Compositional Inductive Invariant Inference via Assume-Guarantee Reasoning

IAN DARDIK, Carnegie Mellon University, USA
EUNSUK KANG, Carnegie Mellon University, USA

A common technique for verifying the safety of complex systems is the *inductive invariant* method. Inductive invariants are inductive formulas that overapproximate the reachable states of a system and imply a desired safety property. However, inductive invariants are notoriously complex, which makes *inductive invariant inference* a challenging problem. In this work, we observe that inductive invariant formulas are complex primarily because they must be closed over the transition relation of an entire system. Therefore, we propose a new approach in which we decompose a system into components, assign an assume-guarantee contract to each component, and prove that each component fulfills its contract by inferring a *local inductive invariant*. The key advantage of local inductive invariant inference is that the local invariant need only be closed under the transition relation for the component, which is simpler than the transition relation for the entire system. Once local invariant inference is complete, system-wide safety follows by construction because the conjunction of all local invariants becomes an inductive invariant *for the entire system*. We apply our compositional inductive invariant inference technique to two case studies, in which we provide evidence that our framework can infer invariants more efficiently than the global technique. Our case studies also show that local inductive invariants provide modular insights about a specification that are not offered by global invariants.

ACM Reference Format:
Ian Dardik and Eunsuk Kang. 2025. Compositional Inductive Invariant Inference via Assume-Guarantee Reasoning. 1, 1 (July 2025), 33 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

A common approach to verifying the safety of complex systems, such as distributed protocols, is to find an *inductive invariant*. An inductive invariant is a logical formula that overapproximates the reachable state space of a system (invariance), is closed with respect to the system's transition relation (inductiveness), and also implies safety; together, these conditions imply that the system is safe. However, inductive invariant formulas can be notoriously large and complex, even for smaller systems. As a result, the important problem of *inductive invariant inference* remains a significant research challenge.

Over the past several decades, researchers have developed principled approaches for traversing the large space of candidate inductive invariant formulas. One common type of method is the *incremental* approach [3, 15, 24, 25, 31, 47] that accumulates non-inductive invariants–referred to as *lemmas*–until their conjunction becomes inductive. Yet another approach is to efficiently perform a bounded enumeration of candidate formulas [19, 51]. Techniques such as these have made inductive invariant inference applicable to a larger range of systems, and have been used to automatically verify sophisticated distributed protocols such as Paxos [27].

Authors' Contact Information: Ian Dardik, idardik@andrew.cmu.edu, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA; Eunsuk Kang, eunsukk@andrew.cmu.edu, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA.

ACM XXXX-XXXX/2025/7-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Despite progress in taming the search space, state-of-the-art techniques still struggle to infer complex inductive invariants that are necessary for verifying large systems. Our key observation is that inductive invariants become large and complex primarily because the formula must be closed under the transition relation of the entire system. Therefore, instead of attempting to synthesize a global invariant for the entire system, in this paper, we propose a *compositional* approach that aims to find multiple simpler inductive invariants. Our approach involves decomposing a specification into components and separately inferring a *local inductive invariant* for each component. Local inductive invariants are only required to be closed under the transition relation of the corresponding component, rather than the entire system. The advantage to local inductive invariant inference is that each component has a simpler transition relation than the overall system and, therefore, presents a simpler inference task.

Our proposed compositional approach is based on a novel *assume-guarantee reasoning* [45, 14] theory. Assume-guarantee reasoning is a verification paradigm in which each component C is assigned an individual proof obligation $\langle\alpha\rangle\, C\, \langle\gamma\rangle$ called a *contract*. Intuitively, a contract establishes that C guarantees the property γ under the assumption α. In our theory, the goal is to infer a local inductive invariant for each component to show that it fulfills its contract.

In order to define an assume-guarantee theory that allows for proof via local inductive invariant, we consider both *action-based* and *state-based* behavioral semantics. The need to consider both semantic paradigms arises because assume-guarantee contracts are a means for both *composition* and *proof localization*. On the one hand, action-based semantics are convenient for composing contracts because components can be treated algebraically and composed via parallel composition. For example, action-based composition styles from CSP [21] and the π-calculus [35] are often used in assume-guarantee reasoning for concurrent and distributed systems [29, 30]. On the other hand, proofs via (local) inductive invariants require state-based semantics because inductive invariants are formulas over state variables.

In this paper, we introduce a two-layered assume-guarantee theory that offers the advantages of both semantic paradigms. The top layer of the theory is action-based and intended for compositional reasoning, while the bottom layer is state-based and intended for proof via local inductive invariant. In the two-layer approach, we decompose a system into action-based contracts in the top layer theory, which we then translate to state-based contracts in the bottom-layer theory to be proved. The key to translating contracts from the action-based theory to the state-based theory is a novel logic language that we propose called *Symbolic Fluent Linear Temporal Logic* (*SFL*). SFL formulas are action-based, but can be translated to a semantically equivalent state-based formula. In other words, given an action-based contract $\langle\alpha\rangle\, C\, \langle\gamma\rangle$ where α and γ are SFL formulas, we can translate it to an equivalent state-based contract $\langle\!\langle A\rangle\!\rangle\, C\, \langle\!\langle G\rangle\!\rangle$ where A and G are state-based formulas.

Using our two-layered assume-guarantee theory, we present **the first divide-and-conquer approach to inductive invariant inference**. The approach involves four steps, which are shown in Fig. 1. Given a system that we wish to verify, the first step is to decompose it into a set of components. The second step is to find SFL formulas α and γ for each component C to create a candidate action-based contract $\langle\alpha\rangle\, C\, \langle\gamma\rangle$. Third, for each component, we infer a local inductive invariant that proves its corresponding state-based contract $\langle\!\langle A\rangle\!\rangle\, C\, \langle\!\langle G\rangle\!\rangle$. Local inductive invariant inference reduces to the usual inductive invariant inference problem, which can be accomplished using existing search-based techniques. Therefore, our work is complementary to–rather than a replacement for–existing work in inductive invariant inference. The fourth and final step is to take the conjunction of all local inductive invariants; by construction, the resulting formula is an inductive invariant *for the entire system.*

We evaluate our technique on two case studies of distributed protocols, the first of which is the classic Two Phase Commit Protocol [17]. The second case study is an industrial-scale protocol of a

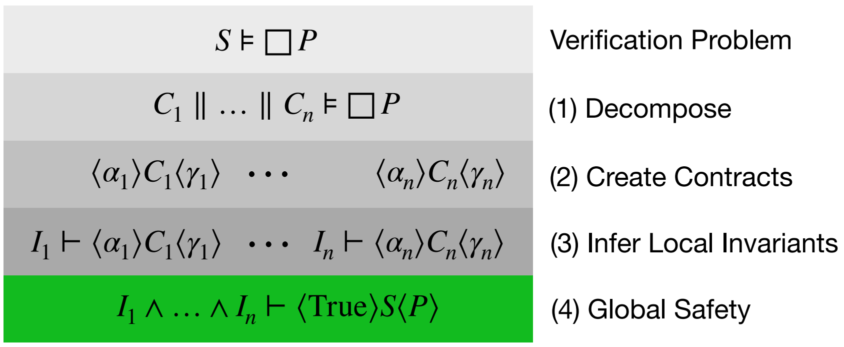
$S \vDash \Box P$
Verification Problem
$C_1 \parallel \dots \parallel C_n \vDash \Box P$
(1) Decompose
$\langle\alpha_1\rangle C_1\langle\gamma_1\rangle \cdots \langle\alpha_n\rangle C_n\langle\gamma_n\rangle$
(2) Create Contracts
$I_1 \vdash \langle\alpha_1\rangle C_1\langle\gamma_1\rangle \cdots I_n \vdash \langle\alpha_n\rangle C_n\langle\gamma_n\rangle$
(3) Infer Local Invariants
$I_1 \wedge \dots \wedge I_n \vdash \langle\text{True}\rangle S\langle P\rangle$
(4) Global Safety

Fig. 1. Overview of our compositional inductive invariant inference framework. Each layer in the figure represents an equivalent verification problem. From top to bottom, the figure shows the original verification problem, decomposition of the system S into components, creation of assume-guarantee contracts for each component, local inductive invariant inference for each component, and proof of global safety by construction. The notation $I \vdash \langle\alpha\rangle C\langle\gamma\rangle$ indicates that I is a local inductive invariant for the assume-guarantee contract.

Raft [40] style algorithm that runs at MongoDB [36]. While the MongoDB case study is beyond the reach of all existing automatic inductive invariant inference techniques, we use our compositional inference framework to infer an inductive invariant semi-automatically. Our results in the two case studies provide evidence that our divide-and-conquer approach can be more efficient than the global approach for inductive invariant inference. Additionally, we show that the artifacts from compositional inference–local inductive invariants and assume-guarantee contracts–provide valuable insights into the behaviors of the system that the global approach does not offer.

In summary, our contributions are:

(1) A two-layered assume-guarantee theory, whose contracts we prove using the novel proof by local inductive invariant method,
(2) Symbolic Fluent Linear Temporal Logic (SFL), a logic language for specifying action-based behavioral properties of parameterized systems,
(3) A compositional verification framework driven by a divide-and-conquer inductive invariant inference algorithm,
(4) Two case studies in which we show the efficacy of our verification framework and highlight the insights that the compositional approach offers about the verified system.

The rest of the paper is structured as follows. In Sec. 2 we introduce the running example and background information for the paper. In Sec. 3 we introduce the bottom layer state-based theory, followed by the top layer action-based theory in Sec. 4. Using the two layered assume-guarantee theory, we introduce our compositional inductive invariant inference framework in Sec. 5. We present the case studies in our evaluation in Sec. 6, followed by a survey of related work in Sec. 7. Finally, we conclude in Sec. 8 with a discussion of the limitations of this paper and future work.

2 Background

We now introduce background information, including the running example, the TLA$^+$ formal specification language, parameterized systems, and the inductive invariant proof method.

Running Example: Toy-2PC. We now introduce a toy version of the classic Two Phase Commit protocol [17], Toy-2PC, which we use as a running example throughout the paper. Toy-2PC is a distributed database commit protocol in which a *transaction manager* (TM) communicates with a set of *resource managers* (RMs) in attempt to commit a database transaction. The protocol unfolds over two phases, which we now describe. In the first phase, each RM starts in the *working* state as it attempts to commit the transaction; any RM that can commit a transaction sends a *prepared* message to the TM. In the second phase, the TM will issue a *commit* message to each RM if they are all prepared, or an *abort* message otherwise. The safety property for the protocol is *consistency*, meaning that no two RMs should disagree as to whether a transaction was committed or aborted. For simplicity, Toy-2PC assumes a perfect network, while the original protocol allows messages to be delayed, dropped, and reordered.

TLA⁺. We will use the TLA⁺ formal specification language [26] for the running example as well as our case studies. TLA⁺ is based on first-order logic (FOL) and temporal logic, and is often used for specifying parameterized symbolic transition systems and their temporal properties.

For example, Fig. 2 shows a specification for Toy-2PC. In Fig. 2, lines 1 and 2 of the specification respectively declare parameters (via the CONSTANT keyword) and state variables (via the VARIABLES keyword). Lines 3-6 specify the initial state predicate, while the remainder of the specification encodes the actions of the symbolic transition relation. The symbolic transition relation is the formula $\exists r \in RMs : Prepare(r) \lor Commit(r) \lor Abort(r) \lor SilentAbort(r)$; however, for brevity, we will omit the transition relation formula from the specifications shown in this paper. In the actions, prime operators (') are used to denote the value of a variable at the following time step and the UNCHANGED keyword explicitly indicates the frame condition for an action. The notation on line 4 uses square brackets to define $rmState$ as a function, while the notation using EXCEPT on line 11 changes one value of the function $rmState$. Line 8 accesses the value of the $rmState$ function, while line 13 uses angle brackets to specify a tuple.

The safety property $Consistent$ for the Toy-2PC protocol can be written as an invariant in TLA⁺, shown in Fig. 3. An invariant is a formula of the form $\square P$, where $\square$ is the *always* temporal operator and P is a non-temporal FOL formula. Oftentimes, we will also refer to non-temporal FOL formulas as invariants, in which case the outer $\square$ is implied; this is the case for the formula for $Consistent$. One can prove that the Toy-2PC protocol satisfies $\square Consistent$ using the inductive invariant proof method, which we describe later in this section.

Parameterized Symbolic Transition Systems. A parameterized symbolic transition system (PSTS) is an indexed triple $(vars, Init, Next)^p$, where p is a (possibly empty) set of parameters, $vars$ is a set of state variables, and $Init$ and $Next$ are first-order logic formulas for the initial-state constraint and the symbolic transition relation respectively. Syntactically, $Init$ may reference the parameters in p and the variables in $vars$, while $Next$ may reference the parameters in p and the variables in $vars \cup vars'$, where $vars' = \{v' \mid v \in vars\}$. For example, the TLA⁺ specification for Toy-2PC in Fig. 2 defines a PSTS $(vars, Init, Next)^{\{RMs\}}$, where $vars$ and $Init$ are defined in Fig. 2, and $Next$ is the (existentially quantified) disjunction of the actions in the specification $\exists r \in RMs : Prepare(r) \lor \ldots$ Parameters are also associated with domains, e.g., RMs is associated with a countably infinite set of possible resource managers.

We will use the notation $Act\ S$ to denote the alphabet of a PSTS S, $Params(S)$ for the set of parameters, and $SV(S)$ for the set of state variables. We consider actions to be *event formulas*, meaning that we will treat them as both events (e.g., in alphabets) and formulas (e.g., in the transition relation). We also overload $Params(F)$ and $SV(F)$ to denote the set of parameters and the set of state variables for a formula F. As an example, we have $Act\ Toy2PC = \{Prepare(rm), Commit(rm), Abort(rm), SilentAbort(rm)\}$, $Params(Toy2PC) = \{RMs\}$, and $SV(Toy2PC) = \{rmState, tmState, tmPrepared\}$. Notice that, as event formulas, actions such as $Prepare(rm)$ appear in $Act\ Toy2PC$, but also have formula definitions in Fig. 2.

```
──────────────────── MODULE ToyTwoPhase ────────────────────
CONSTANT RMs
VARIABLES rmState, tmState, tmPrepared

Init ≜
    ∧ rmState = [rm ∈ RMs ↦ "working"]
    ∧ tmState = "init"
    ∧ tmPrepared = {}

Prepare(rm) ≜
    ∧ rmState[rm] = "working"
    ∧ tmState = "init"
    ∧ rmState' =
        [rmState EXCEPT ![rm] = "prepared"]
    ∧ tmPrepared' = tmPrepared ∪ {rm}
    ∧ UNCHANGED ⟨tmState⟩

Commit(rm) ≜
    ∧ tmState ∈ {"init", "commit"}
    ∧ tmPrepared = RMs
    ∧ rmState' = [rmState EXCEPT ![rm] = "commit"]
    ∧ tmState' = "commit"
    ∧ UNCHANGED ⟨tmPrepared⟩

Abort(rm) ≜
    ∧ tmState ∈ {"init", "abort"}
    ∧ rmState' = [rmState EXCEPT ![rm] = "abort"]
    ∧ tmState' = "abort"
    ∧ UNCHANGED ⟨tmPrepared⟩

SilentAbort(rm) ≜
    ∧ rmState[rm] = "working"
    ∧ rmState' = [rmState EXCEPT ![rm] = "abort"]
    ∧ UNCHANGED ⟨tmState, tmPrepared⟩
─────────────────────────────────────────────────────────────
```

Fig. 2. The TLA$^+$ specification for Toy-2PC.

$$Consistent \triangleq \forall\, r1,\ r2 \in RMs : \neg(rmState[r1] = \text{“abort”} \land\ rmState[r2] = \text{“commit”})$$

Fig. 3. The definition for $Consistent$, the key safety property of Toy-2PC.

We will consider *state-based* and *action-based* behaviors in this paper. State-based and action-based behaviors are sequences of states and actions respectively, where a state is an assignment to variables and an action is an *event formula* as described above. We use subscripts to denote the ith state or action of a behavior. If τ is a state-based behavior and $i \geq 0$, then $\tau_i \models P$ if and only if the formula P is valid when replacing each (non-primed) variable in P with the value assigned to by τ_i. If σ is an action-based behavior, $i \geq 0$, and s and t are states, then $(s, t) \models \sigma_i$ if and only if the event formula σ_i is valid when replacing each non-primed variable in σ_i with the values assigned to by s, and each primed variable with the value assigned to by t.

A behavior satisfies a PSTS if and only if it satisfies all of its *instances*. An instance of a PSTS is a symbolic transition system $(vars, Init, Next)^{\hat{p}}$, where $\hat{p}$ is a function that maps each parameter to a subset of its domain. For example, $\hat{p} = [RMs \mapsto \{rm_1, rm_2\}]$ corresponds to an instance for Toy-2PC. Given a state-based behavior τ and a symbolic transition system $S = (vars, Init, Next)^{\hat{p}}$, $\tau \models S$ if and only if $\tau_0 \models Init$ and, for each $i \geq 0$, $(\tau_i, \tau_{i+1}) \models Next$. Given an action-based behavior σ and a symbolic transition system S, $\sigma \models S$ if and only if there exists a state-based behavior τ such that $\tau \models S$ and, for each $i \geq 0$, $(\tau_i, \tau_{i+1}) \models \sigma_i$. We will use the notation $\mathcal{L}(S)$ to denote the language of S, which is the set of all action-based behaviors that satisfy S. In a given state s, we say that an action a is *enabled* if there exists a state t such that $(s, t) \models a$. We call S *deterministic* if, for all states s, t_1, and t_2 and any action a, it is the case that $(s, t_1) \models a$ and $(s, t_2) \models a$ implies $t_1 = t_2$.

