Abstract Patterns of Compositional Reasoning

Nina Amla[1], E. Allen Emerson[2], Kedar Namjoshi[3], and Richard Trefler[4]

[1] Cadence Design Systems
[2] Univ. of Texas at Austin *
[3] Bell Labs, Lucent Technologies
[4] Univ. of Waterloo **

Abstract. Compositional Reasoning — reducing reasoning about a concurrent system to reasoning about its individual components — is an essential tool for managing proof complexity and state explosion in model checking. Typically, such reasoning is carried out in an *assume-guarantee* manner: each component guarantees its behavior based on assumptions about the behavior of other components. Restrictions imposed on such methods to avoid unsoundness usually also result in incompleteness — i.e., one is unable to prove certain properties. In this paper, we construct an abstract framework for reasoning about process composition, formulate an assume-guarantee method, and show that it is sound and semantically complete. We then show how to instantiate the framework for several common notions of process behavior and composition. For these notions, the instantiations result in the first methods known to be complete for mutually inductive, assume-guarantee reasoning.

1 Introduction

A large system is typically structured as a composition of several smaller components that interact with one another. An essential tool for the formal analysis of such systems is a *compositional reasoning* method — one that reduces reasoning about the entire system to reasoning about its individual components. This is particularly important when applying model checking [10, 25] to a concurrent composition of interacting, non-deterministic processes, where the full transition system can have size exponential in the number of components. This *state explosion* problem is one of the main obstacles to the application of model checking. Compositional reasoning techniques (see e.g., [12, 11]) are particularly useful for ameliorating state explosion, since they systematically decompose the model checking task into smaller, more tractable sub-tasks. A typical *assume-guarantee* style of reasoning (cf. [9, 17, 2, 6, 21, 23]), establishes that the composition of processes P_1 and P_2 refines the composition of Q_1 and Q_2 if P_1 composed

* This author's research is supported in part by NSF grants CCR-009-8141 & ITR-CCR-020-5483, and SRC Contract No. 2002-TJ-1026.
** Sponsored in part by an Individual Discovery Grant from NSERC of Canada.

with Q_2 refines Q_1, and Q_1 composed with P_2 refines Q_2. Here, Q_1 and Q_2 act as mutually inductive hypotheses.

However, existing methods for compositional reasoning can be hard to apply, for a number of reasons. Firstly, they are often bound to a particular syntax for describing processes, and particular notions of behavior and composition. Thus, it is not clear if a reasoning pattern devised for one such choice also applies to another. Another key factor is that several methods are known to be incomplete [23]. The completeness failure can usually be traced back to restrictions placed to avoid unsound, semantically circular reasoning with assumptions and guarantees. As argued in [11], completeness is an important property for a proof method. An incomplete method can be a serious impediment in practice, since it can make it impossible to prove that a correct program is correct. Moreover, the completeness failures demonstrated in [23] all occur for simple and common programming patterns. Lastly, safety and liveness properties are handled differently by most methods. For instance, the above method is *not* sound if both Q_1 and Q_2 include liveness or fairness constraints. It appears that there is a delicate balance between adding enough restrictions to avoid unsound circular reasoning, while yet allowing enough generality to ensure that the method is complete for both safety and liveness properties.

This paper addresses these problems in the following way. First, we construct an abstract, algebraic framework to reason about processes and composition. We formulate a mutually inductive, assume-guarantee method that applies to both safety and liveness properties, and show that it is sound and complete, all within the abstract setting. The framework makes explicit all assumptions needed for these proofs, uses as few assumptions as possible, and clarifies the key ideas used to show soundness and completeness. Our proof method extends the one given above with a soundness check for liveness properties. We show that a simple extension of the proof method in [6], which replaces Q_1 in the second hypothesis with its safety closure, is also complete. The two methods are closely related, but we show that ours is more widely applicable.

We then show how the abstract framework can be concretized in several different ways, obtaining a sound and complete method for each instantiation. In this paper, we discuss interleaving and fully synchronous composition, and notions of process behavior that include liveness, fairness, branching and closure under stuttering. The resulting instantiations are the first mutually inductive, assume-guarantee methods known to be semantically complete for general properties. That such diverse notions of composition and behavior can be handled in a common framework may seem surprising. To a large extent, this is due to the key property that, in each case, composition is represented as a conjunction of languages (cf. [4, 2]). The abstract framework thus provides a clean separation between the general axioms needed for soundness and completeness, and the assumptions needed for their validity in specific contexts. It simplifies and unifies a large body of work, and allows one to easily experiment with — and prove correct — different patterns of compositional reasoning.

Related Work: Methods for compositional reasoning about concurrent processes have been extensively studied for nearly three decades. Assume-guarantee reasoning was introduced by Chandy and Misra [9] and Jones [16] for analyzing safety properties. These methods were extended to some progress properties in the following decade (e.g., [24]; the book [11] has a comprehensive historical survey). More recently, Abadi and Lamport [2] and McMillan [21] extended the methods to temporal liveness properties. However, as shown by Namjoshi and Trefler [23], these extensions are not complete, usually for liveness properties of simple programs. Building on McMillan's formulation, they present a complete method for model checking of linear time temporal logic properties.

The methods presented in this paper apply to a process refinement methodology. In this setting, the method of [2] for asynchronous composition is incomplete [23]. Our methods are complete for asynchronous composition, both with and without closure under stuttering. Alur and Henzinger propose a method in [6] for the Reactive Modules language. We show that our new formulation, and a slight extension of their method are complete for this setting. Henzinger et. al. [15] showed how the same pattern of reasoning applies also to simulation-based refinement of Moore machines. Our proof method is different, and applies somewhat more generally (e.g., to Mealy machines). A major contribution of this paper, we believe, is the demonstration that all of these instantiations can be obtained from a single abstract pattern of reasoning. There is work by Abadi and Plotkin [4], Abadi and Merz [3], Viswanathan and Viswanathan [26], and Maier [18] on similar abstract formulations, but none of these result in complete methods. For a (non-standard) notion of completeness, Maier [19] shows that sound, circular, assume-guarantee rules cannot be complete, and that complete rules (in the standard sense) must use auxiliary assertions.

2 Abstract Compositional Reasoning

Notation. We use a notation popularized by Dijkstra and Scholten [13]. In the term $(\mathcal{Q}x : r(x) : p(x))$, $\mathcal{Q}$ is a quantifier, $r(x)$ is the range for variable x, and $p(x)$ is the term being operated on. The operator $[\phi]$ (read as "box") universally quantifies over the free variables of ϕ. Proof steps are linked by a transitive connective such as $\equiv$ or $\Rightarrow$, with an associated hint. For convenience, we move freely between set-based and predicate-based notations. For instance, $a \in S$ may be written as the predicate $S(a)$, and $[A \Rightarrow B]$ represents $A \subseteq B$.

