# Automated Compositional Proofs for Real-Time Systems*

Carlo A. Furia, Matteo Rossi,
Dino Mandrioli, and Angelo Morzenti

**Abstract**

We present a framework for formally proving that the composition of the behaviors of the different parts of a complex, real-time system ensures a desired global specification of the overall system. The framework is based on a simple compositional rely/guarantee circular inference rule, plus a methodology concerning the integration of the different parts into a whole system. The reference specification language is the TRIO metric linear temporal logic.

The novelty of our approach with respect to existing compositional frameworks — most of which do not deal explicitly with real-time requirements — consists mainly in its generality and abstraction from any assumptions about the underlying computational model and from any semantic characterizations of the temporal logic language used in the specification. Moreover, the framework deals equally well with continuous and discrete time. It is supported by a tool, implemented on top of the proof-checker PVS, to perform deduction-based verification through theorem-proving of modular real-time axiom systems.

As an example of application, we show the verification of a real-time version of the old-fashioned but still relevant "benchmark" of the dining philosophers problem.

## 1 Introduction

Formal methods are increasingly recognized to be a useful tool for the development of applications, as they allow their users to precisely verify the correctness of systems in their early development phases, before uncaught mistakes become overly costly to fix. One drawback often attributed to formal methods, however, is that they do not "scale up", i.e., when the system grows in complexity, they are too cumbersome and unwieldy to be used effectively. A natural solution to this problem is to apply well-known software engineering principles such as

*A preliminary version of this paper appeared in the *Proceedings of the 8th International Conference on Fundamental Approaches to Software Engineering* (FASE'05), Lecture Notes in Computer Science, vol. 3442, pp. 326–340, Springer-Verlag, 2005.

modularity and separation of concerns not only to design, but also to the verification of formal models. A compositional framework can help in this regard, in that it allows one to focus on the single parts of the system at first, and then analyze their mutual interactions at a later moment with a smaller effort than would be required if all aspects (local and global) of the application were taken into account at once.

This paper presents a compositional inference rule and methodology for the TRIO language [5] that is suitable to formally prove the correctness of the behavior of a modular system from the behavior of its components. TRIO is a metric temporal logic for modeling and analyzing time-critical systems, and has been used in a number of industrial projects. Its advanced modular features are useful in writing specifications of complex systems. Our framework combines these features with a compositional inference rule through a methodology that facilitates its practical use in structured specifications.

Our approach belongs to the general framework of descriptive formalisms, where a specification consists of a set of logic formulas and the verification consists of formally demonstrating that certain desired properties follow deductively from the specification formulas. This framework is indeed very general and abstract, as it does not rely on any specific semantic assumption and is independent of any notion of underlying computational model (to be considered only later, when moving from specification towards implementation). In fact, proofs in our framework should be intended as in classic logic deduction [22], and are supported and semi-automated by the theorem proving tool PVS [28]. Also, even if the reference language is TRIO, the results can be easily extended to any logic formalisms for the description of real-time axiom systems.

The paper is structured as follows: Section 2 briefly introduces the TRIO language; Section 3 presents a proof-oriented compositional framework for TRIO; Section 4 applies the framework to a timed version of Dijkstra's dining philosophers problem [10]; Section 5 reviews the most important compositional rules and frameworks in the literature, and points out where our approach differs from previous works on this subject; Section 6 draws conclusions and outlines future research.

# 2 TRIO

TRIO [5, 17, 25] is a general-purpose specification language suitable for describing real-time systems. It is a first-order linear temporal logic that supports a metric on time. TRIO formulas are built out of the usual first-order connectives, operators and quantifiers, as well as a single basic modal operator, called Dist, that relates the *current time*, which is left implicit in the formula, to another time instant: given a time-dependent formula $F$ (i.e., a term representing a mapping from the time domain to truth values) and a term $t$ indicating a time distance, the formula $\mathrm{Dist}(F, t)$ specifies that $F$ holds at a time instant whose distance is exactly $t$ time units from the current instant. $\mathrm{Dist}(F, t)$ is in turn also a time-dependent formula, as its truth value can be evaluated for any current

time instant, so that temporal formulas can be nested as usual.

In this paper, we do not formally specify a semantics for the interpretation of TRIO formulas (the interested reader can refer to [17, 24]), limiting ourselves to the informal intuitive description of the Dist operator given above. In fact, one of the distinguishing features of our compositional framework — presented in the following sections — is its being *syntactic*, that is based on an inference rule which is proved, and can be applied, by exploiting only syntactic manipulations of formulas, with minimal assumptions on how the syntactic items are actually interpreted. Therefore, the following discussion is almost totally independent of how the modal operator Dist is interpreted, and of which computational model is chosen. In particular, TRIO formulas can be interpreted over both dense and discrete time models, and the results of this paper hold regardless of the chosen time model. The only assumptions we make on the time model is that it is a metric, totally ordered set. In practice, one usually chooses the sets of naturals $\mathbb{N}$ and integers $\mathbb{Z}$ for discrete-time models, and the rationals $\mathbb{Q}$ and reals $\mathbb{R}$ for dense-time models.

For convenience in the writing of specification formulas, it is customary to define a number of *derived* temporal operators from the basic Dist, through propositional composition and first-order logic quantification. Table 1 shows those used in this paper.

| Operator | Definition |
|---|---|
| $\text{Past}(F, t)$ | $t \geq 0 \wedge \text{Dist}(F, -t)$ |
| $\text{Futr}(F, t)$ | $t \geq 0 \wedge \text{Dist}(F, t)$ |
| $\text{Som}(F)$ | $\exists d : \text{Dist}(F, d)$ |
| $\text{Alw}(F)$ | $\forall d : \text{Dist}(F, d)$ |
| $\text{AlwP}(F)$ | $\forall d > 0 : \text{Past}(F, d)$ |
| $\text{AlwF}(F)$ | $\forall d > 0 : \text{Futr}(F, d)$ |
| $\text{Lasted}(F, t)$ | $\forall d \in (0, t) : \text{Past}(F, d)$ |
| $\text{Lasts}(F, t)$ | $\forall d \in (0, t) : \text{Futr}(F, d)$ |
| $\text{Within}(F, t)$ | $\exists d \in (0, t) : \text{Past}(F, d) \vee \text{Futr}(F, d) \vee F$ |
| $\text{WithinP}(F, t)$ | $\exists d \in (0, t) : \text{Past}(F, d)$ |
| $\text{WithinF}(F, t)$ | $\exists d \in (0, t) : \text{Futr}(F, d)$ |
| $\text{Since}(F, G)$ | $\exists d > 0 : \text{Lasted}(F, d) \wedge \text{Past}(G, d)$ |
| $\text{Until}(F, G)$ | $\exists d > 0 : \text{Lasts}(F, d) \wedge \text{Futr}(G, d)$ |
| $\text{UpToNow}(F)$ | $\begin{cases} \exists d > 0 : \text{Lasted}(F, d) & \text{if time is dense} \\ \text{Past}(F, 1) & \text{if time is discrete} \end{cases}$ |
| $\text{NowOn}(F)$ | $\begin{cases} \exists d > 0 : \text{Lasts}(F, d) & \text{if time is dense} \\ \text{Futr}(F, 1) & \text{if time is discrete} \end{cases}$ |
| $\text{Becomes}(F)$ | $\text{UpToNow}(\neg F) \wedge (F \vee \text{NowOn}(F))$ |

Table 1: TRIO derived temporal operators

Notice that TRIO operators predicating on intervals do not include the interval endpoints by default. We define variations that may or may not include such endpoints by using the subscripts $i$ (included) or $e$ (excluded). Thus, we have the following definitions (that will be used in what follows): $\text{AlwP}_\text{i}(F) \equiv \forall d \geq 0 : \text{Past}(F, d)$, $\text{AlwP}_\text{e}(F) \equiv \text{AlwP}(F)$, $\text{WithinF}_\text{ei}(F, t) \equiv \exists d \in (0, t] : \text{Futr}(F, d)$, $\text{WithinF}_\text{ii}(F, t) \equiv \exists d \in [0, t] : \text{Futr}(F, d)$, etc. Also notice that any free variable

is implicitly universally quantified at the outermost level, and so is any TRIO formula with respect to time (with the Alw operator).

The TRIO specification of a system consists of a set of basic *items*, which are primitive elements, such as predicates, time-dependent values, functions, etc., representing the elementary model of some aspects of the system. It is often convenient and natural, especially when dealing with continuous time, to use constrained time-dependent items with a particular temporal behavior, such as *states* and *events*. Events are predicates that are true only at isolated time instants; states are predicates which are true on non-empty time intervals (see [16] for a more precise definition and for motivations on their use). After declaring the basic items, the behavior of a system over time is formally described by introducing a set of TRIO formulas, which state how the items are constrained and how they vary, in a purely descriptive (or declarative) fashion.

To specify large and complex systems, and to support encapsulation, reuse and information hiding at the specification level, TRIO has the usual object-oriented constructs such as classes, inheritance and genericity (see e.g. [23]).[1] The basic encapsulation unit is the *class*. Classes can be simple or structured. Simple classes are collections of parameters, basic items, and formulas. Structured classes, in addition, contain instances of other classes, called *modules*. An instance of a simple class is, in logical terms, a model of the class' formulas (i.e., an interpretation satisfying the conjunction of all formulas). An instance of a structured class recursively contains instances of its composing modules. Items of different modules can be *connected*: whenever two items are connected, they are implicitly defined as logically equivalent. If a more complicated semantics to represent connections in a system is needed, one may represent all the relevant information in an *ad hoc* class. For more details on the object-oriented features of TRIO we refer the reader to [25].

Each TRIO class also has an *interface*, defined as the set of items and formulas that are externally *visible*. The user can declare each item to be visible or non-visible; henceforth all formulas predicating on visible items *only* are considered as visible, while all other formulas are considered as non-visible outside the class. Classes and their interfaces are synthetically represented with a graphical notation, where classes are pictured as boxes, visible items are denoted by lines crossing the class box, whereas non-externally visible items stay entirely within the box boundaries. For example, Figure 1 represents a simple class, while Figure 2 represents a structured class and the connections among items of its modules.

For each TRIO class, formulas are divided into three categories: *axioms*, *assumptions* and *theorems*. When necessary, we will refer to these formulas as (TRIO) "axiom, assumption, theorem formulas", respectively. From a methodological point of view, axioms postulate the basic behavior of the system, assumptions express constraints we must prove by means of other parts of the system (external to the current class) and theorems describe properties that are derived from other formulas. Therefore, the verification of a specification

[1] In this paper, however, we are not using TRIO's inheritance mechanisms.

would basically consist in proving theorems from axioms and assumptions, after proving the latter. In Section 3 we illustrate the compositional methodology that exploits these language features.

We now illustrate TRIO's features by introducing the specification of the *dining philosophers problem* [10]. In a nutshell, the problem considers a set of philosophers sitting around a table, competing for the acquisition of some shared resources, namely one fork for each pair of philosophers. When one succeeds in acquiring the forks on both sides, he/she can eat for some time, after which the forks are released for others to use. The non-eating activity is usually referred to as "thinking". In the traditional formulation of the problem, the goal is to prove the absence of deadlocks, which may arise in acquiring the shared forks. In our specification, instead, the goal is to prove a real-time, global, non-starvation property for all the philosophers, i.e., the fact that, under suitable assumptions, every philosopher eats within a given time.

Our solution is based on a **philosopher** class and assumes a continuous time model. Continuous time introduces some peculiar difficulties in the specification and verification phases, which we will handle by exploiting an axiomatic/deductive approach. For reference, we give the full specification of the example in the Appendix; the interested reader can also find the complete TRIO/PVS [16, 12] proofs in [15].

