A Compositional World

a survey of recent works on compositionality in formal methods

Carlo Alberto Furia

March 2005

Abstract

We survey the most significant literature about compositional techniques for concurrent and real-time system verification. We especially focus on abstract frameworks for rely/guarantee compositionality that handles circularity, but also consider different developments.

Contents

1 Definitions of Compositionality 3

2 Early Works on Compositional Methods 6

3 General Circular Rely/Guarantee Compositional Rule 7

4 Abstract Frameworks for Rely/Guarantee Compositionality 8
4.1 Abadi and Lamport, 1993 [2] 8
4.2 Abadi and Lamport, 1995 [4] 10
4.3 Abadi and Plotkin, 1993 [6] 12
4.4 Abadi and Merz, 1995 [5] 13
4.5 Xu, Cau, and Collette, 1994 [85] 14
4.6 Cau and Collette, 1996 [18] 15
4.7 Jonsson and Tsay, 1996 [51] 16
4.8 Tsay, 2000 [82] 18
4.9 Namjoshi and Trefler, 2000 [71] 19
4.10 Viswanathan and Viswanathan, 2001 [84] 21
4.11 Maier, 2001 [62] 23
4.12 Maier, 2003 [63] 24
4.13 Amla, Emerson, Namjoshi, and Trefler, 2003 [10] 26

5 Abstract Non-Rely/Guarantee Methods 29
5.1 A Compositional Proof Method for Assertion Networks 29
5.2 The Lazy Approach to Compositionality 29
5.3 Decomposition of Real-Time Transition Systems 32
5.4 A Semantic Framework for Compositionality 33

6 Compositionality in Model Checking 35

7 Compositionality for Your Own Formalism 38
7.1 Reactive Modules 38
7.2 TTM/RTTL 39
7.3 Interval-Based Logics 41

This report surveys the most significant works about compositional formal techniques and methods for concurrent and real-time systems which have been published, approximately, during the last decade. In particular, the focus is on abstract frameworks for rely/guarantee compositionality for temporal logic languages, although other contributions are also considered.

The report is organized as follows. Section 1 introduces compositionality by providing some definitions of the term and by exposing the most relevant aspects of the problem of compositional verification. Section 2 briefly reports about the papers which have introduced the idea of compositionality, and especially the rely/guarantee paradigm, for the verification of concurrent systems. Section 3 presents an abstract general circular rely/guarantee compositional rule which is the template for most rely/guarantee compositional rules proposed in the literature on the subject. Section 4 reviews the contribution in the recent literature about abstract frameworks for rely/guarantee compositionality, showing similarities and common traits of the various inference rules, trying to fit them to the previously introduced template rule. Section 5 reviews some approaches to compositional verifications that are alternative to the (widely used) rely/guarantee method. Section 6 presents the essential traits of the compositional methods for model checking. Finally, Section 7 presents a (short) selection of compositional methods embedded in some disparate formalisms (in particular, other than temporal logics).

1 Definitions of Compositionality

The term *compositionality* and the idea of composition encompass a large variety of methods and techniques that aim at describing how to compose systems to form larger systems, and how to manage the resulting complexity of the overall system. Clearly, this broad objective can be considered from several, disparate, very different perspectives. In order to understand this better, and to focus on the issue that are of our interest, let us try to identify the dimensions that characterize the compositionality problem.

Let us first attempt a very general (and hence necessarily, to some extent, vague) definition of compositionality. In lay terms, the Merriam-Webster Dictionary of English [66] defines the verb *compose* as "to form by putting together". This basic definition encloses the idea at the root of compositionality: forming a system by putting together smaller subsystems.

A more technical definition reflecting this idea is that usually adopted in the philosophy of language, as reported by Janssen [48]:

> "The meaning of a compound expression is a function of the meanings of its parts and of the rule by which the parts are combined".

This compositionality principle is also called *Frege's principle*, from the name of the author who first formulated the principle [32].

This very same idea can be adapted to program verification: de Roever et al. [26, pg. 48] define compositionality as:

> "That a program meets its specification should be verified on the basis of specifications of its constituent components only, without additional need for information about the interior construction of those components".

Therefore, in compositional verification we aim at verifying the global specification of a system by *composing* the specifications which are local to the various components of the system.

Adopting a rather liberal — but still technical — definition, compositionality consists in bringing the well-known engineering (and software engineering in particular) practices of modularization and abstraction from the specification to the verification domain. Hence, we propose the following general definition of compositionality:

> "The techniques and methods that permit the modularization of the verification process of a large system".

As it is simple to realize, a huge number of different techniques and methods can be classified under the label "compositionality". In order to discern among the members of this large variety, and to focus on those which are of more interest to us, let us list the relevant aspects to be considered when analyzing a compositional method.

Time Model This aspect considers continuous or discrete time, real-time (i.e. metric) or untimed (i.e. without metric) time models, the linear or branching nature of time, etc.

Computational Model This aspect considers issues such as whether we adopt a semantics based on states, on transition systems, an agent-based view, etc.

Model of Concurrency Choosing the model of concurrency involves considering issues such as synchronous vs. asynchronous, interleaving semantics, lower-level models of concurrency such as shared-variables concurrency and message passing systems.

Methodology Basically, we have the two aspects of refinement (a.k.a. compositionality in strict sense [87, 25], decomposing [4], *a priori* or top-down development) and module composition/reuse (a.k.a. modularity [25] or modular completeness [87], composing [4], *a posteriori* or bottom-up development).

Specification Model This aspect refers to the abstraction mechanisms introduced in the model that constitutes the specification. For example we may use an assertional (a.k.a. denotational) approach, or an operational one, or a dual-language one (which combines an operational formalism with an assertional one), etc.

Implementation Language This aspect is only relevant if we are verifying implemented programs. In such case, the formal framework should give a formal semantics to the programming language.

Syntax vs. Semantics This refers to the general approach of the compositional method we are considering. We say it is *syntactic* if it is defined in terms of the language used in the formal description of the model, using rules that refers to the syntactic structure of the formulas, without reference to semantic concepts such as safety/liveness characterization, closures, etc. On the contrary, we label a method *semantic* if its applicability and justification are based on these latter concepts, usually expressed in terms of set-theoretic concepts.

Approach to Managing Complexity To the extremes, we can identify two ways in which a compositional method can mitigate the complexity of verification of a composite system. The first approach tries to restrict the model sacrificing expressiveness for simplicity. In other words, the user cannot specify all kind of modules with this approach, but when he/she can do so, the method brings a provable simplification in the complexity of verification. The other approach instead leaves the user with more expressiveness, generality and freedom, but in some cases the method may be of no practical usefulness, if not even detrimental. Obviously, what we encounter in practice in most methods is usually a trade-off between these two approaches, so that the distinction is really a fuzzy one.

How to Make a Module Compositional This aspect chooses among the rely/guarantee paradigm (dating back to Jones [50] and Misra and Chandi [67]), or the lazy methodology proposed by Shankar [79], or other original approaches such as the ones by Hooman [46], Manna et al. [16], etc.

Semantics of Composition The choice made by most authors is to take composition as conjunction of specifications for denotational formalism, and parallel composition for operational ones. Other, less common, variants are possible.

Table 1 schematically summarizes the above aspects. In italic are shown the choices we focused on in our own contribution to compositionality in [34]. There, we present a framework which works equally well for continuous and discrete time, and relies on no assumptions about the computational and concurrency model. Moreover, it is primarily syntactic, considers linear time and abstract specifications, that is no implementation aspect is considered.

Issue	Possible choices
Time Model	*continuous*, *discrete*, *real-time*, untimed, *linear*, branching, etc.
Computational Model	state-based, agent-based, transition system, etc.
Concurrency Model	synchronous, asynchronous, interleaving, message passing, shared variables, etc.
Methodology	refine or *compose*
Specification	*denotational*, operational, dual-language, etc.
Implementation	language used for implementation
Syntax vs. Semantics	*language* vs. (semantic) model
Manage Complexity	restrict vs. *guide*
Compositionality Model	*rely/guarantee*, lazy, hybrid, etc.
Composition Semantics	*conjunction*, parallel, etc.

Table 1: Dimensions of the Compositional World

As a final remark, note that compositionality is not always effective in reducing the complexity of specification and verification. In particular, as noted in [26], noncompositional methods are more effective in some scenarios, and in particular when a high coupling between modules is present, so that it is difficult to encapsulate the modules with adequate abstractions. An important example of such situation is the case with mutual-exclusion protocols and algorithms; in such cases global methods such as that by Owicki and Gries [77][1], coupled with heuristics for simplifications as in the work by Feijen and van Gasteren [28], usually give the best practical results.

2 Early Works on Compositional Methods

The first attempts at the formal verification of programs date back to the origins of computer science itself, with the pioneering works of Goldstine and von Neumann [36], and Turing [83], in the second half of the '40s. According to de Roever et al. [26], a constant trend in developing methods for proving program correctness has been from *a posteriori* nonstructured methods to structured compositional techniques. For a detailed historical account of compositional and noncompositional methods and techniques for program verification, we refer the reader to the aforementioned book by de Roever et al. [26], and to the surveys by de Roever and Hooman [24, 47, 25].

This development is rather easy to sketch for *sequential* program verification, where the first method is due to Floyd in 1967 [30] and is noncompositional, whereas Hoare reformulated the method in an axiomatic compositional style in 1969 [41].

On the other hand, the first practical noncompositional methods for the verification of *concurrent* programs are due to Ashcroft [13], Owicki and Gries [77], and Lamport [56], during the mid '70s. Concurrency raises several technical difficulties which render the development of methods, as well as their compositional extensions, harder than in the sequential case. Indeed, the first compositional methods for concurrent program verification appeared at the end of the '70s, in the contributions by Francez and Pnueli [31], Jones [50] and Misra and Chandy [67].

In particular, the works by Jones [50] and by Misra and Chandy [67] introduced the rely/guarantee paradigm to specify open reactive systems. It was Jones who introduced the terminology *rely/guarantee* in [49, 50], where he proposed a method for the shared-variable model of concurrency. His work was based on the previous work by Francez and Pnueli [31].

On the other hand, Misra and Chandy proposed in [67] a similar paradigm for concurrency models based on synchronous communication, and called it *assumption/commitment*.

Later, Abadi and Lamport proposed to call both compositional paradigms *assume/guarantee* [2], since they embody the same idea. Several of the works reviewed in Section 4 show in detail how the two paradigms of rely/guarantee and assumption/commitment can be unified. Because of this reason, in all of this paper we will not distinguish in the terminology between rely/guarantee, assumption/commitment and assume/guarantee: we choose to use only the term "rely/guarantee", in accordance with the name given to the first formulation of the principle [49].

Finally, in this short recount of first works on compositionality for concurrent systems, it is worth mentioning the work by Stark, who formulated the first

[1]Similar methods for different formal models of concurrency are for example [56] for inductive assertion networks and [12, 61] for synchronous message passing models.

theory of *refinement* for concurrent processes in 1988 [80].

3 General Circular Rely/Guarantee Compositional Rule

We present a general "template" compositional inference rule which:

- treats composition (rather than decomposition and refinement);
- uses specifications in the rely/guarantee style;
- is circular (when a compositional rule treats circularity is also called "strong", according to [4]).

We are considering the following setting. We have two abstract connectives, represented by the symbols $\sqsubseteq$ and $\sqcap$, to describe entailment and composition. More precisely, the *compose* connective $\sqcap$ permits to put together two specifications to form one. For example, the formula $A \sqcap B$ represents a specification obtained by putting together the specifications A and B. On the other hand, the *entail* connective $\sqsubseteq$ links two specifications in a manner that represents some form of realization or implication relation. So, for example, a formula $A \sqsubseteq B$ represents the fact that A "satisfies" (in a certain sense) the specification B. In this section we give no formal meaning to these connectives and to the idea of module and specification, since we are only introducing a syntactic characterization to express certain formulas and inference rules.

Let us now compose $n > 0$ modules to form a larger system. Each module i has a specification which consists of an assumption E_i and a guarantee M_i, for $i = 1, \ldots, n$. Moreover, the composite global system also has a global assumption E and a global guarantee M. We formulate an inference rule that deduces the validity of the global specification as a consequence of the local specifications. More precisely, we have the following.

Proposition 1 (General Composition Inference Rule). *If:*

1. *for all* $i = 1, \ldots, n$: $E_i \sqsubseteq M_i$
2. $E \sqcap \bigsqcap_{i=1}^{n} M_i \sqsubseteq \bigsqcap_{i=1}^{n} E_i$
3. $E \sqcap \bigsqcap_{i=1}^{n} M_i \sqsubseteq M$

then: $E \sqsubseteq M$.

The informal meaning of the hypotheses of Proposition 1 is as follows:

1. every component satisfies its rely/guarantee local specification;
2. the composition of the guarantees of all the components satisfies (i.e. discharges) the assumptions of all the components, under the hypothesis that the global assumption also holds;
3. the composition of the guarantees of all the components satisfies (i.e. implements) the global guarantee.

Under these hypotheses, the proposition concludes that the global rely/guarantee specification $E \sqsubseteq M$ holds.

In general, the deduction of Proposition 1 is unsound, because of the circularity between the discharging of assumptions in hypothesis 2 and the local specifications in hypothesis 1. In order to build a sound rule, we have to strengthen the proposition by introducing some additional assumptions, of some kind. Usually, the various works introduce *semantic* assumptions about the underlying computational model, about the time model, or about the class of the formulas representing the assumptions and guarantees.

4 Abstract Frameworks for Rely/Guarantee Compositionality

This section surveys several works about abstract compositional frameworks based on rely/guarantee inference rules that handle circularity. We try to show how all of these frameworks are fundamentally based on an inference rule which is a particular instance of the general rule of Proposition 1, to which each framework adds specific additional hypotheses for soundness.

The works are presented roughly in chronological order, but listing sequentially works by the same authors which can be considered successive developments over the same ideas.

4.1 Abadi and Lamport, 1993 [2]

This work presents a purely semantic analysis of compositionality, independent of any particular specification language or logic. This permits a general analysis of compositionality. However, the approach also has some drawbacks, which arise mainly from two aspects. On the one hand, the analysis is probably too abstract, that is it is far from immediate applicability with a given formalism. This is also briefly acknowledged by the authors themselves in [4]. By "too abstract" we also imply that its theoretical results have mostly intellectual (rather than practical) interest. In fact, we consider compositionality a *practical* simplification of difficult problems. On the other hand, the analysis done by Abadi and Lamport still requires a number of focused semantic assumptions, which mimic the features of the semantics of *some* specification formal language. As a result, mapping this general framework onto a given specification language may introduce additional complications and intricacies, due to the fact that a given language usually comes with its own semantics, which may be difficult to adapt or circumvent to match the general hypotheses of the paper.

