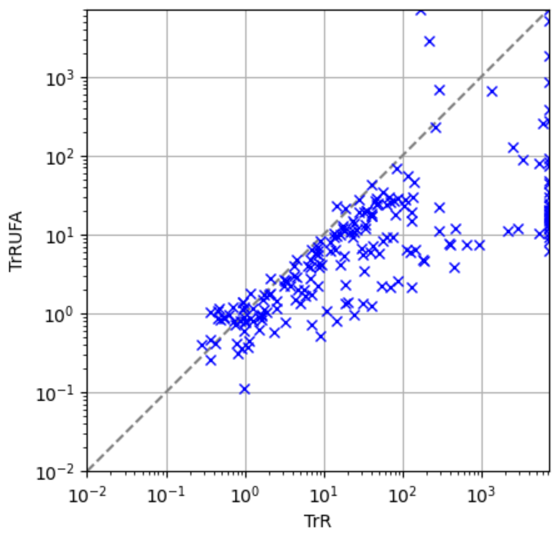
(a) Comparison between TrR+F and TrRuFA over valid instances.



(b) Comparison between TrR and TrRuFA over valid instances.



(c) Comparison between TrR and TrRuFA over "different" results.



(d) Comparison between TrR and TrRuFA over all the instances.

Figure 6. Scatter plots comparing TrR,TrR+F and TrRuFA

6.1.1. *Simple asynchronous models.* We considered two variations of the example model depicted in Figure 1: one version considers all three components with both the possible outcome of failure and success. The second version is composed only of components $c1$ and $c2$ and the global property states that eventually the message is sent.

The second model we are considering is a simple representation of a Wheel Brake System with two Braking System Control Units (BSCU) connected to two input braking pedals, a Selection Switch and an actual hydraulic component that performs the actual brake. The

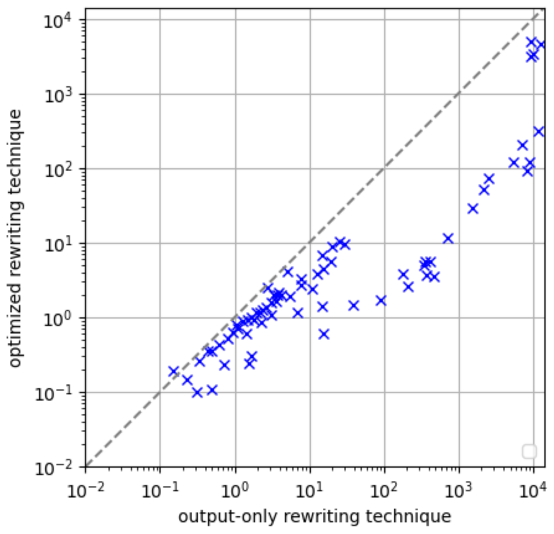


FIGURE 7. Comparison between TrRuFA and event-based rewriting of [BBC+09]

Model	Algorithm	Result	Time (s)
Response boolean sequence with size=3	TrR	Invalid	0.74
Response boolean sequence with size=3	TrRuFA	Valid	0.79
Response boolean sequence with size=3	TrR+F	Valid	0.48
Simplified Example (no component c_3)	TrR	Invalid	1.15
Simplified Example (no component c_3)	TrRuFA	Valid	0.87
Simplified Example (no component c_3)	TrR+F	Valid	2.82
Example (Figure 1)	TrR	Valid	20.13
Example (Figure 1)	TrR+F	Valid	5.3
Example (Figure 1)	TrRuFA	Valid	1.37
Response event sequence with size=12	TrR	Invalid	14.61
Response event sequence with size=12	TrRuFA	Valid	0.8
Response event sequence with size=12	TrR+F	Valid	3.74
Universality sequence with size=17	TrR	Invalid	6.63
Universality sequence with size=17	TrRuFA	Valid	2.81
Universality sequence with size=17	TrR+F	Valid	11.17
"Brake when AEB status is active" bugged	TrR	Invalid	48.3
"Brake when AEB status is active" bugged	TrRuFA	Valid	24.63
"Brake when AEB status is active" bugged	TrR+F	Valid	74.72
"Brake when AEB status is active" fixed	TrR	Valid	136.34
"Brake when AEB status is active" fixed	TrRuFA	Valid	17.6
"Brake when AEB status is active" fixed	TrR+F	Valid	62.83

TABLE 1. Subset of quantitative and qualitative results of the experimental evaluation.