The basic items of the class are the event item start for system initialization, and the events take($s$) and release($s$), with $s \in \{\mathrm{l}, \mathrm{r}\}$, indicating, for each philosopher, the action of taking or releasing the left or right fork. Other items are the state eating, which is true exactly when the philosopher is eating, and the states holding($s$) and available($s$), meaning that the philosopher is holding a given fork or that the fork is available (i.e., not held by the adjacent philosopher). For notational convenience, we also introduce the states thinking and hungry, representing the philosopher not eating, and being ready to eat again (after a sufficiently long thinking period), respectively. The **philosopher** class is parametric with respect to three constants $\mathrm{t_e}, \mathrm{T_e}, \mathrm{T_t}$. They denote, respectively, the minimum eating time, the maximum eating time and the thinking time after an eating session, before becoming hungry again. Obviously, we assume $\mathrm{T_e} > \mathrm{t_e} > 0$.

In pursuing a fully modular approach to the specification of the problem, one should also introduce a dedicated class to model the *forks* in the system. We notice, however, that such a class would simply be a connector class, that is, it would not possess any internal dynamics but would just "pass the information" about the forks' state between two adjacent philosophers. Therefore, as the main purpose of the example is to illustrate the composition of rely/guarantee specifications to deduce global properties, we prefer not to introduce a class to model forks: all the relevant information about the shared items is represented by items of the **philosopher** class, and by connections among its instances.

We then formalize the basic behavior of a philosopher through axiom formulas of the **philosopher** class. We postulate that each philosopher always takes and releases both forks simultaneously (axiom **holding_synch**); consequently, if only one fork is available, the philosopher waits till the other fork becomes

available as well.[2]

**Axiom 1 (philosopher.holding_synch).** $\mathsf{holding}(\mathrm{l}) \Leftrightarrow \mathsf{holding}(\mathrm{r})$

A philosopher is "hungry" when he/she has not eaten for a period of at least $\mathrm{T_t}$, as in the following axiom.

**Axiom 2 (philosopher.hungry).** $\mathrm{Lasted}(\neg\mathsf{eating}, \mathrm{T_t}) \Leftrightarrow \mathsf{hungry}$

If the philosopher is hungry and if the forks are available, two situations are possible: either he/she takes both forks, or nondeterministically one of his/her neighbors takes a fork at that very time, so that the fork is not available anymore and the philosopher "loses his/her turn". This is formalized by axiom **acquire**.

**Axiom 3 (philosopher.acquire).**
$\mathsf{hungry} \wedge \mathrm{UpToNow}(\mathsf{available}(\mathrm{l}) \wedge \mathsf{available}(\mathrm{r}))$
$\Rightarrow\ (\mathsf{take}(\mathrm{l}) \wedge \mathsf{take}(\mathrm{r})) \vee \mathrm{NowOn}(\neg\mathsf{available}(\mathrm{l}) \vee \neg\mathsf{available}(\mathrm{r}))$

Axioms **eat_till_release** and **eating_duration** state that, when a philosopher succeeds in acquiring both forks, he/she eats for a time duration of more than $\mathrm{t_e}$ and less than $\mathrm{T_e}$ time units, after which he/she releases both forks.

**Axiom 4 (philosopher.eat_till_release).**
$\mathrm{Becomes}(\mathsf{holding}(\mathrm{l}) \wedge \mathsf{holding}(\mathrm{r}))$
$\Rightarrow\ (\exists t > \mathrm{t_e} : \mathrm{Lasts}(\mathsf{eating}, t) \wedge \mathrm{Futr}(\mathsf{release}(\mathrm{l}) \wedge \mathsf{release}(\mathrm{r})\,, t))$

**Axiom 5 (philosopher.eating_duration).** [3] $\mathrm{Lasted}(\mathsf{eating}, t) \Rightarrow t < \mathrm{T_e}$

Whenever a philosopher holds both forks we consider him/her eating. This corresponds to predicate eating being true, and is formalized by axiom **eating_def**.

**Axiom 6 (philosopher.eating_def).** $\mathsf{holding}(\mathrm{l}) \wedge \mathsf{holding}(\mathrm{r}) \Leftrightarrow \mathsf{eating}$

The state eating is false whenever the philosopher is "thinking".

**Axiom 7 (philosopher.thinking_def).** $\mathsf{thinking} \Leftrightarrow \neg\mathsf{eating}$

A thinking session (i.e., one in which the philosopher does not hold both forks) which has just begun lasts at least $\mathrm{T_t}$ time units (axiom **thinking_duration**).

**Axiom 8 (philosopher.thinking_duration).**
$\mathrm{Becomes}(\mathsf{thinking}) \Rightarrow \mathrm{Lasts}(\mathsf{thinking}, \mathrm{T_t})$

[2] We adopted this simplification because we are not interested here in deadlock analysis, as discussed above. If needed, a conceptually similar but longer treatment could be applied, by allowing a philosopher to take one fork at a time, and imposing suitable fairness rules (such as timeouts) on the time he/she can hold it without acquiring the other one.

[3] Recall that free variables (i.e., $t$) are implicitly universally quantified at the outermost level.

Axioms **taking** and **putting** describe the consequences of a take and release action, respectively. More precisely, a fork $s$ can be taken when (up to now) it is available and not already held; in this case, the state holding($s$) stays true until a release for the same fork is issued. On the other hand, a fork $s$ can be released when (up to now) it is held; in this case, the state holding($s$) stays false until a take for the same fork is issued.

**Axiom 9 (philosopher.taking).**
take($s$) $\wedge$ UpToNow($\neg$holding($s$)) $\wedge$ UpToNow(available($s$))
$\Rightarrow$ Until(holding($s$), release($s$))

**Axiom 10 (philosopher.putting).**
release($s$) $\wedge$ UpToNow(holding($s$)) $\Rightarrow$ Until($\neg$holding($s$), take($s$))

Figure 1 illustrates the items and interface of the **philosopher** class. Thus, for instance, holding($s$) and eating are both visible items; therefore axiom **eating_def** is also visible. eating has to be declared as visible because, even if it is not used directly in the connections between philosophers, the global non-starvation property that we are going to state predicates on it. The same holds for the start event, whereas the thinking state is visible since a theorem predicating on it will be used in proving the assumptions of other classes.

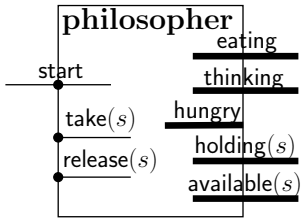


Figure 1: Interface of the **philosopher** class

We compose $N \geq 2$ instances of the **philosopher** class into the new composite class **dining_N**. The $N$ modules of the **philosopher** class are instantiated in an array **Philosophers** indexed by the range $[0..N-1]$. The modules are connected so that the **available** item of each philosopher corresponds to the negation of the **holding** item of the philosopher on his/her left/right, as pictured in Figure 2.

In Section 4, a global property of this set of philosophers will be proved, namely that each philosopher eats regularly for a certain amount of time. Note that all the instances in the array have the same values for the parameters $\mathrm{t_e}, \mathrm{T_e}, \mathrm{T_t}$. This ensures that the timed behavior of the various philosophers is the same, thus permitting to prove the global property. We leave the analysis of the impact of relaxing such constraint to future work.

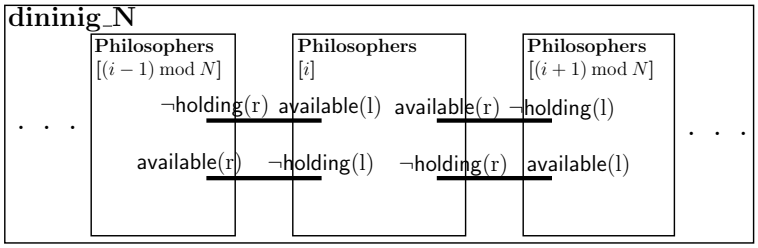


Figure 2: Interface of the **dining_N** class

# 3 A Compositional Framework in TRIO

In this section, we introduce a compositional framework for descriptive logic languages based on axiomatic/deductive techniques, and for the TRIO language in particular. The framework consists essentially of a methodology to specify and verify modular systems. On the one hand, the methodology exploits some TRIO language features to allow for an effective and natural way of structuring a large specification into classes. On the other hand, some specific compositional requirements are satisfied by introducing an *ad hoc* operator (to express temporal formulas where the behavior of a module depends on some properties of its actual environment), and by providing an inference rule to handle specifications written using this operator when integrating modules into an overall system.

Let us briefly sketch the rationale behind the compositional framework. The specification of a large system is structured into classes. The fundamental behavior of the items of each class is captured by axiom formulas. The derived behavior of each class, which is to be verified, can be expressed by theorem formulas. In general, according to the rely/guarantee paradigm, we may need to relate the derived behavior of a class with certain properties of the (external) environment of that class. This can be achieved in essentially two ways. If the assumptions on the environment are temporally closed formulas (i.e., they express time-invariant properties), one may use TRIO assumption formulas to represent them. If, on the other hand, one has to express assumptions on the environment that are not invariant but temporally depend on the derived behavior they guarantee,[4] an *ad hoc* operator is provided: the $\twoheadrightarrow$ operator (called *time progression*) for the TRIO language, introduced in Section 3.2. Then, the specification is completed by composing some classes together, as modules of the overall (global) system.

The verification burden for the global system consists essentially of two tasks. First, the assumptions that have been introduced according to the rely/guarantee paradigm have to be proved. For the sake of methodological distinction, we will generally use — whenever the object is an assumption (rather than a theorem) — the verb "discharge" to mean "prove" (e.g., [30]). Assumptions ex-

[4]This is likely to happen when specifying real-time systems.

pressed as closed TRIO formulas are discharged by means of formulas of other classes. When doing this, it is important to avoid circularities, in order to guarantee the soundness of the whole deductive process. Conversely, assumptions expressed by means of the time progression operator are discharged using an *ad hoc* inference rule which handles circularities, according to the semantics of the operator. Thus, discharging these assumptions amounts to proving the truth of the premises of the inference rule, which can still be done by means of ordinary (temporal) deductive techniques. After discharging all the assumptions, the second stage of the global verification process consists in proving the theorems from axioms, assumptions, and already proved theorems. This is done first for theorems that are local (i.e., they can be proved within a single class), and then for global theorems (i.e., involving properties emerging from the combined behavior of some modules). This meets the overall verification goal.

## 3.1 Rely/Guarantee Specifications

Let us consider how to write the specification of a TRIO class $C$. The basic behavior of $C$ is defined in terms of axioms over both visible and non-visible items, which rely on no assumptions, since they just state the very basic behavior of the class. Then, we state a number of *derived* properties of the class as theorems, that are going to be proved in the verification process. Since in general a class describes an *open* component, it often happens that the truth of some theorem depends on assumptions about how the actual class environment behaves. In other words, some properties hold only if the modules that will constitute the external environment of the class satisfy some requirements. Therefore, according to the rely/guarantee paradigm, we make these assumptions explicit in writing the specification of a TRIO class, which becomes therefore a *rely/guarantee specification*.

Now, we distinguish between two different kinds of assumptions that one may need to express. The first (and simplest) case is that of assumptions which can be expressed as *temporally closed* formulas, that is formulas whose truth is invariant over the whole temporal axis. In such cases, the assumptions can be stated using the TRIO language construct of the *assumption formula*. The other case, namely when the assumption is not expressed as a temporally closed formula, will be handled by using an *ad hoc* operator, discussed in Section 3.2.

Let us now introduce some notation that will help us describe these ideas more precisely. Let $\mathcal{AX}_C$, $\mathcal{AS}_C$ and $\mathcal{TH}_C$ denote the disjoint sets of axioms, assumptions and theorems of class $C$, respectively, and let $\mathcal{F}_C = \mathcal{AX}_C \cup \mathcal{AS}_C \cup \mathcal{TH}_C$ be the set of all formulas of $C$. Furthermore, for each set of formulas $\mathcal{F}_C$, we define $\mathcal{F}_C^{\vee} \subseteq \mathcal{F}_C$ as the set of *visible* formulas in $\mathcal{F}_C$, i.e., the formulas predicating over visible items only (see Section 2). Therefore, the complete specification of $C$ is represented by the formula $\mathcal{AX}_C \cup \mathcal{AS}_C \vdash \mathcal{TH}_C$, which expresses the fact that the truth of the theorems of class $C$ follows deductively from its axioms and assumptions.