Let us briefly outline the main semantic hypotheses of the model.

- Behaviors of described systems are infinite discrete sequences of states from a certain (unspecified) finite set.
- Any state change is triggered by agents. Agents can be divided into sets, which reflect their "nature". Notice that agents are an abstraction which typically translates to separated input and output variables, in a lower level formalism.
- An interleaving semantics for composition of behaviors is assumed.

- Stuttering-equivalence is also required, that is two behaviors are considered equivalent if they are equal after replacing every maximal sequence composed entirely of the same state with one single instance of that state.
- Fairness conditions (also known as progress properties) are usually introduced in a specification.
- Initialization predicates are also considered, so that the system has a notion of "start".

Let us now see how the general inference rule of Proposition 1 is modified by introduction of additional hypotheses.

- Each module is characterized by a set of agents that perform the state transition the module is responsible of. The agent sets for the various modules are assumed to be disjoint.
- Each environment assumption must be a safety property, constraining only agents which do not belong to any module (i.e. they belong to the environment). The authors note that, since it is well known that any property in temporal logic can be expressed as the intersection of a safety and a liveness property, it is possible to move the liveness part of the environment assumption to the guarantee part of the rely/guarantee specification. However, this consideration is of intellectual interest only, as the authors themselves admit in [4]. In fact, again, we must bear in mind that compositionality only has its strengths in being a *practically* usable technique, which is not the case if it forces one to write specifications in an unnatural and obfuscated manner. Moreover, notice that the requirement on safety is not immediately extensible to metric temporal logic, where the classification safety/liveness should be extended and/or reconsidered.
- The operators $\sqsubseteq$ and $\sqcap$ of the abstract rule of Proposition 1 are mapped to the set-theoretic operators $\subseteq$ and $\cap$. This is the most natural choice, since we are dealing with a semantic model where properties are sets.
- A rely/guarantee specification is written using the operator $\Rightarrow$, where $P \Rightarrow Q$ is a shorthand for the set-theoretic expression $(\mathcal{B} - P) \cup Q$, where $\mathcal{B}$ denotes the set of all behaviors.
- Condition 1 is replaced by one involving the *realizable part* of the rely/guarantee expression $E_i \Rightarrow M_i$, denoted as $\mathcal{R}(E_i \Rightarrow M_i)$. Without introducing too many technical details, let us just say that this requires the property to be implementable, that is to be the outcome of a deterministic strategy.
- In condition 2, we consider the safety closure $\mathcal{C}(M_i)$ of the M_i. Moreover, we assume that the safety closure of each M_i does not constrain any agent outside the agent set for the module i.
- Condition 3 is not applied, since the rule actually proves $E \sqsubseteq \bigsqcap_{i=1}^{n} M_i$ instead of $E \sqsubseteq M$.

To sum up, the soundness is based on a discrete state structure, on which to perform induction, on the safety of the environment assumptions and on the distinction between input and output variables. Here it is the inference rule given in [2], after removing explicit mentions to the agents.

Proposition 2 (Abadi and Lamport, 1993 [2]). *If:*

1. *for all $i = 1, \dots, n$: $\mathcal{R}(E_i \Rightarrow M_i)$*
2. *$E \cap \bigcap_{i=1}^{n} \mathcal{C}(M_i) \subseteq \bigcap_{i=1}^{n} E_i$*
3. *E and E_i, for all $i = 1, \dots, n$, are safety properties.*

then: $\mathcal{R}(E \Rightarrow \bigcap_{i=1}^{n} M_i)$.

The rule was given some (partial) tool support in [37], using the theorem prover HOL.

4.2 Abadi and Lamport, 1995 [4]

With reference to the specification language TLA, Abadi and Lamport propose in [4] a sound composition theorem. As it is customary, the compose operator $\sqcap$ is embodied by logical conjunction $\wedge$, and the entail operator $\sqsubseteq$ is embodied by logical implication $\Rightarrow$. The following assumptions are introduced to render the deduction rule sound.

First, general restrictions are implicit in the use of TLA as a specification language, and namely:

- Although time variables can take values over a continuous domain (i.e. the reals $\mathbb{R}$), the semantic of the language is defined in terms of discrete sequences of states, where a state is an assignment of values to the set of variables. Therefore, there is a considerable, and intrinsic, "amount of discreteness" in the description of a system with TLA. This is also shown by the important role given to invariance under stuttering.
- In TLA, there is the notion of initialization of variables, and therefore of "beginning" of the life of a system.
- Usually, it is assumed that the set of all variables can be partitioned into two subsets e and m of input and output variables. The specification is such that the assumption formulas only modify variables in e, while the guarantee formulas only modify variables in m. Actually, how to treat the more general case is discussed briefly in [4], although it still has some limitations.
- It is usually assumed that a specification is written in a preferred "canonical" form. Although the results usually hold also for non-canonical specifications, the use of canonical form is often required to render the proofs practically feasible.
- Although TLA can handle both interleaving and non-interleaving composition, the two cases must usually be treated separately in the correctness proofs.

Second, the following additional assumptions are introduced in the rule.

- The local specifications are written using an *ad hoc* operator, the $\overset{+}{\rhd}$ "while-plus" operator. Quoting [4], a specification

> "is expressed by the formula $E \overset{+}{-\!\!\!\rhd} M$, which means that, for any n, if the environment satisfies E through "time" n, then the system must satisfy M through "time" $n+1$".

  Notice that such an operator requires the notion of discrete "time" and permits some form of induction on such a discrete structure with a beginning.

- Condition 2 is strengthened by requiring it to hold for the *safety closures* of the involved formulas. The *safety closure* of a formula F is denoted as $\mathcal{C}(F)$ and is the strongest (i.e. smallest) safety property which is implied by F. So, condition 2 is modified by replacing E with $\mathcal{C}(E)$ and each M_i with the corresponding $\mathcal{C}(M_i)$. The authors introduce some lemmas that permit to get rid of the closures in the formulas, under additional technical conditions.

- Condition 3 is coupled with a similar one, where the safety closures of the formulas are considered. More precisely, the M_i's and M are replaced by the $\mathcal{C}(M_i)$'s and $\mathcal{C}(M)$, respectively. Moreover, E is replaced by $\mathcal{C}(E)_{+v}$, a formula which asserts that, if $\mathcal{C}(E)$ ever becomes false, then the values of all the specification variables v stay unchanged always in the future. As for the previous condition, rules for handling the closures and the $+$ operator are proposed.

To sum up, the soundness in the compositional rule is gained by considering specifications written in a canonical form, exploiting an induction (using the $\overset{+}{-\!\!\!\rhd}$ operator) over a discrete structure of states with a beginning, and considering safety characterizations of some of the assumptions and guarantees formulas of the specification, handling these safety characterizations with *ad hoc* rules and "tricks". More explicitly, here it is the composition rule.

Proposition 3 (Abadi and Lamport, 1995 [4]). *If:*

1. *for all* $i = 1, \ldots, n$: $E_i \overset{+}{-\!\!\!\rhd} M_i$
2. $\mathcal{C}(E) \wedge \bigwedge_{i=1}^{n} \mathcal{C}(M_i) \Rightarrow \bigwedge_{i=1}^{n} E_i$
3. *(a)* $E \wedge \bigwedge_{i=1}^{n} M_i \Rightarrow M$

   (b) $\mathcal{C}(E)_{+v} \wedge \bigwedge_{i=1}^{n} \mathcal{C}(M_i) \Rightarrow \mathcal{C}(M)$

then: $E \overset{+}{-\!\!\!\rhd} M$.

Finally, Abadi and Lamport also show how a decomposition theorem, to be used with a refinement methodology, can be derived from their composition theorem. The framework was given some tool support in developing the case study described in [55].

We notice how the rule proposed by Abadi and Lamport introduces several assumptions on the computational model, and on how a specification, and the corresponding proof, "should be written". These restrictions, together with the necessary methodology, reduce generality and immediacy of use.

4.3 Abadi and Plotkin, 1993 [6]

This work is an abstract study of compositionality, in two frameworks based on intuitionistic and linear logics. The work treats compositionality at the semantic level, and considers safety properties only.

Let us first consider the analysis based on intuitionistic logic. The framework and the results closely resemble those of [2], with restriction to safety properties for both the assumptions and the guarantees. With respect to [2], an additional theoretical result is the reduction of the compositional inference rule to the the case for just one module. Let us state this result more explicitly. The semantic model is the same as that of [2], with the restriction to safety properties only. It can be shown that such a model is an intuitionistic one, where a suitable implication $\rightarrow$ can be defined. This implication maps the entail operator $\sqsubseteq$, whereas the compose operator $\sqcap$ is mapped to the conjunction $\wedge$, as usual. With these definitions and semantic assumptions, the following very simple circular inference rule is proved sound.

Proposition 4 (Abadi and Plotkin, 1993 [6]). *If:*

1. $E \rightarrow M$
2. $M \rightarrow E$
3. *E and M are safety properties*

then M holds.

Note that the rule of Proposition 4 is actually not compositional, since it involves one module only; alternatively, we could consider it as a degenerate compositional rule. An interesting fact is that we can derive the general compositional rule of [2], restricted to the intuitionistic framework, from Proposition 4, using only propositional reasoning and induction. This result is interesting *per se*, since it basically shows that the only catch in this kind of compositional reasoning with safety properties lies in disentangling the circularity in the discharging of assumptions: doing so with one module suffices. However, it has no practical impact, since it is a specialization of previously seen results by other means.

Let us now consider the other framework based on linear logic. Let us just sketch the main results, skipping several details. In this framework, the behaviors of a module are represented by a process, which is a prefix-closed set of sequences of state transitions. A property is a set of processes. The composition of (safety) properties is defined by the tensor product operator $\otimes$, defined in terms of parallel composition of processes (although the relation in this case is not simply logical conjunction). It is the tensor product that maps the compose operator $\sqcap$. Instead, the $\sqsubseteq$ operator maps to the connective $\multimap\!\!\diamond$, where the formula $E \multimap\!\!\diamond_I M$ represents those processes which cannot be distinguished from those in M whenever the environment process is taken from E and the initialization condition is in the set I. We do not discuss these definitions with more detail. The composition rule for linear logic goes as follows.

Proposition 5 (Abadi and Plokin, 1993 [6]). *If:*

1. $\bigotimes_{i=1}^{n}(E_i \mathrel{-\!\!\diamond}_{I_i} M_i)$

2. for all $i = 1, \ldots, n$, *the property* $E \otimes \bigotimes_{i \neq j=1}^{n} M_j$ *subsumes the property* $M_i \mathrel{-\!\!\diamond}_{I_i} E_i$

3. $I \subseteq \bigcap_{i=1}^{n} I_i$

4. E, M, *and for all* $i = 1, \ldots, n$ E_i *and* M_i *are safety properties*

then $E \mathrel{-\!\!\diamond}_I \bigotimes_{i=1}^{n} M_i$.

Although it introduces several technicalities, we can recognize that the rule is structurally similar to the one for intuitionistic logic, and to those of other works on compositionality. Similarly to what is done in the intuitionistic framework, the rule of Proposition 5 can be deduced from the case of just one module. We do not discuss variations of this result discussed in the paper.

The paper by Abadi and Plotkin shows quite clearly these basic facts about compositional reasoning:

- technical problems arise in proving the soundness because of the presence of circularity;
- these problems can be solved by restricting our attention to safety properties, since proving soundness reduces to proving a sort of "initial validity", which can be regarded as an implicit way of breaking the circularity;
- finally, induction is used to "propagate" the initial validity to any instant in the future.

These basic facts basically apply to most works on compositionality for temporal logics, since the seminal works by Misra and Chandy [67] and Jones [50, 49].

4.4 Abadi and Merz, 1995 [5]

This work introduces an abstract syntactic framework in which to express compositional rules. Actually, even if it is based on an intuitionistic logic, it underlines some of the assumptions about the computational model which were also shared by other works such as [2, 4], so that the statement that the logic is independent of particular computational models is not absolutely true.

The framework consists of a propositional intuitionistic logic, with the usual logic connectives, plus a new connective $\overset{+}{\rightarrow}$ which clearly represents TLA's $\overset{+}{\rhd}$ and similar ones. Even if the framework should be primarily syntactic, it actually introduces some assumptions on the structures on which the logic is to be interpreted. More precisely, these assumptions are:

- a pre-order relation $\subseteq$ is given on the sets of behaviors;
- the relation is such that atomic propositions are true on downward-closed sets of behaviors; that is, for any atomic proposition π and behaviors σ, σ' such that $\sigma \subseteq \sigma'$, if $\sigma' \models \pi$ then also $\sigma \models \pi$.
- the relation $\subset$ obtained from $\subseteq$ by removing reflexivity is well-founded on the set of all behaviors; notice that this permits induction along computations, similarly to the aforementioned models.

Moreover, notice that these assumptions are satisfied whenever the interpretation structures are Kripke models (which are standard models for intuitionistic logic).

Under these assumptions, Abadi and Merz prove a number of basic properties of the operators of the logic. However, a sound compositional rule on the model of Proposition 1 cannot be proved without further semantic assumptions and specializations. More specifically, the authors choose to instantiate the abstract framework with two temporal logics, namely TLA and a fragment of CTL*. Sound compositional rules can be proved only *after* such instantiations are done. Hence, the idea of an abstract framework is appealing in its syntactic approach, but is not detailed enough to implement a sound circular rule.

The rule for TLA is basically an illustration of the one of [4] using the new formalism. In fact, the corresponding additional semantic notions are introduced beforehand. Also notice that the relation $\subseteq$ of the intuitionistic logic corresponds to the prefix order on the set of sequences of states.

The rule for the subset of CTL* tries instead to mimic the rule for TLA with the branching-time logic, by replacing the prefix order between sequences of states with the subtree relation between trees of states. In particular, the notion of safety and suitable operators are introduced for CTL*, so that properties which are syntactically very similar to those of TLA are retained. It must be said that the treatment of compositionality with branching time is peculiar, very different from that of linear time and somewhat more problematic. For example, composition of modules is in general not the logic conjunction of the specifications. This is out of the scope of the present survey, and is only tentative in the work of Abadi and Merz. We notice however, that it is the object of very recent work by other authors (e.g. [70]).

To sum up, Abadi and Merz present an abstract framework to present compositional rules for some languages. The soundness of those rules is only proved within the languages themselves (using their own semantic assumptions), since the framework is too weak in this respect. However, one advantage of this approach is that it is not limited to temporal logics. The authors briefly discuss this aspect in the conclusion. Under this aspect, the approach is indeed more general than most others.