Algorithm	N. ALL	N. VAL	N. INV	N. UNK	< 1 sec	< 60 sec	< 600 sec
ALL	624	476	88	60	82	448	535
TrR	208	91	86	31	26	133	167
TrR+F	208	182	1	25	21	126	168
TrRUFA	208	203	1	4	35	189	200

Table 2. Summary result for algorithms

top-level property states that if one of the two pedals is pressed eventually the hydraulic component will brake.

6.1.2. *Pattern models.* We took from [BT22] some benchmarks based on Dwyer LTL patterns [DAC70]. The considered LTL patterns are the following: *response*, *precedence chain* and *universality* patterns. The models compose the pattern formulas in two ways: as a sequence of n components linked in a bus and as a set of components that tries to write on the output port. These patterns are parametrized on the number of components involved in the composition.

6.1.3. *EVA AUTOSAR contracts.* We took the experimental evaluation from the tool EVA[CCG$^+$23] and we adapted it for our compositional reasoning. The models are composed of various components for which the scheduling is of two types: event-based and cyclic. Event-based scheduling forces the execution of a component when some of its input variables change while cyclic scheduling forces the execution of the component every n time units.

In addition to these models, we also considered a variation of a subset of instances (described in Section 1.1.2) and we relaxed the scheduling constraints allowing some components to become unresponsive at some point to the original constraints. We forced an event-based scheduled component (Brake Actuator) to be scheduled when its input changes but only until it ends; moreover, we forced a cyclic scheduled component (Brake Watchdog) to run every n times units until it ends. Finally, assuming that one of the two components runs infinitely often (i.e. does not become unresponsive), we check if the composition still holds.

6.2. **Experimental evaluation results.** Figure 6 provide an overall comparison between TrR, TrR+F and TrRuFA in terms of results and execution time. In these plots, the colour determines the validity results: blue aggregates all the results without distinction between valid and invalid; green considers valid instances of both techniques (if the other algorithm returns valid, timeout is optimistically believed to be true); yellow consider instances in which the two techniques had different results.

Overall, we checked 624 instances (208 for each rewriting); 406 of these instances were proven valid, 88 were proven invalid and 60 instances timed out. The general statistics can be found in Table 2.

Table 1 shows some relevant instances of the experimental evaluation highlighting their execution time and their validity results.

Qualitative results. As we expected, there are differences in validity results between TrR and TrRuFA. The pattern models cannot guarantee the validity of the composition with the weak semantics without additional constraints over the composition. Intuitively, it suffices that a single component is not scheduled to violate bounded response properties.

Similarly, we compared the validity results of the 3 rewritings on a simplified version of the example of Section 1.1.1; this variation of the model contains only components c_1 and c_2. The simplified version was proved valid using TrRuFA and TrR+F while a counterexample was found using TrR. The simplified version is found invalid when c_1 is scheduled only a finite amount of times failing to deliver the message to component c_2. On the other hand, when we correct the model by adding the component c_3, the composition is proved valid in all the 3 cases.

For what regards the EVA benchmarks, TrR and TrRuFA gave the same validity results. That occurred because the scheduling constraints were quite strong; in particular, the cyclic constraints implicitly forced components, such as the watchdog, to run infinitely often. The event-based scheduled component could have a finite execution but only if their input did not change from some point on. Differently, updating the constraints by removing these implicit infinite executions makes the property invalid using TrR. To fix these invalid results, we updated the system assuming that at least one of the two components runs infinitely often and we relaxed the global property increasing the timed bound for the brake to occur from 2 time units to 3. Finally, the corrected model was proved in all the 3 cases.