2.1 Processes, Closure, and Composition

The abstract space of *processes* is denoted by $\mathcal{P}$. The set of abstract process *behaviors*, $\mathcal{B}$, is assumed to be equipped with a partial order $\preceq$ (read as "prefix"), and partitioned into non-empty subsets of finite behaviors, $\mathcal{B}_*$, and infinite behaviors, $\mathcal{B}_\infty$. We make the following assumptions about the behavior space.

WF $\mathcal{B}_*$ is downward closed under $\preceq$, and $\prec$ is well-founded on $\mathcal{B}_*$.

By downward closure, we mean that any prefix of a behavior in $\mathcal{B}_*$ is also in $\mathcal{B}_*$. An *initial* behavior is a finite behavior with no strict prefix. The set of initial elements, which is non-empty by the well-foundedness assumption, is denoted by $\mathcal{B}_0$. In our concretizations, behaviors are either computations or computation trees, under the standard prefix ordering, so that the **WF** assumption is satisfied. Initial behaviors then correspond to initial states of a process. The semantics of an abstract process P is a subset of $\mathcal{B}$. We call this the *language* of P and denote it by $\mathcal{L}(P)$. The finite behaviors in the language are denoted by $\mathcal{L}_*(P)$ and the infinite behaviors by $\mathcal{L}_\infty(P)$. These subsets must satisfy the following condition.

L1 Every finite prefix of a behavior in $\mathcal{L}(P)$ is a finite behavior of P.

This condition can be expressed succinctly by the notion of the limit of a set of behaviors. For a set S, $lim\,(S) = \{x \mid (\forall y : y \preceq x \wedge y \in \mathcal{B}_* : y \in S)\}$. The condition **L1** is $[\mathcal{L}(P) \;\Rightarrow\; lim\,\mathcal{L}_*(P)]$. The *closure* of a subset S of behaviors, denoted by $cl(S)$, is the set $\{x \mid (\forall y : y \in \mathcal{B}_* \wedge y \preceq x : (\exists z : y \preceq z : z \in S))\}$. I.e., an element x is in $cl(S)$ iff every finite prefix y of x has an "extension" z that is in S. It is not hard to show that cl is monotonic, idempotent, and weakening. We call a set S where $[cl(S) \;\equiv\; S]$ a *safety* property, and a set S where $[cl(S) \;\equiv\; true]$ a *liveness* property in analogy with the definitions of temporal safety and liveness in [5].

Lemma 0 *(cf. [5, 20]) Any set of behaviors, S, can be expressed as the intersection of the safety property $cl(S)$ and the liveness property $(\neg cl(S) \vee S)$.* $\square$

The main process *composition* operator is denoted by $/\!\!/$, and maps a finite subset of $\mathcal{P}$ to $\mathcal{P}$. The *process closure* operator, CL, has signature $CL : \mathcal{P} \to \mathcal{P}$. The process *choice* operator, $+$, also maps a finite subset of $\mathcal{P}$ to $\mathcal{P}$. We say that process P *refines* process Q, written as $P \models Q$, provided that $[\mathcal{L}(P) \;\Rightarrow\; \mathcal{L}(Q)]$. We assume that these operators enjoy the following properties.

P1 *Composition is conjunction of languages*: $[\mathcal{L}(/\!\!/ i : P_i) \;\equiv\; (\wedge\, i : \mathcal{L}(P_i))]$. This implies the corresponding assertions for $\mathcal{L}_*$ and $\mathcal{L}_\infty$.
P2 *Choice is disjunction of languages*: $[\mathcal{L}(+i : P_i) \;\equiv\; (\vee\, i : \mathcal{L}(P_i))]$.
P3 *Closure represents language closure*: $[\mathcal{L}(CL(P)) \;\equiv\; cl(\mathcal{L}(P))]$.

Thus, $/\!\!/$ and $+$ are associative and commutative. To state the circular reasoning method, we also need the concepts of behavior equivalence and non-blocking.

Behavior equivalence: For each process P, we assume the existence of an equivalence relation $\sim_P$ on $\mathcal{B}_*$. This relation is used to state when two behaviors are to be considered equivalent relative to P — for example, in a concrete setting, two computations that agree on values of the external variables of P would be equivalent. We define the closure function, $\langle \sim_P \rangle$, induced by this relation as $\langle \sim_P \rangle(S) = \{x \mid (\exists y : y \in S \wedge x \sim_P y)\}$, for any subset S of behaviors. This must have the property below.

BEQ For any process P, $\mathcal{L}(P)$ is closed under $\sim_P$: $[\langle \sim_P \rangle(\mathcal{L}(P)) \Rightarrow \mathcal{L}(P)]$.

Non-blocking: Process Q *does not block* a process P iff

(a) for every initial behavior of P there is a matching initial behavior of $P /\!\!/ Q$. Formally, $[\mathcal{B}_0 \wedge \mathcal{L}_*(P) \Rightarrow \langle \sim_P \rangle(\mathcal{B}_0 \wedge \mathcal{L}_*(P /\!\!/ Q))]$ and
(b) every extension by P of a finite behavior of $P /\!\!/ Q$ has a matching extension by $P /\!\!/ Q$. Formally, for any x, y, if $x \in \mathcal{L}_*(P)$ and $y \prec x$ and $y \in \mathcal{L}_*(P /\!\!/ Q)$, then $x \in \langle \sim_P \rangle(\mathcal{L}_*(P /\!\!/ Q))$.

Note: This completes the list of postulates. It may also be interesting to know what we do *not* postulate: we do not assume that process languages exhibit either *machine-closure* or *receptivity*, and we do not require the behavior equivalence relations to be a congruence relative to $\preceq$. It is the lack of the latter restriction that lets us concretize the framework to stuttering closed languages.

2.2 Compositional Reasoning

The aim of a compositional reasoning method is to provide a systematic way of breaking down a proof that a composition $(/\!\!/ i : P_i)$ refines a target process T into proof steps that reason individually about each P_i. This is done by abstracting the other processes into an environment for P_i. To uniformly handle such environments, we generalize the original problem into showing that $(/\!\!/ i : P_i)$ refines T when constrained by an environment process, E. The choice composition operator can be handled quite simply: $E /\!\!/ (+i : P_i) \models T$ if, and only if, (from **P2**), for every i, $E /\!\!/ P_i \models T$. We assume that, in the typical case, the composition $(/\!\!/ i : P_i)$ is too large to be handled directly, by model checking methods, but that T and E are small. Consider the two process case: $E /\!\!/ P_1 /\!\!/ P_2 \models T$.

A non-circular reasoning method shows that $E /\!\!/ P_1 /\!\!/ P_2 \models T$ by finding Q such that (a) $E /\!\!/ P_1 \models Q$, and (b) $Q /\!\!/ P_2 \models T$. Soundness follows immediately from **P1** and the transitivity of $\Rightarrow$. This method is most useful when the interaction between P_1 and P_2 is in one direction only. For bi-directional interaction, it is often convenient to reason in a mutually inductive manner, as in the following (syntactically) circular reasoning method, where Q_1 and Q_2 supply the mutually inductive hypotheses. For $i \in \{1, 2\}$, we write $\hat{\imath}$ for the other index.