Let us map these ideas to the philosophers example. Section 2 showed the axioms of the **philosopher** class, formalizing its basic postulated behavior. A

derived property of the class is that there is always a time interval in which both forks are available to the philosopher. This is the first step in ensuring that the philosopher will eventually be able to acquire both forks and eat. In particular, the axioms allow us to prove this availability property for a time interval which occurs no later than $\mathrm{T_t} + 2\mathrm{T_e}$ time units from the current instant. This bound is sufficient to prove all of the following derived properties. The above property is expressed by the following TRIO theorem formula.

**Theorem 11 (philosopher.fork_availability).**
$\mathrm{WithinF_{ei}}(\mathrm{UpToNow}(\mathsf{available}(\mathrm{l}) \wedge \mathsf{available}(\mathrm{r}))\,, \mathrm{T_t} + 2\mathrm{T_e})$

Clearly, the validity of this theorem cannot be guaranteed regardless of the behavior of the environment of this class. Therefore, we introduce three assumption formulas that suffice to deduce theorem **fork_availability**. First of all, we assume that, at any given time, each fork becomes available within $\mathrm{T_t} + \mathrm{T_e}$ time units or is already available and remains so for a sufficiently long (i.e., $\geq \mathrm{T_t}$) amount of time.

**Assumption 12 (philosopher.availability).**
$(\exists t \geq \mathrm{T_t} : \mathrm{Lasts}(\mathsf{available}(s)\,, t)) \vee \mathrm{WithinF_{ei}}(\mathrm{Becomes}(\mathsf{available}(s))\,, \mathrm{T_t} + \mathrm{T_e})$

Second, we assume that each fork is available, for a non-empty time interval, within $\mathrm{T_e}$ time units. This is basically like assuming that the adjacent philosophers eat for no longer than $\mathrm{T_e}$ time units.

**Assumption 13 (philosopher.availability_2).**
$\mathrm{WithinF}(\mathrm{UpToNow}(\mathsf{available}(s))\,, \mathrm{T_e})$

Finally, when a fork becomes available, we assume it to stay so for (at least) $\mathrm{T_t}$ time units; this corresponds to assuming that the thinking time of the neighbor philosophers is not shorter than $\mathrm{T_t}$.

**Assumption 14 (philosopher.lasting_availability).**
$\mathrm{Becomes}(\mathsf{available}(s)) \Rightarrow \mathrm{Lasts}(\mathsf{available}(s)\,, \mathrm{T_t})$

Notice that each of these formulas expresses a temporally-closed property (for instance, considering assumption **availability_2**, at *any given time* each fork is available within $\mathrm{T_e}$ time units).

Finally, let us introduce another theorem of the **philosopher** class which will be used in its verification. The truth of this theorem is a direct consequence of the axioms of the class, requiring no assumptions. It states that any philosopher is always in one of two situations: either he/she is hungry (i.e., $\mathrm{T_t}$ time units without eating have passed), or no more than $\mathrm{T_t} + \mathrm{T_e}$ time units have elapsed since the last time he/she ate or he/she began eating.

**Theorem 15 (philosopher.always_eating_or_not).** $\mathsf{hungry} \vee$
$\mathrm{WithinP_{ii}}((\exists t > \mathrm{t_e} : \mathrm{Lasted}(\mathsf{eating}, t)) \vee \mathrm{Becomes}(\mathsf{eating})\,, \mathrm{T_t} + \mathrm{T_e})$

## 3.2 A Rely/Guarantee Operator and Inference Rule

To formalize rely/guarantee specifications where the assumption is not a temporally closed formula, it is convenient to introduce an *ad hoc* temporal operator which we call *time progression* operator.

Let $P$ and $Q$ be any two time-dependent formulas. We define the time progression operator $\twoheadrightarrow$ as follows:

$$P \twoheadrightarrow Q \quad \triangleq \quad \begin{cases} \mathrm{AlwP_e}(P) \Rightarrow \mathrm{AlwP_i}(Q) \wedge \mathrm{NowOn}(Q) & \text{if time is dense} \\ \mathrm{AlwP_e}(P) \Rightarrow \mathrm{AlwP_i}(Q) & \text{if time is discrete} \end{cases}$$

Informally, $P \twoheadrightarrow Q$ means that $Q$ lasts at least as long as $P$ does and even "a bit longer". The NowOn conjunct is not needed in the discrete-time definition, as the inclusion of the current instant in the right-hand side of $\mathrm{AlwP_i}$ operator suffices to ensure that $Q$ lasts one (discrete) time step longer than $P$.

Now, we consider rely/guarantee specifications expressed in terms of the $\twoheadrightarrow$ operator. Therefore, if $E$ is the environment assumption and $M$ is the guarantee, the rely/guarantee specification is written as $E \twoheadrightarrow M$. This specifies that, whenever the assumption $E$ is true up to some time $t$, the guarantee $M$ is true up to time $t + \epsilon$, where $\epsilon > 0$. We note explicitly that we now admit temporally open formulas as assumptions and guarantees.

In the remainder of this subsection we prove some relevant properties of the $\twoheadrightarrow$ operator, and in particular an inference rule to deduce the validity of formulas from a set of rely/guarantee specifications written using the time progression operator. This inference rule handles a sound discharging of the assumptions in a set of $\twoheadrightarrow$ formulas, and will be used in Section 3.3 to verify the composition of a set of modules containing rely/guarantee specifications written using the time progression operator.

Let us first state the following property of the $\twoheadrightarrow$ operator, which states some conditions under which it is possible to "prolong" the truth of some item in the future, from its truth in a base interval in the past, by successive applications of a time progression operator and of logical implication.

**Lemma 1.** *For any time-dependent formulas $P$, $Q$, and $R$, if:*

1. $\mathrm{Som}(\mathrm{AlwP_e}(P))$
2. $\mathrm{Alw}(Q \wedge R \Rightarrow P)$
3. $\mathrm{Alw}(P \twoheadrightarrow Q)$

*then:* $\mathrm{Alw}(R \twoheadrightarrow Q)$.

*Proof.* We split the proof into two parts, for dense and discrete time models, respectively.

**Proof for dense time models.** Let us assume $\mathrm{Alw}(P \twoheadrightarrow Q)$ and, by the definition of the $\twoheadrightarrow$ operator, $\mathrm{AlwP_e}(R)$ true at a generic time instant $t$. We prove that $\mathrm{AlwP_i}(Q) \wedge \mathrm{NowOn}(Q)$ is true at $t$. See Figure 3 for an informal graphical representation of the proof.

Since $\mathrm{Som}(\mathrm{AlwP_e}(P))$, let $u_0$ be the time instant at which $\mathrm{AlwP_e}(P)$ holds. If $u_0 \geq t$, by considering $\mathrm{Alw}(P \twoheadrightarrow Q)$ we deduce that $\mathrm{AlwP_i}(Q) \wedge \mathrm{NowOn}(Q)$ is true for all time instants less than or equal to $u_0$. This simply concludes this branch of the proof.

Let us now assume $u_0 < t$. We know that $P$ is true for all instants less than $u_0$. Moreover, since $\mathrm{Alw}(P \twoheadrightarrow Q)$, there exists some quantity $\epsilon_0 > 0$ such that $Q$ is true for all instants less than $u_1 = u_0 + \epsilon_0$. Now, it is either $u_1 > t$ or $u_1 \leq t$. In the first case the proof is concluded, since we have shown that $Q$ lasts longer than $R$ does. Instead, if $u_1 \leq t$, we can iterate the reasoning to new instants $u_2 = u_1 + \epsilon_1, u_3, \ldots$ until we obtain a $u_n > t$. Notice, in fact, that both hypotheses (2–3) of the lemma hold on the whole temporal axis, because of the Alw operator. Therefore, for any $u_i \leq t$ such that $Q$ holds up to $u_i$, we are able to find another $u_{i+1} > u_i$ such that $Q$ holds up to $u_{i+1}$ as well. In fact, $R$ holds up to $u_i$ as well, since it holds up to $t \geq u_i$. Thus, by hypothesis (3) we deduce that $P$ holds up to $u_i$. Next, we consider hypothesis (2) at $u_i$: since $\mathrm{AlwP_e}(P)$ at $u_i$, then $Q$ and $\mathrm{NowOn}(Q)$ at the same instant. Therefore, by the definition of the NowOn operator, there exists a $\epsilon_i > 0$ such that $Q$ holds up to $u_i + \epsilon_i = u_{i+1} > u_i$.

We now prove that the sequence of points $u_i$ cannot accumulate before time $t$, but it must eventually grow beyond $t$. We show this by contradiction: assume that $Q$ holds up to time $\delta < t$ only (note that the following reasoning holds regardless of whether $Q$ holds *exactly* at $\delta$ or not, i.e., whether the interval of validity of $Q$ is open or closed to the right). Surely $R$ also holds up to the absolute instant $\delta$, since it holds until time $t$. Now, by hypothesis (2), we realize that $P$ is also true for all time instants less than $\delta$. Since $\mathrm{Alw}(P \twoheadrightarrow Q)$, $Q$ lasts until beyond $P$, contradicting the hypothesis that $Q$ holds up to $\delta$ only.

Since $Q$ lasts for all instants less than $u_n > t$, the proof is concluded, since it is demonstrated that $\mathrm{AlwP_i}(Q) \wedge \mathrm{NowOn}(Q)$ holds at $t$. □

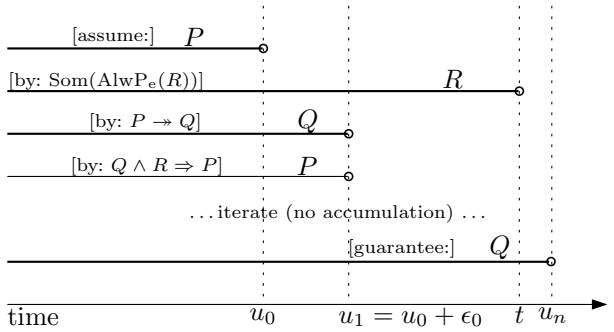


Figure 3: A graphical representation of Lemma 1.

**Proof for discrete time models.** Let us assume $\mathrm{Alw}(P \twoheadrightarrow Q)$ and, by the definition of the $\twoheadrightarrow$ operator, $\mathrm{AlwP_e}(R)$ true at a generic time instant $t$. We prove that $\mathrm{AlwP_i}(Q)$ is true at $t$.

Since $\mathrm{Som}(\mathrm{AlwP_e}(P))$, let $u_0$ be the time instant at which $\mathrm{AlwP_e}(P)$ holds. Let us consider the sequence of points $u_i = u_0 + i$, for all $i = 0, \ldots, (t - u_0)$. By induction on $i$, we show that $\mathrm{AlwP_i}(Q)$ holds at all time instants $u_i$.

The base case $i = 0$ is subsumed by the hypotheses, since $\mathrm{AlwP_e}(P)$ is true at $u_0$ and $P \twoheadrightarrow Q$ also holds at $u_0$.

The inductive step requires us to prove that $\mathrm{AlwP_e}(Q)$ holds at time $u_{i+1}$, by assuming that it holds a $u_i$, for $u_i < t$ (otherwise $u_{i+1}$ is undefined). Since $u_i < t$, it is also $\mathrm{AlwP_i}(R)$ at $u_i$ by hypothesis. Then, by hypothesis (2), $\mathrm{AlwP_i}(P)$ holds at the same time $u_i$. From the definition of the $\mathrm{AlwP_i}$ operator, and by the discreteness of time, this last fact can be rewritten equivalently as $\mathrm{AlwP_e}(P)$ at time $u_i + 1 = u_{i+1}$. Since $\mathrm{Alw}(P \twoheadrightarrow Q)$, it follows immediately that $\mathrm{AlwP_i}(Q)$ also holds at time $u_{i+1}$, thus proving the inductive step.