4.5 Xu, Cau, and Collette, 1994 [85]

This work introduces yet another variation of the compositional circular rule. The rule is introduced to show that it subsumes two families of compositional rules, that is those for models of concurrency based on shared variables, and those for message passing models. The earliest works on composition for these models are the well-known contributions by Misra and Chandy [67] for message passing concurrency and by Jones [50, 49] for shared variables. For our purposes, we will not account for the two specific models of concurrency, but we will just discuss and present the unifying rule of [85].

The general framework is actually very similar to the well-known Abadi and Lamport's [2, 4]. More precisely, the following choices are made.

- The compose $\sqcap$ and entail $\sqsubseteq$ operators are embodied by conjunction $\wedge$ and implication $\Rightarrow$, respectively.

- Rely/guarantee specifications are written using the *spiral* operator $\hookrightarrow$, which has basically the same semantics as the $\overset{+}{\rhd}$ of Abadi and Lamport.
- The assumptions are taken to be safety properties.
- The guarantees are expressed by the conjunction of a safety part M^S and a non-safety part M^R.
- We do not detail the treatment of agents, which is however very similar to that of [2].

Therefore, the inference rule goes as follows.[2]

Proposition 6 (Xu, Cau, and Collette, 1994 [85]). *If:*

1. *for all* $i = 1, \ldots, n$*:* $(E_i \hookrightarrow M_i^S) \wedge (E_i \Rightarrow M_i^R)$
2. $E \wedge \bigwedge_{i=1}^n M_i^S \Rightarrow \bigwedge_{i=1}^n E_i$
3. *(a)* $\bigwedge_{i=1}^n M_i^S \Rightarrow M^S$
   (b) $E \wedge \bigwedge_{i=1}^n M_i^R \Rightarrow M^R$
4. $E \Rightarrow \bigwedge_{i=1}^n E_i$ *at the initial time (i.e. 0)*

then: $(E \hookrightarrow M^S) \wedge (E \Rightarrow M^R)$.

As it can be clearly seen, the rule has the structure we are familiar with. Therefore, the main contribution of this work is indeed the unification framework for the two different models of concurrency. In particular, it is shown that the general semantic framework, and the usual requirements on safety and "progression" operators can account for independently developed models of parallelism. Finally, it is speculated that a unifying rule may be useful in integrating the two models of concurrency in a single specification.

4.6 Cau and Collette, 1996 [18]

This paper is basically a revision and extension of the work already presented in [85] to unify compositional rely/guarantee inference rules for the two models of shared variables and message passing concurrency.

More precisely, the presentation relies on a purely semantic framework where a computation is a sequence $\sigma = \chi_0 \xrightarrow{l_1} \chi_1 \xrightarrow{l_2} \chi_2 \xrightarrow{l_3} \cdots$ of configurations connected by labeled transitions. Clearly, the configurations would correspond to states and messages, respectively in the shared variables (a.k.a. state-based) and message passing formalisms. On the other hand, the label of each transition would represent the agent which is responsible for that transition.

With these basic concepts, specifications and properties are represented as sets of computations. Safety sets are those which are closed under prefixes and limit, according to the standard definitions (see e.g. [7]). Being in a set-theoretic framework, we represent $\sqsubseteq$ as the subset relation $\subseteq$.

Composition (that is the $\sqcap$ operator in our abstraction) is represented with the abstract semantic $\otimes$ operator, that merges two behaviors into one (representing the composition of the behaviors). The $\otimes$ operator is assumed to respect simple properties, namely that if $\sigma = \sigma_1 \otimes \sigma_2$:

[2]Actually, [85] presents the rule for the case of two modules only (i.e. $n = 2$). Instead, we present here its most natural generalization to arbitrary n.

- $|\sigma_1| = |\sigma_2|$, that is only behaviors of equal length can be composed
- the result of the composition is a behavior with the same length of those composed, that is $|\sigma| = |\sigma_1| = |\sigma_2|$;
- any prefix of length k of σ is the composition of the prefixes of length k of σ_1 and σ_2, for all k.

The operator $\otimes$ is generalized to sets of behaviors (i.e. properties) as obvious: given properties P_1, P_2, we define their composition $P_1 \otimes P_2$ as the set of behaviors $\{\sigma | \sigma = \sigma_1 \otimes \sigma_2 \wedge \sigma_1 \in P_1 \wedge \sigma_2 \in P_2\}$. This abstract merge operator encompasses various models of composition, and in particular conjunction as well as disjunction.

Finally, rely/guarantee specifications are assumed to be characterized by a triple of properties, which we indicate as E, M^S and M^R. As in the other work [85], E is assumed to be a safety property, M^S is the safety part of the guarantee, and M^R is its non-safety part. We denote by M the set of behaviors which are both in M^S and in M^R, that is $M = M^S \cap M^R$. The link between the assumption and guarantee of a given module is formalized using an *ad hoc* operator, denoted as $@\!\!\longrightarrow$. Its semantics is basically the same as that of Abadi and Lamport's $\overset{+}{\rhd}$, abstracted in the present framework.

We are now ready to present the abstract semantic inference rule of [18], which is proved sound in the abstract setting. As usual, we do not represent some details in order to fit the rule to the setting of our general rule of Proposition 1, and to convey only the very general intuition behind the rule.

Proposition 7 (Cau, and Collette, 1996 [18]). *If:*

1. *(a)* $E_1 @\!\!\longrightarrow M_1$
   (b) $E_2 @\!\!\longrightarrow M_2$
2. $E \otimes M_1^S \otimes M_2^S \subseteq E_1 \otimes E_2$
3. *(a)* $M_1^S \otimes M_2^S \subseteq M^S$
   (b) $E \otimes M_1^R \otimes M_2^R \subseteq M^R$

then: $E @\!\!\longrightarrow M$.

Afterwards, the authors demonstrate how the rule can be instantiated into the two settings of shared variables and message passing concurrency, which are therefore reduced to special cases of this general rule. We do not present in more detail the results in [18], as the same comments and considerations made for the previous work [85] hold for this one as well.

4.7 Jonsson and Tsay, 1996 [51]

This paper analyzes compositionality with linear temporal logic from a syntactic perspective. The outcome is a compositional inference rule which, although similar in principle to that of Abadi and Lamport [4], is presented using a syntactic formulation, which has the advantage of being simpler to use in practice, over purely semantic approaches.

The basic idea of the approach is still to rely on safety properties to render sound the compositional circular reasoning, but to enforce the safety characterization by requiring that the formulas of the specification are written according to some syntactic restrictions. In fact, in LTL it is possible to syntactically characterize safety. In principle, the same syntactic approach could be pursued with TLA as well, but probably with poor results, from the perspective of practical usefulness. On the other hand, the syntactic characterization in LTL is often more viable.

The syntactic characterization requires the specifier to write the formulas for assumptions and guarantees in a canonical form, to enforce the underlying semantic hypotheses. The requirements for the canonical form are the following ones.[3]

- Assumptions E are expressed with a LTL formula of the form $\Box(\exists x : \boxminus H_E)$, where:
  - H_E is a past formula, that is a formula without future operators;
  - x is a tuple of flexible variables, considered as internal variables (i.e. non-visible, existential quantification representing hiding);
  - $\Box H_E$ must be stuttering extensible, another technical requirement we do not discuss in detail.

  Under these conditions, the formula E is a safety formula. Actually, the requirement of H_E being a past formula could be relaxed to the requirement to be a *historical* formula, but with some additional complications.

- Guarantees M are expressed with a LTL formula of the form $\exists y : \Box H_M \land L_M$, where:
  - H_M is a past formula;
  - y is a tuple of flexible, internal variables;
  - $\Box H_M$ must be stuttering extensible;
  - L_M is a liveness property;
  - (H_M, L_M) is machine-closed, that is $\mathcal{C}(\Box H_M \land L_M) \Leftrightarrow \Box H_M$.

- Rely/guarantee specifications are expressed using the operator $\rhd$, where $E \rhd M$ is defined as $\Box(\widetilde{\circleddash}(\exists x : \boxminus H_E) \Rightarrow (\exists y : \boxminus H_M)) \land (E \Rightarrow M)$.

Under these conditions, we can calculate syntactically the safety closures of the specification formulas. In fact it turns out that:

- $\mathcal{C}(\exists x : \boxminus H_E) = \Box(\exists x : \boxminus H_E)$
- $\mathcal{C}(\exists y : \Box H_M \land L_M) = \Box(\exists y : \boxminus H_M)$

Finally, assuming as usual that the operators $\sqsubseteq$ and $\sqcap$ of the template Proposition 1 are mapped to implication $\Rightarrow$ and conjunction $\land$ respectively, we can formulate Jonsson and Tsay's compositional rule.

[3]The temporal operators are LTL's usual ones, and namely $\Box$, $\boxminus$, $\widetilde{\circleddash}$ for, respectively, always in the future, always in the past, in the previous instant (if it exists). Moreover, $\mathcal{C}(F)$ represents the safety closure of a formula F.

Proposition 8 (Jonsson and Tsay, 1996 [51]). *If:*

1. *for all* $i = 1, \dots, n$*:* $E_i \rhd M_i$
2. $\Box\,((\exists x : \boxminus H_E) \wedge (\exists y_1, \dots, y_n : \boxminus \bigwedge_{i=1}^{n} H_{M_i}) \Rightarrow (\exists x_1, \dots, x_n : \boxminus \bigwedge_{i=1}^{n} H_{E_i}))$
3. *(a)* $\Box\,(\widetilde{\circleddash}(\exists x : \boxminus H_E) \wedge (\exists y_1, \dots, y_n : \boxminus \bigwedge_{i=1}^{n} H_{M_i}) \Rightarrow (\exists x_1, \dots, x_n : \boxminus H_M))$
   (b) $E \wedge \bigwedge_{i=1}^{n} M_i \Rightarrow M$

then: $E \rhd M$*.*

Discussions on how to actually discharge the hypotheses of the inference rule are introduced, with reference to the usual proof methodologies for temporal logics. Moreover, the same example of a queue of [4] is done, resulting in a simpler compositional proof. The work presented in this paper was slightly extended in a later work by Tsay [82], and given some tool support with PVS in [81].

To sum up, Jonsson and Tsay try to identify general conditions under which the semantic assumptions of the earlier works on compositionality can be expressed syntactically in LTL, resulting in inference rules which are simpler to apply and require less *ad hoc* methodology. Actually, not all semantic assumptions are rendered syntactically (for example machine closure and stuttering extensibility conditions are still required explicitly). Moreover, the syntactic restrictions may render the specifications harder to write. However, it is true that some aspects are indeed simplified and the application of the rule in some cases promises to be practically simpler.

4.8 Tsay, 2000 [82]

This paper is an extension of [51]. Therefore, we just analyze the novelties. Before doing that, we highlight the fact that the analysis is deliberately simplified, namely by not allowing shared variables (i.e. variables that can be modified by more than one module) and not considering quantification over flexible variables (e.g. hiding).

First of all, the author considers what he calls *weak* form of rely/guarantee specifications. A weak form uses a weaker form of the $\rhd$ operator. Informally, a weak rely/guarantee specification, that we denote as $E \rhd_w M$, says that if E holds for some prefix of the computation, then M also holds for the *same* prefix. If E and M are written in canonic form (see [51]), then $E \rhd_w M$ is defined syntactically as $\Box(\boxminus H_E \Rightarrow H_M) \wedge (E \Rightarrow M)$. However, this weak operator cannot be used with circular rules; on the other hand a very simple non-circular verification rule is proposed, which is not even compositional.

Another issue briefly touched upon is the use of liveness properties also in the assumption formulas. For this case, and with the same definitions of [51], an asymmetric (i.e. where only one module is allowed to have a liveness property) circular rule for the composition of two modules is given. Note that we are now using the standard progression operator $\rhd$, not its weak form. It is assumed that:

- the local assumption for module 1 is strictly a safety property in the canonical form $E_1 = \Box H_{E_1}$, where H_{E_1} is a past formula;

- the local assumption for module 2 can have a liveness property in the form $E_2 = \Box H_{E_2} \wedge L_{E_2}$;
- the local guarantees are in the form $M_i = \Box H_{M_i} \wedge L_{M_i}$ for $i = 1, 2$;
- the global assumption and guarantee are in the form $E = \Box H_E \wedge L_E$ and $M = \Box H_M \wedge L_M$.

Note that the only property that must strictly be a safety property is the assumption of module 1. The rule is therefore the following.

Proposition 9 (Tsay, 2000 [82]). *If:*

1. *for all* $i = 1, 2$*:* $E_i \rhd M_i$
2. *(a)* $\Box\,(\boxminus H_E \wedge \boxminus(H_{M_1} \wedge H_{M_2}) \Rightarrow H_{E_1} \wedge H_{E_2})$
   (b) $E \wedge M_1 \Rightarrow E_2$
3. *(a)* $\Box\,(\widetilde{\ominus}\boxminus H_E \wedge \boxminus(H_{M_1} \wedge H_{M_2}) \Rightarrow H_M)$
   (b) $E \wedge M_1 \wedge M_2 \Rightarrow M$

then: $E \rhd M$.

Besides being an extension of [51], this paper points out certain interesting facts about circularity, and its relationship with the requirement of a "progression" operator in a nice way.

4.9 Namjoshi and Trefler, 2000 [71]

Namjoshi and Trefler present an interesting analysis concerning the *completeness* of the compositional inference rules found in the literature, with specific reference to that of Abadi and Lamport [4], and McMillan [65]. Since those rules are the basis for most other existing rules in the literature, the analysis can draw rather general results. Moreover, in doing so they tackle the problem of circularity from yet another perspective, showing a reason why circular rules are not needed, at least in principle.

First of all, we have to state formally what are the compositional rules considered in the paper. To render the comparison simpler, let us stick to the case of just two modules. The first rule is the circular one by Abadi and Lamport [4], which we already know of. The second one is a very simple non-circular rule, which we can state as follows.[4]

Proposition 10 (Non-Circular Rule, [71]). *If:*

1. *for all* $i = 1, 2$*:* $E_i \Rightarrow M_i$
2. *(a)* $E \Rightarrow E_1$
   (b) $M_1 \Rightarrow E_2$
3. $M_1 \wedge M_2 \Rightarrow M$

[4]Actually, the exact formulation of the rules of [71] would require a notion of computational model and several semantic notions. For simplicity, we present here only a crude abstraction of a slight modification of that rule.

then: $E \Rightarrow M$.

The formulas M_1 and E_2 are called *auxiliary assertions* in this framework, since they serve to break the proof into two suitable parts, in a way that permits a sound deduction.