Circular Reasoning I: To show that $E /\!\!/ P_1 /\!\!/ P_2 \models T$, find Q_1, Q_2 such that (a) for some i, Q_i does not block $P_{\hat{\imath}}$, and $[\sim_{P_i} \Rightarrow \sim_{Q_i}]$, (b) $P_1 /\!\!/ Q_2 \models Q_1$, (c) $Q_1 /\!\!/ P_2 \models Q_2$, and (d) $E /\!\!/ Q_1 /\!\!/ Q_2 \models T$.

This method is easily shown to be complete for processes P_1 and P_2 such that one does not block the other: given that $E /\!\!/ P_1 /\!\!/ P_2 \models T$, choose P_1 for Q_1

```
P1::   var x: (0,1);  trans. x' = y
P2::   var y: (0,1);  trans. y' = x
T::    var x,y:(0,1);  fair inf. often (x=1) and inf. often (y=1)
Q1::   var x:(0,1);  fair inf. often (x=1)
Q2::   var y:(0,1);  fair inf. often (y=1)
```

Fig. 1. Unsoundness for Liveness

and P_2 for Q_2. It is sound for finite behaviors — the non-blocking hypothesis (a) enables a mutual induction on finite behaviors using (b) and (c).

However, this method is *not sound in general* — it is unsound when both Q_1 and Q_2 define liveness properties. Since it is difficult to exhibit the unsoundness in the abstract, consider the concrete setting of Figure 1, where $/\!\!/$ is interpreted as synchronous composition, and the behavior of a process is the set of its computations. Each process is specified by its initial condition, a transition relation (a primed variable refers to its next value), and a fairness constraint on infinite computations (unspecified components are *true*, i.e., unconstrained).

The composition $P_1 /\!\!/ P_2$ *does not* refine T, since it has a computation where x and y are both always 0. However, consider the processes Q_1 and Q_2 in Figure 1. As their initial and transition conditions are unconstrained, they are non-blocking, satisfying hyp. (a). The fairness condition of Q_2 requires that $y = 1$ is true infinitely often in any computation of $P_1 /\!\!/ Q_2$, so the update $x' = y$ in P_1 forces $x = 1$ to be true infinitely often, ensuring that hyp. (b) holds. Hyp. (c) holds for a similar reason. Hyp. (d) holds since Q_1 and Q_2 together satisfy T. Thus, the method leads to the *unsound* conclusion that $P_1 /\!\!/ P_2$ does refine T.

2.3 A sound and complete method

The previous example shows that a method based on mutual induction is not always sound for liveness properties; however, it is hard to ascribe a reason for the failure. One possible explanation is that the problem arises from the lack of an inductive structure: in the example, both Q_1 and Q_2 restrict only the "future" of a computation. This observation has led to several methods (e.g., [2, 21, 23]) where a temporal next-time operator supplies the necessary well-founded structure. Here, we adopt a different strategy, and augment the previous method with an additional check to ensure that the reasoning is sound.

Circular Reasoning II: To show that $E /\!\!/ P_1 /\!\!/ P_2 \models T$, find Q_1, Q_2 such that the conditions (a)-(d) from **Circular Reasoning I** hold, and additionally,
(e) for some i, $E /\!\!/ P_i /\!\!/ CL(T) \models (T + Q_1 + Q_2)$.

To gain some intuition for the new hypotheses (e), consider the earlier soundness failure. The reasoning fails because the error computation π, where x and y are both always 0, is *ignored* by hypotheses (b) and (c), since neither Q_1 nor Q_2

contain π. The "missing" computations are those, such as π, that belong to $\mathcal{L}(P_1 \mathbin{/\!\!/} P_2)$, but not to either of $\mathcal{L}(Q_1)$ or $\mathcal{L}(Q_2)$. We want these computations to also be part of $\mathcal{L}(T)$. As is shown by the soundness proof below, hypotheses (a)-(d) ensure that all computations of $P_1 \mathbin{/\!\!/} P_2$ belong to the safety part of the language of T. Thus, the missing computations must belong to the liveness part of T. A direct statement of this condition is: $[\mathcal{L}(E) \wedge \mathcal{L}(P_1 \mathbin{/\!\!/} P_2) \wedge \neg(\mathcal{L}(Q_1) \vee \mathcal{L}(Q_2)) \Rightarrow (\neg cl(\mathcal{L}(T)) \vee \mathcal{L}(T))]$. However, this includes $P_1 \mathbin{/\!\!/} P_2$, which is just what we are trying to avoid reasoning about directly.

We show in the soundness proof that one can replace $P_1 \mathbin{/\!\!/} P_2$ with a *single* process, P_i, resulting in a stronger condition, but *without sacrificing completeness*. Rearranging the new condition, we get that, for some i, $[\mathcal{L}(E) \wedge \mathcal{L}(P_i) \wedge cl(\mathcal{L}(T)) \Rightarrow (\mathcal{L}(Q_1) \vee \mathcal{L}(Q_2) \vee \mathcal{L}(T))]$. Put in terms of the process operators using **P1-P3**, this is just condition (e). For our example, both $\mathcal{L}(E)$ and $cl(\mathcal{L}(T))$ are *true* (i.e., the set of all computations), and condition (e) does not hold — as expected — for either P_1 or P_2, because π belongs to both processes, but not to either of T, Q_1, or Q_2. We show completeness first, which is the simpler half of the proof.

Theorem 0 ***(Completeness)*** *For processes P_1, P_2 such that one of the processes does not block the other, and for any process T, if $E \mathbin{/\!\!/} P_1 \mathbin{/\!\!/} P_2 \models T$, then this can be shown using the* ***Circular Reasoning II*** *method.*
Proof. Choose $Q_1 = P_1$ and $Q_2 = P_2$. Condition (a) holds by the non-blocking assumption. Condition (b) becomes $P_1 \mathbin{/\!\!/} P_2 \models P_1$ which, by the definition of $\models$ and assumption **P1**, is equivalent to $[\mathcal{L}(P_1) \wedge \mathcal{L}(P_2) \Rightarrow \mathcal{L}(P_1)]$, which is trivially true. Conditions (c) and (d) can be dealt with in a similar manner. Condition (e), for $i = 1$, simplifies, using **P1-P3**, to $[\mathcal{L}(E) \wedge \mathcal{L}(P_1) \wedge cl(\mathcal{L}(T)) \Rightarrow \mathcal{L}(T) \vee \mathcal{L}(P_1) \vee \mathcal{L}(P_2)]$, which is true trivially. $\square$

Note: The completeness proof, as may be expected, considers the worst case, where $Q_1 = P_1$ and $Q_2 = P_2$. This is unavoidable in general, because it is a manifestation of the state explosion problem. However, the language of a process is usually given by its external input-output behavior, which can be much less complex than its full behavior. Thus, Q_i, which represents external behavior, can be smaller than P_i, and yet be an adequate replacement for it in the method.