Finally, since $u_{t-u_0} = u_0 + (t - u_0) = t$, we have shown that $\mathrm{AlwP_i}(Q)$ holds at $t$. □

Now, by exploiting the above lemma, let us prove a useful inference rule for rely/guarantee specifications written using the $\twoheadrightarrow$ operator. For $i = 1, \ldots, m$, let $E_i$, $M_i$, $E$, and $M$ be generic time-dependent formulas. The following inference rule formulates conditions under which the formula $E \twoheadrightarrow M$ is true, assuming that all the formulas $E_i \twoheadrightarrow M_i$ hold. In Section 3.3, we will show how to use this rule to infer the validity of some global rely/guarantee specification (in the form $E \twoheadrightarrow M$) from the validity of a set of local rely/guarantee specifications (in the form $E_j \twoheadrightarrow M_j$).

**Proposition 2 (Rely/Guarantee Compositional Inference Rule).**
*If, for* $i = 1, \ldots, m$*:*

1. $\mathrm{Som}(\mathrm{AlwP_e}(E_i))$ *(that is,* $E_i$ *is* initialized*)*
2. $E \wedge \bigwedge_{j=1,\ldots,m} M_j \Rightarrow E_i$
3. $\bigwedge_{j=1,\ldots,m} M_j \Rightarrow M$
4. $\mathrm{Alw}\Big(\bigwedge_{j=1,\ldots,m}(E_j \twoheadrightarrow M_j)\Big)$

*then:* $\mathrm{Alw}(E \twoheadrightarrow M)$.

*Proof.* Assume $\mathrm{Alw}\Big(\bigwedge_{j=1,\ldots,m}(E_j \twoheadrightarrow M_j)\Big)$. It is simple to realize, by considering the definition of the time progression operator, that this implies $\mathrm{Alw}\Big(\Big(\bigwedge_{j=1,\ldots,m} E_j\Big) \twoheadrightarrow \Big(\bigwedge_{j=1,\ldots,m} M_j\Big)\Big)$. Moreover, hypothesis (2) implies that $\mathrm{Alw}\Big(E \wedge \bigwedge_{j=1,\ldots,m} M_j \Rightarrow \bigwedge_{j=1,\ldots,m} E_j\Big)$, since it holds for every $i = 1, \ldots, m$.

Finally, hypothesis (1) implies that $\mathrm{Som}\Big(\mathrm{AlwP_e}\Big(\bigwedge_{=1,\ldots,m} E_i\Big)\Big)$, since there exists a base interval such that the conjunction of the $E_i$'s is true on it.[5]

Therefore, we can apply Lemma 1 by substituting $\bigwedge_{j=1,\ldots,m} E_j$ for $P$, $\bigwedge_{j=1,\ldots,m} M_j$ for $Q$ and $E$ for $R$. We get:
$\mathrm{Alw}\Big(\Big(\bigwedge_{j=1,\ldots,m} E_j\Big) \twoheadrightarrow \Big(\bigwedge_{j=1,\ldots,m} M_j\Big)\Big) \Rightarrow \mathrm{Alw}\Big(E \twoheadrightarrow \Big(\bigwedge_{j=1,\ldots,m} M_j\Big)\Big)$.

Finally, by combining it with hypothesis (3) and with the definition of the time progression operator, we get the desired result. □

### 3.2.1 (In)Completeness of the Inference Rule

An inference rule is defined by its hypotheses (also called premises) and by its consequent (or conclusion). In the case of Proposition 2, the hypotheses are the four formulas (1–4), while the consequent is the last formula, whose truth is inferred from the hypotheses. An inference rule is *complete* if, for any system (defined in TRIO by a set of axioms), any formula that holds for the system is also a consequence of direct applications of the inference rule only.

In the literature, the completeness issue has been investigated for some classes of compositional inference rules (see [26, 21]). It has been shown that, as it is often the case with formal languages in general, there is some trade-off between completeness and simplicity and ease of use, and that several widely used compositional rules are indeed incomplete. Likewise the inference rule of Proposition 2 is *incomplete*, as we prove next.

**Proposition 3.** *The inference rule of Proposition 2 is* incomplete.

*Proof.* Let $m = 2$. Let $\epsilon$ and $\mu$ be two primitive Boolean-valued time-dependent items, and let us consider $E = \epsilon$ and $M = \mu$ as the two time-dependent formulas that appear in the consequent of Proposition 2. We just have to show a system for which $E \twoheadrightarrow M$ holds at every time, but such that it is not possible to find any $E_1, E_2, M_1, M_2$ that make the premises of the rule true.

Let us consider the system in which $\mu$ is always false, while $\epsilon$ is true only at some time $u$ internal[6] to the time domain. This could be defined in TRIO by the two axioms $\mathrm{Alw}(\neg\mu)$ and $\mathrm{Som}(\epsilon \wedge \mathrm{AlwP}(\neg\epsilon) \wedge \mathrm{AlwF}(\neg\epsilon))$.[7] Notice that $E \twoheadrightarrow M$ (that is, equivalently, $\epsilon \twoheadrightarrow \mu$) always holds in the system, as $\mathrm{AlwP_e}(E)$ is always false; it is also plain that this can be proved using standard deductive rules. However, no application of the inference rule of Proposition 2 alone can prove this fact, as we now show by contradiction. Assume that there exist $E_1, E_2$ such that $\mathrm{Som}(\mathrm{AlwP_e}(E_1 \wedge E_2))$ holds (an immediate consequence of hypothesis (1) holding). Thus, there exists a time instant $t$ at which $\mathrm{AlwP_e}(E_1 \wedge E_2)$ holds. Next, we consider hypothesis (4), for $E_1$ and $E_2$, at $t$. Because of the definition

[5] Consider the intersection of the intervals on which each of the $E_i$'s is individually true; this intersection is non-empty, since all intersected intervals are unbounded on the left, because of the AlwP operator.

[6] I.e., such that it is not on the boundaries, if they exist.

[7] For simplicity, assume a doubly infinite time domain such as $\mathbb{Z}$ or $\mathbb{R}$, so that there are no (finite) boundaries.

of the $\twoheadrightarrow$ operator, we can infer, in particular, that $M_1 \wedge M_2$ holds at $t$ (this holds regardless of whether the time model is dense or discrete). But, in order for hypothesis (3) to be true, it must be that $M_1 \wedge M_2$ is always false (otherwise the implication would be false, as $M = \mu$ is always false), which is clearly a contradiction. □

Let us analyze the features of our rule that caused incompleteness. Incompleteness arose from two basic facts. First, the $\twoheadrightarrow$ operator encompasses an AlwP operator in its premise, which is false whenever $E$ is true on finite intervals or isolated points, while hypotheses (2–3) introduce requirements that involve point-wise "instantaneous" implication between formulas; thus it is possible to devise situations in which formulas with $\twoheadrightarrow$ hold, while formulas with $\Rightarrow$ do not. Second, hypothesis (1) states an initialization condition which also involves an interval which is unbounded on the left (as in the definition of the AlwP operator). This, combined with hypothesis (4), determines that $M_1, M_2$ must be true somewhere; this rules out cases such as that considered in the incompleteness proof. A modification of the inference rule to achieve completeness should necessarily address these facts.

To conclude, let us briefly discuss the relevance of completeness of an inference rule, within a certain compositional framework. This would put the incompleteness of our inference rule into perspective, showing that it is probably not necessarily an undesirable shortcoming.

It is often held (see e.g., [9, 4]) that the completeness of an inference rule is a particularly desirable feature, as an incomplete rule would make it impossible to prove a correct statement. However, a complete rule is likely to be more difficult to be applied in practice than an incomplete one. As also noted in [21], such a difficulty is particularly undesirable when dealing with automated verification techniques, where hypotheses which can be checked efficiently are the most important requirement for an inference rule.

In addition, even when the focus is on human-assisted verification (as in our case), completeness may not be an indispensable feature. More precisely, we notice that in most compositional frameworks, the rely/guarantee inference rule of the framework is the *only* inference rule to be applied to carry out compositional reasoning. Hence, in such frameworks an incomplete rule does indeed prevent to infer true facts about the composition of modules. On the contrary, the compositional framework of the present paper is broader, and the rule we introduced in Proposition 2 is not the only way to prove global properties of a modular system. In particular, that rule is responsible only for handling specific rely/guarantee constraints that employ the $\twoheadrightarrow$ operator, as will be shown more explicitly in Section 3.3. So, the rest of the framework — which relies on temporally-closed assumption formulas and usual TRIO techniques — can be used to handle situations where the inference rule falls short due to incompleteness. This can be achieved as TRIO admits a relatively complete [6] set of inference rules, which can be exploited to prove any provable statement (this straightforward result is presented in [29]).

Finally, complete compositional rules are inevitably *trivially complete*. This

means that there are cases where the inference rule is applied by picking one of the "local" formulas $E_i, M_i$ (for some $i$) to coincide with the "global" formulas $E, M$. Such cases — where the system decomposition is a trivial one — are unavoidable in all compositional rules, which cannot escape the worst-case complexity inherent in modular reasoning [8, 9]. Hence, a compositional method must be one which yields good results *in practice* for commonly encountered cases: its failure on *some* pathological instances is unavoidable, and whether it is imputable to incompleteness or not is a matter which can be often overlooked.

### 3.3 Composing TRIO Modules

This section completes the description of the compositional framework by formalizing the composition of TRIO classes which include both assumption formulas and rely/guarantee formulas written using the time progression operator. In a nutshell, we describe a method to prove that the assumptions of each class are met by the classes that constitute its actual environment. This is done by usual deductive techniques for temporally closed assumption formulas, while resorting to the inference rule of Proposition 2 to handle rely/guarantee formulas. After discharging all the assumptions, it is possible to proceed on proving the theorems that are *global*, that is are about the combined behavior of the composed classes.

Let us consider $n \geq 2$ TRIO classes $C_1, \ldots, C_n$. For all $i = 1, \ldots, n$, class $C_i$ has a rely/guarantee specification expressed synthetically by the formula $\mathcal{AX}_i \cup \mathcal{AS}_i \vdash \mathcal{TH}_i$. Let $C_{\text{glob}}$ be a class that encapsulates one instance for each of the $n$ classes $C_i$ as modules of $C_{\text{glob}}$. The composition of the $n$ modules is described by the logical conjunction of all the local specification formulas. Therefore TRIO modules are compositional, in that the semantics of the composition of classes is given by the logical conjunction of the semantics of the classes which are put together. In general, class $C_{\text{glob}}$ adds its own axioms, assumptions and theorems, to those of its enclosed modules (this also allows for the recursive application of the method). Hence, $C_{\text{glob}}$ is described by the formula $\mathcal{AX}_{\text{glob}} \cup \mathcal{AS}_{\text{glob}} \cup \bigcup_{j=1,\ldots,n} \mathcal{F}_j \vdash \mathcal{TH}_{\text{glob}}$.

Our overall verification goal can be detailed as follows.

- Verify each local (i.e., within each class $C_i$) specification.
- Discharge the local assumptions, that is prove that the assumptions of each class are satisfied by its actual environment (i.e., the other classes in the system).
- Verify the global specification, that is prove the global theorems from the local visible formulas, global axioms and assumptions.

It is simple to realize that the basic problem involved in the discharging of assumptions is that such reasoning involves a *circularity* between assumptions and other formulas of the modules. As circularity prevents the verification process

from being sound, these invalid deductions must be ruled out. To this end, we introduce two "circularity-breaking" mechanisms, depending on what kind of assumptions we are considering. For assumptions written as temporally closed TRIO formulas, we require an *explicit* breaking of circularity: the verifier has to organize the discharging of these assumptions in such a way that circularity is simply avoided. Conversely, for assumptions appearing in rely/guarantee specifications using the $\twoheadrightarrow$ operator, we can exploit the inference rule of Proposition 2, which performs an *implicit* circularity breaking, thus ensuring that circularities arising in the hypotheses of the inference rule do not compromise the soundness of the final deduction. All this must be done while respecting the requirements on the visibility of formulas.