Quantitative results. We provided a comparison between $\mathcal{R}^{\theta}$ and $\mathcal{R}^{\mathcal{F}}$ in term of impact of the transformation on the verification time. Since $\mathcal{R}^{\mathcal{F}}$ assumes that each local components is executed infinitely often, we used the rewriting $\mathcal{R}^{\theta}$ equipped with fairness assumption (TrR+F). Figure 6a shows the comparison between TrR+F and TrRuFA. In this case, TrRuFA is more efficient than TrR+F. This is not surprising because the generated formula assuming infinite local execution can be significantly smaller than the general one. In general, we see that, regardless of the result, TrRuFA can prove the property faster. We think that having to check all the possible combinations of finite/infinite local execution provides a significant overhead in the verification.

In figure 7, we compared TrRuFA with another rewriting for the composition of temporal properties from [BBC$^+$09]. The comparison has been done over a subset of the *pattern* models. The rewriting of [BBC$^+$09] is in principle similar to TrRuFA; it assumes infinite executions of local component but, contrary to our approaches, in [BBC$^+$09] the asynchronous composition is supported only considering shared synchronization events[3]. Due to the expressive limitations of the other approach, we applied the evaluation over a smaller set of instances. It should be noted that the technique of [BBC$^+$09] already introduces fairness assumptions of the infinite execution of local components; therefore, we don't need to manipulate/complicate that rewriting in this evaluation. It is clear from the figure that TrRuFA outperforms this other rewriting.

[3]These events are basically Boolean variables shared between two components. When one of these variable becomes true, both the components run. Although it is not detailed in the paper, we do support these events natively by adding additional scheduling constraints in α

7. Conclusions

In this paper, we considered the problem of compositional reasoning for asynchronous systems with LTL properties over input and output variables in which local components are not assumed to run infinitely often. We introduced a semantics based on the truncated semantics of Eisner and Fisman for LTL to reason over finite executions of local components. We proposed a new rewriting of LTL formulas that allows for checking compositional rules with temporal satisfiability solvers. We then provided an optimized version of such rewriting.

In the future, we will consider various directions for extending the framework including real-time and hybrid specifications, optimizations based on the scheduling of components and other communication mechanisms such as buffered communication. Moreover, we will generalise our compositional reasoning for assume/guarantee contracts based on [CT15b], for which we hypothesise that assumptions must be treated with strong semantics.

References

[BBC+09] Nikola Benes, Lubos Brim, Ivana Cerná, Jirí Sochor, Pavlína Vareková, and Barbora Buhnova. Partial Order Reduction for State/Event LTL. In *IFM*, 2009.

[BCB+21] Jan Baumeister, Norine Coenen, Borzoo Bonakdarpour, Bernd Finkbeiner, and César Sánchez. A Temporal Logic for Asynchronous Hyperproperties. In *CAV (1)*, volume 12759 of *Lecture Notes in Computer Science*, pages 694–717. Springer, 2021.

[BCGT25] Alberto Bombardelli, Alessandro Cimatti, Alberto Griggio, and Stefano Tonetta. *Another Look at LTL Modulo Theory over Finite and Infinite Traces*, pages 419–443. Springer Nature Switzerland, Cham, 2025. doi:10.1007/978-3-031-75783-9_17.

[BCL+10] Lei Bu, Alessandro Cimatti, Xuandong Li, Sergio Mover, and Stefano Tonetta. Model Checking of Hybrid Systems Using Shallow Synchronization. In John Hatcliff and Elena Zucca, editors, *Formal Techniques for Distributed Systems*, pages 155–169, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. doi:10.1007/978-3-642-13464-7_13.

[BCMT14] Marco Bozzano, Alessandro Cimatti, Cristian Mattarei, and Stefano Tonetta. Formal safety assessment via contract-based design. In Franck Cassez and Jean-François Raskin, editors, *Automated Technology for Verification and Analysis*, pages 81–97, Cham, 2014. Springer International Publishing.