Soundness requires a more complex proof. First, we show, using well-founded induction on $\prec$, and hypotheses (a)-(d), that the finite language of $P_1 \mathbin{/\!\!/} P_2$ is contained in the finite language of $Q_1 \mathbin{/\!\!/} Q_2$. This is used to prove that any behavior of $E \mathbin{/\!\!/} P_1 \mathbin{/\!\!/} P_2$ is a safe behavior of T. We then utilize hypothesis (e) to conclude that all behaviors of $E \mathbin{/\!\!/} P_1 \mathbin{/\!\!/} P_2$ are behaviors of T.

Lemma 1 *For processes P_1, P_2, Q_1, Q_2, and T satisfying the conditions of the* ***Circular Reasoning II*** *method, $[\mathcal{L}_*(P_1 \mathbin{/\!\!/} P_2) \Rightarrow \mathcal{L}_*(Q_1 \mathbin{/\!\!/} Q_2)]$.*
Proof. This proof is based on well-founded induction on the set of finite behaviors. Both the base and inductive cases make use of the non-blocking hypothesis (a), and the mutual induction supplied by hypotheses (b) and (c) on the processes Q_1 and Q_2. Without loss of generality, we assume that hyp. (a) holds for the pair (P_1, Q_2); the proof is symmetric in the other case.

(Base case: initial elements) For any initial behavior x, x is in $\mathcal{L}_*(P_1 \mathbin{/\!\!/} P_2)$ iff (by **P1**), it belongs to both $\mathcal{L}_*(P_1)$ and $\mathcal{L}_*(P_2)$. As Q_2 does not block P_1, x is in $\langle \sim_{P_1} \rangle(\mathcal{B}_0 \wedge \mathcal{L}_*(P_1 \mathbin{/\!\!/} Q_2))$. From Hyp. (b), x belongs to $\langle \sim_{P_1} \rangle(\mathcal{B}_0 \wedge \mathcal{L}_*(Q_1))$. As $[\sim_{P_1} \Rightarrow \sim_{Q_1}]$ from Hyp. (a), using **BEQ** and the monotonicity of $\langle \sim_P \rangle$, x belongs to $\mathcal{L}_*(Q_1)$. Since x belongs to both $\mathcal{L}_*(P_2)$ and $\mathcal{L}_*(Q_1)$, using **P1** and Hyp. (c), it belongs to $\mathcal{L}_*(Q_2)$, and thus, (by **P1**) to $\mathcal{L}_*(Q_1 \mathbin{/\!\!/} Q_2)$.

(Inductive case) Consider any non-initial behavior x in $\mathcal{L}_*(P_1 \mathbin{/\!\!/} P_2)$. Let y be a strict prefix of x. By **WF**, y is a finite behavior, so, by **L1**, it belongs to $\mathcal{L}_*(P_1 \mathbin{/\!\!/} P_2)$. From the inductive hypothesis, y belongs to $\mathcal{L}_*(Q_1 \mathbin{/\!\!/} Q_2)$. It follows from **P1**, that y belongs to $\mathcal{L}_*(P_1 \mathbin{/\!\!/} Q_2)$. As Q_2 does not block P_1 (Hyp. (a)), and x belongs to $\mathcal{L}_*(P_1)$, it follows that x belongs to $\langle \sim_{P_1} \rangle(\mathcal{L}_*(P_1 \mathbin{/\!\!/} Q_2))$. Reasoning now as in the earlier case, it follows that x belongs to $\mathcal{L}_*(Q_1 \mathbin{/\!\!/} Q_2)$. □

Lemma 2 *For any process P, $[cl(\mathcal{L}(P)) \equiv \mathit{lim}\, \mathcal{L}_*(P)]$.*

Proof. For any x, $x \in cl(\mathcal{L}(P))$ iff (by definition of closure) $(\forall y : y \in \mathcal{B}_* \wedge y \preceq x : (\exists z : y \preceq z \wedge z \in \mathcal{L}(P)))$. By condition **L1**, as $[\mathcal{L}_*(P) \Rightarrow \mathcal{L}(P)]$, this is equivalent to $(\forall y : y \in \mathcal{B}_* \wedge y \preceq x : y \in \mathcal{L}_*(P))$ — i.e., x is in $\mathit{lim}\, \mathcal{L}_*(P)$. □

Theorem 1 ***(Soundness)*** *For processes E, P_1, P_2, Q_1, Q_2, and T satisfying the conditions of **Circular Reasoning II**, $[\mathcal{L}(E \mathbin{/\!\!/} P_1 \mathbin{/\!\!/} P_2) \Rightarrow \mathcal{L}(T)]$.*

Proof. The decomposition of $\mathcal{L}(T)$ into safety and liveness components gives us a natural decomposition of this proof into safety and liveness proofs. The safety proof shows that $[\mathcal{L}(E \mathbin{/\!\!/} P_1 \mathbin{/\!\!/} P_2) \Rightarrow cl(\mathcal{L}(T))]$, while the liveness proof shows that $[\mathcal{L}(E \mathbin{/\!\!/} P_1 \mathbin{/\!\!/} P_2) \Rightarrow (\neg cl(\mathcal{L}(T)) \vee \mathcal{L}(T))]$.

(Safety)

$$
\begin{array}{lll}
 & \mathcal{L}(E \mathbin{/\!\!/} P_1 \mathbin{/\!\!/} P_2) & \\
\Rightarrow & cl(\mathcal{L}(E \mathbin{/\!\!/} P_1 \mathbin{/\!\!/} P_2)) & (\ cl \text{ is weakening }) \\
\equiv & \mathit{lim}\, \mathcal{L}_*(E \mathbin{/\!\!/} P_1 \mathbin{/\!\!/} P_2) & (\text{ Lemma 2 }) \\
\Rightarrow & \mathit{lim}\, \mathcal{L}_*(E \mathbin{/\!\!/} Q_1 \mathbin{/\!\!/} Q_2) & (\text{ Lemma 1; monotonicity of } \mathit{lim}\ ) \\
\equiv & cl(\mathcal{L}(E \mathbin{/\!\!/} Q_1 \mathbin{/\!\!/} Q_2)) & (\text{ Lemma 2 }) \\
\Rightarrow & cl(\mathcal{L}(T)) & (\text{ by hyp. (d); monotonicity of } cl\ )
\end{array}
$$