Let $S_i = (vars_i, Init_i, Next_i)^p$ for $i \in \{1, 2\}$, where $vars_1 \cap vars_2 = \emptyset$. We define parallel composition over PSTSs as follows: $S_1 \parallel S_2 \triangleq (vars_1 \cup vars_2, Init_1 \land Init_2, Next)^p$, where the symbolic transition relation $Next$ is defined as:

```
------------------- MODULE ToyRM -------------------
CONSTANT RMs
VARIABLES rmState

Init ≜ rmState = [rm ∈ RMs ↦ "working"]

Prepare(rm) ≜
   ∧ rmState[rm] = "working"
   ∧ rmState' =
      [rmState EXCEPT ![rm] = "prepared"]

Commit(rm) ≜
   ∧ rmState' =
      [rmState EXCEPT ![rm] = "commit"]

Abort(rm) ≜
   ∧ rmState' =
      [rmState EXCEPT ![rm] = "abort"]

SilentAbort(rm) ≜
   ∧ rmState[rm] = "working"
   ∧ rmState' =
      [rmState EXCEPT ![rm] = "abort"]
====================================================
```

```
------------------- MODULE ToyTM -------------------
CONSTANT RMs
VARIABLES tmState, tmPrepared

Init ≜
   ∧ tmState = "init"
   ∧ tmPrepared = {}

Prepare(rm) ≜
   ∧ tmState = "init"
   ∧ tmPrepared' = tmPrepared ∪ {rm}
   ∧ UNCHANGED ⟨tmState⟩

Commit(rm) ≜
   ∧ tmState ∈ {"init", "commit"}
   ∧ tmPrepared = RMs
   ∧ tmState' = "commit"
   ∧ UNCHANGED ⟨tmPrepared⟩

Abort(rm) ≜
   ∧ tmState ∈ {"init", "abort"}
   ∧ tmState' = "abort"
   ∧ UNCHANGED ⟨tmPrepared⟩
====================================================
```

Fig. 4. Toy-2PC decomposed into two components, $ToyRM$ and $ToyTM$.

$$\bigvee_{a \,\in\, Act\ S} \begin{cases} \exists d \in D \ : \ S_1!a \ \land \ S_2!a & \text{if } a \in Act\ S_1 \text{ and } a \in Act\ S_2 \\ \exists d \in D \ : \ S_1!a \ \land \ S_2!vars' = S_2!vars & \text{if } a \in Act\ S_1 \text{ and } a \notin Act\ S_2 \\ \exists d \in D \ : \ S_1!vars' = S_1!vars \ \land S_2!a & \text{if } a \notin Act\ S_1 \text{ and } a \in Act\ S_2 \end{cases}$$

Here, $\exists d \in D$ quantifies over the parameters associated with each action a (d appears syntactically in a), the operator ! indicates the scope of the definition for a formula, and $Act\ S = (Act\ S_1) \cup (Act\ S_2)$. Essentially, the symbolic transition relation synchronizes actions that are shared between components and interleaves unshared actions. We remark that parallel composition is only defined for PSTSs that have the same parameters p but do not share any state variables. As an example, consider the decomposition of the Toy-2PC specification shown in Fig. 4. In this case, we have $Toy2PC = ToyRM \parallel ToyTM$. We refer to $ToyRM$ and $ToyTM$ as the *components* of the Toy-2PC system.

The Inductive Invariant Proof Method. Because PSTS are families of transition systems, typical model checking techniques, such as explicit-state enumeration and symbolic model checking, are generally insufficient for verification. Instead, verification is usually performed using the inductive invariant proof method. This method proves that a PSTS $(vars, Init, Next)^p$ satisfies an invariant $\Box P$ by finding a formula Inv, called an *inductive invariant*, that satisfies the following three equations:

$$Init \implies Inv \tag{1}$$

$$Inv \land Next \implies Inv' \tag{2}$$

$$Inv \implies P \tag{3}$$

Equation (1) is called *initiation* and (2) is called *consecution*; together, these conditions show that Inv is an overapproximation of the reachable states of the transition system. On the other hand, equation (3) shows that Inv is an underapproximation of the invariant $\Box P$. Together, the three conditions above imply that the transition system is safe. The problem of *inductive invariant inference* is the task of finding an inductive invariant that can be used to verify a system. We refer to any technique that finds Inv directly–without decomposing the system–as a *global inductive invariant inference* technique.

3 State-Based Assume-Guarantee Theory

In this section, we present our *state-based* assume-guarantee theory that occupies the bottom of our two-layered theory. We begin in Sec. 3.1 by defining state-based contracts, as well as the local inductive invariant proof technique for proving that a state-based contract is fulfilled. Subsequently, in Sec. 3.2, we present inference rules for composing state-based contracts and their local inductive invariants. Although the state-based theory is not intended for composition, we use concepts from the action-based theory (e.g., parallel composition over components) to create composition rules. Ultimately, these composition rules are not intended to be used directly for verification, but they will simplify our definitions for the composition rules in the action-based theory. Finally, we dedicate Appendix B to proving that these composition rules are sound.

3.1 State-Based Contracts and Local Inductive Invariants

We will use the notation $\langle\!\langle A\rangle\!\rangle C \langle\!\langle G\rangle\!\rangle$ for contracts in our state-based theory, where G is the guarantee that the component C must satisfy under the assumption A. We base contracts on *invariance*, meaning that the component may assume A at all time steps, but must also guarantee G at all time steps. Therefore, the proof obligation of a contract is to satisfy the invariant G, while only considering those states of C that satisfy A. We capture this formally in the following definition.

Definition 3.1. Let $C = (vars, Init, Next)^p$ be a PSTS, and let A and G be non-temporal FOL formulas such that $SV(A) \cup SV(G) \subseteq vars$ and $Params(A) \cup Params(G) \subseteq p$. A contract $\langle\!\langle A\rangle\!\rangle C \langle\!\langle G\rangle\!\rangle$ is *fulfilled* if and only if $(vars, Init \land A, Next \land A')^p \models \Box G$. We will denote contract fulfillment with the notation $\vdash \langle\!\langle A\rangle\!\rangle C \langle\!\langle G\rangle\!\rangle$.

As an example, consider the $ToyRM$ component in the Toy-2PC protocol, shown in Fig. 4. If we assume–strictly for the sake of demonstration–that no resource manager will ever abort, then it is possible to prove that $ToyRM$ guarantees the safety property $Consistent$, shown in Fig. 3. If we let $NoAbort = \forall\, r \in RMs : rmState[r] \neq \text{“abort”}$, then $\vdash \langle\!\langle NoAbort\rangle\!\rangle\, ToyRM \,\langle\!\langle Consistent\rangle\!\rangle$.

Unfortunately, conventional model-checking techniques cannot be used to show that the above contract is fulfilled because $ToyRM$ is parameterized with an infinite domain. Therefore, we turn our attention to a more powerful technique for showing contract fulfillment called the *local inductive invariant* proof method. In this proof method, one must find a formula I that is an inductive invariant for the component under an assumption A, meaning that A can be assumed to hold at all time steps. We will use the notation $I \vdash \langle\!\langle A\rangle\!\rangle C \langle\!\langle G\rangle\!\rangle$ to mean that I is a local inductive invariant that proves that the corresponding contract is fulfilled.

Definition 3.2. Let $C = (vars, Init, Next)^p$ be a PSTS, $\langle\!\langle A\rangle\!\rangle C \langle\!\langle G\rangle\!\rangle$ be a contract, and I be a non-temporal FOL formula such that $SV(I) \subseteq vars$ and $Params(I) \subseteq p$. Then we write $I \vdash \langle\!\langle A\rangle\!\rangle C \langle\!\langle G\rangle\!\rangle$

to mean that the following three equations are valid:

$$Init \wedge A \implies I \tag{4}$$

$$I \wedge Next \wedge A \wedge A' \implies I' \tag{5}$$

$$I \implies G \tag{6}$$

In the Toy-2PC example, $NoAbort$ serves as a local inductive invariant to prove the contract from above, so we additionally write $NoAbort \vdash \langle\!\langle NoAbort \rangle\!\rangle\, ToyRM \,\langle\!\langle Consistent \rangle\!\rangle$. Finally, we conclude this subsection by showing that finding a local inductive invariant is a sufficient condition for contract fulfillment; we provide a proof for this theorem in Appendix A.

Theorem 3.3. *$I \vdash \langle\!\langle A \rangle\!\rangle\, C \,\langle\!\langle G \rangle\!\rangle$ implies $\vdash \langle\!\langle A \rangle\!\rangle\, C \,\langle\!\langle G \rangle\!\rangle$.*

3.2 Composing Contracts and Local Inductive Invariants

In this section, we present inference rules for composing state-based contracts. On their own, these rules can be challenging to use for verifying large systems. Instead, these inference rules are intended to lay the groundwork for composing action-based contracts in Sec. 4. In particular, we will leverage the soundness theorems for these rules to prove the soundness of the action-based composition rules. At the end of this subsection, we provide a discussion with the explicit reason the rules are tough to use on their own.

The inference rules for composition in this paper are based on *transitivity* of invariance. This style of contract composition resembles that of the popular and successful assume-guarantee verification theory for finite-state systems, originally introduced by Cobleigh et al. [7]. We will exclusively focus on transitive-style contract composition; however, in the future we plan to extend our theory with additional types of composition, e.g., circular composition [1, 16, 33, 38] and conjunctive styles of composition as seen in the Owicki-Gries method [42] and Concurrent Separation Logic (called the disjoint concurrency rule) [39].

The idea behind our transitive composition method is to find an invariant R that serves as the guarantee for one contract and the assumption of another, e.g., as seen in the rule NAIVE-COMP in Fig. 5. The formula R is often referred to as an *assumption*, however we will refer to R as a *bridge formula* (or simply a *bridge*), since we already reserve the term *assumption* for the assumption of a contract. In addition to composing contracts, we also provide inference rules for composing the associated local inductive invariants. The NAIVE-COMP rule, for example, composes two contracts as well as their associated local inductive invariants I_1 and I_2. The composed invariants–that is, the formula $I_1 \wedge I_2$–becomes a local inductive invariant for the composed contract.

Unfortunately, as the name suggests, NAIVE-COMP is a naive rule that cannot be used to prove fulfillment for many valid contracts. The problem with NAIVE-COMP is that the components do not share state variables, and therefore the formula R cannot refer to *any* state variables (by Def. 3.1). In practice, this is a problem because local correctness arguments often rely on assumptions about the correct behaviors of their state variables. For example, in Toy-2PC, NAIVE-COMP cannot be used to show that the $ToyRM$ components satisfies $Consistent$. The reason is because $ToyRM$ only guarantees $Consistent$ if it can assume that the $ToyTM$ component operates correctly; i.e. the $ToyTM$ must only choose to commit if all resource managers have prepared, and can never issue both a commit and an abort message. Unfortunately, this assumption cannot be expressed in terms of $ToyRM$'s state variables.

To address the issue identified above, we propose a *bridge component* that acts as an intermediary between the two components. The key idea–shown by rule BRIDGE-COMP in Fig. 5–is that the components share the bridge component B, so it becomes possible to write a bridge assumption R over the "shared" state variables in B. While the bridge component can be an existing component of

$$\textsc{naive-comp}\quad \frac{I_1 \vdash \langle\!\langle A \rangle\!\rangle C_1 \langle\!\langle R \rangle\!\rangle \qquad I_2 \vdash \langle\!\langle R \rangle\!\rangle C_2 \langle\!\langle G \rangle\!\rangle}{I_1 \land I_2 \vdash \langle\!\langle A \rangle\!\rangle C_1 \parallel C_2 \langle\!\langle G \rangle\!\rangle} \qquad \textsc{bridge-comp}\quad \frac{I_1 \vdash \langle\!\langle A \rangle\!\rangle C_1 \parallel B \langle\!\langle R \rangle\!\rangle \qquad I_2 \vdash \langle\!\langle R \rangle\!\rangle B \parallel C_2 \langle\!\langle G \rangle\!\rangle}{I_1 \land I_2 \vdash \langle\!\langle A \rangle\!\rangle C_1 \parallel B \parallel C_2 \langle\!\langle G \rangle\!\rangle}$$

$$\textsc{aux-comp}\quad \frac{I_1 \vdash \langle\!\langle A \rangle\!\rangle C_1 \parallel B \langle\!\langle R \rangle\!\rangle \qquad I_2 \vdash \langle\!\langle R \rangle\!\rangle B \parallel C_2 \langle\!\langle G \rangle\!\rangle \qquad Aux\ B}{\vdash \langle\!\langle A \rangle\!\rangle C_1 \parallel C_2 \langle\!\langle G \rangle\!\rangle}$$

Fig. 5. Inference rules for composing contracts and inductive invariants in the state-based theory.

```
MODULE ToyB

CONSTANT RMs                                     Commit(rm) ≜
VARIABLES oncePrepare, onceCommit,                 ∧ onceCommit′ =
   onceAbort                                          [onceCommit EXCEPT ![rm] = TRUE]
                                                   ∧ UNCHANGED ⟨oncePrepare, onceAbort⟩
Init ≜
   ∧ oncePrepare = [rm ∈ RMs ↦ FALSE]            Abort(rm) ≜
   ∧ onceCommit = [rm ∈ RMs ↦ FALSE]                ∧ onceAbort′ =
   ∧ onceAbort = [rm ∈ RMs ↦ FALSE]                    [onceAbort EXCEPT ![rm] = TRUE]
                                                   ∧ UNCHANGED ⟨oncePrepare, onceCommit⟩
Prepare(rm) ≜
   ∧ oncePrepare′ =
      [oncePrepare EXCEPT ![rm] = TRUE]
   ∧ UNCHANGED ⟨onceCommit, onceAbort⟩
```

Fig. 6. A bridge component for Toy-2PC.

the system, we propose creating bridge components exclusively with *auxiliary variables*. Auxiliary variables were originally proposed by Owicki and Gries to make their inference system complete [41]. Similarly, auxiliary variables make our theory more widely applicable, though not necessarily complete. We lift the notion of auxiliary variables to *auxiliary components* in the definition below.

Definition 3.4. A component B is an *auxiliary component* if, for any contract $\langle\!\langle A \rangle\!\rangle C \langle\!\langle G \rangle\!\rangle$, we have $\vdash \langle\!\langle A \rangle\!\rangle C \parallel B \langle\!\langle G \rangle\!\rangle$ implies $\vdash \langle\!\langle A \rangle\!\rangle C \langle\!\langle G \rangle\!\rangle$. When B is an auxiliary component we write $Aux\ B$.

In the case that $Aux\ B$, we will refer to a pair (B, R) that is used with the BRIDGE-COMP rule as a *bridge pair*. Using an auxiliary bridge component with the BRIDGE-COMP rule is also sound, in the sense that it proves $\vdash \langle\!\langle A \rangle\!\rangle C_1 \parallel C_2 \langle\!\langle G \rangle\!\rangle$. This fact follows from the AUX-COMP inference rule in Fig. 5. Notice that, in general, the inductive invariant $I_1 \land I_2$ in the conclusion of the BRIDGE-COMP rule will contain auxiliary variables from the bridge component. Therefore, we include AUX-COMP as a separate inference rule because $I_1 \land I_2$ may not be well-defined over $C_1 \parallel C_2$. Despite this technicality, the formula $I_1 \land I_2$ is nevertheless an inductive invariant that proves $\vdash \langle\!\langle A \rangle\!\rangle C_1 \parallel C_2 \langle\!\langle G \rangle\!\rangle$ when using an auxiliary bridge component.

Toy-2PC example. Using our state-based theory, we will compositionally infer an inductive invariant for the Toy-2PC example. We begin by creating a bridge component $ToyB$ (Fig. 6) with three auxiliary variables $oncePrepare$, $onceCommit$, and $onceAbort$. These variables respectively represent whether a $Prepare$, $Commit$, or $Abort$ action has happened at least once in the past. We define the bridge formula $ToyR$ (Fig. 7) over the auxiliary variables in $ToyB$. This bridge formula formally encodes our earlier intuition that, in order to verify consistency, $ToyRM$ must assume

$$
\begin{aligned}
&ToyR \triangleq \\
&\quad \land (\exists r \in RMs : onceCommit[r]) \Rightarrow (\forall r \in RMs : oncePrepare[r]) \\
&\quad \land (\exists r \in RMs : onceAbort[r]) \Rightarrow (\forall r \in RMs : \neg onceCommit[r])
\end{aligned}
$$

$$
\begin{aligned}
&I_{RM} \triangleq \\
&\quad \land\ Consistent \\
&\quad \land \forall r \in RMs : onceCommit[r] \iff rmState[r] = \text{“commit”} \\
&\quad \land \forall r \in RMs : oncePrepare[r] \Rightarrow rmState[r] \neq \text{“working”} \\
&\quad \land \forall r \in RMs : (oncePrepare[r] \land rmState[r] = \text{“abort”}) \Rightarrow onceAbort[r]
\end{aligned}
$$

$$
\begin{aligned}
&I_{TM} \triangleq \\
&\quad \land\ ToyR \\
&\quad \land \forall r \in RMs : r \in tmPrepared \Rightarrow oncePrepared[r] \\
&\quad \land \forall r \in RMs : onceCommit[r] \Rightarrow tmState = \text{“commit”} \\
&\quad \land \forall r \in RMs : onceAbort[r] \Rightarrow tmState = \text{“abort”}
\end{aligned}
$$

Fig. 7. A bridge formula and local inductive invariants for the components of Toy-2PC.

that the $ToyTM$ will only choose to commit if all resource managers have prepared, and also that $ToyTM$ will never issue both a commit and an abort message.