[BCTZ23] Alberto Bombardelli, Alessandro Cimatti, Stefano Tonetta, and Marco Zamboni. Symbolic Model Checking Of Relative Safety LTL Properties. In *IFM 2023: 18th International Conference, IFM 2023, Leiden, The Netherlands, November 13-15, 2023, Proceedings*, pages 302–320, Berlin, Heidelberg, 2023. Springer-Verlag. doi:10.1007/978-3-031-47705-8_16.

[BSST09] Clark Barrett, Roberto Sebastiani, Sanjit A Seshia, and Cesare Tinelli. Satisfiability Modulo Theories. In *Handbook of Satisfiability*, pages 825–885. IOS Press, January 2009. doi:10.3233/978-1-58603-929-5-825.

[BT22] Alberto Bombardelli and Stefano Tonetta. Asynchronous Composition of Local Interface LTL Properties. In *NFM*, pages 508–526, 2022.

[BT26] Alberto Bombardelli and Stefano Tonetta. Asynchronous Composition of LTL Properties over Infinite and Finite Traces - Experimental Evaluation, 2026. doi:10.5281/zenodo.18171622.

[CCD+14] Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. The nuXmv Symbolic Model Checker. volume 8559, pages 334–342, 07 2014. doi:10.1007/978-3-319-08867-9_22.

[CCG+23] Alessandro Cimatti, Luca Cristoforetti, Alberto Griggio, Stefano Tonetta, Sara Corfini, Marco Di Natale, and Florian Barrau. Eva: a tool for the compositional verification of autosar models. In Sriram Sankaranarayanan and Natasha Sharygina, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 3–10, Cham, 2023. Springer Nature Switzerland.

[CDT13] Alessandro Cimatti, Michele Dorigatti, and Stefano Tonetta. OCRA: A tool for checking the refinement of temporal contracts. pages 702–705, 11 2013. doi:10.1109/ASE.2013.6693137.

[CGH94] Edmund M. Clarke, Orna Grumberg, and Kiyoharu Hamaguchi. Another Look at LTL Model Checking. *Formal Methods in System Design*, 10:47–71, 1994.

[CGM$^+$19] Alessandro Cimatti, Alberto Griggio, Enrico Magnago, Marco Roveri, and Stefano Tonetta. SMT-based satisfiability of first-order LTL with event freezing functions and metric operators. *Information and Computation*, 272:104502, 12 2019. doi:10.1016/j.ic.2019.104502.

[CMT11] Alessandro Cimatti, Sergio Mover, and Stefano Tonetta. HyDI: A Language for Symbolic Hybrid Systems with Discrete Interaction. pages 275–278, 08 2011.

[CT15a] Alessandro Cimatti and Stefano Tonetta. Contracts-refinement proof system for component-based embedded systems. *Science of Computer Programming*, 97:333–348, 2015. Object-Oriented Programming and Systems (OOPS 2010) Modeling and Analysis of Compositional Software (papers from EUROMICRO SEAA 12). URL: https://www.sciencedirect.com/science/article/pii/S0167642314002901, doi:10.1016/j.scico.2014.06.011.

[CT15b] Alessandro Cimatti and Stefano Tonetta. Contracts-refinement proof system for component-based embedded systems. *Science of Computer Programming*, 97:333–348, 2015. URL: https://www.sciencedirect.com/science/article/pii/S0167642314002901, doi:10.1016/j.scico.2014.06.011.

[DAC70] Matthew Dwyer, George Avrunin, and James Corbett. Patterns in Property Specifications for Finite-State Verification. *Proceedings - International Conference on Software Engineering*, 02 1970. doi:10.1145/302405.302672.

[dAH01] Luca de Alfaro and Thomas A. Henzinger. Interface automata. In *ESEC / SIGSOFT FSE*, pages 109–120. ACM, 2001.

[DGV13] Giuseppe De Giacomo and Moshe Y. Vardi. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13. AAAI Press, 2013.