(Liveness)

$$
\begin{array}{lll}
 & \mathcal{L}(E \mathbin{/\!\!/} P_1 \mathbin{/\!\!/} P_2) \wedge cl(\mathcal{L}(T)) & \\
\equiv & \mathcal{L}(E \mathbin{/\!\!/} P_1 \mathbin{/\!\!/} P_2) \wedge \mathcal{L}(CL(T)) & (\text{ by } \mathbf{P3}\ ) \\
\equiv & \mathcal{L}(E) \wedge \mathcal{L}(P_1) \wedge \mathcal{L}(P_2) \wedge \mathcal{L}(CL(T)) & (\text{ by } \mathbf{P1}\ ) \\
\Rightarrow & \mathcal{L}(E) \wedge \mathcal{L}(P_1) \wedge \mathcal{L}(P_2) \wedge \mathcal{L}(T + Q_1 + Q_2) & \\
 & & (\text{ by hyp. (e) (pick } i)\ ) \\
\equiv & \mathcal{L}(E) \wedge \mathcal{L}(P_1) \wedge \mathcal{L}(P_2) \wedge (\mathcal{L}(T) \vee \mathcal{L}(Q_1) \vee \mathcal{L}(Q_2)) & \\
 & & (\text{ by } \mathbf{P2}\ ) \\
\Rightarrow & \mathcal{L}(T) \vee (\mathcal{L}(E) \wedge \mathcal{L}(P_2) \wedge \mathcal{L}(Q_1)) \vee (\mathcal{L}(E) \wedge \mathcal{L}(P_1) \wedge \mathcal{L}(Q_2)) & \\
 & & (\ \wedge \text{ over } \vee\text{; dropping conjuncts }) \\
\Rightarrow & \mathcal{L}(T) \vee (\mathcal{L}(E) \wedge \mathcal{L}(Q_1) \wedge \mathcal{L}(Q_2)) & (\text{ by } \mathbf{P1}\text{; hyp. (b) and (c) }) \\
\Rightarrow & \mathcal{L}(T) & (\text{ by hyp. (d) })
\end{array}
$$

□

Note: One can replace hyp. (e) with (e'): for some i, $E \,/\!\!/\, P_i \,/\!\!/\, CL(Q_1) \,/\!\!/\, CL(Q_2) \models (T + Q_1 + Q_2)$, without losing either soundness or completeness. Hyp. (e') is weaker (by hyp. (d)) than (e), so it is more likely to hold. However, it might also be more difficult to check in practice as $CL(Q_1) \,/\!\!/\, CL(Q_2)$ could have size larger than $CL(T)$. Hyp. (e') holds if either $\mathcal{L}(Q_1)$ or $\mathcal{L}(Q_2)$ are safety languages, showing that the first circular reasoning rule is sound in this case.

The new hypothesis also provides a direct link to the reasoning method proposed by Alur and Henzinger in [6]. In our notation, it reads as follows: to show that $P_1 \,/\!\!/\, P_2 \models Q_1 \,/\!\!/\, Q_2$, prove that (i) $P_1 \,/\!\!/\, Q_2 \models Q_1$, and (ii) $P_2 \,/\!\!/\, CL(Q_1) \models Q_2$ (non-blocking is assured by the language definition). This is incomplete as stated, but it can be generalized to showing that $P_1 \,/\!\!/\, P_2 \models T$ by making a *choice* of Q_1, Q_2 such that (i) and (ii) hold, as well as (iii) $Q_1 \,/\!\!/\, Q_2 \models T$. This is sound, by arguments in [6], as well as complete (choose P_i for Q_i).

Both methods are complete, but we may go further and compare their solution sets for $\{Q_i\}$ (i.e., those that satisfy the hypotheses), for fixed $\{P_i\}$ and T. In one direction, if (i)-(iii) hold for $\{Q_i\}$, so do the hypotheses (a)-(e') of **Circular Reasoning II**, for the same choice. Hyp. (a) is the non-blocking property. Hyp. (b),(d) are identical to (i),(iii), respectively. Hyp. (c) holds using (ii) and the weakening property of cl. Finally, Hyp. (e') holds for P_2 using (ii). The converse "almost" holds (assuming non-blocking): from (e) (for P_2) and (c), one can derive that $P_2 \,/\!\!/\, CL(Q_1) \models Q_2 + T$, which is weaker than hyp. (ii). It turns out, in fact, that there are specific $\{P_i\}, \{Q_i\}$, and T for which the hypotheses of our method hold, but those of the generalized [6] method do not. This implies that our method is more widely applicable. Indeed, we had noticed in earlier work [7, 8] that it was not possible to use the generalized Alur-Henzinger rule for Q_i with liveness constraints that were *automatically* generated from property T. In [7], we showed that conditions (a), (d), and (e) *always* hold for the generated Q_i's, thus leaving one to (safely) apply the mutually inductive checks (b) and (c).

3 Concrete Compositional Reasoning

In this section, we show how the abstract framework can be instantiated for a variety of composition operators and language definitions in shared-variable concurrency. For each choice, we show that assumptions **WF**,**L1**,**P1-P3**, and **BEQ** are met. For lack of space, detailed proofs are left to the full version.

We assume available a set of variable names, with associated domains of values. For a variable w, the primed variable w' is used to denote the value of w in the next state. A *process*, P, is given by a tuple (z, ι, τ, Φ), where: z is a finite set of *variables*, partitioned into *local* variables (x), and *interface* variables (y), $\iota(z)$ is an *initial* condition, $\tau(z, z')$ is a *transition* relation, relating current and next values of z, and $\Phi(z)$ is a temporal fairness formula defining a *fairness* condition on the states. A *state* of P is a function giving values to its variables. We write states as pairs (a, b), where a is the value of x and b the value of y. A *path*, π, is a finite or infinite sequence of states where for each i such that $0 < i < |\pi|$, $\tau(\pi_{i-1}, \pi_i)$ holds; $|\pi|$ is the length of π (the number of states on it). A state t is *reachable* from state s by a path π of length n iff $\pi_0 = s$ and $\pi_{n-1} = t$.

A process exhibits *finite local branching* (cf. [1]) if (i) for each b, there are finitely many a such that (a, b) is an initial state, and (ii) for any state (a, b) and any b', there are finitely many a' such that (a', b') is reachable from (a, b) by a path where the value of y is b for all non-final states. We assume that all processes satisfy this restriction. We use temporal logic to define the languages of processes. We only need the operators G (always) and $\overset{\infty}{\mathsf{G}}$ (from some point on). Formally, for a sequence π, and a predicate q on states (transitions), $\mathsf{G}(q)$ holds of π iff q holds for every state (transition) on π. For an infinite sequence π, and a predicate q, $\overset{\infty}{\mathsf{G}}(q)$ holds of π if, from some $k \in \mathbf{N}$, the suffix of π starting at point k satisfies $\mathsf{G}(q)$. The path operator A quantifies over all paths from a state.