Next, we create a local inductive invariant I_{RM} (Fig. 7) for $ToyRM \parallel ToyB$, i.e. we have $I_{RM} \vdash \langle\langle ToyR \rangle\rangle\, ToyRM \parallel ToyB \langle\langle Consistent \rangle\rangle$. Similarly, we create a local inductive invariant I_{TM} (Fig. 7) for $ToyB \parallel ToyTM$, i.e. $I_{TM} \vdash \langle\langle \textsc{true} \rangle\rangle\, ToyB \parallel ToyTM \langle\langle ToyR \rangle\rangle$. The rule AUX-COMP allows us to conclude that the entire protocol (without assumptions) is safe, since $Toy2PC = ToyRM \parallel ToyTM$. Furthermore, the inference rule BRIDGE-COMP allows us to infer that $I_{RM} \land I_{TM}$ is an inductive invariant for $ToyRM \parallel ToyB \parallel ToyTM$. Because $ToyB$ is an auxiliary component, $I_{RM} \land I_{TM}$ is an inductive invariant *for the entire system.*

Purpose of the state-based composition rules. The inference rules from Fig. 5 can be used to show system-wide correctness, but require inventing a bridge component. Unfortunately, bridge components can be complex–especially for large systems–which can ultimately cause these composition rules to be difficult to use in practice. In Sec. 4, we will solve this problem by introducing simpler composition rules in the action-based theory that do not require inventing a bridge component. Therefore, the purpose of the composition inference rules from Fig. 5 is to build upon them in the action-based theory.

4 Action-Based Assume-Guarantee Theory

In this section, we present the action-based theory that sits at the top of our two-layered assume-guarantee theory. The action-based theory has inference rules for composition that are well suited for verification, which we will use in our compositional framework in Sec. 5. In particular, the inference rules for the action-based theory do not require inventing a bridge component. In the state-based theory, bridge components were a necessary intermediary for bridge formulas, since components do not share state variables. However, components share actions, which is the basis for the simplicity of composition in the action-based theory. In the action-based theory, assumptions, guarantees, and bridge formulas are encoded as *action invariants*, which are formulas with action-based semantics. Bridge formulas are required to reference the actions from the shared alphabet of components whose contracts are being composed, which makes an intermediary, such as a bridge component, unnecessary.

MODULE $oncePrepareT$
CONSTANT RMs
VARIABLES x
$Init \triangleq x = [rm \in RMs \mapsto \text{FALSE}]$
$Prepare(rm) \triangleq$
$\quad x' = [x \text{ EXCEPT } ![rm] = \text{TRUE}]$

(a) Definition for $oncePrepareT$.

MODULE $onceCommitT$
CONSTANT RMs
VARIABLES y
$Init \triangleq y = [rm \in RMs \mapsto \text{FALSE}]$
$Commit(rm) \triangleq$
$\quad y' = [y \text{ EXCEPT } ![rm] = \text{TRUE}]$

(b) Definition for $onceCommitT$.

MODULE $onceAbortT$
CONSTANT RMs
VARIABLES z
$Init \triangleq z = [rm \in RMs \mapsto \text{FALSE}]$
$Abort(rm) \triangleq$
$\quad z' = [z \text{ EXCEPT } ![rm] = \text{TRUE}]$

(c) Definition for $onceAbortT$.

$$\begin{aligned} oncePrepare &= (\langle RMs\rangle, x, oncePrepareT) \\ onceCommit &= (\langle RMs\rangle, y, onceCommitT) \\ onceAbort &= (\langle RMs\rangle, z, onceAbortT) \end{aligned}$$

(d) Fluent definitions for ρ_1 and ρ_2.

$$\sigma^1 = \big(Prepare(r_1)\big)$$
$$\sigma^2 = \big(Prepare(r_1), Prepare(r_2), Commit(r_2)\big)$$

(e) Example finite behaviors for the instance $RMs = \{r_1, r_2\}$.

$$\rho_1 \triangleq (\exists r \in RMs : onceCommit(r)) \implies (\forall r \in RMs : oncePrepare(r))$$
$$\rho_2 \triangleq (\exists r \in RMs : onceAbort(r)) \implies (\forall r \in RMs : \neg onceCommit(r))$$

(f) Example SFL formulas.

Fig. 8. Definitions for specifying and verifying Toy-2PC compositionally using SFL.

The advantages of using action-based formalisms for composition are well-known, which is the reason that assume-guarantee verification frameworks often use labeled transition systems for assumptions, guarantees, and bridges [2, 5, 7, 18, 37]. However, using action invariants in our contracts has the unique advantage that it allows our contracts to be translated to state-based contracts, which can then be proved with the local inductive invariant method. Allowing our contracts to be proved via local inductive invariants is a key advantage because it allows us to compositionally verify parameterized systems, which the assume-guarantee theories for labeled transition systems cannot, in general, be used for.

We will begin in Sec. 4.1 by introducing SFL, an action-based logic language that we will use to define action invariants. In Sec. 4.2, we show how to translate action invariants to state-based formalisms. In Sec. 4.3, we introduce our action-based assume-guarantee theory, in which we define contracts, local inductive invariants for action-based contracts, and inference rules for composing action-based contracts. Finally, in Sec. 4.4, we introduce a second type of action-based contract we call a *hybrid contract* that allows the action-based theory to verify state-based properties.

4.1 Symbolic Fluent LTL

We now introduce a novel logic language called *Symbolic Fluent Linear Temporal Logic* (SFL). SFL is based on Fluent Linear Temporal Logic (FLTL) [13], a logic language for specifying temporal properties over *actions*. SFL provides two extensions to FLTL that make it particularly well suited for specifying assumptions and guarantees for parameterized systems: *symbolic fluents* and quantifiers. We will describe these extensions along with the syntax and semantics for the new logic language. At the end of this subsection, we will use the SFL language to identify a class of formulas called *action invariants* that we will use to define action-based contracts in Sec. 4.3.

SFL syntax. Symbolic fluents are an extension of the non-symbolic fluents from FLTL. Non-symbolic fluents are action-based propositions in FLTL that are syntactically analogous to the state-based atomic propositions of Linear Temporal Logic (LTL) [46]. Semantically, however, fluents represent booleans that may turn on or off depending on each action taken. The set of non-symbolic fluents is fixed in each FLTL formula, which makes the language well-suited for specifying properties of systems that have a fixed alphabet, e.g., labeled transition systems. In a parameterized system, however, the alphabet generally depends on the parameters, which makes FLTL ill-suited for specifying properties of such systems. This motivates *symbolic fluents* that accept arguments and semantically correspond to boolean-valued functions. Note that we specifically use the term *argument* for symbolic fluents to differentiate from the *parameters* of a protocol. For the remainder of this paper, we will use the terms *fluent* and *symbolic fluent* synonymously. The formal definition of a fluent is as follows.

Definition 4.1. A symbolic fluent is a triple (s, v, T), where s is a tuple of parameters that specify the type of each argument to the fluent, v is a state variable, and T is a PSTS with the following four requirements:

(1) T has exactly one state variable v.
(2) T is deterministic.
(3) The type of v must always be a boolean-valued function whose arguments have the types given by s.
(4) Every action in $Act\ T$ is enabled in every state of T.

We also define the following notation for the alphabet of a symbolic fluent: $Act\ (s, v, T) = Act\ T$.

Returning to the Toy-2PC example, consider the fluent $oncePrepare$ in Fig. 8d. The fluent accepts one argument with type RMs and refers to a state variable x whose value changes according to the transition system $oncePrepareT$ from Fig. 8a. Intuitively, the fluent represents whether a $Prepare(rm)$ action has occurred at least once in the past, which can be seen from the definition of $oncePrepareT$. We will use the syntax $oncePrepare(rm)$ to access the value of the fluent at a given time step, where rm is an argument to the fluent. For example, after execution of the finite behavior σ^1 from Fig. 8e, $oncePrepare(r_1)$ will evaluate to TRUE, while $oncePrepare(r_2)$ will evaluate to FALSE. Because fluents in SFL accept arguments, we can introduce quantifiers into the language for the purpose of passing quantified variables as arguments to fluents. Based on these extensions, we now define the syntax for SFL formulas.

Definition 4.2. A Symbolic Fluent Linear Temporal Logic (SFL) formula has the syntax in Fig. 9, must be absent of free variables (arguments to fluents), and fluents may not share state variables. For an SFL formula ϕ, $fl(\phi)$ refers to the set of all symbolic fluents in ϕ and $Params(\phi)$ refers to the set of all parameters (each D from Fig. 9) that are quantified over in ϕ. In this paper, we will use Greek letters, namely ϕ, ψ, α, ρ, and γ, to represent SFL formulas.

REMARK 1. *We include standard logical connectives (e.g., $\land$ and $\implies$) and temporal logic operators (e.g., $\Box$ and $\Diamond$) in SFL, defined with the usual syntactic sugar. For example, the "always" operator can be defined in SFL as $\Box\phi = \neg(\text{TRUE}\ \boldsymbol{U} \neg\phi)$.*

For example, consider the SFL formula $\Box\rho_1$, where ρ_1 is defined in Fig. 8f. This formula is an invariant that quantifies over RMs and has two symbolic fluents, $oncePrepare$ and $onceCommit$, that are defined in Fig. 8d. The $onceCommit$ fluent is defined similarly to the $oncePrepare$ fluent described above, the main difference being that $onceCommit$ represents whether a $Commit(rm)$ action has occurred at least once (see the definition for $onceCommitT$ in Fig. 8b). Intuitively, the formula ρ_1 specifies that at all times, if any resource manager commits, then it must be the case that all resource managers have previously prepared.

$$SFL ::= \ SFL\,\mathbf{U}\,SFL \mid \mathbf{X}\,SFL \mid \exists x \in D : SFL \mid \ \neg SFL \mid SFL \vee SFL \mid f(ARG)$$
$$ARG ::= \ x \mid x, ARG$$

Fig. 9. BNF grammar for the SFL Syntax. **U** and **X** are temporal operators, while $\exists$, $\neg$, and $\vee$ are first-order logic connectives. $f(ARG)$ indicates a symbolic fluent f with one or more parameters. In ARG, x represents a quantified variable passed as an argument to a symbolic fluent.

$$\begin{array}{lll}
E \vdash \sigma, i \models \phi \mathbf{U} \psi & \text{iff} & \exists j \in \mathbb{N} : (j \geq i) \wedge (E \vdash \sigma, j \models \psi) \wedge \\
& & (\forall k \in \mathbb{N} : i \leq k < j \implies E \vdash \sigma, k \models \phi) \\
E \vdash \sigma, i \models \mathbf{X}\phi & \text{iff} & E \vdash \sigma, i+1 \models \phi \\
E \vdash \sigma, i \models \exists x \in D : \phi & \text{iff} & \exists y \in D : E[x \mapsto y] \vdash \sigma, i \models \phi \\
E \vdash \sigma, i \models \neg\phi & \text{iff} & \neg(E \vdash \sigma, i \models \phi) \\
E \vdash \sigma, i \models \phi \vee \psi & \text{iff} & (E \vdash \sigma, i \models \phi) \vee (E \vdash \sigma, i \models \psi) \\
E \vdash \sigma, i \models f(r) & \text{iff} & (f|\sigma_0 \ldots \sigma_i)[r[E]]
\end{array}$$

Fig. 10. SFL semantics.

SFL semantics. We now turn our attention to defining the formal semantics of SFL. We use a ternary operator of the form $E \vdash \sigma, i \models \phi$ to express that a (possibly infinite) action-based behavior σ satisfies the SFL formula ϕ at index i with environment E. An environment is an assignment of variables to values, and we use the notation $r[E]$ to mean the result of replacing all variables in r with their corresponding value in E. Additionally, we use the short hand $\sigma \models \phi$ to mean $[] \vdash \sigma, 0 \models \phi$, where $[]$ is the empty environment. We also define the language of an SFL formula ϕ as the set of all its satisfying behaviors: $\mathcal{L}(\phi) = \{\sigma \mid \sigma \models \phi\}$.

We present the formal semantics for SFL in Fig. 10. The semantics for SFL are standard, with the exception of symbolic fluent expressions which we now describe. Intuitively, the meaning of a symbolic fluent (s, v, T) is the boolean-valued function v after the execution of a finite behavior σ of actions in the transition system T. We now provide a notation for accessing the value of a fluent at a given time step.

Definition 4.3. Given a fluent $f = (s, v, T)$ and a finite action-based behavior $\sigma = \sigma_0\sigma_1 \ldots \sigma_i$, we let $f|\sigma$ denote the boolean-valued function v that is the result of executing σ in the transition system T. Note that any action in σ that is not in $Act\ T$ leaves the state variable v unchanged. We remark that this notation is well-defined because σ is finite, T is deterministic, and all actions in T are always enabled.

Fig. 10 uses the notation from Def. 4.3 to define the semantics of a symbolic fluent expression $f(r)$, where f is a fluent and r are the arguments passed to the fluent. In $(f|\sigma_0 \ldots \sigma_i)[r[E]]$ from the definition in Fig. 10, the arguments r contain quantified variables, and therefore $r[E]$ are concrete arguments. Essentially, the semantics of a fluent is the (boolean) value of the function $(f|\sigma_0 \ldots \sigma_i)$ given the concrete arguments $r[E]$.

For example, consider $oncePrepare$ from Fig. 8d as well as the finite behaviors from Fig. 8e. In this case, $oncePrepare|\sigma^1 = [r_1 \mapsto \text{TRUE}, r_2 \mapsto \text{FALSE}]$ and $onceCommit|\sigma^2 = [r_1 \mapsto \text{FALSE}, r_2 \mapsto \text{TRUE}]$. Alternatively, we can also write $(oncePrepare|\sigma^1)[r_1] = \text{TRUE}$, $(oncePrepare|\sigma^1)[r_2] = \text{FALSE}$, etc. We also point out that we have both $\sigma^2 \models \Box\rho_1$ and $\sigma^2 \models \Box\rho_2$.

$$\begin{array}{lll}
\mathcal{R}(\phi\mathbf{U}\psi) = \mathcal{R}(\phi)\mathbf{U}\mathcal{R}(\psi) & \mathcal{R}(\mathbf{X}\phi) = \mathbf{X}\mathcal{R}(\phi) & \mathcal{R}(\exists x \in D : \phi) = \exists x \in D : \mathcal{R}(\phi) \\
\mathcal{R}(\neg\phi) = \neg\mathcal{R}(\phi) & \mathcal{R}(\phi \vee \psi) = \mathcal{R}(\phi) \vee \mathcal{R}(\psi) & \mathcal{R}\big((s, v, T)(r)\big) = v[r]
\end{array}$$

Fig. 11. Definition of $\mathcal{R}$, a syntactic transformation of an SFL formula to a state-based one.

Action invariants. In the remainder of this paper, we will focus on a class of SFL formulas called *action invariants*. An action invariant is a formula of the form $\Box\phi$, where ϕ is a non-temporal SFL formula. For example, $\Box\rho_1$ and $\Box\rho_2$ are both action invariants. In Sec. 4.2, we will show that action invariants can be translated to state-based formalisms. Subsequently, in Sec. 4.3, we will use action invariants for the assumptions and guarantees of the contracts in our action-based assume-guarantee theory.

4.2 Translating Action Invariants to State-Based Formalisms

In this section, we show that action invariants have two state-based counterparts: bridge pairs and transition systems (PSTSs). When we introduce our action-based assume-guarantee theory in the following section (Sec. 4.3), the translation from action invariants to bridge pairs will allow us to translate action-based contracts to state-based contracts. On the other hand, the translation from action invariants to transition systems lets us treat action invariants as algebraic processes, which will help us to define action-based contracts.

Translation to bridge pairs. We now show that action invariants correspond to bridge pairs, which are state-based formalisms that we introduced in Sec. 3.2. The translations are given by two mappings $\mathcal{B}$ (bridge components) and $\mathcal{R}$ (bridge formulas), which we now define.

Definition 4.4. Let ϕ be an SFL formula, then $\mathcal{B}(\phi) = \|\{T \mid (s, v, T) \in \mathit{fl}(\phi)\}$, where $\mathit{fl}(\phi)$ is the set of all fluents in the formula ϕ and the $\|$ operator here indicates the parallel composition of all transition systems in the given set. We provide the formal definition for $\mathcal{R}$ in Fig. 11.

In essence, $\mathcal{B}$ is the parallel composition of all fluents in an SFL formula, while $\mathcal{R}$ is a syntactic transformation that replaces each fluent with its underlying state variable. These maps can be seen as a decoupling of an SFL formula into two parts, where $\mathcal{B}$ represents the dynamics of the fluents and $\mathcal{R}$ represents the invariant itself.