[EFH$^+$03a] Cindy Eisner, Dana Fisman, John Havlicek, Yoad Lustig, Anthony McIsaac, and David Van Campenhout. Reasoning with Temporal Logic on Truncated Paths. In *International Conference on Computer Aided Verification*, 2003. URL: https://api.semanticscholar.org/CorpusID:9153840.

[EFH$^+$03b] Cindy Eisner, Dana Fisman, John Havlicek, Anthony McIsaac, and David Van Campenhout. The Definition of a Temporal Clock Operator. In *ICALP*, volume 2719 of *Lecture Notes in Computer Science*, pages 857–870. Springer, 2003.

[FK18] Dana Fisman and Hillel Kugler. *Temporal Reasoning on Incomplete Paths: 8th International Symposium, ISoLA 2018, Limassol, Cyprus, November 5-9, 2018, Proceedings, Part II*, pages 28–52. 11 2018. doi:10.1007/978-3-030-03421-4_3.

[JT96] B. Jonsson and Yih-Kuen Tsay. Assumption/Guarantee Specifications in Linear-Time Temporal Logic. *Theor. Comput. Sci.*, 167:47–72, 1996.

[KV01] Orna Kupferman and Moshe Y Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001.

[Lam94] Leslie Lamport. Temporal logic of actions. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16:872–923, 05 1994. doi:10.1145/177492.177726.

[Lam97] Leslie Lamport. The Operators of TLA. 06 1997.

[Lat03] Timo Latvala. Efficient Model Checking of Safety Properties. In *SPIN*, volume 2648 of *Lecture Notes in Computer Science*, pages 74–88. Springer, 2003.

[LBA$^+$22] Cong Liu, Junaid Babar, Isaac Amundson, Karl Hoech, Darren D. Cofer, and Eric Mercer. Assume-Guarantee Reasoning with Scheduled Components. In *NASA Formal Methods*, 2022. URL: https://api.semanticscholar.org/CorpusID:248991390.

[LPZ85] O. Lichtenstein, A. Pnueli, and L.D. Zuck. The Glory of the Past. In *Logics of Programs*, pages 196–218, 1985.

[McM99] Kenneth L. McMillan. Circular Compositional Reasoning about Liveness. In *CHARME*, volume 1703 of *Lecture Notes in Computer Science*, pages 342–345. Springer, 1999.

[MP92a] Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems - specification.* Springer, 1992.

[MP92b] Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems - specification.* Springer, 1992.

[PDH99] Corina S. Pasareanu, Matthew B. Dwyer, and Michael Huth. Assume-Guarantee Model Checking of Software: A Comparative Case Study. In *SPIN*, volume 1680 of *Lecture Notes in Computer Science*, pages 168–183. Springer, 1999.

[Pnu77] Amir Pnueli. The Temporal Logic of Programs. pages 46–57, 09 1977. `doi:10.1109/SFCS.1977.32`.

[RBH$^+$01] Willem-Paul Roever, Frank Boer, Ulrich Hannemann, Jozef Hooman, Yassine Lakhnech, Mannes Poel, and Job Zwiers. *Concurrency Verification: Introduction to Compositional and Noncompositional Methods.* 01 2001.

[RR09] Ondrej Rysavy and Jaroslav Rab. A formal model of composing components: The TLA+ approach. *Innovations in Systems and Software Engineering*, 5:139–148, 06 2009. `doi:10.1007/s11334-009-0089-0`.

[Sis85] A. P. Sistla. On characterization of safety and liveness properties in temporal logic. In *Proceedings of the Fourth Annual ACM Symposium on Principles of Distributed Computing*, PODC '85, pages 39–48, New York, NY, USA, 1985. Association for Computing Machinery. `doi:10.1145/323596.323600`.

[Ton17] Stefano Tonetta. Linear-time Temporal Logic with Event Freezing Functions. In *GandALF*, volume 256 of *EPTCS*, pages 195–209, 2017.