The third rule we consider is the one by McMillan [65]. Even if in [71] a generalization of the rule to handle n modules is presented, the generalization has a rather convoluted formulation. Therefore, for the sake of simplicity, we present here the original version of [65] for two modules only, in a minimal setting (that is, without reference to the computational model). Note that the rely/guarantee specifications are now written using the $\rhd$ operator, called *constrains* in that paper. As usual, $E \rhd M$ is true iff, if E is true up to step i of a computation, then M is true at step $i+1$. It can be defined in linear temporal logic, using the *until* operator as $\neg(E\ \mathcal{U}\ \neg M)$, using the dual *release* operator as $\neg E\ \mathcal{R}\ M$, and can be thought as meaning "$\neg E$ precedes M", that is E is falsified before M is. The circular rule is then as follows ($\hat{\imath}$ represent the index "other than" i, that is $\hat{\imath} \in \{1,2\} \setminus \{i\}$).

Proposition 11 (McMillan, 1999 [65]). *If:*

1. *for all* $i = 1,2$*:* $E \Rightarrow (M_{\hat{\imath}} \rhd M_i)$
2. $M_1 \wedge M_2 \Rightarrow M$

then: $E \Rightarrow \Box M$.

Namjoshi and Trefler first show that, while the non-circular rule of Proposition 10 is complete, the other rules of Abadi and Lamport [4] and McMillan (Proposition 11 and [65]) are incomplete, even for simple properties. The incompleteness in McMillan's rule is due to the absence of auxiliary assertions, that is local assumptions E_i's, different than the other module's guarantee M_i. In fact, in a rather different setting, Maier [63] has shown that the use auxiliary assertions is a necessary condition for completeness. Still, it is not sufficient, since Abadi and Lamport's rule [4] does use auxiliary assertions but is nonetheless incomplete, as Namjoshi and Trefler also show in their paper.

Next, the authors show how to strengthen McMillan's circular rule in order to get a complete rule, obviously still keeping soundness and circularity. In order to do that, we have to write the assumptions and guarantees in a (simple) canonical form, where the parts representing the auxiliary assertions are conjoined to the remainder of the specification. So, each assumption E_i is written as $E_i = H_{\hat{\imath}} \wedge G_{\hat{\imath}}$ and each guarantee M_i as $M_i = (H_{\hat{\imath}} \Rightarrow G_i) \wedge H_i$, for some formulas H_i, G_i. The rule is finally as follows.

Proposition 12 (Namjoshi and Trefler, 2000 [71]). *If:*

1. *for all* $i = 1,2$*:* $E \Rightarrow ((H_{\hat{\imath}} \wedge G_{\hat{\imath}}) \rhd ((H_{\hat{\imath}} \Rightarrow G_i) \wedge H_i))$
2. $G_1 \wedge G_2 \Rightarrow M$

then: $E \Rightarrow \Box M$.

Notice that Proposition 12 reduces to Proposition 11 if we take $H_1 = H_2 = true$.

The last aspect of compositionality considered by Namjoshi and Trefler is the translation between circular and non-circular proofs. Since both the non-circular rule of Proposition 10 and the circular rule of Proposition 12 are complete, every compositional proof can be carried out with any of the two, at least in principle. Hence, it is shown how one can translate the application of a rule into the application of the other rule; in other words, it is shown how one should pick the local assumptions and guarantees (possibly including the auxiliary assertions) so that the applicability of one rule follows from the application of the other one to the same system. It is also briefly discussed how this translation is efficient in both directions, that is the translation process builds formulas that are of the same order of magnitude as the formulas from which it translates (i.e. the ones of the original application of the rule). This formalizes an important conclusion: circularity is not needed, at least in principle. Actually, this is no big surprise, since, once again, it is well understood that compositional techniques are only useful in practice, while cannot reduce the worst-case complexity of verification. Therefore, a more complicated rule (and namely a circular one) can bring some benefits in some concrete cases, but is not more efficient than another equally complete rule in the general case.

4.10 Viswanathan and Viswanathan, 2001 [84]

This paper is another attempt at unifying different compositional rules found in the literature under a common framework. More precisely, a rely/guarantee semantics is given in a model based on fixed points; then, an inference rule for such a rely/guarantee model is given. Finally, it is shown how the rules for two different computational models, and namely traces and trace trees, can be expressed as specializations of the given rule for fixed points.

The proposed model is rather non-standard, if compared to the traditional temporal logic frameworks. Let us introduced its essential features. Let Σ be the (uninterpreted) set of all computations $\sigma, \sigma_1, \ldots \in \Sigma$, also called behaviors. A specification S is then a set of computations $S \subseteq \Sigma$. In particular, we consider specifications which are expressible as fixed points. Let F be a (monotonic) operator that maps sets of computations to sets of computations, that is $F : 2^\Sigma \to 2^\Sigma$. We denote the *greatest fixed point* of F as $\nu(F)$, the least fixed point as $\mu(F)$ and any of the two as $\rho(F)$. The kth approximation (i.e. application) of the fixed point $\rho(F)$ is indicated as $[\rho(F)]^k$. For example, using this notation, the greatest fixed point is defined recursively as:

$$[\nu(F)]^0 = \Sigma \qquad [\nu(F)]^{k+1} = F([\nu(F)]^k) \qquad [\nu(F)]^\omega = \bigcap_{k=1}^{\infty} [\nu(F)]^k$$

Under rather general conditions, the applications converge to their fixed points. This is the case with all examples in the paper. Moreover notice that, in order to simplify the presentation, we remove explicit mention to the computations. For example, instead of $\sigma \models [\rho(F)]^k$, we will write simply $[\rho(F)]^k$, considering a computation σ implicitly defined. Although this renders the notation less precise, it is acceptable since we only aim here at conveying the general "flavor" of the rule.

First of all, let us note that we consider the simple case of two modules only. A generalization would probably be feasible, but let us now give the rule exactly as it is in the paper. Let us now consider the conditions introduced with the inference rule.

- Once again, $\sqsubseteq$ is implication $\Rightarrow$ and $\sqcap$ is conjunction $\wedge$.
- Assumptions are written as fixed points $\rho(E)$, whereas guarantees are written as fixed points $\rho'(M)$. ρ and ρ' may or may not be the same kind of fixed point.
- Rely/guarantee specifications are written using the $\rhd$ operator, which is defined similarly to the usual "progression" operators, but with the relaxation that, when the assumption has been satisfied in the past, the guarantee must be satisfied in some future approximation (not necessarily the next one). Formally, $\rho(E) \rhd \rho(M)$ is satisfied iff:

$$\forall k \geq 0 : [\rho(E)]^k \quad \Rightarrow \quad \exists k' > k : [\rho'(M)]^{k'}$$

- The inference rule, basically requires the following conditions:
  - if the global assumption is satisfied currently and one of the local guarantees is satisfied eventually, then the other local assumption is satisfied eventually;
  - if the local guarantee holds then the global guarantee holds eventually;
  - if the global assumption is satisfied, then one of the local guarantees holds eventually. This last requirement is an *explicit* circularity-breaking condition, which was not present in all the other works, where circularity is broken only implicitly. Notice that this explicit requirement is needed to have a rule which is more general than the other ones.

Hence, the rule can be formalized as follows.

Proposition 13 (Viswanathan and Viswanathan, 2001 [84]). *If:*

1. *for all* $i = 1, 2$*:* $\rho(E_i) \rhd \rho'(M_i)$
2. *for all* $i = 1, 2$[5]*:* $\forall k : \forall k' \geq k : [\rho(E)]^k \wedge [\rho'(M_i)]^{k'} \;\Rightarrow\; \exists k'' \geq k : [\rho(E_{\hat{\imath}})]^{k''}$
3. $\forall k : [\rho'(M_1)]^k \wedge [\rho'(M_2)]^k \;\Rightarrow\; \exists k' \geq k : [\rho'(M)]^{k'}$
4. $\forall k : [\rho(E)]^k \;\Rightarrow\; \exists k' \geq k : [\rho'(M_1)]^{k'} \vee [\rho'(M_2)]^{k'}$

then: $\rho(E) \rhd \rho'(M)$.

As we discussed, the framework is specialized for two different models of concurrency, showing that it can encompass well-known compositional rules, independently developed. More precisely, the first model is the one for trace semantics, that is the one for linear temporal logic we are familiar with. In particular, with reference to propositional logic, the rule of "weak until" presented in [71, 65] is derived from the fixed-point one, with a suitable choice for fixed-point operators to represent the usual safety properties. The other model is

[5]As usual $\hat{\imath}$ denotes the "other" index $\hat{\imath} \in \{1, 2\} \setminus \{i\}$.

the one of trace trees for Moore machines with synchronous composition, whose compositional rules were presented in [40], as well as for reactive modules in [8]. We do not give the details of these reductions.

Two other issues presented in the paper still deserve discussion. The first one is the idea of a subsumption rule to derive certain variations of the inference rule, and namely those that use an operator different than $\rhd$ to express rely/guarantee. If the different operator satisfies certain conditions, it is possible to derive a formula for $\rhd$ from the formula with the other operator, thus permitting the applicability of the rule.

The other application consists in specializing the rule for greatest fixed points. If all the ρ are taken to be ν, the explicit circularity-breaking condition 4 of Proposition 13 can be removed, since it is implicit in the semantics of the greatest fixed point operator. This shows how other rules can achieve soundness without explicit circularity-breaking rules.

All in all, the paper by Viswanathan and Viswanathan [84] is interesting in its generalization effort, and in elucidating how some basic facts about circularity-breaking for compositional circular rules come into play and are treated in the most common rules found in the literature.

4.11 Maier, 2001 [62]

This paper introduces another simple compositional framework, and then shows that it can be instantiated to some concrete computational models. The framework is based on set theory, and in this sense it is similar to [2]. However, with respect to [2] it is simpler and based on fewer semantic assumptions, and therefore more abstract.

In a nutshell, the framework is based on downward-closed sets. More precisely, a system is defined by a set of behaviors from a *universe set* $\mathcal{B}$, which is partially ordered (with order relation denoted as $\preceq$), well-founded (i.e. it has a *bottom element* ϵ), and *downward-closed*, that is for all b, b' if $b \in \mathcal{B}$ and $b' \preceq b$ then $b' \in \mathcal{B}$. Every property (or module) is identified with some downward-closed subset $B \subseteq \mathcal{B}$, with certain properties.

Structures called *chains* are introduced to describe the evolution of behaviors. A sequence of behaviors $b_i \in \mathcal{B}$ for $i \in \mathbb{N}$ is called a *chain* whenever:

- it ascends from the bottom, i.e. $b_0 = \epsilon$ and for all $i \in \mathcal{B}$: $b_i \preceq b_{i+1}$;
- it converges to some limit in $\mathcal{B}$, that is the upper bound b of the set $\{b_i | i \in \mathbb{N}\}$ exists and is in $\mathcal{B}$.

With a little abuse of notation, we will denote a chain $(b_i)_{i \in \mathbb{N}}$ simply as b_i.

We need two more notions to introduce the compositional inference rule of the framework in detail. One is the idea of *extension*: given two sets $B_1, B_2 \subseteq \mathcal{B}$ and a chain b_i, we say that B_1, B_2 extend along b_i iff, for all $i \in \mathbb{N}$, if $b_i \in B_1 \cap B_2$ then $b_{i+1} \in B_1 \cup B_2$. The other notion, more familiar, is that of closure: a set $B \subseteq \mathcal{B}$ is closed with respect to a chain b_i if, whenever $b_i \in B$ for all $i \in \mathbb{N}$, the limit of b_i is also in B.

Finally, as it is natural in a set-theoretic framework, composition $\sqcap$ is represented by intersection $\cap$ and entailment $\sqsubseteq$ by the subset relation $\subseteq$.

We are now ready to introduce the abstract compositional rule of the framework. We note that it is proposed in a narrower form than that of the general rule of Proposition 1: it considers the simple case of two modules only; it directly introduces the circularity by putting the guarantee of each module among the assumptions in the specification of the other module; it considers composition into a closed, rather than open, overall system. Here it is the inference rule.

Proposition 14 (Maier, 2001 [62]). *If:*

1. *(a)* $E_1 \cap M_2 \subseteq M_1$

   (b) $E_2 \cap M_1 \subseteq M_2$

2. *for every $b \in \mathcal{B}$, there exists a chain b_i (for $i \in \mathbb{N}$) such that:*

   (a) b_i converges to b

   (b) M_1, M_2 extend along b_i

   (c) M_1, M_2 are closed with respect to b_i

then: $E_1 \cap E_2 \subseteq M_1 \cap M_2$.

Notice that the requirement 2c on the closure of M_1, M_2, together with downward-closure, is a strict analogous of requiring the guarantees to be safety properties, as done in other compositional frameworks. Moreover, the extension condition 2b recalls the idea of a *spiral* operator such as Abadi and Lamport's *while-plus*. This, combined with the well-foundedness of the set of behaviors $\mathcal{B}$, permits the application of the usual induction in proving the soundness of the rule.

The remainder of the paper [62] shows how to instantiate the framework for familiar models of computation. More precisely, it is instantiated to the synchronous models of Moore machines, Mealy machines, and Kripke structures.

The proposed framework is simple enough to be interesting as a general framework. Its major contribution is, in our opinion, to explicitly pin down some basic facts about compositional reasoning at the semantic level. These facts constitute the basis of most works on compositionality and therefore they can be understood better. In this sense, we believe that this work is similar in spirit to [10].

4.12 Maier, 2003 [63]

This paper considers compositional rely/guarantee inference rules in a very abstract setting, in order to draw general conclusions about soundness and completeness of such rules. Contrarily to the other papers considered above, it does not introduce any new inference rule, but it considers the inherent limitations of circular compositional rely/guarantee reasoning in a given abstract framework. Let us first describe such framework and then show the results drawn by Maier.

In this framework, systems (or modules) and modules properties are modeled uniformly as elements of a generic meet-semilattice with one. A meet-semilattice with one $\mathbf{S} = \langle S, \wedge, 1, \leq \rangle$ is a partial order $\langle S, \leq \rangle$ with greatest element $1 \in S$ and such that for any two elements $x, y \in S$ there exists their greatest lower bound, denoted as $x \wedge y \in S$. Intuitively $x \wedge y$ indicates the composition of two modules, or the conjunction of two properties; in other words, it corresponds to the compose operator $\sqcap$ of our abstract setting. The entailment relation $\sqsubseteq$ is instead represented by the refinement relation $x \leq y$ (x refines y); equivalently it represents the fact that module x satisfies property y. In other words, the model assumes only a refinement order relation $\leq$, an associative commutative and idempotent operation $\wedge$ which respects the order, and a discrete structure S over which to interpret (in particular, no computational model is required).