3.1 Linear-time Interleaving Semantics

In the interleaving semantics, the *language* of a process, P_i, is defined relative to an *external context* for P_i. The context is represented by a set of variables, w_i, disjoint from z_i (we write z_i instead of z_{P_i} for clarity). The language is a subset of the set of sequences, both finite and infinite, of valuations to $z_i \cup w_i$. We denote the language by the expression $\mathcal{L}^{\Box}(P_i)(z_i; w_i)$, where ";" is used to separate the process' variables from its context. The set of finite computations ($\mathcal{L}^{\Box}_*(P_i)(z_i; w_i)$) is defined by the temporal logic formula below, where, for a set X of variables, $\mathit{unch}(X) \equiv (\forall x : x \in X : x' = x)$.

$$\iota_i(z_i) \wedge \mathsf{G}((\tau_i(z_i, z_i') \wedge \mathit{unch}(w_i)) \vee \mathit{unch}(x_i))$$

This formula ensures that the first state is an initial one, every transition is either that of P_i and leaves the context w_i unchanged, or is an "environment" transition that leaves the local variables of P_i unchanged. The set of infinite sequences in the language, denoted by $\mathcal{L}^{\Box}_\infty(P_i)(z_i; w_i)$, is defined by the same formula (interpreted over infinite sequences), together with the constraint $\Phi_i(z_i)$ which ensures that the fairness condition of P_i holds. The full language, $\mathcal{L}^{\Box}(P_i)(z_i; w_i)$, is given by $\mathcal{L}^{\Box}_*(P_i)(z_i; w_i) \cup \mathcal{L}^{\Box}_\infty(P_i)(z_i; w_i)$. The *external language* of process P_i for context w_i is denoted by $\mathcal{L}^{\Box}_{ext}(P_i)(y_i; w_i)$. It is defined as the projection of the language of P_i on its interface and context variables: i.e., $\mathcal{L}^{\Box}_{ext}(P_i)(y_i; w_i) \equiv (\exists x_i : \mathcal{L}^{\Box}(P_i)(z_i; w_i))$. Here, existential quantification refers to the choice of a *sequence* of values for x_i. Two computations are *behavior equivalent* relative to P_i iff their projections on $y_i \cup w_i$ are identical. The ordering $\preceq$ is defined as the prefix ordering on finite sequences. This choice satisfies **WF**. From the language definitions, every finite prefix of a sequence in $\mathcal{L}^{\Box}_\infty(P)$ satisfies the initial and transition conditions, and is thus in $\mathcal{L}^{\Box}_*(P)$, so that **L1** holds for $\mathcal{L}^{\Box}_{ext}(P)$. As $\mathcal{L}^{\Box}_{ext}(P)$ is defined over the non-local variables of P, it satisfies **BEQ**.

Interleaving Composition For a set of processes $\{P_i\}$, their *interleaving composition* Q is denoted as $([]i : P_i)$. It is defined provided that the local variables of each process are disjoint from the variables of other processes; i.e., $x_i \cap z_j = \emptyset$ for $i \neq j$. The process Q has local variables $(\cup i : x_i)$, interface variables $(\cup i : y_i)$, initial condition $(\wedge i : \iota_i(z_i))$, and fairness condition $(\wedge i : \Phi_i(z_i))$.

For any i, let X_i be the set of local variables of the other processes; formally, $X_i = x_Q \setminus x_i$. Let Y_i be the set of interface variables of other processes that are *not shared* with the interface variables of process P_i; formally, $Y_i = y_Q \setminus y_i$. Let $Z_i = X_i \cup Y_i$. The transition relation of Q is defined to be $(\vee i : \tau_i(z_i, z_i') \wedge \mathit{unch}(Z_i))$. This definition implies that a transition of process P_i, leaves unchanged the values of all variables that are not shared with P_i.

It is important to note that we do not distinguish between the usage of shared variables, such as read-only or write-only variables. All shared variables can be both read and written to by the sharing processes. Since this is the most general case, our results apply to more specific situations as well.

Non-blocking: The condition $[\sim_{P_i} \Rightarrow \sim_{Q_i}]$ in hyp. (a) of the **Circular Reasoning II** method is equivalent to saying that y_{Q_i} is a subset of y_{P_i}. Furthermore, only the first part of the non-blocking definition (for initial states) is needed, because, for any x, y: if x is a finite computation in $\mathcal{L}^{[]}_{ext}(P)$, $y \prec x$ and $y \in \mathcal{L}^{[]}_{ext}(P \,[]\, Q)$ then y can be extended to a computation z in $\mathcal{L}^{[]}_{ext}(P \,[]\, Q)$ such that z and x agree on the external variables of P by following the transitions in x while keeping the local variables of Q unchanged.

Theorem 2 ***(P1** for $\mathcal{L}^{[]}$)* *Let $\{P_i\}$ be a set of processes such that $Q = ([]i : P_i)$ is defined, and let w be a context for Q. Then, $[\mathcal{L}^{[]}(Q)(z_Q; w) \equiv (\wedge i : \mathcal{L}^{[]}(P_i)(z_i; w \cup Z_i))]$.* $\square$

The previous theorem shows that asynchronous composition corresponds to conjunction of languages. We are usually interested only in the externally visible behavior of a process. Abadi and Lamport showed in [2] that a similar theorem applies to the external languages, under restrictions which ensure that the external language is closed under stuttering (i.e., finite repetition of states). However, there are applications such as the analysis of timing diagrams [8], where one wants to count the number of events in an asynchronous system, and this is not a stuttering-closed language. We therefore show the analogous theorem under a different non-interference assumption on composition.

The *mutex* assumption: A set of processes $\{P_i\}$ satisfies this assumption if in any jointly enabled transition, at most one process changes its local state. This may be realized in practice by using a turn-based interleaving scheduler, possibly implemented by the processes themselves, and guarding each transition with a check that it is the process' turn.

Theorem 3 ***(P1** for $\mathcal{L}^{[]}_{ext}$)* *Let $\{P_i\}$ be a set of processes such that $Q = ([]i : P_i)$ is defined, and let w be a context for Q. Under the mutex assumption for $\{P_i\}$, $[\mathcal{L}^{[]}_{ext}(Q)(y_Q; w) \equiv (\wedge i : \mathcal{L}^{[]}_{ext}(P_i)(y_i; w \cup Y_i))]$.*$\square$

Stuttering Equivalence: Stuttering refers to finite repetition of identical values. Two sequences are stuttering equivalent if they are identical up to such finite repetition. The language of a process is closed relative to stuttering; however, its external language is not, as existential quantification does not preserve closure in general. The stuttering closed external language of P_i for context w is denoted by $\tilde{\mathcal{L}}^{[\,]}_{ext}(P_i)(y_i; w)$, and is defined as $(\tilde{\exists} x_i : \mathcal{L}^{[\,]}(P_i)(z_i; w))$. Here, $\tilde{\exists}$ is a stuttering closed version of $\exists$, defined by: for a sequence π over a set of variables W, $\pi \in (\tilde{\exists} X : S(X, W))$ iff there exists a sequence ξ defined over $X \cup W$ such that π and ξ are stuttering equivalent on X, and $\xi \in S$.