In general, each of the $n$ component classes may have one or more rely/guarantee formulas of the form $E \twoheadrightarrow M$ among its theorems. For each class $C_j$, $j = 1, \ldots, n$, let $\mathcal{TH}_j^{\mathrm{rg}} \subseteq \mathcal{TH}_j$ be the set of theorems we consider in the form $E \twoheadrightarrow M$ of class $C_j$, i.e., for each theorem formula $F \in \mathcal{TH}_j$, $F \in \mathcal{TH}_j^{\mathrm{rg}}$ if $F$ can be written as $F \equiv E \twoheadrightarrow M$ for some formulas $E$ and $M$. Notice however that *any* formula can be possibly written using the $\twoheadrightarrow$ operator, as any temporally closed formula $G$ is equivalent to $\mathrm{Alw}(\mathrm{true} \twoheadrightarrow G)$, so the choice of which formulas to consider in $\mathcal{TH}_j^{\mathrm{rg}}$ is partially up to the specifier.[8] Let us define $m$ to be the number of rely/guarantee formulas over all classes: $m = \sum_{j=1,\ldots,n} |\mathcal{TH}_j^{\mathrm{rg}}|$. Moreover, $\mathcal{TH}_j^{\mathrm{nrg}}$ is defined as the complement set $\mathcal{TH}_j \setminus \mathcal{TH}_j^{\mathrm{rg}}$ for all $j = 1, \ldots, n$. The composite class $C_{\mathrm{glob}}$ also has its own rely/guarantee formula $E_{\mathrm{glob}} \twoheadrightarrow M_{\mathrm{glob}}$ among its theorems $\mathcal{TH}_{\mathrm{glob}}$.

Next, let us define what is a *dependency* between two formulas. Let us consider a formal proof $\pi$: it consists of a finite sequence of formulas, together with their *justifications* (see, for example, [22]). We say that a formula $\chi$ *directly depends upon* another formula $\phi$ in the proof $\pi$, and write $\phi \rightsquigarrow_\pi \chi$, if and only if $\phi$ appears before $\chi$ in the proof and $\chi$ is the result of the application of an inference rule which uses $\phi$. The transitive closure $\overset{+}{\rightsquigarrow}$ ("depends upon") of the $\rightsquigarrow$ relation is defined as usual. The notion of dependency can be extended to a set of proofs $\Pi$: for any two formulas $\phi, \chi$ we say that $\phi \overset{+}{\rightsquigarrow}_\Pi \chi$ if and only if there exists a proof $\pi \in \Pi$ such that $\phi \overset{+}{\rightsquigarrow}_\pi \chi$. Note that the we will require the $\overset{+}{\rightsquigarrow}_\Pi$ relation to be irreflexive for the set $\Pi$ of formal proofs constituting the verification process; this requirement will be expressed *a posteriori*.

Finally, the verification of the composite specification proceeds according to the following steps.

1. Verify each local specification, that is prove that for all $k = 1, \ldots, n$:

$$\mathcal{AX}_k \cup \mathcal{AS}_k \vdash \mathcal{TH}_k$$

   From our perspective, this step is considered to be atomic, but obviously the compositional approach can be applied recursively to each module.

[8] Moreover, for the sake of simplicity, we do not address explicitly more convoluted cases where rely/guarantee specifications in the form $E \twoheadrightarrow M$ appear as subformulas of larger formulas.

2. Show that the local assumptions can be discharged by means of global formulas, visible formulas of other classes, and local axioms and theorems.[9] Formally, this corresponds to proving that for all $k = 1, \ldots, n$:

$$\mathcal{AX}_k \cup \mathcal{TH}_k \cup \mathcal{F}_{\text{glob}} \cup \bigcup_{\substack{j=1,\ldots,n \\ j \neq k}} \mathcal{F}_j^{\vee} \vdash \mathcal{AS}_k$$

3. Prove that the global non-rely/guarantee theorems (i.e., not involving the $\twoheadrightarrow$ operator) follow from the local visible formulas and from the global axioms, assumptions and other (i.e., rely/guarantee) theorems. In formulas, this means proving:

$$\mathcal{AX}_{\text{glob}} \cup \mathcal{AS}_{\text{glob}} \cup \mathcal{TH}_{\text{glob}}^{\text{rg}} \cup \bigcup_{j=1,\ldots,n} \mathcal{F}_j^{\vee} \vdash \mathcal{TH}_{\text{glob}}^{\text{nrg}}$$

4. Show that each local rely/guarantee formula has assumptions $E_k$ which satisfy the initialization condition (as in hypothesis (1) of Proposition 2). In order to prove the initialization condition, we allow one to use global and local formulas, plus any visible formula of any other class of the system. This corresponds to proving that for all $k = 1, \ldots, m$: for all $j = 1, \ldots, n$: if $(E_k \twoheadrightarrow M_k) \in \mathcal{TH}_j^{\text{rg}}$ then:

$$\mathcal{F}_{\text{glob}} \cup \mathcal{F}_j \cup \bigcup_{i=1,\ldots,n} \mathcal{F}_i^{\vee} \vdash \text{Som}(\text{AlwP}_{\text{e}}(E_k))$$

5. Show that each local rely/guarantee formula has assumptions that can be discharged by means of global and local formulas, or by the global assumption, or by means of guarantees of other classes. This corresponds to hypothesis (2) in Proposition 2. Formally, prove that for all $k = 1, \ldots, m$: for all $j = 1, \ldots, n$: if $(E_k \twoheadrightarrow M_k) \in \mathcal{TH}_j^{\text{rg}}$ then:

$$\mathcal{F}_{\text{glob}} \cup \mathcal{F}_j \cup \bigcup_{i=1,\ldots,n} \mathcal{F}_i^{\vee} \vdash \text{Alw}\left(E_{\text{glob}} \wedge \bigwedge_{i=1,\ldots,m} M_i \Rightarrow E_k\right)$$

6. Show that the global guarantee follows from the local guarantees of all modules and from global formulas and local visible formulas of any class. This corresponds to hypothesis (3) of Proposition 2, i.e., prove that:

$$\mathcal{F}_{\text{glob}} \cup \bigcup_{j=1,\ldots,n} \mathcal{F}_j^{\vee} \vdash \text{Alw}\left(\bigwedge_{j=1,\ldots,m} M_j \Rightarrow M_{\text{glob}}\right)$$

[9] Of course, if an assumption can be proved just from local axioms and theorems it should be a theorem, but it might happen that local formulas contribute, together with external ones, to the proof of an assumption.

7. Be sure that in all the above proofs (i.e., steps (1–6)) there are no circular dependencies among any two closed formulas. This is the explicit circularity-breaking requirement: formally, it corresponds to checking that in the set $\Pi$ of all the above proofs, for all formulas $\phi \in \bigcup_{k=1,\ldots,n}(\mathcal{AS}_k \cup \mathcal{TH}_k) \cup \mathcal{TH}_{\text{glob}}$:

$$\neg(\phi \overset{+}{\rightsquigarrow}_{\Pi} \phi)$$

i.e., the relation $\overset{+}{\rightsquigarrow}_{\Pi}$ is *irreflexive*.

From the application of the above steps, thanks to the inference rule of Proposition 2 and the absence of circularities, the global verification of the composite specification is soundly completed. In fact, steps (1–3) achieve the verification of the theorems in $\mathcal{TH}^{\text{nrg}}_{\text{glob}}$: since $\overset{+}{\rightsquigarrow}_{\Pi}$ is irreflexive, it induces a partial ordering on closed formulas. Thus, any global ordering of formulas compatible with the partial ordering defines a chain of proofs which all are sound, since they rely on ordinary (sound) inference rules and have no circular dependencies. Then, steps (4–6) apply the inference rule of Proposition 2 by discharging its hypotheses by means of other formulas; since Proposition 2 is also a sound inference rule, the theorem in $\mathcal{TH}^{\text{rg}}_{\text{glob}}$ is soundly proved as well, completing the verification of the global class.

# 4 Compositional Dining Philosophers

This section illustrates the use of the rely/guarantee paradigm, by building the compositional proofs of some relevant properties of the *dining philosophers problem*. Even if the example does not constitute an "industrial-strength in-the-large" case study, we believe that, after several decades of successful application, it is still an insightful and thought-provoking example to assess the validity of our compositional rule. Section 4.3 presents other considerations about the significance of the example.

All details of the proofs have been checked with the encoding of the TRIO language in the PVS proof checker [28] (see [16, 12] for some details of this encoding), even if we present them succinctly and in human-readable form. The interested reader can find some more details about the proofs, as well as the full PVS encoding of them, in [14].

## 4.1 One Rely/Guarantee Philosopher

Formulas **availability**, **availability_2** and **lasting_availability** of Section 3.1 express the assumptions that each philosopher makes about the behavior of his/her neighbors. In turn, the philosopher must guarantee to them that he/she will not be unfair and will periodically release the forks. This requirement is expressed by the two theorems **taking_turns** and **taking_turns_2**, that are analogous to the assumptions **availability** and **availability_2**, while assumption **lasting_availability** corresponds to axiom **thinking_duration** (see Section 2).

**Theorem 16 (philosopher.taking_turns).**
$(\exists t \geq \mathrm{T_t} : \mathrm{Lasts}(\neg \mathsf{holding}(s), t)) \vee \mathrm{WithinF_{ei}}(\mathrm{Becomes}(\neg \mathsf{holding}(s)), \mathrm{T_t} + \mathrm{T_e})$

**Theorem 17 (philosopher.taking_turns_2).**
$\mathrm{WithinF}(\mathrm{UpToNow}(\neg \mathsf{holding}(s)), \mathrm{T_e})$

Up to this point, only assumptions expressed as temporally closed formulas (e.g., assumption **availability**) have been considered. Now, we have to express a new fundamental local rely/guarantee property of each philosopher: it is a non-starvation property requiring that, under the assumption of a regular availability of the forks, we can guarantee that, after the system starts, the philosopher eats regularly. In such a case, it is convenient to express the assumption as a time-dependent formula, and link it to the guarantee using the $\twoheadrightarrow$ operator: we relate the availability of the forks in the past to the occurrence of the eating sessions in the immediate future. Hence, let us take $E_k = \mathrm{WithinF_{ei}}(\mathrm{UpToNow}(\mathsf{available}(\mathrm{l}) \wedge \mathsf{available}(\mathrm{r})), \mathrm{T_t} + 2\mathrm{T_e})$ and $M_k = \mathrm{SomP_i}(\mathsf{start}) \Rightarrow (\exists t > \mathrm{t_e} : \mathrm{Within_{ii}}(\mathrm{Lasts}(\mathsf{eating}, t), \mathrm{T_t} + 2\mathrm{T_e}))$. Thus, the following theorem expresses the local non-starvation property in rely/guarantee form: as long as both forks are available within a bounded time interval (i.e., $\mathrm{T_t} + 2\mathrm{T_e}$), the philosopher is able to eat for a sufficiently long (i.e., $> \mathrm{t_e}$) time, within the same bound, provided the system has started.

**Theorem 18 (philosopher.regular_eating_rg).**
$\mathrm{WithinF_{ei}}(\mathrm{UpToNow}(\mathsf{available}(\mathrm{l}) \wedge \mathsf{available}(\mathrm{r})), \mathrm{T_t} + 2\mathrm{T_e})$
$\twoheadrightarrow \quad (\mathrm{SomP_i}(\mathsf{start}) \Rightarrow (\exists t > \mathrm{t_e} : \mathrm{Within_{ii}}(\mathrm{Lasts}(\mathsf{eating}, t), \mathrm{T_t} + 2\mathrm{T_e})))$

This theorem completes the specification of the **philosopher** class.

We can finally verify the whole specification, according to the method described in Section 3.3. Let us start from step (1), which prescribes to prove each local specification, i.e., each local theorem. Besides the lastly introduced theorem **regular_eating_rg**, we have a total of four theorems in class **philosopher**. Theorems **taking_turns[_2]** and **always_eating_or_not** are proved directly from the axioms of the class, while theorem **fork_availability** relies on some of the assumptions of the class. Moreover, in all proofs we assume that the thinking time of each philosopher is larger than twice the eating time: $\mathrm{T_t} > 2\mathrm{T_e}$.[10] This condition allows one to avoid the race conditions, and is referred to as assumption **TCCs** (for "Type Correctness Constraints", *à la* PVS). Similarly to what was discussed when presenting Theorem **fork_availability**, one might try to determine which are the "weakest" inequalities that have to be assumed for the following proof to hold. A detailed discussion of these issues is out of the scope of the present paper, and we leave it to future work.