In this setting, we define an *inference rule* as any rule R in the form:

$$R : \frac{\phi_1 \cdots \phi_n}{\psi} \text{ if } \Gamma$$

where $\Phi = \{\phi_1, \cdots, \phi_n\}$ is a set of formulas called *premises*, ψ is a formula called *conclusion* and Γ is a relation called *side condition*. The meaning of R is as expected: if the premises and the side condition are true, then the truth of the conclusion follows logically. Notice that the language in which one can express the formulas is rather restricted in the given setting, so that the side condition is needed to express more powerful constraints (relations are strictly more expressive than formulas in the given setting).

Given a inference rule R, we say that R is:

sound iff for all evaluations α, $\alpha \models \Phi$ and $\alpha \models \Gamma$ implies $\alpha \models \psi$, i.e. the rule draws true facts from true premises;

complete iff for all evaluations α, $\alpha \models \psi$ and $\alpha \models \Phi$ implies $\alpha \models \Gamma$, i.e. the rule is applicable whenever premises and conclusions hold (in other words, the side condition is not "too restrictive");

assume-guarantee iff for all premises $\phi \in \Phi$, the left-hand side of ψ and the right-hand side of ϕ do not share any variables, i.e. we can identify "assumptions" and "guarantees" without ambiguities;

circular iff it is assume-guarantee and in general $\Phi \not\models \psi$, i.e. the side condition is strictly needed to guarantee soundness;

compositional iff it is assume-guarantee and (informally) it never considers the system composition (i.e. the conjunction of all modules) in the premises and in the side-condition (in other words, we can consider each module independently of the others).

Assuming the aforementioned definitions and structures, the author shows that if the semilattice **S** contains forks (which are certain special structures) of sufficient width (and, in particular, of infinite width), then any compositional circular assume-guarantee rule is either unsound or incomplete. Since soundness is obviously indispensable, we must either give up completeness or compositionality. In the case of automatic verification, compositionality is probably more important, since it permits to really divide the burden of verification among modules, whenever possible, thus lowering the required computational effort. The author argues that in the case of manual verification, completeness may be more important, since the human user can always, at least in principle, invent a suitable system decomposition to correctly carry out the verification task.

Since the proposed framework is a very abstract one, it is important to understand if and how the results drawn for it can be extended to more concrete frameworks. The author claims that most rely/guarantee inference rule presented in the literature can be transformed into rules over meet-semilattices with forks of infinite width, while preserving properties such as soundness and compositionality. Hence, the limitation of completeness must hold also for the original rule.

Brief examples are shown of rules for Moore or Mealy machines, such as those in [40, 62], or for temporal logics, such as those in [4, 5, 51]. In particular, the presence of forks of infinite width follows quite naturally from the fact that the aforementioned frameworks are interpreted over sets of strings of infinite length: such structures provide such forks in all non-trivial cases. However, notice that more expressive formalisms cannot be interpreted over meet-semilattices in all circumstances. For instance, as far as we understand correctly, metric temporal logics over a continuous time domain cannot be fully (i.e. without any lessening of expressive power) translated using the simple formalism of [63], since such translation is possible only for interpretations over algebraic (and discrete) structures.

Finally, the author considers a different definition of completeness than that introduced in the paper. Such other definition is named *backward* completeness, since it is useful to characterize rules which enable backward reasoning. It is used, among others, in [71]. Indeed, we must point out the fact that the notion of completeness introduced in this paper is a non-standard one, as also briefly pointed out in [10], while backward completeness is what is usually considered a *standard* definition of completeness. A rule $R : \Phi/\psi$ if Γ is backward complete iff for all evaluations α, if $\alpha \models \psi$ then $\alpha' \models \Phi$ and $\alpha' \models \Gamma$, for some evaluation α' which agrees with α on all the variables of ψ. In other words, backward complete rules admit the use of *auxiliary variables*, i.e. variables appearing in the premises but not in the conclusion, and of assertions on them; such auxiliary variables increase the expressive power of the inference rules. Backward completeness implies completeness; therefore, every sound and backward complete compositional circular assume-guarantee rule necessarily employs some auxiliary variables. Notice that the presence of auxiliary variables also increases the complexity of the proofs, since one also needs to "guess" the value of auxiliary assertions about the system.

All in all, this paper brings forward an interesting analysis of some basic issues concerning rely/guarantee inference rules. In order to be able to apply the conclusions of the paper to concrete formalisms, it is important to always carefully consider, case by case, if and how the definitions characterizing the framework can express suitably in less abstract settings. More precisely, things should probably be reconsidered with more expressive frameworks (in particular with continuous time formalisms) or with different definitions for inference rules and for completeness.

4.13 Amla, Emerson, Namjoshi, and Trefler, 2003 [10]

This paper constitutes an attempt to build a generic (i.e. sufficiently abstract) sound and complete compositional inference rule, following an approach which is different from those of most previous works. In fact, nearly all methods render the circular reasoning of Proposition 1 sound by introducing some sort of *progression* operator, which justifies an induction on the set of behaviors prefixes. On the other hand, this paper does not rely on any *ad hoc* operator, but rather it introduces some precise semantic assumptions among the conditions on the rules. These additional constraints also permit a well-founded induction on the set of behaviors prefixes, thus rendering the inference rule sound.

Actually, the focus of the paper is on refinement, so the setting is different than that of the other works we considered. However, since the approach is novel, we review it anyway, briefly detouring from our basic setting. Actually, refinement (and decomposition) can be regarded as a special case of composition, under reasonable conditions; for example, Abadi and Lamport in [4] derive their decomposition theorem as a corollary of the composition theorem. However, the differences in the two settings are such that the terminology and notational conventions of the composition setting, which we have considered thus far, sound strange and do not provide intuition in the decomposition setting. Therefore, we are going to introduce some new notations, in order to provide the intuition needed to understand the core results of this paper.

The first difference of the decomposition setting is that we do not have rely/guarantee specifications of modules, but rely/guarantee only refers to the style of reasoning employed by the inference rule. For simplicity, let us consider the simple case of two modules, as it is done in [10]. The goal of decomposition reasoning is to show that the composition of the two low-level specifications M_1^L and M_2^L of the modules (e.g. their implementations), implements a certain property M, possibly under a certain global environment specification E. In general, the composition of the two low-level specifications is very complicated, therefore the direct proof of the above is unfeasible or too costly (usually, we consider automated verification, but the same holds for manual verification). On the contrary, rely/guarantee reasoning gives conditions under which we can consider only the composition of the high-level specifications M_1 and M_2 of the modules, which is in general simpler than its low-level counterpart, together with one low-level specification *at a time*, that is avoiding to consider directly the composition of the two low-level specifications. The reasoning is rely/guarantee since we usually require that M_1^L refines M_1, *assuming* M_2 holds, and *vice versa* for the other module. Notice that it is not sufficient to just require that M_i^L refines M_i, since this is in general not the case: usually, the refinement relation holds only when the other module is a proper environment to the former one.

The reduction to the composition framework can be understood by taking the M_i's of the composition setting as the M_i^L's of the decomposition setting, and the E_i's as the M_i's of the decomposition setting. However, notice that the E_i's now represent the assumption of one module about the behavior of the other, which is needed for the refinement relation to hold, and not a condition under which the module behaves properly. We believe this brief recount is enough to give the general idea that lies beneath the link between composition and decomposition.

We are now ready to introduce the abstract framework of [10]; as usual we introduce only the relevant aspects, while avoiding some details for simplicity. We consider the set of all behaviors (i.e. computations) $\mathcal{B}$. The following assumptions characterize the set of behaviors:

- $\mathcal{B}$ is partitioned into the two non-empty subsets of finite and infinite behaviors;
- $\mathcal{B}$ is partially ordered by an order "prefix" relation $\preceq$;
- the set of finite behaviors is downward closed under $\preceq$ (i.e. every prefix of a finite behavior is also a finite behavior);

- $\prec$ is well-founded on the set of finite behaviors; notice that this implies the existence of an initial condition, that serves as base for inductions.

As usual, a specification is a subset of $\mathcal{B}$, that is a set of behaviors, with the additional condition that every finite prefix of a behavior in the specification is also a (finite) behavior in the specification. This requirement basically extends downward closure to specifications. We overlook the requirement of behavior equivalence and the non-blocking condition, for the sake of simplicity. Notice that all these conditions are semantic, and consider algebraic (and discrete) structures; in fact, the classical framework of state sequences is a straightforward realization of these assumptions.

Finally, the usual choice of representing composition $\sqcap$ as conjunction $\wedge$ of specifications, and entailment $\sqsubseteq$ as implication $\Rightarrow$ is made; notice that in this case entailment represents *refinement*.

We are now ready to state the decomposition rule of [10]. In a nutshell, the rule aims at showing that $E \wedge M_1^L \wedge M_2^L$ refines M, without considering explicitly the conjunction $M_1^L \wedge M_2^L$ in the hypotheses of the rule. In order to do that, we require:

1. that the low-level specification of each module, when composed with the high-level specification of the other, refines the high-level specification of the former module;

2. that the composition of the high-level specifications refines M, under the assumption E;

3. that the composition of one (anyone will do) of the low-level specifications with the (safety) closure of M refines one of the high-level specifications, or M itself. Notice that this last condition is the semantic requirement that allows to break the circularity, giving a base case for the induction, since it requires that one low-level specification refines a high-level one, without assuming anything about the other module.

The formal statement of the rule, using our notation, follows. The authors prove in [10] that the rule is both sound and complete for the given setting.

Proposition 15 (Amla, Emerson, Namjoshi, and Trefler, 2003 [10]). *If:*

1. (a) $M_1^L \wedge M_2 \Rightarrow M_1$
   (b) $M_2^L \wedge M_1 \Rightarrow M_2$

2. $E \wedge M_1 \wedge M_2 \Rightarrow M$

3. *for some* $i \in \{1, 2\}$*:* $E \wedge M_i^L \wedge \mathcal{C}(M) \Rightarrow M \vee M_1 \vee M_2$

then: $E \wedge M_1^L \wedge M_2^L \Rightarrow M$.

All in all, the proposal of Amla et al. is, to the best of our knowledge, the first example of compositional rely/guarantee reasoning which presents an abstract framework not relying on an *ad hoc* operator for expressing rely/guarantee specifications, but only exploiting semantic assumptions. Moreover, the proposed rule is a complete one.

5 Abstract Non-Rely/Guarantee Methods

This section considers some examples of compositional method that adopt paradigms other than the rely/guarantee one, and therefore cannot be reduced to the general scheme of Proposition 1. In particular, we are considering abstract methods, rather than those tailored for a very specific formal language, some of which are instead described in Section 7.

5.1 A Compositional Proof Method for Assertion Networks

A compositional proof method based on an abstract semantic setting is investigated by de Boer and de Roever in [23], with reference to assertion networks. Assertion networks are transition diagrams that represent programs and they were first proposed by Floyd [30] for sequential programs. The work presented in [23] considers the extension of such diagrams to the treatment of concurrent programs.

More precisely, [23] considers state-based reasoning about concurrent programs that have *synchronous communication.* The method relies on two ingredients:

- compositional inductive assertion networks, that are used to describe the sequential parts of the system;
- compositional proof rules, to deduce global properties of the system, resulting from the concurrent interaction of the local networks.

The basic idea is to introduce auxiliary variables in the assertional description of a network that represent *logical histories.* A logical history simulates the local communication history of some component.

Each component is specified by a Hoare-like triple such as $\{\phi\}P\{\psi\}$, where ϕ, ψ are predicates over the logical histories variables, representing properties of such histories respectively before and after the program P (represented by a synchronous transition diagram) is executed. Finally, the compositional method is based on an inference rule for the parallel composition $P_1 \parallel P_2$ of two programs P_1, P_2. Under certain conditions on the predicates ϕ_i, ψ_i and on the variables involved in the communication channels, the following inference rule is sound and complete for synchronous transition diagrams.

$$\frac{\{\phi_1\}P_1\{\psi_1\}, \{\phi_2\}P_2\{\psi_2\}}{\{\phi_1 \cap \phi_2\}P_1 \parallel P_2\{\psi_1 \cap \psi_2\}}$$

Finally, the paper discusses some modifications to the rule to handle *nested parallelism.*

Various other modifications and extensions of the method, to handle different models of concurrency, as well as other methodologies, are thoroughly discussed in the book [26].

5.2 The Lazy Approach to Compositionality

The lazy paradigm is an approach to compositionality alternative to the rely/guarantee one, proposed by Shankar in [79].

The motivation for the study of an alternative paradigm stems from the fact that, according to Shankar, the rely/guarantee approach is often difficult to apply in practice, since it requires to anticipate several details of the guarantees of a component so that they are strong enough to permit the discharging of the assumptions of the other modules. On the contrary, one would often prefer to produce a weak compositional specification, adding the necessary details only in later phases of development. Lazy compositionality proposes to solve this problem by simply *lazily* postponing the discharging of the assumptions to later phases of development, when all the necessary details naturally come into the picture.

Under this respect, the lazy approach is more liberal than the rely/guarantee approach, since it does not enforce a pre-defined way of performing verification, but simply allows for the use of conventional techniques and methods. Moreover, lazy composition is mainly a *methodological* approach to compositionality, whereas rely/guarantee has a strong technical connotation. In a nutshell, lazy compositionality combines a *refinement* methodology with a compositional methodology. The result is an approach which is quite general and can be applied to a large variety of formal specification languages and models. At the same time, its generality can also be a weakness, since the verification burden is implicitly subsumed to other aspects of system analysis, rather than exposed and dealt with directly.

In a hypothetical classification of methods for the verification of large systems, the lazy approach has an "intermediate" degree of compositionality, in between truly compositional approaches (and namely the rely/guarantee approach), and non-compositional (a.k.a. "global") approaches (namely the Owicki-Gries method [77] and derived methods). Indeed, we claim that several "non-strictly-compositional" methods fall in this intermediate region, and in particular those described in this section (i.e. Section 5).

Shankar elicits the differences between lazy compositionality and rely/guarantee compositionality in five points, namely:

1. components are not treated as black-boxes;

2. composition is not necessarily conjunction;

3. environment assumptions are specified as abstract components (rather than properties);

4. no rely/guarantee proof obligations are generated;

5. composition can yield inconsistent specifications.

Actually, we note that these differences apply only in a typical setting where we specify both a computational model and a formal language to express properties of computations in the model. For example, the "black-box" view referred to in point 1 implies a distinction between the implementation of a module (that is its description in the computational model) and some of its properties (which are described in some logic language). Similarly, "composition as conjunction" of point 2 is, in this case, a characteristic of the computational model, and an "abstract component" (in point 3) is, again, the implementation (in the computational model) of an abstract specification.

