An Algebra of Alignment for Relational Verification

TIMOS ANTONOPOULOS, Yale University, USA
ERIC KOSKINEN, TON CHANH LE, RAMANA NAGASAMUDRAM, DAVID A. NAUMANN, and MINH NGO, Stevens Institute of Technology, USA

Relational verification encompasses information flow security, regression verification, translation validation for compilers, and more. Effective alignment of the programs and computations to be related facilitates use of simpler relational invariants and relational procedure specs, which in turn enables automation and modular reasoning. Alignment has been explored in terms of trace pairs, deductive rules of relational Hoare logics (RHL), and several forms of product automata. This article shows how a simple extension of Kleene Algebra with Tests (KAT), called BiKAT, subsumes prior formulations, including alignment witnesses for forall-exists properties, which brings to light new RHL-style rules for such properties. Alignments can be discovered algorithmically or devised manually but, in either case, their adequacy with respect to the original programs must be proved; an explicit algebra enables constructive proof by equational reasoning. Furthermore our approach inherits algorithmic benefits from existing KAT-based techniques and tools, which are applicable to a range of semantic models.

CCS Concepts: • **Theory of computation → Logic and verification**; **Algebraic semantics**; **Program reasoning**.

Additional Key Words and Phrases: relational verification, hyperproperties, program algebra, Kleene algebra with tests

ACM Reference Format:
Timos Antonopoulos, Eric Koskinen, Ton Chanh Le, Ramana Nagasamudram, David A. Naumann, and Minh Ngo. 2023. An Algebra of Alignment for Relational Verification. *Proc. ACM Program. Lang.* 7, POPL, Article 20 (January 2023), 31 pages. https://doi.org/10.1145/3571213

1 INTRODUCTION

A number of important program requirements are not trace properties but can be defined as 2-properties. For example, secure information flow says that any two executions with the same "low" (non-secret) inputs have the same low outputs. Continuity says two executions from very close inputs produce very close outputs. Pairs of executions are also the basis for relations between programs, such as extensional equivalence, refinement, or simulation.

One way to prove such a relational property is to prove a strong functional property of each program, and show that the relational property is a consequence. However, from early work on refinement [de Roever et al. 2001; de Roever and Engelhardt 1998] through explicit studies of relational program logic [Benton 2004; Francez 1983] to current work on automated relational reasoning (e.g. see [Beckert and Ulbrich 2018]) it has been clear that one should align intermediate points in programs and their executions, in order to decompose the reasoning. In many cases it is much better to reason in terms of a well chosen alignment, which can make it possible to reason using logically simple relational invariants such as conjunctions of equalities between variables.

Authors' addresses: Timos Antonopoulos, Yale University, USA; Eric Koskinen; Ton Chanh Le; Ramana Nagasamudram; David A. Naumann; Minh Ngo, Stevens Institute of Technology, USA.

2475-1421/2023/1-ART20
https://doi.org/10.1145/3571213

This is especially important for automated reasoning, since restricted fragments such as linear arithmetic facilitate techniques like Horn clause solving to find invariants. It is also important in the many situations where a functional specification is not available. Moreover, alignment enables use of relational specifications and summaries for procedures.

For these reasons, alignment appears in various guises such as the same-structure ("diagonal") proof rules of Benton's relational Hoare logic [Benton 2004] and program product constructions based on syntax [Barthe et al. 2011a; Sousa and Dillig 2016] and on product automata [Barthe et al. 2013; Churchill et al. 2019]. For the moment let us use the term ***aligned product*** for such constructions. The proof of a given relational property has two parts: the chosen aligned product satisfies the property, and the aligned product is ***adequate***, in the sense that it represents all computations of the program(s) of interest. Verification can be automated by searching for possible alignments and checking whether they can be proved to satisfy the property, or by synthesizing an alignment in tandem with attempting to prove the property [Antonopoulos et al. 2019; Churchill et al. 2019; Farzan and Vandikas 2019; Shemer et al. 2019; Unno et al. 2021].

The current state of the art offers a number of variations on these ideas, in disparate formulations. In this paper we introduce an algebra for alignment products which makes it possible to account for adequacy by equational reasoning. Our algebra, called BiKAT, is a simple extension of Kleene Algebra with Tests (KAT) [Kozen 1997], itself an algebra of programs. Our representation of alignment products is thus immediately amenable to various program analysis tools and may serve as a framework for procedures that search for alignments.

In KAT, a partial correctness assertion $\{p\}c\{q\}$ is expressed by an equation: $p; c; \neg q = 0$. Now consider what we call the $\forall\forall$ ***relational judgment***, written $c|c' : R \approx S$, meaning that: from a pair of states related by R, terminated executions of c and c' respectively, end in states related by S. The relational judgment can be expressed by the similar equation $R; \langle c \mid c' \rangle; \neg S = 0$, using the BiKAT form $\langle c \mid c' \rangle$ that represents pairs of executions. We show that rules of relational Hoare logics can be derived in BiKAT, which applies to many semantic models, not just the specific semantic models used in prior works such as those cited above.

Not all relational properties have the termination-insensitive $\forall\forall$ (2-safety) form described above. For programs which may exhibit nondeterminacy, typical requirements involve existential quantification. For example, possibilistic noninterference says that given two low-indistinguishable states, and a terminated execution from the first, there is a terminated execution from the second, with low-indistinguishable final state [Clarkson and Schneider 2010; Sabelfeld and Myers 2003]. As we explain, attempts to formulate such properties in the algebra lead to a three-run equation which we have found somewhat unwieldly. So, in addition we provide a usable reduction to equations on two-run alignment products.