In the Toy-2PC example, we can translate the action invariant $\Box(\rho_1 \wedge \rho_2)$ (Fig. 8f) to the corresponding state-based formalisms. Up to a variable renaming, we have $\mathcal{B}(\Box(\rho_1 \wedge \rho_2)) = \mathit{ToyB}$ (Fig. 6) and $\mathcal{R}(\Box(\rho_1 \wedge \rho_2)) = \Box\,\mathit{ToyR}$ (Fig. 7). In this example, $\mathcal{B}$ maps an action invariant to an auxiliary component. The following theorem shows that $\mathcal{B}$ will, in general, map action invariants to auxiliary components. This result formally demonstrates that action invariants correspond to bridge pairs; we provide a proof in Appendix C.

Theorem 4.5. *Let $\Box\phi$ be an action invariant, then $\mathcal{B}(\Box\phi)$ is an auxiliary component.*

Translation to a transition system. We now show that action invariants correspond to transition systems. We provide this connection formally with the mapping $\mathcal{T}$, which we define in terms of the previously introduced maps $\mathcal{B}$ and $\mathcal{R}$.

Definition 4.6. Let $\Box\phi$ be an action invariant and also let $\mathcal{B}(\Box\phi) = (\mathit{vars}, \mathit{Init}, \mathit{Next})^p$. Then, $\mathcal{T}(\Box\phi) = (\mathit{vars}, \mathit{Init} \wedge \mathcal{R}(\phi), \mathit{Next} \wedge \mathcal{R}(\phi)')^p$.

Intuitively, $\mathcal{T}(\Box\phi)$ is a transition system version of the action invariant $\Box\phi$. We will show formally that these two are semantically equivalent (Thm. 4.7) after a brief example. Returning to the Toy-2PC example, $\mathcal{T}(\Box(\rho_1 \wedge \rho_2))$ is a transition system that represents $ToyB$ restricted to the behaviors of $\Box\, ToyR$. This intuition helps to see that $\mathcal{T}(\Box(\rho_1 \wedge \rho_2)) \parallel ToyRM \models \Box Consistent$ and also that $\mathcal{L}(ToyTM) \subseteq \mathcal{L}(\mathcal{T}(\Box(\rho_1 \wedge \rho_2)))$. In this example, $\mathcal{T}(\Box(\rho_1 \wedge \rho_2))$ appears to take the place of a bridge formula, acting as an assumption for $ToyRM$ to satisfy $Consistent$ and as a guarantee for $ToyTM$. Indeed, we will make this intuition precise in the following section (Sec. 4.3) by defining action-based contracts in terms of $\mathcal{T}$.

We now present the following theorem that shows that action invariants are semantically equivalent to their transition system counterpart given by $\mathcal{T}$. We provide a proof for this theorem in Appendix D.

Theorem 4.7. *For any action invariant $\Box\phi$, we have $\mathcal{L}(\Box\phi) = \mathcal{L}(\mathcal{T}(\Box\phi))$.*

4.3 Assume-Guarantee Theory with Action-Based Contracts

We now present our action-based assume-guarantee theory. We begin by defining contracts and the local inductive invariant proof method in this theory. Subsequently, we provide inference rules for composing contracts and their associated invariants.

Action-based contracts and local inductive invariants. We will use the notation $\langle\alpha\rangle C \langle\gamma\rangle$ for contracts in our action-based theory, where γ is the guarantee that C must satisfy under the assumption α. The meaning of a contract is that the language of the component C, when restricted to the assumption α, must be contained in the language of the guarantee γ. In typical assume-guarantee verification paradigms, restricting the component to the assumption is achieved algebraically, i.e. by taking the parallel composition of the assumption and the component. Similarly, we will restrict the component to the assumption by taking the parallel composition of C and $\mathcal{T}(\Box\alpha)$, the latter being semantically equivalent to the assumption by Thm. 4.7. We now define action-based contracts from this intuition.

Definition 4.8. Let $C = (vars, Init, Next)^p$ be a PSTS. Also let α and γ be non-temporal SFL formulas such that $Params(\alpha) \cup Params(\gamma) \subseteq p$ and $Act\ \alpha \cup Act\ \gamma \subseteq Act\ C$. Then a contract $\langle\alpha\rangle C \langle\gamma\rangle$ is *fulfilled* if and only if $\mathcal{L}(\mathcal{T}(\Box\alpha) \parallel C) \subseteq \mathcal{L}(\Box\gamma)$. We will denote contract fulfillment with the notation $\vdash \langle\alpha\rangle C \langle\gamma\rangle$.

For example, $\langle \text{TRUE} \rangle\, ToyTM \langle \rho_1 \wedge \rho_2 \rangle$ is an action-based contract. Proving that the contract is fulfilled, however, requires the local inductive invariant method. We can use the local inductive invariant method by translating the contract to a state-based contract, which we accomplish by translating the assumption and guarantee using the maps $\mathcal{B}$ and $\mathcal{R}$ from the prior section (Sec. 4.2). For an action-based contract $\langle\alpha\rangle C \langle\gamma\rangle$, the corresponding state-based contract is $\langle\!\langle \mathcal{R}(\alpha) \rangle\!\rangle \mathcal{B}(\Box\alpha) \parallel C \parallel \mathcal{B}(\Box\gamma) \langle\!\langle \mathcal{R}(\gamma) \rangle\!\rangle$. We will formally show that this choice of translation is appropriate below when we prove Thm. 4.10. Based on this translation, we provide the following definition for showing that a contract is fulfilled by a local inductive invariant.

Definition 4.9. Let α and γ be non-temporal SFL formulas. We define $I \vdash \langle\alpha\rangle C \langle\gamma\rangle$ to be fulfilled if and only if $I \vdash \langle\!\langle \mathcal{R}(\alpha) \rangle\!\rangle \mathcal{B}(\Box\alpha) \parallel C \parallel \mathcal{B}(\Box\gamma) \langle\!\langle \mathcal{R}(\gamma) \rangle\!\rangle$. Note that, for any non-temporal SFL formula ϕ, $\mathcal{B}(\Box\phi) = \mathcal{B}(\phi)$, and hence we will usually write $\mathcal{B}(\phi)$ below instead of $\mathcal{B}(\Box\phi)$ for brevity.

The contract from the example above can be proved using the invariant I_{TM} from Fig. 7. In other words, we have $I_{TM} \vdash \langle \text{TRUE} \rangle\, ToyTM \langle \rho_1 \wedge \rho_2 \rangle$. The following theorem that shows that the local inductive invariant method is sufficient for showing that action-based contracts are fulfilled; we provide proof in Appendix E.

Theorem 4.10. *$I \vdash \langle\alpha\rangle C \langle\gamma\rangle$ implies $\vdash \langle\alpha\rangle C \langle\gamma\rangle$*

$$\textsc{sfl-comp}\ \frac{I_1 \vdash \langle\alpha\rangle C_1\langle\rho\rangle \qquad I_2 \vdash \langle\rho\rangle C_2\langle\gamma\rangle}{I_1 \wedge I_2 \vdash \langle\alpha\rangle C_1 \parallel \mathcal{B}(\rho) \parallel C_2\langle\gamma\rangle} \qquad \textsc{sfl-safe}\ \frac{I_1 \vdash \langle\alpha\rangle C_1\langle\rho\rangle \qquad I_2 \vdash \langle\rho\rangle C_2\langle\gamma\rangle}{\vdash \langle\alpha\rangle C_1 \parallel C_2\langle\gamma\rangle}$$

Fig. 12. Assume-guarantee inference rules for transitive composition with an SFL bridge formula ρ.

Composing contracts and local inductive invariants. We now present inference rules for composing action-based contracts based on transitivity of action invariance. The idea is that composition depends on finding an action invariant ρ–a bridge formula–that acts as the assumption for one contract and the guarantee for the other.

Based on this intuition, we present the inference rule SFL-COMP in Fig. 12 that composes contracts and their associated local inductive invariants. Notice that, in the conclusion of SFL-COMP, we include $\mathcal{B}(\rho)$ to ensure that $I_1 \wedge I_2$ is well-defined, given that I_1 and I_2 may reference the (auxiliary) variables in $\mathcal{B}(\rho)$. We also point out that, by Def. 3.1, the alphabet of a bridge formula must be contained in the shared alphabet of the two components whose contracts are being composed. Because the actions of a bridge formula are shared between the two components, the action-based inference rules avoid an intermediary such as a bridge component. Furthermore, the inference rule SFL-SAFE in Fig. 12 shows that SFL-COMP is sound, in the sense that it proves the composition of the components to be safe. We prove that the rules SFL-COMP and SFL-SAFE are sound in Appendix F.

4.4 Hybrid Assume-Guarantee Contracts

In the prior section (Sec. 4.3), we introduced an action-based assume-guarantee theory. However, this theory is not sufficient for verifying that specifications satisfy state-based properties. For example, using action-based contracts alone, one cannot show that the Toy-2PC protocol satisfies the state invariant $\Box\mathit{Consistent}$ because action-based guarantees are action invariants.

Consequently, in this section, we introduce *hybrid* contracts, in which the assumption is an action invariant and the guarantee is a state invariant. We will define contracts and the local inductive invariant proof method in this theory, after which we will provide inference rules for composition. We will show that, with the addition of hybrid contracts, our action-based theory can verify state-based properties of systems.

Hybrid contracts and local inductive invariants. We will use the notation $\langle\alpha\rangle C\langle G\rangle$ for hybrid contracts, where G is a state-based guarantee that C must satisfy under the action-based assumption α. We use the same notation for action-based and hybrid contracts for uniformity, since we will use both types of contracts for verifying a single system. The meaning of a hybrid contract is that the component C, when restricted to the assumption α, must always satisfy the guarantee G. Similarly to Sec. 4.3, we will restrict the component to the assumption algebraically, by taking the parallel composition of C and $\mathcal{T}(\Box\alpha)$. However, we require the guarantee to hold in a state-based fashion. We now define hybrid contracts from this intuition.

Definition 4.11. Let $C = (\mathit{vars}, \mathit{Init}, \mathit{Next})^p$ be a PSTS. Also, let α be a non-temporal SFL formula and G be a non-temporal FOL formula such that such that $\mathit{Params}(\alpha) \cup \mathit{Params}(G) \subseteq p$, $\mathit{Act}\ \alpha \subseteq \mathit{Act}\ C$, and $SV(G) \subseteq \mathit{vars}$. A hybrid contract $\langle\alpha\rangle C\langle G\rangle$ is *fulfilled* if and only if $\mathcal{T}(\Box\alpha) \parallel C \models \Box G$. We will denote contract fulfillment with the notation $\vdash \langle\alpha\rangle C\langle G\rangle$.

For example, $\langle\rho_1 \wedge \rho_2\rangle \mathit{ToyRM}\langle\mathit{Consistent}\rangle$ is a hybrid contract. Proving fulfillment requires the inductive invariant method; much like in the action-based theory, we translate the hybrid contract to a state-based contract in order to use the local inductive invariant method from Sec. 3.1. For a hybrid contract $\langle\alpha\rangle C\langle G\rangle$, the corresponding state-based contract is $\langle\!\langle\mathcal{R}(\alpha)\rangle\!\rangle\mathcal{B}(\Box\alpha) \parallel C\langle\!\langle G\rangle\!\rangle$.

$$\textsc{hybrid-comp}\ \frac{I_1 \vdash \langle\alpha\rangle\, C_1 \langle\rho\rangle \qquad I_2 \vdash \langle\rho\rangle\, C_2 \langle G\rangle}{I_1 \wedge I_2 \vdash \langle\alpha\rangle\, C_1 \parallel \mathcal{B}(\rho) \parallel C_2 \langle G\rangle} \qquad \textsc{hybrid-safe}\ \frac{I_1 \vdash \langle\alpha\rangle\, C_1 \langle\rho\rangle \qquad I_2 \vdash \langle\rho\rangle\, C_2 \langle G\rangle}{\vdash \langle\alpha\rangle\, C_1 \parallel C_2 \langle G\rangle}$$

Fig. 13. Assume-guarantee inference rules for transitive composition with an SFL bridge formula ρ.

This translation is similar to the action-based contract translation, except the state-based formula G does not need to be translated. Based on this translation, we provide the following definition for showing that a contract is fulfilled by a local inductive invariant.

Definition 4.12. Let α be a non-temporal SFL formula and let G be a non-temporal FOL formula. We define $I \vdash \langle\alpha\rangle\, C \langle G\rangle$ to be fulfilled if and only if $I \vdash \langle\!\langle \mathcal{R}(\alpha) \rangle\!\rangle \mathcal{B}(\Box\alpha) \parallel C \langle\!\langle G \rangle\!\rangle$. Similarly to Sec. 4.3, we will prefer to write $\mathcal{B}(\phi)$ instead of $\mathcal{B}(\Box\phi)$ since the two are equivalent.

The contract from above can be proved using the invariant I_{RM} from Fig. 7. In other words, we have $I_{RM} \vdash \langle\rho_1 \wedge \rho_2\rangle\, ToyRM \langle Consistent\rangle$. The following theorem that shows that the local inductive invariant method is sufficient to show that a hybrid contract is fulfilled; we provide proof in Appendix G.

Theorem 4.13. *$I \vdash \langle\alpha\rangle\, C \langle G\rangle$ implies $\vdash \langle\alpha\rangle\, C \langle G\rangle$*

Composing contracts and local inductive invariants. Hybrid contracts are intended to be composed with action-based contracts. Therefore, we present the inference rule hybrid-comp in Fig. 13 that composes an action-based contract with a hybrid contract, as well as their associated local inductive invariants. Similarly to Sec. 4.3, we include $\mathcal{B}(\rho)$ in the conclusion of the rule hybrid-comp to ensure that the local invariant is well-defined. The inference rule hybrid-safe in Fig. 13 additionally shows that hybrid-comp is sound, in the sense that it proves the composition of the components to be safe. We prove that the two inference rules hybrid-comp and hybrid-safe are sound in Appendix H.

We now return to the Toy-2PC example. Recall from above $I_{RM} \vdash \langle\rho_1 \wedge \rho_2\rangle\, ToyRM \langle Consistent\rangle$, and also from Sec. 4.3 that $I_{TM} \vdash \langle \text{TRUE}\rangle\, ToyTM \langle\rho_1 \wedge \rho_2\rangle$. By the hybrid-safe rule, we can conclude that $\vdash \langle\text{TRUE}\rangle\, ToyTM \parallel ToyRM \langle Consistent\rangle$, i.e. that the entire system is safe. Furthermore, the hybrid-comp rule shows that $I_{TM} \wedge I_{RM}$ is an inductive invariant *for the entire system.*

5 Compositional Inductive Invariant Inference

Using the two-layered assume-guarantee theory from the prior two sections, we now present a framework for compositional inductive invariant inference. The problem statement is as follows: given a system S and an invariant $\Box P$, find an inductive invariant that proves $S \models \Box P$. Our compositional inference framework solves this problem with the following four steps: (1) system decomposition, (2) bridge formula inference, (3) local inductive invariant inference, and (4) global safety. The visual in Fig. 1 shows the four steps at a high level; here, we provide details for each step, including examples using Toy-2PC.

(1) System decomposition. In this step, we decompose S into components $C_1, \ldots, C_n$ such that $S = C_1 \parallel \cdots \parallel C_n$. For example, in Toy-2PC, we decompose $Toy2PC$ into two components ($n = 2$), $ToyRM$ and $ToyTM$ from Fig. 4.

(2) Bridge formula inference. The goal of this step is to find formulas $\phi_1, \ldots, \phi_{n+1}$ to create candidate contracts $\langle\phi_i\rangle\, C_i \langle\phi_{i+1}\rangle$. The formulas $\phi_1, \ldots, \phi_n$ are action invariants with the requirement that $\phi_1 = \text{TRUE}$. The final formula ϕ_{n+1} is required to be the state-based safety property, i.e. $\phi_{n+1} = P$.

Consequently, the first $n-1$ contracts are action-based, while the final contract is hybrid. In the Toy-2PC example, $\phi_1 = \textsc{true}$, $\phi_2 = \rho_1 \land \rho_2$, and $\phi_3 = \mathit{Consistent}$, making the candidate contracts $\langle\textsc{true}\rangle\, TM\, \langle\rho_1 \land \rho_2\rangle$ and $\langle\rho_1 \land \rho_2\rangle RM \langle \mathit{Consistent}\rangle$.

(3) Local inductive invariant inference. For each candidate contract $\langle\phi_i\rangle\, C_i\, \langle\phi_{i+1}\rangle$, the goal is to infer a local inductive invariant I_i such that $I_i \vdash \langle\phi_i\rangle\, C_i\, \langle\phi_{i+1}\rangle$. By Def. 4.9, the first $n-1$ contracts can be translated to the state-based contract $I_i \vdash \langle\!\langle \mathcal{R}(\phi_i)\rangle\!\rangle \mathcal{B}(\phi_i) \parallel C_i \parallel \mathcal{B}(\phi_{i+1})\langle\!\langle \mathcal{R}(\phi_{i+1})\rangle\!\rangle$. By Def. 4.12, we can also translate the final contract to the hybrid contract $I_i \vdash \langle\!\langle \mathcal{R}(\phi_i)\rangle\!\rangle \mathcal{B}(\phi_i) \parallel C_i \langle\!\langle \phi_{i+1}\rangle\!\rangle$. In both cases, we obtain a state-based contract that entails a local inductive invariant inference problem for finding I_i.