On the contrary, in this survey we are mainly interested in more abstract works that focus solely on the logical aspects, independently of any computational model (such as our own contribution [34]). In such models, only the points 4–5 apply. Therefore, let us describe those in some more detail.

No rely/guarantee proof obligations are generated. In fact, in lazy compositionality the discharging of assumptions is simply lazily postponed to when the necessary specification details are present. More precisely, the environment assumption is embedded in the specification by means of a component that represents explicitly the abstract environment. The goal of the following refinement steps is to show that the overall system subsumes the abstract environment of that component, thus showing that the explicit environment is superfluous and can be eliminated without affecting the functionality of the component (in fact, the rest of the system does play the role of the assumed environment).

Composition can yield inconsistent specifications. This feature is a consequence of the previous characteristic: since we lazily postpone the discharging of the assumptions, tentatively assuming that they are indeed dischargeable, it may happen that our suppositions show to be false according to any possible refinement of the system. In such cases, the specification is globally inconsistent and we must emend some previous specification choice.

We now finally describe how lazy compositionality is actually carried out, with reference to two modules (the generalization to an arbitrary number of modules is rather straightforward). Let us consider two modules with specifications P_1 and P_2. Note that we do not say anything about how these local specifications should be written; in particular they could also be rely/guarantee specifications, even if we will not use the rely/guarantee *methodology* to treat them. When composing these two modules, we explicitly add to the specification of each module the assumptions that the other module makes on its environment. So, if the two modules assume an environment with properties E_1 and E_2, respectively, then we strengthen P_1 to $P_1 \wedge E_2$ and P_2 to $P_2 \wedge E_1$. Then, we compose these two strengthened modules; in doing so, it is clear that the environment assumptions of each module are satisfied by its actual environment, by construction. More explicitly, in a setting where composition is conjunction, we end up with the system $(E_2 \wedge P_1) \wedge (E_1 \wedge P_2)$, which simply assumes explicitly the validity of the environment assumptions.

Then, the goal of lazy compositional verification, is to show that $(E_2 \wedge P_1) \wedge (E_1 \wedge P_2)$ can be refined to simply $P_1 \wedge P_2$: this would show explicitly that the environment assumptions are satisfied in the system which thus guarantees its functionalities. A drawback of this approach is that a module cannot be refined independently of the others, since each module is enlarged to comprehend the environment assumptions of the other modules, which must also be preserved in the refinement.

[79] also shows some examples and details on how the lazy approach can actually be pursued in a semantic setting similar to that of TLA [57]. This requires familiar inductive proof techniques that we do not discuss here.

All in all, we believe that the liberal lazy approach can be a useful alternative to rely/guarantee compositionality in some cases, even if it basically amounts to shifting the burden of compositional verification to other, standard, methodologies and techniques for verification.

5.3 Decomposition of Real-Time Transition Systems

Finkbeiner, Manna and Sipma present in [29] a framework for the modular specification and verification of concurrent systems, based on the formalism of fair transition systems. In particular, verification does not rely on *ad hoc* compositional proof rules, such as those of the works presented in Section 4. On the contrary, rely/guarantee formulas can be simply expressed using implications in linear temporal logic, and assumptions are discharged either by properties of other components, using standard deduction techniques, or by (future) refinements of the system. Until then, assumptions are simply carried over without being discharged. Therefore, this framework falls in the category of lazy methodology, discussed above in Section 5.2. More precisely, the lazy methodology is applied to a complex formalism that permits code reuse, composition, and information hiding. Note that the framework encompasses both composition and decomposition in its liberal approach, although the focus is mainly on the latter.

The basic computational model that is considered is *fair transition systems* [64]: a fair transition system describes a closed concurrent system. A *transition module* is a variation of a transition system to describe open components in an overall closed system: when composing transition modules to create a transition systems, visible transitions of the various modules which share the same label synchronize. Since in general a module can have an arbitrary environment, we say that a property (expressed, in this framework, in linear temporal logic) is *modularly valid* whenever it holds for the module in *any* environment. More precisely, the environment is only constrained not to modify the private and output variables of the module and not to synchronize on the internal transitions of the module.

Even if the requirement for modular validity is clearly a strong one, we can still have non trivial properties to be modularly valid by, for example, formulating them using the rely/guarantee form in temporal logic. For instance, using implication to represent the link between the assumption and the guarantee, if a module guarantees the property M whenever the environment guarantees the property E, then the property $E \Rightarrow M$ is clearly true of any environment: in particular it is true of all environments that do not satisfy E.

Then, a transition system can be declared as a collection of modules. More precisely, the formalism permits to compose modules by parallel composition, and to modify them using operations such as conditional implementation, hiding and renaming of variables, restricting and augmenting of the visible variables (thus providing a form of inheritance in the declarations of modules). All these operations can also be declared recursively (i.e. recursive definitions are allowed), and permit specification code reuse.

The verification of properties of a composite system applies standard techniques, plus their extensions and generalizations to handle modules. In particular, a set of simple inference rules to handle inheritance and refinement is provided; they rely on the known theory of simulation and refinement mappings (see e.g. [1]). In particular, as briefly discussed above, the framework permits to discharge assumptions of a module lazily, using those standard techniques. Another interesting notion is that of *interface* (i.e. *modular*) *abstraction*. Informally speaking, a module M *modularly simulates* another module N whenever M is simpler than N, and they have the same behavior at the interface. Therefore, in any global proof, M can be substituted for N, yielding an equivalent but simpler system. Even more, the *simplest* (with respect to the simulation preorder) element which modularly simulates a given module is well defined and can be generated automatically in the transition system formalism; it is called the *interface abstraction*. This is useful in simplifying composite systems and global proofs of properties.

Finally, the formalism is exemplified with the aid of the example of a multi-level arbiter.

Bjørner, Manna, Sipma and Uribe in [16] discuss what is basically an extension of the framework discussed above to treat real-time. Namely, transition modules are augmented with a variable representing (continuous) time to yield *clocked transition modules*. This approach to real-time is similar to that of Abadi and Lamport [3], among others.

A technical feature which must be considered in real-time modeling is that of *Zenoness* (see [35, 3]). The problem is that non-Zenoness is not preserved under parallel composition of clocked modules: it may happen that the composition of two modules is Zeno even if each module is non-Zeno. Therefore, the paper introduces the notion of *receptiveness*: roughly speaking, a module is receptive if it is non-Zeno whenever it acts in a "reasonably cooperative" environment. Receptiveness is preserved under parallel composition.

The formalism is illustrated with the classical example of the Generalized Railroad Crossing (see e.g. [38]).

Both formalisms (i.e. clocked [16] and unclocked [29] transition modules) are supported by the STeP (Stanford Temporal Prover [15]) tool, expressly designed to support specification and verification of these modular systems adopting various techniques (in particular, both theorem proving and model checking).

Finally, Cau proposes in [17] a composition and refinement framework somewhat similar to that presented above, but with reference to dense time temporal logics. The resulting framework is rather complicated and convoluted, and uses some of the ideas about compositionality presented in [18].

5.4 A Semantic Framework for Compositionality

Hooman presents in [46] a framework to support the top-down design of distributed real-time systems. The framework is based on assertions (i.e. it is denotational) and also *mixed*, that is a specification in the framework can contain both assertions and programming constructs (namely computations).

The framework is semantic, in that it is based on descriptions given by simple *semantic primitives*. Moreover, it is parametric with respect to the time model, which can be, in particular, discrete or continuous. More precisely, the only assumption on the time model is that it is a strictly ordered set. The basic semantic primitive used to describe the behavior of a system (or of a module) is the *observation function*, which is simply a function from the time domain to a set of events: thus, it associates each time instant to the primitive facts that occur at that instant. A *computation* represents a history of a module, and it is constituted by a pair $(\alpha, \mathcal{OBS})$. α is a set of observable events, which are the events that may be observed during the behavior of the component. $\mathcal{OBS}$ is a set of observation functions of the events in α, that is a set of mappings from time to subsets of α (note that we have more than a single observation function, since we allow for nondeterminism).

Two different definitions of parallel composition of two computations are given. Consider two computations $(\alpha_1, \mathcal{OBS}_1)$ and $(\alpha_2, \mathcal{OBS}_2)$. According to the first definition, the composition of the two computations is obtained by taking the union $\alpha_1 \cup \alpha_2$ of the observable events, and the pointwise (i.e. at all time points) union of the observation functions, requiring that these functions agree on the events shared by both components (i.e. which are in $\alpha_1 \cap \alpha_2$). The second definition of parallel composition allows for "open" computations where the observed events can also be not in α: this means that they include arbitrary environment behaviors or, in other words, that $\mathcal{OBS}$ is now a mapping from time to subsets of all the possible events. In this case, the parallel composition is defined simply by the union $\alpha_1 \cup \alpha_2$ of the observable events, and the (pointwise) intersection of the observation functions. The two definitions of parallel composition, although different, are strictly related, and simple conditions, basically involving the removal of the environment observables, permit to pass from a representation to the other. We indicate parallel composition with the operator $//$.

In Hooman's framework, a specification is a particular form of computation, defined by a pair $(\alpha, \mathcal{A})$. $\mathcal{A}$ is an assertion, that is a predicate over observation functions. Thus, the specification is a computation which has α as set of observable events, and all observation functions which satisfy the assertion $\mathcal{A}$. Note that no particular structure is given to a specification, and in particular the rely/guarantee paradigm is not adopted. Therefore, parallel composition basically reduces to set intersection, without allowing for mutual dependencies between components. On the other hand, *refinement* between specifications is indicated by the operator $\Rrightarrow$.

Now, the core of the framework consists in formulating a composition rule that permits to deduce soundly the specification of a parallel composition of two specifications. Let us consider two specifications $(\alpha_1, \mathcal{A}_1)$ and $(\alpha_2, \mathcal{A}_2)$. Then, their parallel composition is a refinement of the specification given by the union of the α's and the logical conjunction of the assertions, under a simple condition. Exactly, the soundness condition is that the assertion $\mathcal{A}_i$ only depends on the events in α_i, for $i = 1, 2$; this amounts to requiring that the two specifications are decoupled, and one does not predicate about the behavior of the events that belong to the other. Under this simple condition, the following refinement relation holds:

$$(\alpha_1, \mathcal{A}_1)//(\alpha_2, \mathcal{A}_2) \Rrightarrow (\alpha_1 \cup \alpha_2, \mathcal{A}_1 \wedge \mathcal{A}_2)$$

The paper also briefly discusses the hiding operation to selectively remove observable events from a specification or computation.

As it is clear even from this short presentation, Hooman's framework is very simple and quite general. Nonetheless, even if the semantic primitives are very basic notions, they naturally refer to semantic models such as those based on sequences of events and interleaving semantics (e.g. such as Abadi and Lamport's [4]). One advantage of the simplicity of the framework is that it can be easily implemented in a theorem prover. In fact, the paper completely formalizes the framework in PVS [78], and uses the formalization in proving properties of a hybrid system. A less formal version of a very similar framework was proposed by the same author in [44].

Finally, this framework, or variations closely related to it (such as those described in [42, 44]), have been used in the development of some significant case studies, namely an arbitration protocol [43] and a bus protocol [45].

6 Compositionality in Model Checking

The problem with the scalability of formal methods does not affect only deductive methods, such as those that we focused on elsewhere in this report, but also automata-based automated methods, and namely model checking techniques. The limitations of scalability of model checking manifest themselves in the *state explosion problem*: under conditions that happen often in practice, the size of the representation of the parallel composition of modules is exponential in the size of the individual components. Hence, composing modules of manageable sizes yields a global system whose size in unmanageable and cannot be verified automatically.

Compositional techniques try to soothe this problem by permitting the verification of components in isolation, while allowing to infer global properties of the overall system. For a thorough analysis of compositional model checking techniques we refer the reader to the work by Kupferman and Vardi [54], which is especially focused on rely/guarantee techniques and algorithmic and complex-theoretical aspects. [54] also contains several pointers to related literature on compositionality for model checking. Another in-depth analysis of recent compositional techniques for automata-based verification is the book by Juan and Tsai [52]. In particular, besides briefly surveying a large spectrum of compositional techniques for automata-based formalisms, it especially focuses on the models of Multiset Labeled Transition Systems and Timed Petri Nets. The book also considers experimental aspects, practically comparing some model checking tools on a set of common verification problems.

On the other hand, in this section, we just briefly present in a very plain manner some directions in applying compositional techniques to model checking, not limited to the rely/guarantee approach. The exposition follows mainly the nice survey by Berezin, Campos and Clarke [14] which, of course, contains much more details.

Let us present very shortly the essential elements of the model checking problem for a generic system. Consider a system whose state is an assignment of values to the (Boolean) variables from a set V. Let V' be a duplicated set of variables, corresponding to the variables in V at the next step, that is after the next transition is taken. The behavior of a component can be fully specified by a *transition function* $N(V, V')$ that describes all the constraints that each variable must satisfy in a legal transition of the system. Note that a transition function can be expressed as a Boolean propositional formula (over variables in V and V').

Now, let us consider a composite system, and denote by V the set of *all* variables in the system. Moreover, let $N_1, \dots, N_n$ be the transition functions for its n modules. The modules can be composed synchronously or asynchronously. In *synchronous* composition, the global transition function can be expressed as the *conjunction* of all transition functions of the n modules. In *asynchronous* composition, the global transition function can be expressed as a *disjunction* of terms, each of which represents the case of just one transition taken, while all the other variables remaining unchanged. (This is the interleaving model of asynchronous composition).

A fundamental operation in model checking consists in computing the *image* (or pre-image) of a set of states. Given a set of states S, the image of S is the set of all successors of the states in S. S can also be expressed as a Boolean formula, representing the truth assignments to the variables. Thus, the image of S can be represented by the formula:

$$\{V' \mid \exists V(S(V) \wedge N(V, V'))\}$$

In fact, it represents all the assignments to variables in V' (i.e. states) such that there exists another assignment V which is a state and a transition to the assignments in V' exists. In general, when N is given as the composition of (local) transition functions, finding the image has a complexity which is exponential in the size of the local transition functions. Thus, compositional methods must aid to reduce this complexity in most practical cases.

We can group the most important techniques for doing so into four main categories: partitioning, lazy composition, interface abstraction, and rely/guarantee techniques. Roughly speaking, we can say that they are listed from the simplest to the hardest, where simple means "largely automatable", whereas hard means "requiring substantial human assistance"; due to the trade-off between automation and efficiency, they also are from the least efficient to the most efficient, the latter meaning bringing the best results in terms of complexity reduction. Also notice that the terminology used in this section is slightly different than that used for deductive methods, which was illustrated at the beginning of this report.