In summary, the whole *proof dependencies* for the **philosopher** class are displayed in Figure 4: whenever we use a formula $\alpha$ in deducing the validity of another formula $\beta$, there is a proof dependency between the two formulas, graphically represented by an arrow going from $\alpha$ to $\beta$. For simplicity, we did not

[10] After all, they are philosophers, not gourmands! (Unless they are Epicureans, one may argue...).

represent the dependencies from axioms **thinking_def** and **hungry** in Figure 4, as they just define elementary equivalences. For the sake of brevity, we do not

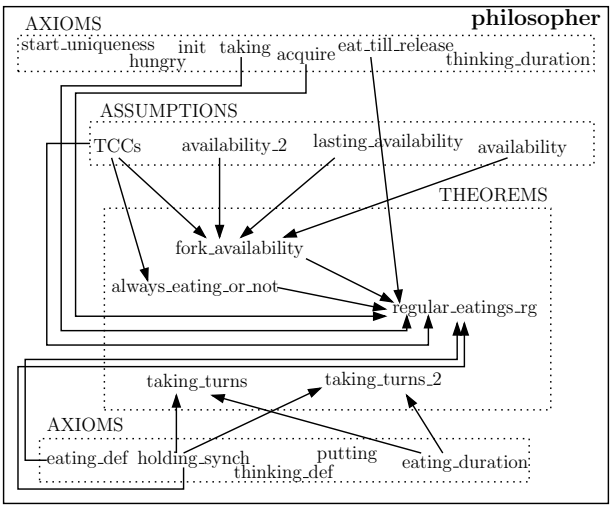


Figure 4: Proof dependencies in class **philosopher**

show the proof of any of these theorems, except for **regular_eating_rg** which is proved below, and whose overall case structure is given in Figure 5. The interested reader can find the proofs of the other theorems, in PVS format, in [15].

*Proof of* ***philosopher.regular_eating_rg****.* By exploiting the definition of the $\twoheadrightarrow$ operator, we assume $\mathrm{AlwP_e}(\mathrm{WithinF_{ei}}(\mathrm{UpToNow}(\mathsf{available(l)} \wedge \mathsf{available(r)}), \mathrm{T_t} + 2\mathrm{T_e}))$ as hypothesis and set our goal to proving $\mathrm{AlwP_i}(F)$ and $\mathrm{NowOn}(F)$ separately, where $F$ is $F \equiv (\exists t > \mathrm{t_e} : \mathrm{Within_{ii}}(\mathrm{Lasts}(\mathsf{eating}, t), \mathrm{T_t} + 2\mathrm{T_e})) \vee \mathrm{AlwP_i}(\neg\mathsf{start})$, an equivalent statement of the implication. First of all, we notice that, because of theorem **fork_availability** (11), we can actually strengthen our current hypothesis to be $\mathrm{AlwP_i}(\mathrm{WithinF_{ei}}(\mathrm{UpToNow}(\mathsf{available(l)} \wedge \mathsf{available(r)}), \mathrm{T_t} + 2\mathrm{T_e}))$, that is including the current instant. Now, let us first prove $\mathrm{AlwP_i}(F)$ from the hypothesis, the axioms and the other (already proved) theorems of the class. Let $t$ be the generic time instant at which the hypothesis holds. We have to prove that $F$ holds for all time instants less than or equal to $t$, so let $u \leq t$ a generic time instant before $t$. Let us consider theorem **always_eating_or_not** (15) at time $u$ and perform a case analysis. The proof is split into two branches whether $\mathsf{hungry} \equiv \mathrm{Lasted}(\neg\mathsf{eating}, \mathrm{T_t})$ or $\mathrm{WithinP_{ii}}(\mathrm{Becomes}(\mathsf{eating}) \vee \ldots, \mathrm{T_t} + \mathrm{T_e})$ holds at $u$.

The first branch considers the hypothesis instantiated at time $u$, that is $\mathrm{WithinF_{ei}}(\mathrm{UpToNow}(\mathsf{available(l)} \wedge \mathsf{available(r)}), \mathrm{T_t} + 2\mathrm{T_e})$ at $u$. Therefore, let us make explicit the existentially quantified time variable of the WithinF operator and name it $f$. Notice that $0 < f \leq \mathrm{T_t} + 2\mathrm{T_e}$ and we can write that

$\mathrm{UpToNow}(\mathsf{available}(\mathrm{l}) \wedge \mathsf{available}(\mathrm{r}))$ at time $u + f$. Now, the proof is further split into two branches whether or not the state eating never becomes true for all time instants since $u$ to $u + f$. In the first case $\mathrm{Lasts}_{\mathrm{ie}}(\neg\mathrm{Becomes}(\mathsf{eating}), f)$ at $u$. Therefore, we can deduce from basic properties of state items that eating is always false from $u$ to $u+f$. eating was also false for $\mathrm{T_t}$ time units in the past at $u$ in this branch of the proof. Therefore, a short time before $u+f$ the philosopher is hungry, and the forks are available in the immediate past and in the immediate future. Let $u + f - \epsilon$ be this time instant (where $\epsilon > 0$ is sufficiently small) and consider axiom **acquire** (3) at this time. We immediately conclude that both forks are taken at $u + f - \epsilon$ and therefore $\mathrm{Becomes}(\mathsf{holding}(\mathrm{l}) \wedge \mathsf{holding}(\mathrm{r}))$ is true at $u + f - \epsilon$. Furthermore, we can consider axiom **eat_till_release** (4) at time $u + f - \epsilon$ and deduce that there exists $r > \mathrm{t_e}$ such that $\mathrm{Lasts}(\mathsf{eating}, r)$ holds at $u+f-\epsilon$. Since $|f-\epsilon| \leq \mathrm{T_t}+2\mathrm{T_e}$ this branch of the proof is concluded.

The other branch of the proof considers the case in which there is a time instant $g$ between $u$ and $u + f$ where $\mathrm{Becomes}(\mathsf{eating})$ is true. Since eating becomes true at $g$, we can apply axiom **eat_till_release** (4) and deduce that there exists $r > \mathrm{t_e}$ such that $\mathrm{Lasts}(\mathsf{eating}, r)$ holds at $g$. Since $u \leq g < u + f \leq u + (\mathrm{T_t} + 2\mathrm{T_e})$, this branch of the proof is also concluded.

Let us now consider the case in which $\mathrm{WithinP_{ii}}(\mathrm{Becomes}(\mathsf{eating}) \vee (\exists t > \mathrm{t_e} : \mathrm{Lasted}(\mathsf{eating}, t)), \mathrm{T_t} + \mathrm{T_e})$ holds at $u$. This means that there is a generic time instant $v : u - (\mathrm{T_t} + \mathrm{T_e}) \leq v \leq u$ where $\mathrm{Becomes}(\mathsf{eating})$ or $\exists t > \mathrm{t_e} : \mathrm{Lasted}(\mathsf{eating}, t)$ holds. In the first case, axiom **eat_till_release** (4) lets us conclude that eating lasts for a time at least as long as $\mathrm{t_e}$ starting from time instant $v$. Since $v \geq u - (\mathrm{T_t} + \mathrm{T_e}) \geq u - (\mathrm{T_t} + 2\mathrm{T_e})$, this branch of the proof is concluded. In the second case, eating was true for $r > \mathrm{t_e}$ time units, starting from $v$ in the past; hence the whole branch of the proof is concluded.

The case $\mathrm{NowOn}(F)$ is proved similarly to the first branch, but with different base time instantiations and some technicalities that we do not discuss here. □

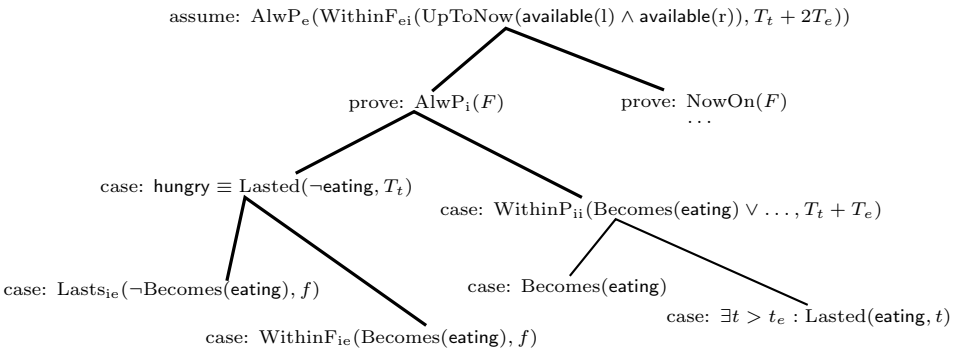


Figure 5: Proof structure for Theorem **regular_eating_rg**

This concludes the proof of $\mathcal{AX}_{\mathbf{phil}} \cup \mathcal{AS}_{\mathbf{phil}} \vdash \mathcal{TH}_{\mathbf{phil}}$, i.e., step (1) of Section 3.3.

## 4.2 A Table of Philosophers

Let us now complete the *composite specification* of the dining philosophers. More precisely, we formulate the global property to be verified, which is expressed by theorem **liveness_rg** of the composite class **dining_N**. It simply states that each philosopher in the array eats regularly, unless he/she has not started yet. Notice that in our example $E_{\text{glob}} = E_{\textbf{dining\_N}} = \text{true}$ since the composite system is closed, and $M_{\text{glob}} = M_{\textbf{dining\_N}}$ coincides with the following formula.

**Theorem 19 (dining_N.liveness_rg).**
$\forall k \in [0..N-1] : \quad (\ \text{SomP}_{\text{i}}(\textbf{Philosophers}[k].\mathsf{start})$
$\Rightarrow\ (\exists t > \text{t}_{\text{e}} : \text{Within}_{\text{ii}}(\text{Lasts}(\textbf{Philosophers}[k].\mathsf{eating}, t)\ , \text{T}_{\text{t}} + 2\text{T}_{\text{e}}))\ )$

This finally completes the global composite specification, so that we are now able to consider step (2) of the verification process. Each local assumption is discharged by a visible theorem or axiom of the modules adjacent to the current philosopher, as shown in Figure 6. Therefore, step (2) is completed without cir-

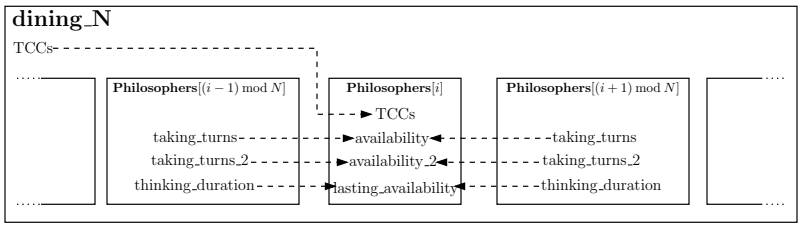


Figure 6: Discharging of assumptions in global class **dining_N**

cularities involved, since **thinking_duration** is an axiom and **taking_turns[_2]** are both proved directly from axioms local to each class (i.e., they do not rely circularly on other assumptions).

Step (3) is empty in our example, since the only global theorem we have to prove is theorem **liveness_rg** in $\mathcal{TH}^{\text{rg}}_{\textbf{dining\_N}}$.

Let us consider step (4), where we have to prove that local rely/guarantee assumption $E_k$ is initialized, that is we have to show that hypothesis (1) of Proposition 2 holds. Since the local theorems have already been proved without circularities, we can use **fork_availability** to complete this step. The theorem simply states that the desired property $E_k = \text{WithinF}_{\text{ei}}(\text{UpToNow}(\mathsf{available}(\text{l}) \wedge \mathsf{available}(\text{r})), \text{T}_{\text{t}}+2\text{T}_{\text{e}})$ always holds, which subsumes the initialization condition.