We describe how to solve the local inductive invariant inference problem for the first $n-1$ contracts; the solution for the final hybrid contract is similar. Suppose that $\mathcal{B}(\phi_i) \parallel C_i \parallel \mathcal{B}(\phi_{i+1}) = (\mathit{vars}, \mathit{Init}, \mathit{Next})^p$, then we construct $D_i = (\mathit{vars}, \mathit{Init} \land \mathcal{R}(\phi_i), \mathit{Next} \land \mathcal{R}(\phi_i)')^p$. By Def. 3.1, any inductive invariant that proves $D_i \models \Box\mathcal{R}(\phi_{i+1})$ also serves as a local inductive invariant I_i for the corresponding contract. However, verifying $D_i \models \Box\mathcal{R}(\phi_{i+1})$ reduces to a global inductive invariant inference problem, which can be solved using off-the-shelf tools.

For Toy-2PC, the state-based proof obligations are to find I_1 and I_2 such that $I_1 \vdash \langle\!\langle\textsc{true}\rangle\!\rangle\, \mathit{ToyTM} \parallel \mathit{ToyB} \langle\!\langle \mathit{ToyR}\rangle\!\rangle$ and $I_2 \vdash \langle\!\langle \mathit{ToyR}\rangle\!\rangle\, \mathit{ToyB} \parallel \mathit{ToyRM} \langle\!\langle \mathit{Consistent}\rangle\!\rangle$. The proof obligation for the first contract reduces to finding an inductive invariant I_1 for $\mathit{ToyTM} \parallel \mathit{ToyB} \models \Box\, \mathit{ToyR}$, while the proof obligation for the second contract reduces to finding an inductive invariant I_2 for $\mathcal{T}(\Box(\rho_1 \land \rho_2)) \parallel \mathit{ToyRM} \models \Box\, \mathit{Consistent}$. In this case, $I_1 = I_{TM}$ and $I_2 = I_{RM}$ are sufficient choices.

We also point out that local inductive invariant inference can be parallelized because each contract constitutes an independent proof obligation. We will utilize this fact during our evaluation in Sec. 6.

(4) Global safety. Let I be the conjunction of each local inductive invariant, i.e. $I = I_1 \land \cdots \land I_n$. Also, let B be the parallel composition of all bridge components, i.e. $B = \mathcal{B}(\phi_1) \parallel \cdots \parallel \mathcal{B}(\phi_n)$. Then, by the rules SFL-COMP and HYBRID-COMP, we can infer that $I \vdash \langle\textsc{true}\rangle S \parallel B \langle P\rangle$. Furthermore, by the rules SFL-SAFE and HYBRID-SAFE, we can infer that $\vdash \langle\textsc{true}\rangle S \langle P\rangle$. In other words, the entire system is safe and I is an inductive invariant for the entirety of S. In the Toy-2PC example, $I_{TM} \land I_{RM}$ is an inductive invariant I for the entire protocol.

6 Evaluation

We evaluate our inductive invariant inference method by applying it to two case studies of distributed protocols, both of which are specified in TLA$^+$. We compare the compositional approach to the global approach using Endive [47] as a baseline. Endive is a state-of-the-art tool for automatic inductive invariant inference for TLA$^+$ specifications. In addition, we use Endive as our off-the-shelf tool for local inductive invariant inference, as described in our compositional framework (Sec. 5).

The first case study is the full version of the Two Phase Commit protocol [17], in which we compare automatic inductive invariant inference using both the global and compositional methods. The second case study is an industrial-scale protocol of a Raft [40] style algorithm that runs at MongoDB [36]. To the best of our knowledge, this case study is beyond the reach of any existing algorithm for automatic inductive invariant inference, including Endive. However, with the compositional approach, we infer an inductive invariant for the protocol semi-automatically. Between the two case studies, we provide evidence that, in comparison to the global approach, the compositional approach can be more efficient for inductive invariant inference and results in more concise proofs. Additionally, we show that the artifacts from compositional inference, including

$$\begin{aligned}\rho_r \triangleq\ & \land\ (\exists rm \in RMs : onceRcvCommit(rm)) \implies (\forall rm \in RMs : onceSndPrepare(rm))\\ & \land\ (\exists rm \in RMs : onceRcvAbort(rm)) \implies (\forall rm \in RMs : \neg onceRcvCommit(rm))\end{aligned}$$

$$\begin{aligned}\rho_e \triangleq\ & \land\ (\exists rm \in RMs : onceSndCommit(rm)) \implies (\forall rm \in RMs : onceRcvPrepare(rm))\\ & \land\ (\exists rm \in RMs : onceSndAbort(rm)) \implies (\forall rm \in RMs : \neg onceSndCommit(rm))\end{aligned}$$

Fig. 14. Definitions for the bridge formulas in the 2PC protocol.

$$\begin{aligned}J_g \triangleq\ & \forall rm \in RM : (tmState = \text{“}init\text{”} \land [type \mapsto \text{“}Prepared\text{”}, theRM \mapsto rm] \in msgs) \implies\\ & \quad (rmState[rm] = \text{“}prepared\text{”})\\ J_l \triangleq\ & \forall rm \in RMs : onceSndPrepare[rm] \implies rmState[rm] \neq \text{“}working\text{”}\end{aligned}$$

Fig. 15. J_g is a conjunct of the global inductive invariant for 2PC while J_l is a conjunct of a local inductive invariant for 2PC. Both formulas specify that resource managers cannot abort after sending a prepare message.

bridge formulas and local inductive invariants, provide insights into the behavior of the protocols beyond those of the global approach.

We manually decomposed all specifications in our case studies; automatic specification decomposition is well-studied [6, 9, 34, 37] and beyond the scope of this paper. We also manually created all bridge formulas in the case studies. Bridge formula inference is a nontrivial task whose automation is a significant research problem on its own and is beyond the scope of this paper. We also include a machine checked proof for each inductive invariant written in the TLA$^+$ Proof System (TLAPS) [8] proof language. All experiments and proofs were run on MacOS version 14.7.6 with an M1 Pro chip, and are available in our supplementary materials.

6.1 Case Study 1: Two Phase Commit

In this case study, we verify the (full) Two Phase Commit (2PC) protocol [17]. We compare automatic inductive invariant inference between our compositional framework and the global approach.

Protocol description. The 2PC protocol allows messages to be delayed, dropped, and reordered, while Toy-2PC assumes a perfect network. Rather than single $Prepared$, $Commit$ and $Abort$ actions, this protocol models sending and receiving each message over a network. Therefore, 2PC includes the actions $SndPrepared$ and $RcvPrepared$, etc. The protocol also includes an additional state variable $msgs$ that records all messages that have been sent across the network; message delay, dropping, and reordering are modeled as a receiver ignoring a message for some period of time (e.g., indefinitely for a dropped message).

Inductive invariant inference. For global inference, we use the inductive invariant and the proof reported in the Endive paper [47]. For compositional inference, we use the framework presented in Sec. 5. We outline the four steps of the framework below.

Step (1) *system decomposition.* We decompose the system into three components RM, Env, and TM. The components respectively represent the resource managers, the environment, and the transaction manager.

Step (2) *bridge formula inference.* The candidate contracts are $\langle \text{TRUE} \rangle TM \langle \rho_e \rangle$, $\langle \rho_e \rangle Env \langle \rho_r \rangle$, and $\langle \rho_r \rangle RM \langle Consistent \rangle$, where the bridge formulas ρ_r and ρ_e are shown in Fig. 14. The fluents in

Table 1. Comparison between global and compositional inductive invariant inference for 2PC. All times from Endive are reported in seconds.

Specification	Endive Time	Proof Size	Inv. Size
$TwoPhase$ (global)	34.82	107	10
RM	15.44	8	5
Env	30.43	86	8
TM	6.13	8	5

ρ_r and ρ_e follow the *once* pattern, e.g., from Fig. 8d; we therefore omit the explicit definition of the fluents for brevity. Intuitively, ρ_r specifies the acceptable behavior of the network, while ρ_e specifies the acceptable behavior of the transaction manager. The network requirement ρ_r is conceptually the same as $\rho_1 \wedge \rho_2$ (Fig. 8f) from the Toy-2PC protocol, except ρ_r describes the requirement in terms of messages that the resource managers can receive from the network. Likewise, the transaction manager requirement ρ_e is conceptually the same as $\rho_1 \wedge \rho_2$, except ρ_e describes messages that the transaction manager may send over the network.

Step (3) *local inductive invariant inference.* We first translate each of the three contracts to their state-based counterpart. After, we use Endive to infer a local inductive invariant. We include a TLAPS proof for each inductive invariant that Endive returned.

Step (4) *global safety.* By proof rules SFL-COMP and HYBRID-COMP, the conjunction of the three local inductive invariants is an inductive invariant for the entire 2PC protocol. Furthermore, we infer that the protocol is safe by rules SFL-SAFE and HYBRID-SAFE.

Results. We summarize our results in Table. 1. For each specification, we include the run-time for Endive in seconds (Endive Time), the size of the corresponding proof (Proof Size), and the number of conjuncts in the inductive invariant (Inv. Size) as a rough measure for its size. When reporting the size of the proof, we count only the number of proof steps to avoid counting irrelevant lines that result due to formatting.

Discussion. **Efficiency of invariant inference.** In terms of run-time for automatic inductive invariant inference, the global approach is the least efficient at 34.82 seconds. By parallelizing local inductive invariant inference, as recommended in Sec. 5, compositional inference is performed more efficiently (30.43 seconds) than the global approach. This comparison provides evidence that compositional inference can be more efficient than the global approach.

Proof conciseness. Table. 1 shows that the proof sizes for the local inductive invariants, both individually and combined, are smaller than the proof for the global invariant. These results provide evidence that the proofs for local inductive invariants may be more concise than the proofs for global invariants. We also point out that the number of conjuncts in each local inductive invariant is smaller than the number in the global invariant; however, we consider the number of conjuncts to be a rough, secondary measure for evaluating the conciseness of the proof for an inductive invariant.

Insights from compositional inference. In addition to their value for verification, inductive invariants are also valuable for the insights they provide about a system. The artifacts from compositional inference–bridge formulas and local inductive invariants–also provide insights about systems, some of which are not found in global inductive invariants. For example, the two conjuncts of both bridge formulas ρ_r and ρ_e offer a concise modular specification of the two phases of the 2PC protocol: the first conjuncts specify that all resource managers attempt to prepare in the first phase before a commit, while the second conjuncts specify that the transaction manager chooses to either commit or abort in the second phase. However, no single part of the global inductive invariant offers this concise insight.

Local invariants offer concise local insights. Furthermore, we observe that local insights tend to be encoded more concisely in local invariants in comparison with global invariants. For example, a key requirement for the 2PC protocol is that once each resource manager sends a prepare message, it can no longer silently abort. The global inductive invariant and the local inductive invariant for RM both encode this requirement differently; we show the relevant conjunct from each invariant in Fig. 15. Notice that the global invariant refers to the state variables from all three components of the system (RM, Env, and TM), while the local invariant refers only to the state variables and actions for the relevant component (RM). By referring to only the relevant component, the local inductive invariant reduces the scope of the protocol that is needed to understand the key requirement.

6.2 Case Study 2: Mongo Static Raft

In this case study, we verify Mongo Static Raft (MSR) [48], an industrial-scale protocol based on Raft [40] that runs at MongoDB [36]. We describe and compare our effort to infer an inductive invariant for MSR using both the global and compositional approaches.

Protocol description. MSR is a distributed consensus protocol that uses leader elections to maintain a consistent replicated state machine. The protocol begins with an election, where a server within a cluster is decided to be the *primary*. Each server updates its metadata with the primary server and the *term*–a natural number that increases with each election–for which the primary will serve. The primary server accepts client requests and tracks them in a local data structure called a *log*. The servers in the cluster replicate the contents of their log in gossip style to the other nodes in the cluster. Once a majority of servers add a client request to their log, the request is considered to be *committed* and each server adds it to their state machine.

The safety property for the protocol is state machine safety (SMS), which we show formally in Fig. 16d. This property specifies that, for any two commits c_1 and c_2, if they share the same index ($c_1.ind = c_2.ind$), then the two commits must be identical ($c_1 = c_2$). In general, a commit c records the index in the log ($c.ind$) and the term in which the client request occurred ($c.term$). The MSR specification uses the term ($c.term$) as an abstract representation for the contents of a client request.

In the MSR protocol, the network can drop or reorder messages and can also cause the servers to be partitioned. However, servers are expected to act in a failstop manner, meaning that failed servers do not produce faulty or corrupt messages. When a leader fails or if the network is partitioned, the protocol will elect a new primary in a new term. All servers have the ability to "catch up" other servers in a lower term by rolling back extraneous log entries and adding correct ones. Additionally, the protocol derives its name from the assumption that its configuration–the set of nodes in the cluster–is static.

In the TLA$^+$ encoding of the protocol [48], the set of servers is given as a parameter. The specification includes the following state variables: $committed$, log, $state$, and $currentTerm$, which represent the set of commits, the log, whether each server is a primary, and the current election term. The variable $committed$ keeps track of the set of all commits throughout the protocol, while the remaining three variables represent data structures that are local to each server.

Inductive invariant inference. We attempted to use Endive to automatically infer a global inductive invariant for MSR. However, Endive exits with failure for global inference because it could not find predicates to eliminate all counter examples to induction (CTIs). Therefore, we manually constructed a global inductive invariant for MSR, which we proved with TLAPS.

MODULE $commitAtTermIndT$

$Init \triangleq v_1 = [x_0 \in \mathbb{N} \mapsto [x_1 \in \mathbb{N} \mapsto \text{FALSE}]]$

$CommitEntry(s, ind, term) \triangleq$

$\quad v_1' = [v_1 \text{ EXCEPT } ![term][ind] = \text{TRUE}]$

(a) Fluent definition for *commitAtTermInd*.

MODULE $leaderAtTermServT$

$Init \triangleq v_2 = [x_0 \in \mathbb{N} \mapsto [x_1 \in Server \mapsto \text{FALSE}]]$

$BecomeLeader(s, term) \triangleq$

$\quad v_2' = [v_2 \text{ EXCEPT } ![term][s] = \text{TRUE}]$

(b) Fluent definition for *leaderAtTermServ*.

MODULE $currentLeaderT$

$Init \triangleq v_3 = [x_0 \in Server \mapsto [x_1 \in \mathbb{N} \mapsto \text{FALSE}]]$

$GetEntries(s_1, s_2) \triangleq v_3' = [v_3 \text{ EXCEPT } ![s_1] = [x_0 \in \mathbb{N} \mapsto \text{FALSE}]]$

$RollbackEntries(s_1, s_2) \triangleq v_3' = [v_3 \text{ EXCEPT } ![s_1] = [x_0 \in \mathbb{N} \mapsto \text{FALSE}]]$

$BecomeLeader(s, term) \triangleq v_3' = [[v_3 \text{ EXCEPT } ![s] = [x_0 \in \mathbb{N} \mapsto \text{FALSE}]] \text{ EXCEPT } ![s][term] = \text{TRUE}]$

(c) Fluent definition for *currentLeader*.

$$StateMachineSafety \triangleq \forall c_1, c_2 \in committed : (c_1.ind = c_2.ind) \implies c_1 = c_2$$

(d) The key safety property $StateMachineSafety$, or SMS for short.

$$\rho_c \triangleq \forall i \in \mathbb{N} : \exists t_1 \in \mathbb{N} : \forall t_2 \in \mathbb{N} : commitAtTermInd(t_2, i) \implies (t_1 = t_2)$$

$$\rho_l^* \triangleq \forall t \in \mathbb{N} : \forall s_1, s_2 \in Server : \\ (leaderAtTermServ(t, s_1) \land leaderAtTermServ(t, s_2)) \implies (s_1 = s_2)$$

(e) ρ_c is the bridge formula for the $Committed$ and Log components. ρ_l^* is one conjunct of ρ_l, the bridge formula for the Log and $StateTerm$ components.

Fig. 16. Definitions for the safety property and bridge formulas in the Mongo Static Raft protocol.

With our compositional framework, we inferred an inductive invariant for MSR semi-automatically. We also used Endive in attempt to automatically infer each local inductive invariant. Endive completed successfully for one component and failed for two components, where the failures were also because the tool could not eliminate all CTIs. We now describe our compositional inference effort in detail for the four steps from Sec. 5.

Step (1) *system decomposition.* We decompose MSR into three components $Committed$, Log, and $StateTerm$. The components respectively represent the set of commits, the log for each server, and the state and term metadata for each server.

Step (2) *bridge formula inference.* The candidate contracts are $\langle \text{TRUE} \rangle StateLog \langle \rho_l \rangle$, $\langle \rho_l \rangle Log \langle \rho_c \rangle$, and $\langle \rho_c \rangle Committed \langle SMS \rangle$. In Fig. 16e, we show the bridge formula ρ_c as well as one exemplar conjunct ρ_l^* for the bridge formula ρ_l. We show the entire bridge formula ρ_l in Appendix I, Fig. 19. We show the fluent definitions (transition systems only) for *commitAtTermInd*, *leaderAtTermServ*, and *currentLeader* in Figures 16a, 16b, and 16c respectively. The definitions for the remaining fluents are included in Appendix I.