Partitioning Partitioning consists in distributing the existential quantification of the image relation over several independent subformulas, the value of each of which can be computed independently of the others, thus reducing the verification effort.

This is especially simple for asynchronous composition, since existential quantification distributes over disjunctions. More explicitly, if the relation we must evaluate is $\exists V(S(V) \wedge (N_1(V, V') \vee \cdots \vee N_n(V, V')))$, we can rewrite it as:

$$\exists V(S(V) \wedge N_1(V, V')) \vee \cdots \vee \exists V(S(V) \wedge N_n(V, V'))$$

each of whose terms can be evaluated independently.

On the other hand, in synchronous composition the transition function is a conjunction of transition functions in the form $\exists V(S(V) \wedge (N_1(V, V') \wedge \cdots \wedge N_n(V, V')))$, and in general existential quantification does not distribute over conjunctions. However, partitioning is often still possible in practice since transition functions often exhibit *locality*, that is most N_is depend only on a small number of variables in V and V'. Thus, we can partition into independent quantifications over subsets of variables which only appear in some terms. Various techniques have been devised to implement efficiently such partitioning. As it is expected, in general finding an efficient partitioning is a computationally intractable problem, even if many heuristics which perform well in practice have been developed and successfully used.

Lazy Composition Lazy parallel composition is a method, alternative to that described in the previous paragraph, for partitioning the global transition relation into subformulas which can be evaluated independently and more efficiently. However, lazy composition is not an exact method, since it relies on a simplified transition relation to compute the partitioning. The simplified (a.k.a. restricted) transition relation agrees with the "real" transition relation on "most important" states, but may disagree on other ones. However, the simplified transition relation is simpler to partition, and can be computed more efficiently. Then, the discrepancies between the simplified and real relations can be checked *a posteriori*, to verify that they are irrelevant for the specific instance of the problem. Of course, it may happen that this check fails: in such cases the lazy technique has been unsuccessful. However, efficient lazy techniques, which behave well on most problem instances, have been developed.

Interface Abstraction This technique is based on the observation that processes often communicate only through a small number of shared variable. For example, consider two processes P_1 and P_2 with transition functions N_1 and N_2, respectively, and which communicate using a set $\sigma \subset V$ of variables. P_1 observes the behavior of P_2 only through variables in σ. Thus, for verification purposes, we can replace P_2 by an equivalent, but simpler, process A_2 which is indistinguishable from P_2 with respect to the variables in σ. Then, verifying the parallel composition of P_1 and A_2 in general involves a much simpler global transition function, but yields results which are equivalent to verifying the parallel composition of P_1 and P_2, for properties involving only variables in σ. When using this technique, the most important thing is to define an effective notion of interface equivalence, which is both easy to compute and yields good results in terms of verification complexity. Extensively studied examples of such equivalences are bisimulation equivalence and stuttering equivalence.

Rely/Guarantee Techniques In rely/guarantee reasoning, each component is verified separately. In order to do so, we assume that the environment behaves in a certain manner, described by a logic formula (rather than a component, as in interface abstraction). Then, suitable inference rules permit to infer the global validity of the overall system, as discussed extensively in this survey. The peculiarity of rely/guarantee for model checking is that it is usually based on a preorder relations between models that captures the notion of "more behaviors". In our framework of Section 3, the preorder would implement the abstract entailment relation $\sqsubseteq$, so that discharging the assumptions correspond to checking a preorder relation. Conversely, the relation must be suitable for algorithmic verification if one wants to facilitate model checking. Still, the method is only semi-automated, since the user has to provide a suitable choice for assumptions and guarantees, which cannot be completely automated.

7 Compositionality for Your Own Formalism

In the last decade (or more), the need for compositionality, to allow the scalability of formal methods, has become indisputable in the research community. As a consequence of that, almost every new formal language encompasses some form of compositional technique or permits compositional verification. At the same time, much work has also been devoted to the introduction of compositional techniques and methods for existing formalisms, possibly modeling these methods on some of the general frameworks discussed in Section 4.

An extensive report of all these developments is definitely out of the scope of this paper, which focuses on general frameworks. Therefore, this section just briefly samples a few examples of applications of compositionality to some formalisms. More references can be found, for example, in [27] and in [26].

7.1 Reactive Modules

Alur and Henzinger present in [8] a formal model for concurrent systems: the reactive modules. This model is suitable for representing concurrent systems with both synchronous and asynchronous components. It is especially tailored to the specification and verification of digital components; in fact it allows for descriptions of components which resemble those of a hardware description language (e.g. VHDL or System C).

Each module is a collection of variables (possibly including a *tick* variable to model time lapsing), divided into the three classes *private*, *interface* (variables visible to the environment), and *external* (variables updated by the environment). Various models of (a)synchrony are also supported, in order to allow for the translation of different programming languages and hardware description languages into reactive modules.

The semantics of a reactive module is defined in terms of traces of observations. First, a meaning for module execution is defined by imposing some restrictions on the ordering of the variables of a module with respect to their reciprocal dependencies when being updated (in particular, the order relation must be asymmetric). Then, the semantics of a module is defined in terms of states (a set of variables values) and transitions between states (when a state can be the result of an update execution of another state).

A refinement relation (called *implementation relation*) is defined in terms of trace equivalence restricted to observable variables. Special classes of modules can also be defined according to certain characteristics of the definition (e.g. stuttering invariance, reactive behavior, etc.).

Reactive modules can be composed both spatially and temporally. *Spatial composition* involves the usual operations of variable renaming and hiding, and parallel composition. Simple proof rules for compositional rely/guarantee reasoning are provided, especially for proving refinement of composite modules. *Temporal composition* involves the operations of round abstraction and triggering. Round abstraction means that several temporally consecutive rounds are combined into a single "scaled" one. Triggering, on the other hand, splits a round into intermediate rounds so that the module sleeps through them until some predefined variables change their values.

Finally, some typical issues of transition modules, namely fairness, safety and receptiveness, are discussed and applied to the framework of reactive modules. This completes the presentation of this unified, modular and hierarchical framework for describing reactive components.

The framework is definitely interesting and efficient for describing hardware components, where one familiar with a hardware description language can rather easily transfer its experience to the use of reactive modules.

Henzinger, Qadeer and Rajamani present in [39] a commented case study of application of the reactive modules framework [8] with the MOCHA tool [9], consisting of a hardware pipeline. The case study offers the opportunity to introduce some methodological remarks and some "learned lessons" about the practical applicability of rely/guarantee reasoning and refinement checking (i.e. model checking of refined modules). The bottom line is twofold. On the one hand, the success of rely/guarantee reasoning depends crucially on the construction of suitable *abstraction* modules. The modules must be neither too complex (otherwise modular verification is no simpler than unstructured verification) nor too abstract (otherwise we cannot exploit the composition rules to deduce the validity of the global specification from the local specifications). On the other hand, the success of refinement checking depends crucially on the construction of suitable *witness* modules (which are roughly the analogous of refinement mappings to show the correspondence of variables between a module and its refinement). The careful application of both principles permits the automatic verification of otherwise untreatable models, as in the example presented in the paper.

7.2 TTM/RTTL

Ostroff presents in [76] a structured compositional design method for discrete real-time systems. The method is based on the TTM/RTTL framework [73, 74], where modules are described as TTM transition systems, and specifications are written using the real-time temporal logic RTTL. The framework also supports model checking and theorem proving techniques embedded in the toolset *State-Time* [75], based on the STeP [15] tool.

First, the TTM/RTTL framework is introduced. RTTL (Real-Time Temporal Logic) is obtained from (untimed, i.e. without a metric) linear-time temporal logic by adding a fair *tick* transition. Then, states (defined as mappings from the set of system variables to values) are linked by transitions (which model events occurrences or time lapsing). Properties are expressed using variations of the usual temporal logic operators that include quantitative bounds on time (i.e. number of occurrences of tick transitions), similarly to, for example, Koymans [53]. TTMs (Timed Transition Models) are an extension of Fair Transition Systems [64] for real-time modeling. As usual, their semantics is defined in terms of trajectories satisfying some fairness constraints. Modules are defined as compositional extensions of TTMs. Notice that this extension is different from [19], where the focus is on deductive methods and different restrictions are imposed on the definition of a module (in particular, the transitions must be self-disabling). A module is made of an *interface stub* (consisting of all the variables shared with the environment), a *body* (a TTM whose statements are not visible outside the module) and a *specification* (a RTTL formula in the interface variables, expressing the required visible behavior of the module).

The parallel composition of two modules gives a new module, whose interface contains some of the variables of the interfaces of the two composed modules (while some other variables may be hidden in the composite module). The body is given by ordinary composition between TTMs (i.e. similar to that of Fair Transition Systems [64]). The specification is given by the logical conjunction of the specifications of the two composed modules. Modules of this framework are compositional. A composition rule is given to infer the validity of a global specification for a composite module from the validity of the specifications that are local to the composing modules. The composition rule can be used both bottom-up (to reuse preexisting modules) and top-down (by devising a suitable decomposition to split the verification burden among submodules). The paper also sketches a methodology that suggests the use of model checking to verify the modular validity of the composed modules, and of deductive methods to prove that the global system requirement follows from the specification of the modules. As an aside, when model checking a module, it may be useful to constrain the evolution of the variables of its environment according to the description of the modules that will actually constitute its environment, provided they are fixed. This reduces the state-space of the model to be checked, thus reducing the complexity of the automatic procedure.

The analysis of module refinement aims at formulating conditions under which we can replace a module's body by another one, without affecting the observed behavior at the interface. The notion of program equivalence developed for Fair Transition Systems [64] is not suitable for real-time modules, since it considers as equivalent modules whose *timed* behavior is instead different (in other words, the tick transition is not handled properly). Therefore, the author develops a *state-event* notion of *observational equivalence*, taken from [59, 58, 60]. The notion of (weak) state-event bisimulation is introduced, where sequences of pairs (state, transition) are considered. Two pairs, one referring to the first module and one to the other, are considered bisimilar if either they lead directly to equal next pairs, or they lead to equal next pairs after a sequence of transitions which do not involve visible variables (these are called "unobservable moves"). Finally, if the similarity between pairs holds for all the (state, transition) pairs of both modules, then the modules can be considered equivalent with respect to the refinement relation. Whenever two modules are equivalent, the modular validity of any formula for one module implies the modular validity of the same formula for the other module; in other words, all the properties shown to hold for a module also hold for the refinement of that module. This equivalence is also compositionally consistent, in that we can replace a module with an equivalent one without affecting the behavior of the other module being composed with it.

Finally, a case study is described to show in practice the applicability of the proposed framework and methodology. In particular, the author points out the fact that the design of composite real-time systems often requires a combination of top-down and bottom-up approaches, since an inherently composite system has no "top" function and involves properties which are "emergent" from the whole system, rather than being easy to partition.

Ostroff's methodology is rather comprehensive and encompasses various techniques in a unified framework. The proposed framework for composition and decomposition is similar, under several aspects, to other well-known frameworks for rely/guarantee compositional reasoning, such as [4, 64], with the important addition of quantitative real-time. Its main limitations are the fact that it considers only discrete time and that the use of a tick transition to model time lapsing may be considered unnatural and bring a description of the evolution of the state of the system which is not intuitively very appealing.

7.3 Interval-Based Logics

Compositionality for the interval-based logic ITL (Interval Temporal Logic [68]) is discussed by Moszkowski in [69]. The paper introduces a compositional methodology based on the rely/guarantee paradigm, combined with a technique based on fixpoints.

Duration Calculus [20, 22, 21] is basically an extension of Interval Temporal Logic to dense time domains. Compositionality is introduced in Duration Calculus by Xu and Swarup in [86]. They adopt the rely/guarantee paradigm, and precisely they introduce a rule modeled after that of [85, 18].

Another work focusing on compositionality for Duration Calculus is that by Olderog and Dierks [72], where they show how a specification written in an implementable subset of Durations Calculus can be decomposed into an untimed (i.e. non real-time) part in parallel composition with a set of (real-time) timers.

References

[1] Martín Abadi and Leslie Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, 1991.

[2] Martín Abadi and Leslie Lamport. Composing specifications. *ACM Transactions on Programming Languages and Systems*, 15(1):73–132, 1993.

[3] Martín Abadi and Leslie Lamport. An old-fashioned recipe for real time. *ACM Transactions on Programming Languages and Systems*, 16(5):1543–1571, 1994.

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

[5] Martín Abadi and Stephan Merz. An abstract account of composition. In Jirí Wiedermann and Petr Hájek, editors, *Proceedings of the 20th International Symposium on Mathematical Foundations of Computer Science (MFCS'95)*, volume 969 of *Lecture Notes in Computer Science*, pages 499–508. Springer-Verlag, 1995.

[6] Martín Abadi and Gordon D. Plotkin. A logical view of composition. *Theoretical Computer Science*, 114(1):3–30, June 1993.

[7] Bowen Alpern and Fred B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, 1985.

[8] Rajeev Alur and Thomas A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15:7–48, 1999.

[9] Rajeev Alur, Thomas A. Henzinger, Freddy Y. C. Mang, Shaz Qadeer, Sriram K. Rajamani, and Serdar Tasiran. MOCHA: Modularity in model checking. In Alan J. Hu and Moshe Y. Vardi, editors, *Proceedings of the 10th International Conference on Computer Aided Verification (CAV'98)*, volume 1427 of *Lecture Notes in Computer Science*, pages 521–525. Springer-Verlag, 1998.

[10] Nina Amla, E. Allen Emerson, Kedar Namjoshi, and Richard Trefler. Abstract patterns of compositional reasoning. In Roberto M. Amadio and Denis Lugiez, editors, *Proceedings of the 14th International Conference on Concurrency Theory*, volume 2761 of *Lecture Notes in Computer Science*, pages 431–445. Springer-Verlag, 2003.

[11] Ignacio Angelelli, editor. *Gottlob Frege. Kleine Schriften.* George Olms, 1967.

[12] Krzysztof R. Apr, Nissim Francez, and Willem-Paul de Roever. A proof system for Communicating Sequential Processes. *ACM Transactions on Programming Languages and Systems*, 2(3):359–385, 1980.

[13] Ed A. Ashcroft. Proving assertions about parallel programs. *Journal of Computer and System Sciences*, 10:110–135, 1975.

[14] Sergey Berezin, Sérgio Campos, and Edmund M. Clarke. Compositional reasoning in model checking. In de Roever et al. [27], pages 81–102.

[15] Nikolaj S. Bjørner, Anca Browne, Edward Y. Chang, Michael Colón, Arjun Kapur, Zohar Manna, Henny B. Sipma, and Tomás E. Uribe. STeP: deductive-algorithmic verification of reactive and real-time systems. In Rajeev Alur and Thomas A. Henzinger, editors, *Proceeding of the 8th International Conference on Computer Aided Verification (CAV'96)*, volume 1102 of *Lecture Notes in Computer Science*, pages 415–418. Springer-Verlag, 1996.