Step (5) requires to discharge the $E_k$'s by means of other formulas, in order to fulfill hypothesis (2) of Proposition 2. In this case as well, theorem **fork_availability** for module $k$ works correctly since it predicates the validity of the $E_k$'s over the whole temporal axis.

Step (6) is also very simple, since $M_{\text{glob}} = \forall k \in \{1, \ldots, n\} : M_k$ in our case, so that the implication of this step holds trivially. As a consequence, hypothesis (3) of Proposition 2 is shown to hold.

As discussed above and shown in the proof dependencies pictures, no circularities arise in proving the local formulas, so we conclude that theorem **liveness_rg** soundly holds as a consequence of the inference rule of Proposition 2 and according to the steps in Section 3.3.

## 4.3 Discussion

As hinted above, the actual verification was carried out with PVS support: this section aims at sketching an analysis of the complexity of the process. First, however, let us notice that the bare number of proof commands is hardly a complexity measure of some interest for comparing compositional rules and methods, as it is often strongly influenced by factors which do not pertain to the methodology, such as the encoding of the logic in the proof checker, the number of additional auxiliary lemmas generated in building the proofs, etc. Conversely, it would be meaningful to compare our proof with a non-compositional one, carried out with the same basic TRIO/PVS prover, in order to understand the benefits of a compositional methodology in building our verification, and in order to assess the significance of the philosophers example as a benchmark for compositional verification. Indeed, there are cases where non-compositional methods may perform better in practice then compositional ones; in such cases the additional verification burden required by a compositional framework is not traded off by any benefit in proof simplification or reuse [9, 20].

Under this respect, let us compare the complexity of our verification with the cost of a non-compositional one. The basic problem with a non-compositional proof is that we cannot exploit encapsulation and reuse. Therefore, there is no distinction between local and global items and everything is "flattened" at the same level of visibility. In the case of the dining philosopher problem, we can overcome this problem by "simulating" modularization at the global level. In other words, we have to carefully parameterize each item with respect to an index which separates different "instances" of the philosopher.

Moreover, and most importantly, we must devise a way to replace the use of the $\twoheadrightarrow$ operator by temporally closed formulas only. As we pointed out above, writing rely/guarantee formulas using the $\twoheadrightarrow$ operator permits to use an *ad hoc* inference rule which can handle circularities between assumptions and guarantees of the various local formulas without need for an *explicit* circularity breaking to be provided by the verifier. Therefore, replacing specifications written using the time progression operator is more difficult the tighter the circularity between the local $E_k$'s and $M_k$'s is. This is usually the case when we compose modules which are instances of the *same class*: having the same formulas by definition, the circularity is likely to arise between connected classes for symmetry reasons. This was the case in the philosopher example, albeit with a significant simplification, namely the fact that the local assumptions $E_k$'s can be shown to hold over the whole temporal axis, as discussed above. This simplification made it possible to build a non-compositional solution, though still with considerable effort. More generally, we point out that a system made of a (possibly large) set of instances of the same class, connected with some circularities, is repre-

sentative of a vast category of real systems: in particular, those representing communication protocols between a large number of hosts, such as peer-to-peer protocols. Thus, we believe that the philosophers problem can be regarded as a good abstraction of some of the core problems arising in verifying such systems.

We carried out the unstructured, non-compositional proof of the philosophers problem;[11] the result has been a proof of length roughly comparable to — or even a bit shorter than — the compositional one, but fragmented into more intermediate lemmas, with more assumptions and more intricate proof dependencies. Another feature that distinguishes the compositional proof from the non-compositional one is the fact that the former is repetitive while the latter is intricate. In other words, the compositional proof has a transparent, intelligible structure made of several similar parts, indicating that it is indeed simpler to manage for the human user who can easily understand when previous proof patterns can be applied again with minor modifications. All in all, even if the number of proof commands was not dramatically different in the two proofs, the complexity of the non-compositional one, considering also the mental effort and difficulty in managing the proof, was much greater. Furthermore, our experience with the compositional proof of the same property has guided and helped the building of the non-compositional one: we believe that doing the non-compositional proof first would have been really hard and time-consuming.

# 5 Related Works

A compositional analysis technique applies some, possibly formal, method to infer global properties of a large, complex system through a hierarchical and iterative process that exploits the system's modular structure. A general (and historical) introduction to compositional methods can be found in [8, 9], while the reader can refer to [13] for a comprehensive survey about compositionality for temporal logics. Without aiming at exhaustiveness, this section briefly reviews some of the most important contributions about compositional reasoning and shows how the approach of this paper differs from them.

An issue still largely unexplored in the present literature on compositionality is the consideration of hard real-time aspects, which require a metric modeling of time. A noticeable exception is Ostroff in [27], where the metric temporal logic RTTL is embedded in a compositional framework. Ostroff's work is also valuable in providing an overall framework, which includes an inference rule as well as a usable methodology. Nonetheless, the approach is rather different from ours, being focused on refinement aspects rather than on *a posteriori* composition that exploits reuse. Furthermore, time is treated as a separate variable and is discrete, while in our approach time is an implicit item of the language and can be either continuous or discrete.

The need for compositionality has become indisputable in the formal methods community, so that almost every newly introduced formalism encompasses some sort of compositional technique or permits compositional specifications.

[11] It is available in PVS form [15].

However, to the best of our knowledge, all proposed compositional frameworks are deeply rooted on some particular, often restrictive, semantic assumptions, and depend explicitly on the underlying computational model. In this regard, formalisms typically assume either an interleaving semantics (e.g., [1, 11, 26]) or a synchronous semantics (e.g., [3]) for the concurrent components of the system.

A rather different compositional framework to support the top-down development of real-time systems based on logical formulas at the semantic level is studied by Hooman [18]. In a sense, Hooman's framework is independent of semantic assumptions, even if its set-theoretic model of semantic primitives naturally relates to interleaving semantics models. However, the framework is focused on the refinement (i.e., decomposition) aspect and basically consists of an inference rule that allows one to deduce that the decomposition of a module into its refined parts correctly implements the original (unrefined) module. Another important difference between Hooman's framework and ours is that the former does not adopt the rely/guarantee paradigm, which is instead an often convenient, natural, and widely used paradigm to write specifications of open modules which rely on a constrained behavior of the environment to function correctly.

More typical solutions to the problem of formulating a sound rely/guarantee compositional rule involve the use of an *ad hoc* operator to write rely/guarantee specifications so that they satisfy certain specific characterizations. This solution was pursued also in our framework, with the time progression $\twoheadrightarrow$ operator and the related inference rule. However, the first main difference between our compositional framework and other ones using also a rely/guarantee operator is that most other frameworks consist just of one single inference rule. Often, this is a consequence of them being tailored mostly to simple use by automated techniques, especially model checking. On the contrary our approach provides the user with complementary mechanisms to express rely/guarantee specifications, which are integrated with some of TRIO's language features to support encapsulation and reuse.

For example, an *ad hoc* operator is introduced by Abadi and Lamport [1], who analyze the rely/guarantee compositional paradigm using TLA as the reference specification language. The authors use the TLA operator $\stackrel{+}{\triangleright}$ to write rely/guarantee specifications that can be soundly composed. A crucial difference between TLA's $\stackrel{+}{\triangleright}$ and TRIO's $\twoheadrightarrow$ is that our time progression operator is applied in inference rules independently of any assumption on the semantics of processes and also of any semantic characterization of formulas (its application does not need notions such as safety, closure, etc., which are instead integral part of (among others) Abadi and Lamport's framework). This renders our framework purely *syntactic* (i.e., independent of any semantic assumptions) and hence very general. In particular, even if the inference rule of [1] is usable for general properties, the conditions of the rule are hard to prove if they are not safety properties; such a distinction does not apply to a syntactic rule such as ours.

Abadi and Merz [2] propose an abstract generalization of rely/guarantee inference rules, in an attempt to treat compositionality syntactically. To this

extent, a modal operator to write rely/guarantee inference rules is introduced with minimal semantic assumptions. However, the use of the operator in inference rules and the ensuing soundness proofs are possible only *after* the abstract framework is specialized by choosing a semantic model and a computational model. On the contrary, in our framework the soundness of the inference rule is proved without assumptions of this kind.

An attempt to get rid of the difficulties inherent in applying rules which follow the "semantic approach" (such as Abadi and Lamport's, as well as several others) is the work by Jonsson and Tsay [19], and Tsay [31]. Their approach is based on plain LTL, and aims at introducing a syntactical characterization of the semantic properties used to guarantee the soundness of compositional reasoning. In particular, they handle safety requirements by imposing that the assumptions and guarantees are written in a *canonical form*; when using this form, it is possible to translate the semantic requirements of a compositional inference rule into syntactic obligations, which can then be discharged using usual deductive techniques. Our framework also pursues a "syntactic approach" to compositionality. Contrarily to Jonsson and Tsay, however, we do not try to "translate" semantic methods into syntactic ones; instead, we focus exclusively on a simple syntactic approach which avoids intricacies. As a result our rule does not use complex canonical forms, and is more general: in particular we do not have to distinguish between safety and liveness properties, hence we do not need to prescribe a different approach for them.

Amla et al. [4] present an abstract compositional framework which can be considered as a generalization of several concrete compositional frameworks in the literature. In particular, they succeed in formulating a simple inference rule whose soundness does not rely on an *ad hoc* operator. However, their framework still relies on certain semantic assumptions, such as downward closure, on the set of behaviors describing a process. Therefore, our framework does not fit the models in [4].

# 6 Conclusions

We presented a compositional framework for the TRIO specification language that supports verification through automated theorem proving. The framework is based on a formal notion of composition of TRIO modules, which is used to prove that the mutual interactions between components of a complex system guarantee some property for the global application, after the components are integrated into the system. The compositional rule has been proved sound and has been applied to the classic example of Dijkstra's dining philosophers as a simple, but not simplistic, benchmark. The compositional framework has been encoded into the logic of the PVS theorem prover.

With respect to other approaches to compositionality in formal methods, ours emerges as more suitable for real-time modeling, it encompasses both continuous and discrete time to better model physical processes, and it is conceived for axiom systems and deductive verification. Therefore, the approach is very

general and abstracts away from specific assumptions about process semantics and the underlying computational model. Moreover, it encompasses a useful methodology to combine the inference rules with the specific modular features of the TRIO language, resulting in a rich compositional framework.

Future work in this line of research will follow three main directions. First, the framework presented here is being applied to several real-life industrial case studies to experimentally evaluate its effectiveness. Second, alternative weaker — or stronger — inference rules will be investigated. In particular, we are exploring variations and generalizations of the $\twoheadrightarrow$ operator, better suited to be applied on certain classes of systems, different inference rules which do not use a time progression operator at all, and complete inference rules (which may sacrifice some simplicity). Third, the automated support for the framework will be improved and extended.

**Acknowledgements**

The authors would like to thank the anonymous referees for their detailed suggestions, and the guest editors for their efforts in promoting and managing the realization of this special issue.

# References

[1] M. Abadi and L. Lamport. Conjoining specifications. *ACM Transactions on Programming Languages and Systems*, 17(3):507–535, 1995.

[2] M. Abadi and S. Merz. An abstract account of composition. In *Proceedings of MFCS'95*, volume 969 of *Lecture Notes in Computer Science*, pages 499–508, 1995.

[3] R. Alur and T. A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1):7–48, 1999.

[4] N. Amla, E. A. Emerson, K. S. Namjoshi, and R. J. Trefler. Abstract patterns of compositional reasoning. In *Proceedings of CONCUR'03*, volume 2761 of *Lecture Notes in Computer Science*, pages 431–445, 2003.

[5] E. Ciapessoni, A. Coen-Porisini, E. Crivelli, D. Mandrioli, P. Mirandola, and A. Morzenti. From formal models to formally-based methods: an industrial experience. *ACM Transactions on Software Engineering and Methodology*, 8(1):79–113, 1999.