The *sequencing* assumption: This is a semantic formulation of the syntactic constraints considered by Abadi and Lamport in [2]. It holds if, for every jointly enabled transition t of $Q = ([\,] i : P_i)$, there is a finite sequence σ that satisfies the transition condition of Q on all transitions, and σ and t are stuttering equivalent relative to the external variables $y_Q \cup w$.

Theorem 4 ***(P1** for $\tilde{\mathcal{L}}^{[\,]}_{ext}$) (cf. [2]) Let $\{P_i\}$ be a set of processes such that $Q = ([\,] i : P_i)$ is defined, and let w be a context for Q. Under the sequencing assumption, and if Φ_Q is stuttering-closed relative to $y_Q \cup w$,*
$[\tilde{\mathcal{L}}^{[\,]}_{ext}(Q)(y_Q; w) \equiv (\wedge i : \tilde{\mathcal{L}}^{[\,]}_{ext}(P_i)(y_i; w \cup Y_i))]$. $\square$

Process Choice For a set of processes $\{P_i\}$, the process $C = (+i : P_i)$ is defined whenever $([\,] i : P_i)$ is defined, with local variables $(\cup i : x_i) \cup \{c\}$, where c is a fresh variable not in $(\cup i : x_i)$, interface variables $(\cup i : y_i)$, initial condition $(\vee i : c = i \wedge \iota_i(z_i))$ transition relation $(c' = c) \wedge (\wedge i : c = i \Rightarrow \tau_i(z_i, z_i'))$, and fairness condition $(\vee i : \overset{\infty}{\mathsf{G}}(c = i) \wedge \Phi_i(z_i))$. The variable c records the initial (fixed) choice of $\{P_i\}$. The following theorem is an easy consequence.

Theorem 5 ***(P2** for $\mathcal{L}^{[\,]}_{ext}$ and $\tilde{\mathcal{L}}^{[\,]}_{ext}$) For a set of processes $\{P_i\}$ such that $C = (+i : P_i)$ is defined, and a context w for C, (i) $[\mathcal{L}^{[\,]}_{ext}(C)(y_C; w) \equiv (\vee i : \mathcal{L}^{[\,]}_{ext}(P_i)(y_i; w))]$, and (ii) $[\tilde{\mathcal{L}}^{[\,]}_{ext}(C)(y_C; w) \equiv (\vee i : \tilde{\mathcal{L}}^{[\,]}_{ext}(P_i)(y_i; w))]$.* $\square$

Process Closure For a process Q, let $CL(Q)$ be the process $(z_Q, \iota_Q, \tau_Q, \mathit{true})$. I.e., $CL(Q)$ has the same transition structure as Q, but has a trivial fairness constraint. The proof of the following theorem relies on the finite local branching requirement and König's Lemma.

Theorem 6 ***(P3** for $\mathcal{L}^{[\,]}_{ext}$, and for $\tilde{\mathcal{L}}^{[\,]}_{ext}$ (cf. [1])) For any process Q and context w for Q, $[\mathcal{L}^{[\,]}_{ext}(CL(Q))(y_Q; w) \equiv cl(\mathcal{L}^{[\,]}_{ext}(Q)(y_Q; w))]$, and*
$[\tilde{\mathcal{L}}^{[\,]}_{ext}(CL(Q))(y_Q; w) \equiv cl(\tilde{\mathcal{L}}^{[\,]}_{ext}(Q)(y_Q; w))]$. $\square$

3.2 Synchronous Semantics

In the synchronous semantics, all processes in a composition make a transition simultaneously. Synchronous semantics are appropriate for modeling hardware systems, where several components are controlled by a single clock. The *language* of a process, P_i, is a subset of the set of sequences, both finite and infinite, of valuations to z_i. The finite part of the language, denoted by $\mathcal{L}^{\|}_{*}(P_i)(z_i)$, is given by the temporal formula $\iota_i(z_i) \wedge \mathsf{G}(\tau_i(z_i, z_i'))$. The infinite part, denoted by $\mathcal{L}^{\|}_{\infty}(P_i)(z_i)$ is given by sequences satisfying the same formula, but with the additional fairness constraint, $\Phi_i(z_i)$. The full language, denoted by $\mathcal{L}^{\|}(P_i)(z_i)$ is $\mathcal{L}^{\|}_{*}(P_i)(z_i) \cup \mathcal{L}^{\|}_{\infty}(P_i)(z_i)$. The external language of the process, denoted by $\mathcal{L}^{\|}_{ext}(P_i)(y_i)$, is defined, as before, by $(\exists x_i : \mathcal{L}^{\|}(P_i)(z_i))$.

Synchronous Composition For a set of processes $\{P_i\}$, their *synchronous composition* Q is denoted as $(\| \ i : P_i)$. The components of Q are defined as for interleaving composition, except the transition relation, which is defined as $(\wedge i : \tau_i(z_i, z_i'))$, since transitions are simultaneous. Process choice and closure are defined exactly as in the asynchronous case. Similarly, the behavior equivalence relationship from Hyp. (a) of the method is given by $y_{Q_i} \subseteq y_{P_i}$. Completely specified Moore machines are non-blocking by definition. Mealy machines can be made non-blocking by syntactically preventing combinational cycles between the input-output variables of the composed processes (e.g., as in Reactive Modules [6]). In [7], it is shown that these definitions enjoy the properties **P1-P3** for $\mathcal{L}^{\|}_{ext}$; thus, the **Circular Reasoning II** method is applicable.

3.3 Refinement through Fair Simulation

In this part of the paper, we interpret the relation $\models$ as refinement through a simulation relation that includes the effect of fairness constraints cf. [15]. Composition is synchronous, with the operators defined as in the previous section. The difference is that the language of a process is now a set of trees instead of sequences. We assume that each process has the finite branching property.

Tree Languages: A *tree* is a prefix-closed subset of $\mathbf{N}^*$. We consider only finite branching trees. Finite behaviors are finite depth trees. The $\preceq$ ordering on trees is subtree inclusion. The finite branching condition ensures that this ordering is well-founded on finite behaviors, since each finite-depth tree has finitely many nodes. A *labeled tree*, u, is a triple (U, f, Σ), where U is a tree and $f : U \to \Sigma$.