[16] Nikolaj S. Bjørner, Zohar Manna, Henny B. Sipma, and Tomás E. Uribe. Deductive verification of real-time systems using STeP. *Theoretical Computer Science*, 253:27–60, 2001.

[17] Antonio Cau. Composing and refining dense temporal logic specifications. *Formal Aspects of Computing*, 12(1):52–70, 2000.

[18] Antonio Cau and Pierre Collette. Parallel composition of assumption-commitment specifications: A unifying approach for shared variable and distributed message passing concurrency. *Acta Informatica*, 3(2):153–176, 1996.

[19] Edward Chang. Compositional verification of reactive and real-time systems. Ph.D. Thesis CS-TR-94-1522, Stanford University, Department of Computer Science, 1993.

[20] Zhou Chaochen, Charles Anthony Richard Hoare, and Anders P. Ravn. A calculus of duration. *Information Processing Letters*, 40(5):269–276, 1991.

[21] Zhou Chaochen, Dang Van Hung, and Li Xiaoshan. A duration calculus with infinite intervals. In Horst Reichel, editor, *Proceedings of the 10th International Symposium on Fundamentals of Computation Theory (FCT'95)*, volume 965 of *Lecture Notes in Computer Science*, pages 16–41. Springer-Verlag, 1995.

[22] Zhou Chaochen, Anders P. Ravn, and Michael R. Hansen. An extended duration calculus for hybrid real-time systems. In Robert L. Grossman, Anil Nerode, Anders P. Ravn, and Hans Rischel, editors, *Hybrid Systems 1992*, volume 736 of *Lecture Notes in Computer Science*, pages 36–59. Springer-Verlag, 1993.

[23] Frank S. de Boer and Willem-Paul de Roever. Compositional proof methods for concurrency: A semantic approach. In de Roever et al. [27], pages 632–646.

[24] Willem-Paul de Roever. The quest for compositionality — a survey of assertion-based proof systems for concurrent programs (part 1: concurrency based on shared variables). In *Proceedings of the IFIP Working Conference on The Role of Abstract Models in Computer Science.* North-Holland, 1985.

[25] Willem-Paul de Roever. The need for compositional proof systems: A survey. In de Roever et al. [27], pages 1–22.

[26] Willem-Paul de Roever, Frank de Boer, Ulrich Hannemann, Jozef Hooman, Yassine Lakhnech, Mannes Poel, and Job Zwiers. *Concurrency Verification: Introduction to Compositional and Noncompositional Methods.* Cambridge University Press, 2001.

[27] Willem-Paul de Roever, Hans Langmaack, and Amir Pnueli, editors. *Proceedings of the International Symposium: "Compositionality: The Significant Difference" (COMPOS'97)*, volume 1536 of *Lecture Notes in Computer Science.* Springer-Verlag, 1998.

[28] Wim H. J. Feijen and A. J. M. van Gasteren. *On a Method of Multiprogramming.* Springer-Verlag, 1999.

[29] Bernd Finkbeiner, Zohar Manna, and Henny B. Sipma. Deductive verification of modular systems. In de Roever et al. [27], pages 239–275.

[30] Robert W. Floyd. Assigning meanings to programs. In J. T. Schwartz, editor, *Mathematical Aspects of Computer Science*, volume 19 of *Proceedings of Symposia in Applied Mathematics*, pages 19–32. American Mathematical Society, 1967.

[31] Nissim Francez and Amir Pnueli. A proof methods for cyclic programs. *Acta Informatica*, 9:133–157, 1978.

[32] Gottlob Frege. *Logische Untersuchungen. Dritter Teil: Gedankengefüge*, volume 3, chapter Beiträge zur Philosophie des Deutschen Idealismus, pages 36–51. 1923. Reprinted in [11, pp. 378–394]. English translation in [33].

[33] Gottlob Frege. Compound thoughts. In P. Geach and N. Black, editors, *Logical Investigations*, Oxford, 1977. Blackwells.

[34] Carlo A. Furia, Matteo Rossi, Dino Mandrioli, and Angelo Morzenti. Automated compositional proofs for real-time systems. In Maura Cerioli, editor, *Proceedings of Fundamental Aspects of Software Engineering (FASE'05)*, volume 3442 of *Lecture Notes in Computer Science*, pages 326–340. Springer-Verlag, 2005.

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

[36] H. H. Goldstine and John von Neumann. Planning and coding problems for an electronic computer. In A. H. Taub, editor, *Collected Works of John von Neumann*, volume Volume 5, pages 80–235. Pergamon Press, 1963. Original publication year: 1947.

[37] Mark R. Heckman, Cui Zhang, Brian R. Becker, Dave Peticolas, Karl N. Levitt, and Ronald A. Olsson. Towards applying the composition principle to verify a microkernel operating system. In Joakim von Wright, Jim Grundy, and John Harrison, editors, *Proceedings of the 9th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'96)*, volume 1125 of *Lecture Notes in Computer Science*, pages 235–250. Springer-Verlag, 1996.

[38] Constance Heitmeier and Dino Mandrioli, editors. *Formal Methods for Real-Time Computing.* John Wiley & Sons, 1996.

[39] Thomas A. Henzinger, Shaz Qadeer, and Sriram K. Rajamani. You assume, we guarantee: Methodology and case studies. In Alan J. Hu and Moshe Y. Vardi, editors, *Proceedings of the 10th International Conference on Computer-Aided Verification (CAV'98)*, volume 1427 of *Lecture Notes in Computer Science*, pages 440–451. Springer-Verlag, 1998.

[40] Thomas A. Henzinger, Shaz Qadeer, Sriram K. Rajamani, and Serdar Tasiran. An assume-guarantee rule for checking simulation. *ACM Transactions on Programming Languages and Systems*, 24(1):51–64, 2002.

[41] Charles Antony Richard Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[42] Jozef Hooman. *Specification and Compositional Verification of Real-Time Systems*, volume 558 of *Lecture Notes in Computer Science.* Springer-Verlag, 1991.

[43] Jozef Hooman. Compositional verification of a distributed real-time arbitration protocol. *Real-Time Systems*, 6(2):173–206, 1994.

[44] Jozef Hooman. Correctness of real time systems by construction. In *Proceedings of the 3rd International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 863 of *Lecture Notes in Computer Science*, pages 19–40. Springer-Verlag, 1994.

[45] Jozef Hooman. Verifying part of the ACCESS.bus protocol using PVS. In P. S. Thiagarajan, editor, *Proceedings of the 15th Conference on Foundations of Software Technology and Theoretical Computer Science*

(FSTTCS'95), volume 1026 of *Lecture Notes in Computer Science*, pages 96–110. Springer-Verlag, 1995.

[46] Jozef Hooman. Compositional verification of real-time applications. In de Roever et al. [27], pages 276–300.

[47] Jozef Hooman and Willem-Paul de Roever. The quest goes on: a survey of proof systems for partial correctness of CSP. In J. W. de Bakker, Willem-Paul de Roever, and Grzegorz Rozenberg, editors, *Current Trends in Concurrency*, volume 224 of *Lecture Notes in Computer Science*, pages 343–395. Springer-Verlag, 1986.

[48] Theo M. V. Janssen. An overview of compositional translations. In de Roever et al. [27], pages 327–349.

[49] Cliff B. Jones. *Development Methods for Computer Programs Inclusing a Notion of Interference.* PhD thesis, Oxford University Computing Laboratory, 1981.

[50] Cliff B. Jones. Tentative steps towards a development method for interfering programs. *ACM Transactions on Programming Languages and Systems*, 5(4):596–619, 1983.

[51] Bengt Jonsson and Yih-Kuen Tsay. Assumption/guarantee specifications in linear-time temporal logic. *Theoretical Computer Science*, 167(1–2):47–72, 1996.

[52] Eric Y. T. Juan and Jeffrey J. P. Tsai. *Compositional Verification of Concurrent and Real-Time Systems.* Kluwer Academic Publishers, 2002.

[53] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[54] Orna Kupferman and Moshe Y. Vardi. An automata-theoretic approach to modular model checking. *ACM Transactions on Programming Languages and Systems*, 22(1):87–128, January 2000.

[55] Robert P. Kurshan and Leslie Lamport. Verification of a multiplier: 64 bits and beyond. In Costas Courcoubetis, editor, *Proceedings of the 5th International Conference on Computer-Aided Verification (CAV'93)*, volume 697 of *Lecture Notes in Computer Science*, pages 166–179. Springer-Verlag, 1993.

[56] Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, 3(2):125–143, 1977.

[57] Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, 1994.

[58] Mark S. Lawford, Jonathan S. Ostroff, and W. M. Wonham. Model reduction of modules for state-event temporal logics. In *IFIP Joint International Conference on Formal Description Techniques (FORTE-PSTV'96)*, pages 263–278. Chapman & Hall, 1996.

[59] Mark S. Lawford and W. M. Wonham. Equivalence preserving transformations for timed transition models. In *Proceedings of the 31st IEEE Conference on Decision and Control (CDC'92)*, pages 3350–3356, 1992.

[60] Mark S. Lawford, W. M. Wonham, and Jonathan S. Ostroff. State-event labels for labelled transition systems. In *Proceedings of the 33rd IEEE Conference on Decision and Control (CDC'94)*, pages 3642–3648, 1994.

[61] Gary M. Levin and David Gries. A proof technique for Communicating Sequential Processes. *Acta Informatica*, 15:281–302, 1981.

[62] Patrick Maier. A set-theoretic framework for assume-guarantee reasoning. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP'01)*, volume 2076 of *Lecture Notes in Computer Science*, pages 821–834. Springer-Verlag, 2001.

[63] Patrick Maier. Compositional circular assume-guarantee rules cannot be sound and complete. In Andrew D. Gordon, editor, *Proceedings of the 16th International Conference on Foundations of Software Science and Computational Structures (FOSSACS'03)*, volume 2620 of *Lecture Notes in Computer Science*, pages 343–357. Springer-Verlag, 2003.

[64] Zohar Manna and Amir Pnueli. *Temporal Verification of Reactive Systme: Safety.* Springer-Verlag, 1995.

[65] Ken McMillan. Circular compositionl reasoning about liveness. In Laurence Pierre and Thomas Kropf, editors, *Proceedings of the 10th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME'99)*, volume 1703 of *Lecture Notes in Computer Science*, pages 342–345. Springer-Verlag, 1999.

[66] Merriam-Webster, editor. *Merriam-Webster's Collegiate Dictionary.* Merriam-Webster Inc., 10th edition edition, 1998.

[67] Jayadev Misra and K. Mani Chandy. Proof of networks of processes. *IEEE Transactions on Software Engineering*, 7(4):417–426, 1981.

[68] Ben C. Moszkowski. A temporal logic for multilevel reasoning about hardware. *IEEE Computer*, 18(2):10–19, 1985.

[69] Ben C. Moszkowski. Compositional reasoning using interval temporal logic and tempura. In de Roever et al. [27], pages 439–464.

[70] Kedar S. Namjoshi and Richard J. Trefler. Branching time compositional reasoning. Unpublished draft (as of 2004).

[71] Kedar S. Namjoshi and Richard J. Trefler. On the completeness of compositional reasoning. In E. Allen Emerson and A. Prasad Sistla, editors, *Proceedings of the 12th International Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *Lecture Notes in Computer Science*, pages 139–153. Springer-Verlag, 2000.

[72] Ernst-Rüdiger Olderog and Henning Dierks. Decomposing real-time specifications. In de Roever et al. [27], pages 465–489.

[73] Jonathan S. Ostroff. *Temporal Logic for Real-Time Systems*. Advanced Software Development Series. Research Studies Press, 1989.

[74] Jonathan S. Ostroff. Deciding properties of timed transition models. *IEEE Transactions on Parallel and Distributed Systems*, 1(2):170–183, 1990.

[75] Jonathan S. Ostroff. A visual toolset for the design of real-time discrete-event systems. *IEEE Transactions on Control Systems Technology*, 5(3):320–337, 1997.

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

[77] Susan Owicki and David Gries. An axiomatic proof technique for parallel programs. *Acta Informatica*, 6:319–340, 1976.

[78] Sam Owre, John M. Rushby, and Natarajan Shankar. PVS: A Prototype Verification System. In *Proceedings of the 11th International Conference on Automated Deduction (CADE-11)*, volume 607 of *Lecture Notes in Computer Science*, pages 748–752. Springer-Verlag, 1992.

[79] Natarajan Shankar. Lazy compositional verification. In de Roever et al. [27], pages 541–564.

[80] Eugene W. Stark. Proving entailment between conceptual state specifications. *Theoretical Computer Science*, 56(1):135–154, 1988.

[81] Jei-Wen Teng and Yih-Kuen Tsay. Composing temporal-logic specifications with machine assistance. In Keijiro Araki, Stefania Gnesi, and Dino Mandrioli, editors, *Proceedings of the 12th International Symposium of Formal Methods Europe (FME'03)*, volume 2805 of *Lecture Notes in Computer Science*, pages 719–738. Springer-Verlag, 2003.

[82] Yih-Kuen Tsay. Compositional verification in linear-time temporal logic. In Jerzy Tiuryn, editor, *Proceedings of the 3rd International Conference on Foundations of Software Science and Computation Structures (FOSSACS'00)*, volume 1784 of *Lecture Notes in Computer Science*, pages 344–358. Springer-Verlag, 2000.

[83] Alan Turing. On checking a large routine. In *Report of a conference on high-speed automatic calculating machines*. University Mathematical Laboratory, Cambridge, 1949.

[84] Mahesh Viswanathan and Ramesh Viswanathan. Foundations for circular compositional reasoning. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP'01)*, volume 2719 of *Lecture Notes in Computer Science*, pages 835–847. Springer-Verlag, 2001.

[85] Qiwen Xu, Antonio Cau, and Pierre Collette. On unifying assumption-commitment style proof rules for concurrency. In Bengt Jonsson and Joachim Parrow, editors, *Proceedings of the 5th International Conference*

on Concurrency Theory (CONCUR'94), volume 836 of *Lecture Notes in Computer Science*, pages 267–282. Springer-Verlag, 1994.

[86] Qiwen Xu and Mohalik Swarup. Compositional reasoning using the assumption-commitment paradigm. In de Roever et al. [27], pages 565–583.

[87] Job Zwiers. *Compositionality, Concurrency and Partial Correctness — Proof theories for networks of processes, and their relationship*, volume 321 of *Lecture Notes in Computer Science*. Springer-Verlag, 1989.