The fluents *commitAtTermInd* and *leaderAtTermServ* use the *once* style as seen in the 2PC and Toy-2PC protocols. These three fluents respectively represent whether a commit has happened at a specific term and index, and whether a server has been elected leader at a specific term. The fluent *currentLeader*, however, represents servers who believe themselves to be a leader at a specific term;

Table 2. Comparison between global and compositional inductive invariant inference for Mongo Static Raft.

Specification	**Endive Time**	**Proof Size**	**Inv. Size**
MSR (global)	NA	1,686	17
$Committed$	19.80	61	2
Log	NA	123	9
$StateTerm$	NA	188	7

this may be more than one server in the case of a network partition. In the definition for this fluent (Fig. 16c), a server believes itself to be a leader upon a *BecomeLeader* action, but no longer once its log is updated via a *GetEntries* or *RollbackEntries* action.

Step (3) *local inductive invariant inference.* For each contract, we attempted to automatically infer a local inductive invariant with Endive. Endive failed for the Log and $StateTerm$ components, but succeeded for the $Committed$ component. For the two cases that Endive failed, we manually constructed a local inductive invariant. We also include a TLAPS proof for all local inductive invariants.

Step (4) *global safety.* By proof rules SFL-COMP and HYBRID-COMP, the conjunction of the three local inductive invariants is an inductive invariant for the entire MSR protocol. Furthermore, we infer that the protocol is safe by rules SFL-SAFE and HYBRID-SAFE.

Results. We summarize our results for each specification in Table. 2. We include the run-time for Endive in seconds (Endive Time), unless the tool failed in which case we write NA. We also report the size of the proof for the inductive invariant (Proof Size) and the number of conjuncts in the invariant (Inv. Size). Similarly to the 2PC case study (Sec. 6.1), we count only the number of proof steps when we report the proof size.

Discussion. **Efficiency of invariant inference.** Table. 2 shows that Endive fails for the global inductive invariant inference task. In the compositional approach, Endive fails for the *Log* and *StateTerm* components so we manually created local inductive invariants for these two components. However, Endive completed successfully for the *Committed* component, which is smaller and less complex than the other two components. Ultimately, we inferred an inductive invariant semi-automatically with the compositional approach, which provides evidence that compositional inductive invariant inference can be more efficient than global inference. This result also suggests that the compositional approach may benefit from further research into system decomposition, specifically for finding simpler components.

Proof conciseness. Table. 2 shows that the length of the TLAPS proof for each individual component is shorter than the proof for the global invariant by an order of magnitude. This result provides evidence that the proof for local inductive invariants may be shorter than the proofs for global inductive invariants. We also point out that the local inductive invariants have fewer conjuncts, but offer this observation as a rough, secondary metric.

Insights from compositional inference. Much like the 2PC case study, we found that the artifacts from compositional inference provided valuable insights about the case study that are not captured by the global inductive invariant. For example, consider ρ_l^* in Fig. 16e, which is a conjunct of the bridge formula ρ_l. The formula ρ_l^* represents the *one leader per term* property, which is a key lemma at the heart of the correctness argument for Raft [40]. While this property does appear in the global inductive invariant, the fact that it appears in the bridge formula ρ_l offers the more specific insight that *one leader per term* is guaranteed by the $StateTerm$ component and assumed by the

$$
\begin{aligned}
K_g \triangleq\ & \forall s \in Server : (state[s] = Secondary \land LastTerm(log[s]) = currentTerm[s]) \implies \\
&\quad \lor\ \exists p \in Server : \\
&\qquad \land\ state[p] = Primary \\
&\qquad \land\ currentTerm[p] = currentTerm[s] \\
&\qquad \land\ LastTerm(log[p]) \geq LastTerm(log[s]) \\
&\qquad \land\ Len(log[p]) \geq Len(log[s]) \\
&\quad \lor\ \exists p \in Server : \\
&\qquad \land\ state[p] = Primary \\
&\qquad \land\ currentTerm[p] > currentTerm[s] \\
&\quad \lor\ \forall t \in Server : state[t] = Secondary \\
K_l \triangleq\ & \forall t_1, t_2, i \in \mathbb{N} : \forall s \in Server : \\
&\quad (commitAtTermInd[t_1][i] \land currentLeader[s][t_2] \land t_1 \leq t_2) \implies log[s][i] = t_1
\end{aligned}
$$

Fig. 17. K_g is a conjunct from the global invariant and K_l is a conjunct from the local invariant for *Log*. Both formulas specify when an entry must be present in the log.

Log component. This insight could provide the basis for modular optimizations to the protocol, e.g., using a new leader election scheme in the *StateTerm* component that maintains the property.

The presence of important properties such as *one leader per term* in bridge formulas may also be a reason that the local inductive invariant proofs are more concise than the global proof. Using the *Log* component as an example, the intuition for this reasoning is that local inductive invariant inference becomes simpler when it can assume key lemmas such as *one leader per term*, rather than needing to infer them.

Simpler constraints in local invariants. We observe that local inductive invariants can specify simpler constraints when compared with the global invariant. For example, consider the conjunct K_g from the global invariant in Fig. 17. This conjunct specifies an intricate condition for when an entry must appear in the logs of a secondary (non-primary) server. The purpose of this conjunct is to help establish leader completeness properties in the induction proof. However, the compositional approach avoids tying the log to leader completeness properties, which results in a simpler constraint than K_g. The fact that the leader completeness properties do not depend on the log is explicit in the compositional approach because the *Log* component assumes all its leader completeness properties from the bridge formula ρ_l. As a result, the local inductive invariant for *Log* specifies simpler constraints on the log. We show the lone conjunct K_l that constrains the log in Fig. 17.

7 Related Work

Many techniques exist for automatically inferring inductive invariants. Approaches include interpolation [32], ICE learning [11], syntax-driven enumeration of invariants [19, 51, 50], and incremental lemma learning [3, 15, 24, 25, 47]. Theoretical work based on the *Hoare query model* has shown that the incremental approach–specifically with relative induction checks–can learn inductive invariants exponentially faster than with induction checks alone [10]. A more recent approach, based on the observation that counter examples to induction (CTIs) help inductive invariant inference algorithms to find new predicates and vice versa (duality), attempts to balance the search for

CTIs and predicates to achieve progress theorems [44]. While the above techniques have certain advantages, each one experiences scalability issues by attempting to infer an inductive invariant for the transition relation of an entire system. In contrast, our compositional framework divides the transition relation to improve the efficiency of inductive invariant inference. However, the techniques above are complementary to our compositional framework because they can be used for local inductive invariant inference.

Due to the limitations of fully automated inductive invariant inference, frameworks have been proposed for assisting users in finding an inductive invariant. Ivy [43] and similar languages [49] are popular frameworks that encourage a user to write specifications in a decidable fragment of FOL. Kondo [54] is also a tool that requires user assistance, but is specifically designed to strike a balance between automation and manual proof effort. Lamport describes a technique for finding inductive invariants for TLA$^+$ [28] in which a person manually finds CTIs using the TLC model checker [52]. Each of these techniques also runs into scalability issues, both from to the manual efforts involved and the fact that a user attempts to find a global inductive invariant for the entire transition relation.

There is a large body of work for automating compositional verification for finite-state processes. Beginning with the seminal paper of Cobleigh et al. [7], researchers have attempted to find ways to learn and compute assumptions (which we call bridges), both explicitly [2, 5, 7, 18, 37] and implicitly (symbolically) [4, 20]. While these techniques are powerful, they do not, in general, apply to parameterized systems because they are inherently finite-state. Our framework allows assume-guarantee contracts to be proved with local inductive invariants, which allows us to compositionally verify parameterized systems.

The method of Owicki Gries [42], Rely-Guarantee Reasoning [22], and Concurrent Separation Logic [39] are each assume-guarantee instances for proving properties of programs; however, we are interested in verifying declarative specifications in this work.

Past Time Temporal Logic (PTL) [23] is a variant of temporal logic with operators that can specify propositions that happened in the past. For example, the *once* operator specifies that a proposition was true at least once in the past and can be seen as the past-time version of the *eventually* ($\Diamond$) temporal operator. The *once* operator is analogous to the *once* style fluents from the 2PC and Toy-2PC protocols. However, the expressiveness of PTL (and LTL) with actions is limited, which inspired the Fluent LTL specification language [13]. In our work, we introduce SFL that builds upon Fluent LTL by including quantifiers and symbolic fluents that accept arguments. Ultimately, the extensions in SFL make the language appropriate for specifying parameterized systems.

8 Limitations and Future Work

In this paper, we presented a compositional inductive invariant inference framework that is based on a two layer assume-guarantee theory. As part of our presentation, we showed that the inference rules for contract composition are sound. However, we leave a formal treatment of completeness for future work. We also plan to investigate whether the inference rules have the cut elimination property, which may reveal a technique for eliminating auxiliary variables from inductive invariants that are inferred compositionally.

We created all bridge formulas in this paper manually because automated inference of bridge formulas is a nontrivial task. In the future, we plan to create an algorithm for automatic inference of bridge formulas, specifically for the SFL language. We also plan to investigate whether SFL is useful for tasks besides compositional invariant inference. For example, we plan to explore whether we can compute the weakest assumption [12] of a system in SFL, which may be useful for robustness analysis [53].

In our evaluation (Sec. 6), we were able to use compositional inductive invariant inference to verify the Mongo Static Raft protocol semi-automatically. We automatically inferred a local invariant for the smallest component, which suggests that our technique would benefit from more granular decompositions. In the future, we plan to investigate techniques for both automated and more granular decompositions. Ultimately, we plan to fully automate our compositional verification framework in Sec. 5, in which decomposition, bridge formula inference, and local invariant inference are all automated.

Acknowledgments

This project was supported by a TLA$^+$ Foundation grant.

References

[1] Karam Abd Elkader, Orna Grumberg, Corina S. Păsăreanu, and Sharon Shoham. 2018. Automated circular assume-guarantee reasoning. *Form. Asp. Comput.*, 30, 5, (Sept. 2018), 571–595. DOI: 10.1007/s00165-017-0436-0.

[2] Rajeev Alur, P. Madhusudan, and Wonhong Nam. 2005. Symbolic compositional verification by learning assumptions. In *Computer Aided Verification*. Kousha Etessami and Sriram K. Rajamani, (Eds.) Springer Berlin Heidelberg, Berlin, Heidelberg, 548–562. ISBN: 978-3-540-31686-2.

[3] Aaron R. Bradley. 2011. Sat-based model checking without unrolling. In *Verification, Model Checking, and Abstract Interpretation*. Ranjit Jhala and David Schmidt, (Eds.) Springer Berlin Heidelberg, Berlin, Heidelberg, 70–87. ISBN: 978-3-642-18275-4.

[4] Yu-Fang Chen, Edmund M. Clarke, Azadeh Farzan, Ming-Hsien Tsai, Yih-Kuen Tsay, and Bow-Yaw Wang. 2010. Automated assume-guarantee reasoning through implicit learning. In *Proceedings of the 22nd International Conference on Computer Aided Verification* (CAV'10). Springer-Verlag, Edinburgh, UK, 511–526. ISBN: 364214294X. DOI: 10.1007/978-3-642-14295-6_44.

[5] Yu-Fang Chen, Azadeh Farzan, Edmund M. Clarke, Yih-Kuen Tsay, and Bow-Yaw Wang. 2009. Learning minimal separating dfa's for compositional verification. In *Tools and Algorithms for the Construction and Analysis of Systems*. Stefan Kowalewski and Anna Philippou, (Eds.) Springer Berlin Heidelberg, Berlin, Heidelberg, 31–45. ISBN: 978-3-642-00768-2.

[6] Jamieson M. Cobleigh, George S. Avrunin, and Lori A. Clarke. 2006. Breaking up is hard to do: an investigation of decomposition for assume-guarantee reasoning. In *Proceedings of the 2006 International Symposium on Software Testing and Analysis* (ISSTA '06). Association for Computing Machinery, Portland, Maine, USA, 97–108. ISBN: 1595932631. DOI: 10.1145/1146238.1146250.

[7] Jamieson M. Cobleigh, Dimitra Giannakopoulou, and Corina S. PĂsĂreanu. 2003. Learning assumptions for compositional verification. In *Tools and Algorithms for the Construction and Analysis of Systems*. Hubert Garavel and John Hatcliff, (Eds.) Springer Berlin Heidelberg, Berlin, Heidelberg, 331–346. ISBN: 978-3-540-36577-8.

[8] Denis Cousineau, Damien Doligez, Leslie Lamport, Stephan Merz, Daniel Ricketts, and Hernan Vanzetto. 2012. Tla+ proofs. *Proceedings of the 18th International Symposium on Formal Methods (FM 2012), Dimitra Giannakopoulou and Dominique Mery, editors. Springer-Verlag Lecture Notes in Computer Science*, 7436, (Jan. 2012), 147–154. https://www.microsoft.com/en-us/research/publication/tla-proofs/.

[9] Ian Dardik, April Porter, and Eunsuk Kang. 2024. Recomposition: a new technique for efficient compositional verification. In *2022 Formal Methods in Computer-Aided Design (FMCAD)*. TU Wien Academic Press, 130–141.

[10] Yotam M. Y. Feldman, Neil Immerman, Mooly Sagiv, and Sharon Shoham. 2019. Complexity and information in invariant inference. *Proc. ACM Program. Lang.*, 4, POPL, Article 5, (Dec. 2019), 29 pages. DOI: 10.1145/3371073.

[11] Pranav Garg, Christof Löding, P. Madhusudan, and Daniel Neider. 2014. Ice: a robust framework for learning invariants. In *Computer Aided Verification*. Armin Biere and Roderick Bloem, (Eds.) Springer International Publishing, Cham, 69–87. ISBN: 978-3-319-08867-9.

[12] D. Giannakopoulou, C.S. Pasareanu, and H. Barringer. 2002. Assumption generation for software component verification. In *Proceedings 17th IEEE International Conference on Automated Software Engineering*, 3–12. DOI: 10.1109/ASE.2002.1114984.

[13] Dimitra Giannakopoulou and Jeff Magee. 2003. Fluent model checking for event-based systems. *SIGSOFT Softw. Eng. Notes*, 28, 5, (Sept. 2003), 257–266. DOI: 10.1145/949952.940106.

[14] Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, (Eds.) 2018. *Compositional reasoning. Handbook of Model Checking*. Springer International Publishing, Cham, 345–383. ISBN: 978-3-319-10575-8. DOI: 10.1007/978-3-319-10575-8_12.

[15] Aman Goel and Karem Sakallah. 2021. On symmetry and quantification: a new approach to verify distributed protocols. In *NASA Formal Methods*. Aaron Dutle, Mariano M. Moscato, Laura Titolo, César A. Muñoz, and Ivan Perez, (Eds.) Springer International Publishing, Cham, 131–150. ISBN: 978-3-030-76384-8.

[16] Susanne Graf and Bernhard Steffen. 1990. Compositional minimization of finite state systems. In *Proceedings of the 2nd International Workshop on Computer Aided Verification* (CAV '90). Springer-Verlag, Berlin, Heidelberg, 186–196. ISBN: 3540544771.

[17] Jim Gray and Leslie Lamport. 2006. Consensus on transaction commit. *ACM Trans. Database Syst.*, 31, 1, (Mar. 2006), 133–160. DOI: 10.1145/1132863.1132867.

[18] Anubhav Gupta, Kenneth L. McMillan, and Zhaohui Fu. 2007. Automated assumption generation for compositional verification. In *Proceedings of the 19th International Conference on Computer Aided Verification* (CAV'07). Springer-Verlag, Berlin, Germany, 420–432. ISBN: 9783540733676.

[19] Travis Hance, Marijn J. H. Heule, Ruben Martins, and Bryan Parno. 2021. Finding invariants of distributed systems: it's a small (enough) world after all. In *Symposium on Networked Systems Design and Implementation*. https://api.semanticscholar.org/CorpusID:233733235.

[20] Fei He, Bow-Yaw Wang, Liangze Yin, and Lei Zhu. 2014. Symbolic assume-guarantee reasoning through bdd learning. In *Proceedings of the 36th International Conference on Software Engineering* (ICSE 2014). Association for Computing Machinery, Hyderabad, India, 1071–1082. ISBN: 9781450327565. DOI: 10.1145/2568225.2568253.

[21] C. A. R. Hoare. 1978. Communicating sequential processes. *Commun. ACM*, 21, 8, (Aug. 1978), 666–677. DOI: 10.1145/359576.359585.

[22] Cliff Jones. 1983. Specification and design of (parallel) programs. In vol. 83. (Jan. 1983), 321–332.

[23] Hans Kamp. 1968. *Tense Logic and the Theory of Linear Order*. Ph.D. Dissertation. Ucla.

[24] Jason R. Koenig, Oded Padon, Neil Immerman, and Alex Aiken. 2020. First-order quantified separators. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation* (PLDI 2020). Association for Computing Machinery, London, UK, 703–717. ISBN: 9781450376136. DOI: 10.1145/3385412.3386018.