A *computation tree* of a process P is obtained, informally, by unrolling its transition relation starting from an initial state. A *computation tree fragment* (CTF, for short) of process P is obtained from a computation tree of P by either dropping or duplicating some subtrees of the computation tree, while satisfying its fairness constraint along all infinite branches. Formally, a CTF is a labeled tree $u = (U, f, \Sigma)$ where: Σ is the state space of P, $f(\lambda)$ is an initial state of P, $(f(x), f(x.i))$ is a transition of P for each $x \in \mathbf{N}^*$ and $i \in \mathbf{N}$ such that $x, x.i \in U$, and the fairness condition of P holds along all infinite branches of U. Tree fragments represent the result of composing P with arbitrary, nondeterministic environments. The set of CTF's of P forms its (tree) *language*, denoted by $\mathcal{L}^T(P)$. The finite language of P, $\mathcal{L}^T_*(P)$, is therefore, the set of finite-depth trees satisfying the CTL-like formula $\iota_P(z_P) \wedge \mathsf{AG}(\tau_P(z_P, z'_P))$. The infinite language of P, $\mathcal{L}^T_\infty(P)$, is the set of infinite trees satisfying, additionally, $\mathsf{A}(\Phi_P(z_P))$. Let $\mathcal{L}^T(P) = \mathcal{L}^T_*(P) \cup \mathcal{L}^T_\infty(P)$. The external language of M, denoted by $\mathcal{L}^T_{ext}(P)$, is obtained, as before, by projecting out the local variable component of each node label: formally, this is represented as $(\exists x_P : \mathcal{L}^T(P))$.

In [14], it is shown (with different notation) that for non-fair processes P and Q, Q simulates P (in the standard sense [22]) iff $[\mathcal{L}^T_{ext}(P) \;\Rightarrow\; \mathcal{L}^T_{ext}(Q)]$. Based on this correspondence, they propose using language inclusion as the definition of simulation under fairness. We show that this choice satisfies conditions **P1-P3**. Conditions **P1** and **P2** follow quite directly from the CTL formulation of process language. From the prefix ordering, a tree is in the closure of a set of trees S iff every finite depth subtree can be extended to a tree in S. This is called *finite closure* in [20], where it is used to define "universally safe" branching properties. The proof of **P3** relies on the finite-branching property, and König's lemma.

Theorem 7 ***(P1,P2,P3** *for* $\mathcal{L}^T_{ext}$*)* *If* $Q = (\| \, i : P_i)$ *is defined, then* *(i)* $[\mathcal{L}^T_{ext}(Q) \;\equiv\; (\wedge i : \mathcal{L}^T_{ext}(P_i))]$, *(ii)* $[\mathcal{L}^T_{ext}((+i : P_i)) \;\equiv\; (\vee i : \mathcal{L}^T_{ext}(P_i))]$, *and* *(iii) for any* P, $[cl(\mathcal{L}^T_{ext}(P)) \;\equiv\; \mathcal{L}^T_{ext}(CL(P))]$.

4 Conclusions

This paper develops a general framework for designing sound and complete methods for mutually inductive, assume-guarantee reasoning. The key challenge is in balancing, on one hand, the restrictions needed to avoid unsound circular reasoning with liveness properties and, on the other, the generality necessary to ensure completeness. Furthermore, we show how to instantiate this framework for linear-time interleaving composition, and extend it to stuttering closed external languages. We then outline the instantiation for synchronous composition in both linear and branching time semantics. We believe that the resulting rules can be adapted without much difficulty to specific modeling languages.

References

1. M. Abadi and L. Lamport. The existence of refinement mappings. In *LICS*, 1988.
2. M. Abadi and L. Lamport. Conjoining specifications. *ACM Trans. on Programming Languages and Systems (TOPLAS)*, May 1995.
3. M. Abadi and S. Merz. An abstract account of composition. In *MFCS*, volume 969 of *LNCS*. Springer Verlag, 1995.

4. M. Abadi and G. Plotkin. A logical view of composition and refinement. In *POPL*, 1991.
5. B. Alpern and F. Schneider. Defining liveness. *Information Processing Letters*, 21(4), 1985.
6. R. Alur and T. Henzinger. Reactive modules. In *LICS*, 1996.
7. N. Amla, E.A. Emerson, K.S. Namjoshi, and R.J. Trefler. Assume-guarantee based compositional reasoning for synchronous timing diagrams. In *TACAS*, volume 2031 of *LNCS*, 2001.
8. N. Amla, E.A. Emerson, K.S. Namjoshi, and R.J. Trefler. Visual specifications for modular reasoning about asynchronous systems. In *FORTE*, volume 2529 of *LNCS*, 2002.
9. K.M. Chandy and J. Misra. Proofs of networks of processes. *IEEE Transactions on Software Engineering*, 7(4), 1981.
10. E.M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Workshop on Logics of Programs*, volume 131 of *LNCS*, 1981.
11. W-P. de Roever, F. de Boer, U. Hannemann, J. Hooman, Y. Lakhnech, M. Poel, and J. Zwiers. *Concurrency Verification: Introduction to Compositional and Non-compositional Proof Methods.* Cambridge University Press, 2001.
12. W-P. de Roever, H. Langmaack, and A. Pnueli, editors. *Compositionality: The Significant Difference*, volume 1536 of *LNCS*. Springer-Verlag, 1997.
13. E.W. Dijkstra and C.S. Scholten. *Predicate Calculus and Program Semantics.* Springer Verlag, 1990.
14. T.A. Henzinger, O. Kupferman, and S. Rajamani. Fair simulation. In *CONCUR*, volume 1243 of *LNCS*, 1997.
15. T.A. Henzinger, S. Qadeer, S.K. Rajamani, and S. Tasiran. An assume-guarantee rule for checking simulation. *ACM Trans. on Programming Languages and Systems (TOPLAS)*, January 2002.
16. C.B. Jones. *Development methods for computer programs including a notion of interference*. PhD thesis, Oxford University, 1981.
17. R.P. Kurshan. *Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach.* Princeton University Press, 1994.
18. P. Maier. A set-theoretic framework for assume-guarantee reasoning. In *ICALP*, volume 2076 of *LNCS*, 2001.
19. P. Maier. Compositional circular assume-guarantee rules cannot be sound and complete. In *FoSSaCS*, volume 2620 of *LNCS*, 2003.
20. P. Manolios and R. J. Trefler. Safety and liveness in branching time. In *LICS*, 2001.
21. K.L. McMillan. Circular compositional reasoning about liveness. In *CHARME*, volume 1703 of *LNCS*, 1999.
22. R. Milner. An algebraic definition of simulation between programs. In *2nd IJCAI*, 1971.
23. K.S. Namjoshi and R.J. Trefler. On the completeness of compositional reasoning. In *CAV*, volume 1855 of *LNCS*, 2000.
24. P. Pandya and M. Joseph. P-A logic - a compositional proof system for distributed programs. *Distributed Computing*, 1991.
25. J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proc. of the 5th Intl. Symp. on Programming*, volume 137 of *LNCS*, 1982.
26. M. Viswanathan and R. Viswanathan. Foundations for circular compositional reasoning. In *ICALP*, volume 2076 of *LNCS*. Springer Verlag, 2001.