[6] S. A. Cook. Soundness and completeness of an axiom system for program verification. *SIAM Journal on Computing*, 7(1):70–90, 1978. *Corrigendum* in [7].

[7] S. A. Cook. Corrigendum: Soundness and completeness of an axiom system for program verification. *SIAM Journal on Computing*, 10(3):612, 1981.

[8] W.-P. de Roever. The need for compositional proof systems: a survey. In *Proceedings of COMPOS'97*, volume 1536 of *Lecture Notes in Computer Science*, pages 1–22, 1998.

[9] W.-P. de Roever, F. de Boer, U. Hannemann, J. Hooman, Y. Lakhnech, M. Poel, and J. Zwiers. *Concurrency Verification: Introduction to Compositional and Noncompositional Methods*. Cambridge University Press, 2001.

[10] E. W. Dijkstra. Hierarchical ordering of sequential processes. *Acta Informatica*, 1(2):115–138, 1971.

[11] B. Finkbeiner, Z. Manna, and H. B. Sipma. Deductive verification of modular systems. In *Proceedings of COMPOS'97*, volume 1536 of *Lecture Notes in Computer Science*, pages 239–275, 1998.

[12] C. A. Furia. Compositional proofs for real-time modular systems. Laurea degree thesis, Politecnico di Milano, 2003.

[13] C. A. Furia. A compositional world: a survey of recent works on compositionality in formal methods. Technical Report 2005.22, Dipartimento di Elettronica e Informazione, Politecnico di Milano, 2005.

[14] C. A. Furia, M. Rossi, D. Mandrioli, and A. Morzenti. Automated compositional proofs for real-time systems. FASE'05 full version with appendices available online from `http://www.elet.polimi.it/upload/furia`, 2005.

[15] C. A. Furia, M. Rossi, D. Mandrioli, and A. Morzenti. TRIO/PVS proofs of the dining philosophers example. Available online from `http://www.elet.polimi.it/upload/furia` (publications section), 2005.

[16] A. Gargantini and A. Morzenti. Automated deductive requirement analysis of critical systems. *ACM Transactions on Software Engineering and Methodology*, 10(3):255–307, 2001.

[17] C. Ghezzi, D. Mandrioli, and A. Morzenti. TRIO: A logic language for executable specifications of real-time systems. *The Journal of Systems and Software*, 12(2):107–123, 1990.

[18] J. Hooman. Compositional verification of real-time applications. In *Proceedings of COMPOS'97*, volume 1536 of *Lecture Notes in Computer Science*, pages 276–300, 1998.

[19] B. Jonsson and Y.-K. Tsay. Assumption/guarantee specifications in linear-time temporal logic. *Theoretical Computer Science*, 167(1–2):47–72, 1996.

[20] L. Lamport. Composition: A way to make proofs harder. In *Proceedings of COMPOS'97*, volume 1536 of *Lecture Notes in Computer Science*, pages 402–423, 1998.

[21] P. Maier. Compositional circular assume-guarantee rules cannot be sound and complete. In *Proceedings of FOSSACS'03*, volume 2620 of *Lecture Notes in Computer Science*, pages 343–357, 2003.

[22] E. Mendelson. *Introduction to Mathematical Logic*. Chapman & Hall, 1997.

[23] B. Meyer. *Object-Oriented Software Construction*. Prentice Hall, 2nd edition, 1997.

[24] A. Morzenti, D. Mandrioli, and C. Ghezzi. A model parametric real-time logic. *ACM Transactions on Programming Languages and Systems*, 14(4):521–573, 1992.

[25] A. Morzenti and P. San Pietro. Object-oriented logical specification of time-critical systems. *ACM Transactions on Software Engineering and Methodology*, 3(1):56–98, 1994.

[26] K. S. Namjoshi and R. J. Trefler. On the completeness of compositional reasoning. In *Proceedings of CAV'00*, volume 1855 of *Lecture Notes in Computer Science*, pages 139–153, 2000.

[27] J. S. Ostroff. Composition and refinement of discrete real-time systems. *ACM Transactions on Software Engineering and Methodology*, 8(1):1–48, 1999.

[28] S. Owre, J. M. Rushby, and N. Shankar. PVS: A Prototype Verification System. In *Proceedings of CADE-11*, volume 607 of *Lecture Notes in Computer Science*, pages 748–752, 1992.

[29] P. San Pietro. Completezza di un sistema di assiomi per TRIO. Unpublished manuscript (in Italian), 1992.

[30] N. Shankar. Lazy compositional verification. In *Proceedings of COMPOS'97*, volume 1536 of *Lecture Notes in Computer Science*, pages 541–564, 1998.

[31] Y.-K. Tsay. Compositional verification in linear-time temporal logic. In *Proceedings of FOSSACS'00*, volume 1784 of *Lecture Notes in Computer Science*, pages 344–358, 2000.

# A TRIO Specification of the philosopher Class

**class philosopher** (**const** $\mathrm{t_e}$, **const** $\mathrm{T_e}$, **const** $\mathrm{T_t}$)

**signature:**

**visible:** start, eating, holding, available, thinking;

**temporal domain:** $\mathbb{R}$;

**domains:** $\mathrm{S} : \{\mathrm{l}, \mathrm{r}\}$;

**items:**

**event** start, take(S), release(S);

**state** eating, holding(S), available(S), thinking, hungry;

**formulae:**

**vars:** $s : \mathrm{S}, t : \mathbb{R}$;

**axioms:**

**holding_synch:** $\mathsf{holding}(\mathrm{l}) \Leftrightarrow \mathsf{holding}(\mathrm{r})$;

**hungry:** $\mathrm{Lasted}(\neg\mathsf{eating}, \mathrm{T_t}) \Leftrightarrow \mathsf{hungry}$;

**acquire:** $\mathsf{hungry} \wedge \mathrm{UpToNow}(\mathsf{available}(\mathrm{l}) \wedge \mathsf{available}(\mathrm{r}))$
$\Rightarrow\ (\mathsf{take}(\mathrm{l}) \wedge \mathsf{take}(\mathrm{r})) \vee \mathrm{NowOn}(\neg\mathsf{available}(\mathrm{l}) \vee \neg\mathsf{available}(\mathrm{r}))$;

**eat_till_release:** $\mathrm{Becomes}(\mathsf{holding}(\mathrm{l}) \wedge \mathsf{holding}(\mathrm{r}))$
$\Rightarrow\ (\exists t > \mathrm{t_e} : \mathrm{Lasts}(\mathsf{eating}, t) \wedge \mathrm{Futr}(\mathsf{release}(\mathrm{l}) \wedge \mathsf{release}(\mathrm{r}), t))$;

**eating_duration:** $\mathrm{Lasted}(\mathsf{eating}, t) \Rightarrow t < \mathrm{T_e}$;

**eating_def:** $\mathsf{holding}(\mathrm{l}) \wedge \mathsf{holding}(\mathrm{r}) \Leftrightarrow \mathsf{eating}$;

**thinking_def:** $\mathsf{thinking} \Leftrightarrow \neg\mathsf{eating}$;

**thinking_duration:** $\mathrm{Becomes}(\mathsf{thinking}) \Rightarrow \mathrm{Lasts}(\mathsf{thinking}, \mathrm{T_t})$;

**taking:** $\mathsf{take}(s) \wedge \mathrm{UpToNow}(\neg\mathsf{holding}(s)) \wedge \mathrm{UpToNow}(\mathsf{available}(s))$
$\Rightarrow\ \mathrm{Until}(\mathsf{holding}(s), \mathsf{release}(s))$;

**putting:** $\mathsf{release}(s) \wedge \mathrm{UpToNow}(\mathsf{holding}(s)) \ \Rightarrow\ \mathrm{Until}(\neg\mathsf{holding}(s), \mathsf{take}(s))$;

**assumptions:**

**TCCs:** $\mathrm{T_e} > \mathrm{t_e} > 0 \ \wedge\ \mathrm{T_t} > 2\mathrm{T_e}$;

**availability:** $(\exists t \geq \mathrm{T_t} : \mathrm{Lasts}(\mathsf{available}(s), t))$
$\vee\ \mathrm{WithinF_{ei}}(\mathrm{Becomes}(\mathsf{available}(s)), \mathrm{T_t} + \mathrm{T_e})$;

**availability_2:** $\mathrm{WithinF}(\mathrm{UpToNow}(\mathsf{available}(s)), \mathrm{T_e})$;

**lasting_availability:** $\mathrm{Becomes}(\mathsf{available}(s)) \Rightarrow \mathrm{Lasts}(\mathsf{available}(s), \mathrm{T_t})$;

**theorems:**

**taking_turns:** $(\exists t \geq \mathrm{T_t} : \mathrm{Lasts}(\neg\mathsf{holding}(s), t))$
$\vee\ \mathrm{WithinF_{ei}}(\mathrm{Becomes}(\neg\mathsf{holding}(s)), \mathrm{T_t} + \mathrm{T_e})$;

**taking_turns_2:** $\mathrm{WithinF}(\mathrm{UpToNow}(\neg\mathsf{holding}(s)), \mathrm{T_e})$;

**fork_availability:**
$\mathrm{WithinF_{ei}}(\mathrm{UpToNow}(\mathsf{available}(\mathrm{l}) \wedge \mathsf{available}(\mathrm{r})), \mathrm{T_t} + 2\mathrm{T_e})$;

**always_eating_or_not:** $\mathsf{hungry}$
$\vee\ \mathrm{WithinP_{ii}}((\exists t > \mathrm{t_e} : \mathrm{Lasted}(\mathsf{eating}, t))$
$\vee \mathrm{Becomes}(\mathsf{eating}), \mathrm{T_t} + \mathrm{T_e})$;

**regular_eating_rg:**
$\mathrm{WithinF_{ei}}(\mathrm{UpToNow}(\mathsf{available}(\mathrm{l}) \wedge \mathsf{available}(\mathrm{r})), \mathrm{T_t} + 2\mathrm{T_e})$
$\twoheadrightarrow\ (\mathrm{SomP_i}(\mathsf{start}) \Rightarrow (\exists t > \mathrm{t_e} : \mathrm{Within_{ii}}(\mathrm{Lasts}(\mathsf{eating}, t), \mathrm{T_t} + 2\mathrm{T_e})))$;

**end**

# B TRIO Specification of the dining_N Class

**class dining_N** (**const** $\mathrm{t_e}$, **const** $\mathrm{T_e}$, **const** $\mathrm{T_t}$, **const** $N$)
**import:** philosopher;
**signature:**
**visible:** start;
**temporal domain:** $\mathbb{R}$;
**items:** **event** start;
**modules:**
Philosophers: **array** $[0..N-1]$ **of** philosopher
$[\mathrm{t_e}$ **is const** $\mathrm{t_e}$, $\mathrm{T_e}$ **is const** $\mathrm{T_e}$, $\mathrm{T_t}$ **is const** $\mathrm{T_t}]$;
**connections:**
**vars:** $i : [0..N-1]$;
(**direct** Philosophers$[i]$.available(l) ,
$\neg$Philosophers$[(i-1) \mod N]$.holding(r)),
(**direct** Philosophers$[i]$.available(r) ,
$\neg$Philosophers$[(i+1) \mod N]$.holding(l)),
(**broadcast** start, Philosophers.start);
**formulae:**
**vars:** $k : [0..N-1], t : \mathbb{R}$;
**assumptions:**
**TCCs:** $\mathrm{T_e} > \mathrm{t_e} > 0 \ \wedge \ \mathrm{T_t} > 2\mathrm{T_e} \ \wedge \ N \geq 2$;
**theorems:**
**liveness_rg:** $\forall k \in [0..N-1] : (\ \mathrm{SomP_i}(\mathrm{Philosophers}[k].\mathsf{start})$
$\Rightarrow \ (\exists t > \mathrm{t_e} : \mathrm{Within_{ii}}(\mathrm{Lasts}(\mathrm{Philosophers}[k].\mathsf{eating}, t)\ , \mathrm{T_t} + 2\mathrm{T_e})))$;
**end**