[25] Jason R. Koenig, Oded Padon, Sharon Shoham, and Alex Aiken. 2022. Inferring invariants with quantifier alternations: taming the search space explosion. In *Tools and Algorithms for the Construction and Analysis of Systems*. Dana Fisman and Grigore Rosu, (Eds.) Springer International Publishing, Cham, 338–356. ISBN: 978-3-030-99524-9.

[26] Leslie Lamport. 2002. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, (June 2002). https://www.microsoft.com/en-us/research/publication/specifying-systems-the-tla-language-and-tools-for-hardware-and-software-engineers/.

[27] Leslie Lamport. 1998. The part-time parliament. *ACM Trans. Comput. Syst.*, 16, 2, (May 1998), 133–169. DOI: 10.1145/279227.279229.

[28] Leslie Lamport. 2018. Using tlc to check inductive invariance. In https://api.semanticscholar.org/CorpusID:53323076.

[29] Jeff Magee and Jeff Kramer. 2006. *Concurrency: State Models and Java Programs*. (2nd ed.). Wiley Publishing. ISBN: 0470093552.

[30] Patrick Donohoe, (Ed.) 1999. *Behaviour analysis of software architectures. Software Architecture: TC2 First Working IFIP Conference on Software Architecture (WICSA1) 22–24 February 1999, San Antonio, Texas, USA*. Springer US, Boston, MA, 35–49. ISBN: 978-0-387-35563-4. DOI: 10.1007/978-0-387-35563-4_3.

[31] Zohar Manna and Amir Pnueli. 1995. *Temporal verification of reactive systems: safety*. Springer-Verlag, Berlin, Heidelberg. ISBN: 0387944591.

[32] K. L. McMillan. 2003. Interpolation and sat-based model checking. In *Computer Aided Verification*. Warren A. Hunt and Fabio Somenzi, (Eds.) Springer Berlin Heidelberg, Berlin, Heidelberg, 1–13. ISBN: 978-3-540-45069-6.

[33] Kenneth L. McMillan. 1999. Circular compositional reasoning about liveness. In *Proceedings of the 10th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods* (CHARME '99). Springer-Verlag, Berlin, Heidelberg, 342–345. ISBN: 3540665595.

[34] Björn Metzler, Heike Wehrheim, and Daniel Wonisch. 2008. Decomposition for compositional verification. In *Formal Methods and Software Engineering*. Shaoying Liu, Tom Maibaum, and Keijiro Araki, (Eds.) Springer Berlin Heidelberg, Berlin, Heidelberg, 105–125. ISBN: 978-3-540-88194-0.

[35] Robin Milner, Joachim Parrow, and David Walker. 1992. A calculus of mobile processes, i. *Inf. Comput.*, 100, 1, (Sept. 1992), 1–40. DOI: 10.1016/0890-5401(92)90008-4.

[36] MongoDB. 2025. Mongodb github project. (2025). https://github.com/mongodb/mongo.

[37] Wonhong Nam, P. Madhusudan, and Rajeev Alur. 2008. Automatic symbolic compositional verification by learning assumptions. *Form. Methods Syst. Des.*, 32, 3, (June 2008), 207–234. DOI: 10.1007/s10703-008-0055-8.

[38] Kedar S. Namjoshi and Richard J. Trefler. 2010. On the completeness of compositional reasoning methods. *ACM Trans. Comput. Logic*, 11, 3, Article 16, (May 2010), 22 pages. DOI: 10.1145/1740582.1740584.

[39] Peter W. O'Hearn. 2007. Resources, concurrency, and local reasoning. *Theoretical Computer Science*, 375, 1, 271–307. Festschrift for John C. Reynolds's 70th birthday. DOI: https://doi.org/10.1016/j.tcs.2006.12.035.

[40] Diego Ongaro and John Ousterhout. 2014. In search of an understandable consensus algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference* (USENIX ATC'14). USENIX Association, Philadelphia, PA, 305–320. ISBN: 9781931971102.

[41] Susan Owicki. 1976. A consistent and complete deductive system for the verification of parallel programs. In *Proceedings of the Eighth Annual ACM Symposium on Theory of Computing* (STOC '76). Association for Computing Machinery, Hershey, Pennsylvania, USA, 73–86. ISBN: 9781450374149. DOI: 10.1145/800113.803634.

[42] Susan Owicki and David Gries. 1976. An axiomatic proof technique for parallel programs i. *Acta Informatica*, 6, 4, 319–340. ISBN: 1432-0525. DOI: 10.1007/BF00268134.

[43] Oded Padon, Kenneth L. McMillan, Aurojit Panda, Mooly Sagiv, and Sharon Shoham. 2016. Ivy: safety verification by interactive generalization. *SIGPLAN Not.*, 51, 6, (June 2016), 614–630. DOI: 10.1145/2980983.2908118.

[44] Oded Padon, James R. Wilcox, Jason R. Koenig, Kenneth L. McMillan, and Alex Aiken. 2022. Induction duality: primal-dual search for invariants. *Proc. ACM Program. Lang.*, 6, POPL, Article 50, (Jan. 2022), 29 pages. DOI: 10.1145/3498712.

[45] Amir Pnueli. 1985. In transition from global to modular temporal reasoning about programs. In *Logics and Models of Concurrent Systems*. Krzysztof R. Apt, (Ed.) Springer Berlin Heidelberg, Berlin, Heidelberg, 123–144. ISBN: 978-3-642-82453-1.

[46] Amir Pnueli. 1977. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, 46–57.

[47] William Schultz, Ian Dardik, and Stavros Tripakis. 2022. Plain and simple inductive invariant inference for distributed protocols in tla+. In *2022 Formal Methods in Computer-Aided Design (FMCAD)*. IEEE, 273–283.

[48] William Schultz, Siyuan Zhou, Ian Dardik, and Stavros Tripakis. 2021. Design and analysis of a logless dynamic reconfiguration protocol. In *25th International Conference on Principles of Distributed Systems, OPODIS 2021, December 13-15, 2021, Strasbourg, France* (LIPIcs). Quentin Bramas, Vincent Gramoli, and Alessia Milani, (Eds.) Vol. 217. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 26:1–26:16. DOI: 10.4230/LIPICS.OPODIS.2021.26.

[49] James R. Wilcox, Yotam M. Y. Feldman, Oded Padon, and Sharon Shoham. 2024. Mypyvy: a research platform for verification of transition systems in first-order logic. In *Computer Aided Verification*. Arie Gurfinkel and Vijay Ganesh, (Eds.) Springer Nature Switzerland, Cham, 71–85. ISBN: 978-3-031-65630-9.

[50] Jianan Yao, Runzhou Tao, Ronghui Gu, and Jason Nieh. 2022. DuoAI: fast, automated inference of inductive invariants for verifying distributed protocols. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, (July 2022), 485–501. ISBN: 978-1-939133-28-1. https://www.usenix.org/conference/osdi22/presentation/yao.

[51] Jianan Yao, Runzhou Tao, Ronghui Gu, Jason Nieh, Suman Jana, and Gabriel Ryan. 2021. DistAI: Data-Driven automated invariant learning for distributed protocols. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. USENIX Association, (July 2021), 405–421. ISBN: 978-1-939133-22-9. https://www.usenix.org/conference/osdi21/presentation/yao.

[52] Yuan Yu, Panagiotis Manolios, and Leslie Lamport. 1999. Model checking tla+ specifications. In *Correct Hardware Design and Verification Methods*. Laurence Pierre and Thomas Kropf, (Eds.) Springer Berlin Heidelberg, Berlin, Heidelberg, 54–66. ISBN: 978-3-540-48153-9.

[53] Changjian Zhang, David Garlan, and Eunsuk Kang. 2020. A behavioral notion of robustness for software systems. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (ESEC/FSE 2020). Association for Computing Machinery, Virtual Event, USA, 1–12. ISBN: 9781450370431. DOI: 10.1145/3368089.3409753.

[54] Tony Nuda Zhang, Travis Hance, Manos Kapritsos, Tej Chajed, and Bryan Parno. 2024. Inductive invariants that spark joy: using invariant taxonomies to streamline distributed protocol proofs. In *Proceedings of the 18th USENIX Conference on Operating Systems Design and Implementation* (OSDI'24) Article 45. USENIX Association, Santa Clara, CA, USA, 17 pages. ISBN: 978-1-939133-40-3.

$$\textsc{interf-free}\ \frac{I \vdash \langle\!\langle A \rangle\!\rangle C_1 \langle\!\langle G \rangle\!\rangle}{I \vdash \langle\!\langle A \rangle\!\rangle C_1 \parallel C_2 \langle\!\langle G \rangle\!\rangle} \qquad \textsc{trans-inv}\ \frac{I_1 \vdash \langle\!\langle A \rangle\!\rangle C \langle\!\langle R \rangle\!\rangle \qquad I_2 \vdash \langle\!\langle R \rangle\!\rangle C \langle\!\langle G \rangle\!\rangle}{I_1 \wedge I_2 \vdash \langle\!\langle A \rangle\!\rangle C \langle\!\langle G \rangle\!\rangle}$$

Fig. 18. Basic inference rules for proving soundness of the composition rules.

Appendix

A Proof of Thm. 3.3

We now prove Thm. 3.3 which states: $I \vdash \langle\!\langle A \rangle\!\rangle C \langle\!\langle G \rangle\!\rangle$ implies $\vdash \langle\!\langle A \rangle\!\rangle C \langle\!\langle G \rangle\!\rangle$.

Proof. Assume that $I \vdash \langle\!\langle A \rangle\!\rangle C \langle\!\langle G \rangle\!\rangle$. Using the inductive invariant proof method, the three equations in Def. 3.2 establish $(vars, Init \wedge A, Next \wedge A \wedge A')^p \models \Box G$. However, notice that $(vars, Init \wedge A, Next \wedge A \wedge A')^p$ is identical to $(vars, Init \wedge A, Next \wedge A')^p$, from which we can infer $\vdash \langle\!\langle A \rangle\!\rangle C \langle\!\langle G \rangle\!\rangle$ by Def. 3.1. □

B Proof of Soundness for State-Based Contract Composition

In this appendix, we show that the composition rules for the state-based theory are sound. Our proofs are based on two observations, which we encode formally as inference rules. The first observation is that components do not share state variables, and hence are interference-free. This observation is captured by inference rule interf-free, which we present in Fig. 18. The second observation, captured by inference rule trans-inv in Fig. 18, is that each individual component has the *transitivity of invariance* property. We will now show that these two inference rules are sound, which we will then use as lemmas to prove that the composition rules are sound.

Lemma B.1. *interf-free is sound.*

Proof. Suppose that $I \vdash \langle\!\langle A \rangle\!\rangle C_1 \langle\!\langle G \rangle\!\rangle$, then by definition we have the following three facts: (1) $Init_1 \wedge A \implies I$, (2) $I \wedge Next_1 \wedge A \wedge A' \implies I'$, and (3) $I \implies G$. Our goal is to prove that $I \vdash \langle\!\langle A \rangle\!\rangle C_1 \parallel C_2 \langle\!\langle G \rangle\!\rangle$. Because $Init_1 \wedge Init_2 \wedge A \implies Init_1 \wedge A$ and also because of fact (1), we can conclude initiation. Also, due to fact (3), I implies safety. Therefore, it remains to prove consecution.

We will prove consecution by the possible cases on the actions of $C_1 \parallel C_2$. The three cases for an action $a \in Act\ (C_1 \parallel C_2)$ are (i) $a \in (Act\ C_1) \cap (Act\ C_2)$, (ii) $a \in Act\ C_1$ and $a \notin Act\ C_2$, or (iii) $a \notin Act\ C_1$ and $a \in Act\ C_2$. In each possible case, will prove consecution, i.e. that $I \wedge a \wedge A \wedge A' \implies I'$.

(i) In this case, $a \implies Next_1$. Consecution follows due to this as well as fact (2).
(ii) We have $a \implies Next_1$ in this case as well, and therefore consecution also follows due to fact (2).
(iii) In this case, $a \implies SV(C_1)' = SV(C_1)$, where $SV(C_1)' = SV(C_1)$ is an abuse of notation that indicates that the state variables of C_1 are unchanged. Furthermore, by Def. 3.2, we have $SV(I) \subseteq SV(C_1)$ which implies $a \implies SV(I)' = SV(I)$. Finally, consecution follows because $I \wedge SV(I)' = SV(I) \implies I'$.

Therefore, we have shown $I \vdash \langle\!\langle A \rangle\!\rangle C_1 \parallel C_2 \langle\!\langle G \rangle\!\rangle$ as desired. □

Lemma B.2. *trans-inv is sound.*

Proof. Suppose that $I_1 \vdash \langle\!\langle A \rangle\!\rangle C \langle\!\langle R \rangle\!\rangle$ and $I_2 \vdash \langle\!\langle R \rangle\!\rangle C \langle\!\langle G \rangle\!\rangle$. Then we have the following three facts: (1) $Init \implies I_1$ and $Init \implies I_2$, (2) $I_1 \wedge Next \wedge A \wedge A' \implies I_1'$ and $I_2 \wedge Next \wedge R \wedge R' \implies I_2'$, and (3) $I_1 \implies R$ and $I_2 \implies G$. Our goal is to prove that $I_1 \wedge I_2 \vdash \langle\!\langle A \rangle\!\rangle C \langle\!\langle G \rangle\!\rangle$. Initiation and safety follow by facts (1) and (3) respectively. Then, the following equations establish consecution, where the right-hand side column indicates the reason for each implication in parentheses.

$$
\begin{aligned}
& I_1 \land I_2 \land \mathit{Next} \land A \land A' && && (7)\\
\implies & I_1 \land R \land I_2 \land \mathit{Next} \land A \land A' && (I_1 \implies R) && (8)\\
\implies & I_1' \land R \land I_2 \land \mathit{Next} && (I_1 \land \mathit{Next} \land A \land A' \implies I_1') && (9)\\
\implies & I_1' \land R \land R' \land I_2 \land \mathit{Next} && (I_1' \implies R') && (10)\\
\implies & I_1' \land I_2' && (I_2 \land \mathit{Next} \land R \land R' \implies I_2') && (11)
\end{aligned}
$$

In the equations above, the reasons in (8) and (10) follow due to fact (3), while the reasons in (9) and (11) follow due to fact (2). □

The two lemmas above show that the INTERF-FREE and TRANS-INV inference rules (Fig. 18) are sound. We will now use these two lemmas to show that the compositional inference rules (Fig. 5) are sound. We also include a proof that NAIVE-COMP is sound for completness.

THEOREM B.3. *NAIVE-COMP is sound.*

PROOF. Suppose that $I_1 \vdash \langle\!\langle A \rangle\!\rangle C_1 \langle\!\langle R \rangle\!\rangle$ and $I_2 \vdash \langle\!\langle R \rangle\!\rangle C_2 \langle\!\langle G \rangle\!\rangle$. By Lemma B.1 and the fact that $\|$ is commutative, we have $I_1 \vdash \langle\!\langle A \rangle\!\rangle C_1 \parallel C_2 \langle\!\langle R \rangle\!\rangle$ and $I_2 \vdash \langle\!\langle R \rangle\!\rangle C_1 \parallel C_2 \langle\!\langle G \rangle\!\rangle$. Finally, the theorem follows with Lemma B.2 by using the TRANS-INV rule on $C_1 \parallel C_2$. □

THEOREM B.4. *BRIDGE-COMP is sound.*

PROOF. Suppose that $I_1 \vdash \langle\!\langle A \rangle\!\rangle C_1 \parallel B \langle\!\langle R \rangle\!\rangle$ and $I_2 \vdash \langle\!\langle R \rangle\!\rangle B \parallel C_2 \langle\!\langle G \rangle\!\rangle$. By Lemma B.1 and the fact that $\|$ is commutative, we have $I_1 \vdash \langle\!\langle A \rangle\!\rangle C_1 \parallel B \parallel C_2 \langle\!\langle R \rangle\!\rangle$ and $I_2 \vdash \langle\!\langle R \rangle\!\rangle C_1 \parallel B \parallel C_2 \langle\!\langle G \rangle\!\rangle$. Finally, the theorem follows with Lemma B.2 by using the TRANS-INV rule on $C_1 \parallel B \parallel C_2$. □

THEOREM B.5. *AUX-COMP is sound.*

PROOF. Suppose $I_1 \vdash \langle\!\langle A \rangle\!\rangle C_1 \parallel B \langle\!\langle R \rangle\!\rangle$, $I_2 \vdash \langle\!\langle R \rangle\!\rangle B \parallel C_2 \langle\!\langle G \rangle\!\rangle$, and $\mathit{Aux}\ B$. By Thm B.4 and Thm. 3.3, we have $\vdash \langle\!\langle A \rangle\!\rangle C_1 \parallel B \parallel C_2 \langle\!\langle G \rangle\!\rangle$. Finally, because $\mathit{Aux}\ B$, we have $\vdash \langle\!\langle A \rangle\!\rangle C_1 \parallel C_2 \langle\!\langle G \rangle\!\rangle$ by Def. 3.4. □

C Proof of Thm. 4.5

We now prove Thm. 4.5 which states: Let $\Box\phi$ be an action invariant, then $\mathcal{B}(\Box\phi)$ is an auxiliary component.

PROOF. Let $\langle\!\langle A \rangle\!\rangle C \langle\!\langle G \rangle\!\rangle$ be a state-based contract such that $\vdash \langle\!\langle A \rangle\!\rangle C \parallel \mathcal{B}(\Box\phi) \langle\!\langle G \rangle\!\rangle$; by Def. 3.4, the proof obligation is to show that $\vdash \langle\!\langle A \rangle\!\rangle C \langle\!\langle G \rangle\!\rangle$. Let $C = (\mathit{vars}, \mathit{Init}, \mathit{Next})^p$. By Def. 3.1 and the definition of parallel composition, we can infer that $(\mathit{vars}, \mathit{Init} \land A, \mathit{Next} \land A')^p \parallel \mathcal{B}(\Box\phi) \models \Box G$. However, all actions in $\mathcal{B}(\Box\phi)$ are enabled in every state, meaning that $\mathcal{B}(\Box\phi)$ allows all possible behaviors. We can therefore infer $(\mathit{vars}, \mathit{Init} \land A, \mathit{Next} \land A')^p \models \Box G$, from which the theorem follows by Def. 3.1. □

D Proof of Thm. 4.7

We now dedicate this appendix to proving that action invariants are semantically equivalent to their transition system counterpart given by $\mathcal{T}$. Formally, the proof obligation is to show that $\mathcal{L}(\Box\phi) = \mathcal{L}(\mathcal{T}(\Box\phi))$, where $\Box\phi$ is an arbitrary action invariant. This is exactly what we will prove in Thm. 4.7 at the end of this subsection; however, we first prove two helper lemmas.

LEMMA D.1. *There exists a unique state-based behavior in $\mathcal{B}(\Box\phi)$ that corresponds to executing the actions in σ.*

Proof. The behavior exists because all actions are always enabled in $\mathcal{B}(\Box\phi)$ and the behavior is unique because $\mathcal{B}(\Box\phi)$ is deterministic. ☐

Lemma D.2. *Let τ be the state-based behavior in $\mathcal{B}(\Box\phi)$ that corresponds to executing the actions in σ (by Lemma D.1). Then, $\sigma \models \Box\phi$ if and only if $\tau \models \Box\mathcal{R}(\phi)$.*

Proof. The proof obligation is to show that for all $i \geq 0$, we have $\emptyset \vdash \sigma, i \models \phi$ if and only if $\emptyset \vdash \tau_i \models \mathcal{R}(\phi)$. However, it suffices to prove the following stronger statement: for all well-formed environments E (environments that assign values to the free variables in ϕ) and for all $i \geq 0$, we have $E \vdash \sigma, i \models \phi$ if and only if $E \vdash \tau_i \models \mathcal{R}(\phi)$. We will prove this statement by structural induction for SFL over ϕ.

In the base case, we have $\phi = f(r)$, where $f = (s, v, T)$ is a fluent and r are arguments to the fluent. Let a well-formed environment E and $i \geq 0$ be given. Furthermore, let v_i be the value of the variable v in the state τ_i. Then, $E \vdash \sigma, i \models f(r)$ if and only if $(f|\sigma_0 \dots \sigma_i)[r[E]]$ if and only if $v_i[r[E]]$ if and only if $E \vdash \tau_i \models \mathcal{R}(f(r))$.

Since ϕ is a non-temporal formula, it suffices to only consider non-temporal connectives for the inductive step. In the case that $\phi = \psi_1 \vee \psi_2$, let a well-formed environment E (for ϕ) and $i \geq 0$ be given. E must also be well-formed for ψ_1 and ψ_2, since no free-variables are introduced by the disjunction. Therefore, we can invoke the inductive hypothesis to show that $E \vdash \sigma, i \models \psi_1$ if and only if $E \vdash \tau_i \models \mathcal{R}(\psi_1)$ and also $E \vdash \sigma, i \models \psi_2$ if and only if $E \vdash \tau_i \models \mathcal{R}(\psi_2)$. However, this allows us to conclude that $E \vdash \sigma, i \models \phi$ if and only if $E \vdash \tau_i \models \mathcal{R}(\phi)$ by the definition of ϕ and also the definition of $\mathcal{R}$ over disjunctions. The remaining connectives are similar. ☐

We now provide a proof for Thm. 4.7, which states that for any action invariant $\Box\phi$, we have $\mathcal{L}(\Box\phi) = \mathcal{L}(\mathcal{T}(\Box\phi))$.

Proof. We will prove that for any action-based behavior σ, $\sigma \models \Box\phi$ if and only if $\sigma \models \mathcal{T}(\Box\phi)$. Let τ be the state-based behavior in $\mathcal{B}(\Box\phi)$ that corresponds to executing the actions in σ (by Lemma D.1).

$\sigma \models \Box\phi$ if and only if $\tau \models \Box\mathcal{R}(\phi)$	By Lemma D.2.
$\tau \models \Box\mathcal{R}(\phi)$ if and only if $\tau \models \mathcal{T}(\Box\phi)$	By Def. 4.6.
$\tau \models \mathcal{T}(\Box\phi)$ if and only if $\sigma \models \mathcal{T}(\Box\phi)$	By the assumption that τ corresponds to σ in $\mathcal{B}(\Box\phi)$, and hence also $\mathcal{T}(\Box\phi)$. For "only if", also because $\mathcal{T}(\Box\phi)$ is deterministic.

Together, the equations above imply the intended result. ☐

E Proof of Thm. 4.10

We now prove Thm. 4.10 which states: $I \vdash \langle\alpha\rangle C \langle\gamma\rangle$ implies $\vdash \langle\alpha\rangle C \langle\gamma\rangle$

Proof. Assume that $I \vdash \langle\alpha\rangle C \langle\gamma\rangle$. Then, from both Def. 4.9 and Thm. 3.3, we can infer that $\vdash \langle\!\langle \mathcal{R}(\alpha) \rangle\!\rangle \mathcal{B}(\alpha) \parallel C \parallel \mathcal{B}(\gamma) \langle\!\langle \mathcal{R}(\gamma) \rangle\!\rangle$. Let $\mathcal{B}(\alpha) = (vars, Init, Next)^p$. By Def. 3.1 and the definition of parallel composition we can infer $(vars, Init \wedge \mathcal{R}(\alpha), Next \wedge \mathcal{R}(\alpha)')^p \parallel C \parallel \mathcal{B}(\gamma) \models \Box\mathcal{R}(\gamma)$. By Def. 4.6, we see that this is equivalent to the statement $\mathcal{T}(\Box\alpha) \parallel C \parallel \mathcal{B}(\gamma) \models \Box\mathcal{R}(\gamma)$.

Now consider an action-based behavior σ such that $\sigma \models \mathcal{T}(\Box\alpha) \parallel C$; we will show that $\sigma \models \mathcal{T}(\Box\gamma)$ to complete the proof. Because $\mathcal{B}(\gamma)$ has every action enabled in all states, it must also be the case that $\sigma \models \mathcal{T}(\Box\alpha) \parallel C \parallel \mathcal{B}(\gamma)$. By Lemma D.1, there exists a unique state based behavior τ in $\mathcal{B}(\gamma)$ that corresponds to σ. This in turn implies that $\tau \models \mathcal{T}(\Box\alpha) \parallel C \parallel \mathcal{B}(\gamma)$, which also implies that $\tau \models \Box\mathcal{R}(\gamma)$. Because $\tau \models \mathcal{B}(\gamma)$ and $\tau \models \Box\mathcal{R}(\gamma)$, we can infer that $\tau \models \mathcal{T}(\Box\gamma)$. Finally, by the assumption that σ corresponds to τ, we see that $\sigma \models \mathcal{T}(\Box\gamma)$. □

F Soundness of Action-Based Contract Composition

THEOREM F.1. *Rule SFL-COMP is sound.*

PROOF. Suppose that $I_1 \vdash \langle\alpha\rangle C_1\langle\rho\rangle$ and $I_2 \vdash \langle\rho\rangle C_2\langle\gamma\rangle$. By Def. 4.8, we also have $I_1 \vdash \langle\!\langle\mathcal{R}(\alpha)\rangle\!\rangle\mathcal{B}(\alpha) \parallel C_1 \parallel \mathcal{B}(\rho)\langle\!\langle\mathcal{R}(\rho)\rangle\!\rangle$ and $I_2 \vdash \langle\!\langle\mathcal{R}(\rho)\rangle\!\rangle\mathcal{B}(\rho) \parallel C_2 \parallel \mathcal{B}(\gamma)\langle\!\langle\mathcal{R}(\gamma)\rangle\!\rangle$. Using the BRIDGE-COMP inference rule (Thm. B.4), we can infer that $I_1 \wedge I_2 \vdash \langle\!\langle\mathcal{R}(\alpha)\rangle\!\rangle\mathcal{B}(\alpha) \parallel C_1 \parallel \mathcal{B}(\rho) \parallel C_2 \parallel \mathcal{B}(\gamma)\langle\!\langle\mathcal{R}(\gamma)\rangle\!\rangle$. Finally, we see that $I_1 \wedge I_2 \vdash \langle\alpha\rangle C_1 \parallel \mathcal{B}(\rho) \parallel C_2\langle\gamma\rangle$ by Def. 4.8. □

THEOREM F.2. *Rule SFL-SAFE is sound.*

PROOF. Suppose that $I_1 \vdash \langle\alpha\rangle C_1\langle\rho\rangle$ and $I_2 \vdash \langle\rho\rangle C_2\langle\gamma\rangle$. By Thm. F.1, we can infer $I_1 \wedge I_2 \vdash \langle\alpha\rangle C_1 \parallel \mathcal{B}(\rho) \parallel C_2\langle\gamma\rangle$. By Def. 4.8, we also have $I_1 \wedge I_2 \vdash \langle\!\langle\mathcal{R}(\alpha)\rangle\!\rangle\mathcal{B}(\alpha) \parallel C_1 \parallel \mathcal{B}(\rho) \parallel C_2 \parallel \mathcal{B}(\gamma)\langle\!\langle\mathcal{R}(\gamma)\rangle\!\rangle$. However, by Thm. 3.3, we can also infer $\vdash \langle\!\langle\mathcal{R}(\alpha)\rangle\!\rangle\mathcal{B}(\alpha) \parallel C_1 \parallel \mathcal{B}(\rho) \parallel C_2 \parallel \mathcal{B}(\gamma)\langle\!\langle\mathcal{R}(\gamma)\rangle\!\rangle$. The theorem then follows by Def. 4.8. □

G Proof of Thm. 4.13

We now prove Thm. 4.13 which states: $I \vdash \langle\alpha\rangle C\langle G\rangle$ implies $\vdash \langle\alpha\rangle C\langle G\rangle$

PROOF. Assume that $I \vdash \langle\Box\alpha\rangle C\langle G\rangle$. Then, from both Def. 4.9 and Thm. 3.3, we can infer that $\vdash \langle\!\langle\mathcal{R}(\alpha)\rangle\!\rangle\mathcal{B}(\Box\alpha) \parallel C\langle\!\langle G\rangle\!\rangle$. By Def. 3.1 and the definition of parallel composition, we can also infer $(vars, Init \wedge \mathcal{R}(\alpha), Next \wedge \mathcal{R}(\alpha)')^p \parallel C \models \Box G$, where $\mathcal{B}(\Box\alpha) = (vars, Init, Next)^p$. By Def. 4.6, we see that this is equivalent to the statement $\mathcal{T}(\Box\alpha) \parallel C \models \Box G$. Finally, the theorem is proved by Def. 4.8. □

H Soundness of Hybrid Contract Composition

We now prove that the two inference rules HYBRID-COMP and HYBRID-SAFE are sound.

THEOREM H.1. *The rule HYBRID-COMP is sound.*

PROOF. Suppose that $I_1 \vdash \langle\alpha\rangle C_1\langle\rho\rangle$ and $I_2 \vdash \langle\rho\rangle C_2\langle G\rangle$. By Def. 4.8, we have $I_1 \vdash \langle\!\langle\mathcal{R}(\alpha)\rangle\!\rangle\mathcal{B}(\alpha) \parallel C_1 \parallel \mathcal{B}(\rho)\langle\!\langle\mathcal{R}(\rho)\rangle\!\rangle$ and by Def. 4.11 we also have $I_2 \vdash \langle\!\langle\mathcal{R}(\rho)\rangle\!\rangle\mathcal{B}(\rho) \parallel C_2\langle\!\langle G\rangle\!\rangle$. Using the BRIDGE-COMP inference rule (Thm. B.4), we can infer that $I_1 \wedge I_2 \vdash \langle\!\langle\mathcal{R}(\alpha)\rangle\!\rangle\mathcal{B}(\alpha) \parallel C_1 \parallel \mathcal{B}(\rho) \parallel C_2\langle\!\langle G\rangle\!\rangle$. Finally, we see that $I_1 \wedge I_2 \vdash \langle\alpha\rangle C_1 \parallel \mathcal{B}(\rho) \parallel C_2\langle G\rangle$ by Def. 4.11. □

THEOREM H.2. *The rule HYBRID-SAFE is sound.*

PROOF. Suppose that $I_1 \vdash \langle\alpha\rangle C_1\langle\rho\rangle$ and $I_2 \vdash \langle\rho\rangle C_2\langle G\rangle$. By Thm. H.1, we can infer $I_1 \wedge I_2 \vdash \langle\alpha\rangle C_1 \parallel \mathcal{B}(\rho) \parallel C_2\langle G\rangle$. By Def. 4.11, we also have $I_1 \wedge I_2 \vdash \langle\!\langle\mathcal{R}(\alpha)\rangle\!\rangle\mathcal{B}(\alpha) \parallel C_1 \parallel \mathcal{B}(\rho) \parallel C_2\langle\!\langle G\rangle\!\rangle$. However, by Thm. 3.3, we can also infer $\vdash \langle\!\langle\mathcal{R}(\alpha)\rangle\!\rangle\mathcal{B}(\alpha) \parallel C_1 \parallel \mathcal{B}(\rho) \parallel C_2\langle\!\langle G\rangle\!\rangle$. The theorem then follows by Def. 4.11. □

I Bridge and Fluent Definitions for Mongo Static Raft

MODULE $committedThisTermT$

$Init \triangleq v_4 = [x_0 \in \mathbb{N} \mapsto [x_1 \in Server \mapsto \text{FALSE}]]$

$BecomeLeader(s, term) \triangleq$
$\quad v_4' = [x_0 \in \mathbb{N} \mapsto [x_1 \in Server \mapsto \text{FALSE}]]$

$CommitEntry(s, ind, term) \triangleq$
$\quad v_4' = [v_4 \text{ EXCEPT } ![term][s] = \text{TRUE}]$

(a) Fluent definition for *committedThisTerm*.

MODULE $globalCurrentTermT$

$Init \triangleq v_5 = [x_0 \in \mathbb{N} \mapsto \text{FALSE}]$

$BecomeLeader(s, term) \triangleq$
$\quad v_5' = [[x_0 \in \mathbb{N} \mapsto \text{FALSE}]$
$\qquad \text{EXCEPT } ![term] = \text{TRUE}]$

(b) Fluent definition for *globalCurrentTerm*.

MODULE $reqThisTermT$

$Init \triangleq v_6 = [x_0 \in Server \mapsto [x_1 \in \mathbb{N} \mapsto \text{FALSE}]]$

$ClientRequest(s, term) \triangleq$
$\quad v_6' = [v_6 \text{ EXCEPT } ![s][term] = \text{TRUE}]$

$GetEntries(s_1, s_2) \triangleq$
$\quad v_6' = [v_6 \text{ EXCEPT } ![s_1] = [x_0 \in \mathbb{N} \mapsto \text{FALSE}]]$

$RollbackEntries(s_1, s_2) \triangleq$
$\quad v_6' = [v_6 \text{ EXCEPT } ![s_1] = [x_0 \in \mathbb{N} \mapsto \text{FALSE}]]$

$BecomeLeader(s, term) \triangleq$
$\quad v_6' = [x_0 \in Server \mapsto [x_1 \in \mathbb{N} \mapsto \text{FALSE}]]$

(c) Fluent definition for *reqThisTerm*.

$$\begin{aligned}
&\land\ \forall t \in \mathbb{N} : \forall s \in Server : committedThisTerm(t, s) \implies globalCurrentTerm(t)\\
&\land\ \forall t \in \mathbb{N} : \forall s \in Server : reqThisTerm(s, t) \implies currentLeader(s, t)\\
&\land\ \forall t \in \mathbb{N} : \forall s \in Server : committedThisTerm(t, s) \implies currentLeader(s, t)\\
&\land\ \forall t \in \mathbb{N} : \forall s \in Server : currentLeader(s, t) \implies leaderAtTermServ(t, s)\\
&\land\ \forall t_1, t_2 \in \mathbb{N} : \forall s \in Server :\\
&\quad (leaderAtTermServ(t_1, s) \land globalCurrentTerm(t_2)) \implies (t_1 \le t_2)\\
&\land\ \forall t \in \mathbb{N} : \forall s_1, s_2 \in Server :\\
&\quad (leaderAtTermServ(t, s_1) \land leaderAtTermServ(t, s_2)) \implies (s_1 = s_2)
\end{aligned}$$

(d) The entire bridge formula ρ_l for the Log and $StateTerm$ components, including the *one leader per term* conjunct.

Fig. 19. Additional definitions for the MongoStaticRaft protocol.