With this reduction in hand, we discovered that one can derive rules for reasoning about two fundamental and ubiquitous kinds of $\forall\exists$ properties: forward and backward simulation. We derived inference rules for a logic of forward simulation and a logic of backward simulation. In retrospect, these logics seem natural and it is surprising that (to our knowledge) such rules have not appeared in the literature. As often happens, devising an algebraic description of the models of interest leads to new insights about existing ideas, and also results far beyond the initial motivations.

Contributions.

- We define a novel algebra, BiKAT, that can express the alignment of programs in both relational and trace semantic models, building in a simple way on the well understood KAT.
- Using BiKAT, we derive rules of relational Hoare logic for $\forall\forall$ properties, just as Kozen's seminal work derives the rules for conventional Hoare logic [Kozen 2000].

- Using BiKAT equations, we characterize the ∀∃ forward and backward simulation properties, and use these results to derive new proof rules for these properties. To our knowledge this provides the first deductive system for backward simulation judgments.

Outline. Sect. 2 is an overview of the problem and shows BiKAT in action. Sect. 3 briefly reviews KAT. Sect. 4 defines BiKAT and some standard models. Sect. 5 demonstrates direct use of BiKAT on illustrative examples of relational reasoning challenges. Examples are given throughout to show how BiKAT can express and justify alignments used in the literature on relational verification. Sect. 6 derives rules for the ∀∀ judgment. Sect. 7 addresses ∀∃ properties: we develop characterizations in BiKAT and use these to derive proof rules for forward and backward simulation. We also report on explorations for ∃∀ and ∃∃ properties, and sketch a notion of TriKAT that deserves further study. Sect. 8 discusses automation, product programs, expressiveness, and future directions, all in the context of related work.

Accompanying this paper is a Coq formalization that includes the basic definitions and results about BiKATs [Antonopoulos et al. 2022b]. There is also an extended version of the paper with an appendix working out many proofs and examples [Antonopoulos et al. 2022a].

2 EXAMPLES OF ALIGNMENT IN BIKAT

We now introduce our algebra of alignment, through a series of examples that highlight the features of our theory. Recall that Kleene Algebra with Tests (KAT) [Kozen 1997] can be used to represent and reason about the behavior of programs. Atomic program statements can be treated as primitive "actions" in the KAT and conditions can be treated as primitive "tests." For example, consider the following program that computes factorial of `n` and stores it in `r`:

Code: `C1: i:=n; r:=1; while i!=0 do r:=r*i; i:=i-1 od`
KAT: $k_1 : \texttt{i:=n} \cdot \texttt{r:=1} \cdot (\texttt{[i≠0]} \cdot \texttt{r:=r*i} \cdot \texttt{i:=i-1})^* \cdot \neg\texttt{[i≠0]}$

The above KAT expression combines primitives with sequential composition · and Kleene star-iteration $*$. Tests are denoted with square brackets to distinguish from non-test actions (in a KAT tests are a subset of actions). There are various semantic models of KATs, notably relational models, where actions are state relations, and trace models where actions denote sets of traces. KAT permits one to express properties through terms/equations that involve programs and their specifications. For example, KAT subsumes Hoare logic: the KAT equation $p \cdot k_1 \cdot \neg q = 0$ expresses validity of the Hoare triple $\{p\}\texttt{c}_1\{q\}$.

Example 2.1. Dependency or non-interference. Let us now consider a basic ∀∀ property of a single program (2-safety): whether different output results can be observed by varying input values. A basic pre/post 2-safety ∀∀ property, written $C1 \mid C1 : [\texttt{n} \doteq \texttt{n}] \approx\!\!> [\texttt{r} \doteq \texttt{r}]$, requires that for any two executions of $C1$, if they agree on the initial value of `n`, then they will agree on the final value of `r`, regardless of the initial values of the other variables. Here $[\texttt{n} \doteq \texttt{n}]$ means value of `n` in the left program is equal to the value of `n` in the right program. Our notation does not require that the given program(s) have been modified to act on disjoint parts of a single state.

In this paper we will describe an algebra for expressing these and other relational verification problems, as well as the alignments necessary for proving them. We will introduce a relational analog of KAT called *BiKAT*, which allows us to, for example, express this 2-safety judgment as:

$$[\texttt{n} \doteq \texttt{n}] \cdot \langle k_1 \mid k_1 \rangle \cdot \neg[\texttt{r} \doteq \texttt{r}] = 0 \tag{1}$$

The connectives above are relational analogs of the KAT connectives: · is sequential composition, ¬ is negation, and 0 is the empty relation. A key feature of BiKAT is to be able to homomorphically *embed* a unary KAT expression k into a relational setting, that acts according to k on one side and

acts as the identity on the other side. Embedding k on the "left" is denoted $\langle k]$, on the "right" is $[k\rangle$, and we write $\langle k \mid k'\rangle$ to mean $\langle k] \cdot [k'\rangle$. This embedding resembles the popular sequential product (a.k.a. self-composition [Barthe et al. 2004]), but does not require k and k' to act on disjoint variables. A BiKAT is a (special kind of) KAT and, in addition to embedded unary actions/tests, can be equipped with its own relational primitive actions and tests. The agreement $[\mathtt{r} \doteq \mathtt{r}]$ above is a primitive "bitest." Embedding and bitests are sequentially composed together in the equation above with $\cdot$ into an overall equation that characterizes validity of the relational judgment, by requiring that post-disagreement on $\mathtt{r}$ is equivalent to the empty set of behaviors.

While the above is merely a BiKAT problem statement of non-interference, the principle benefits of BiKAT arise when we begin to use our equational theory of BiKAT to derive alignments that ease (or make tractable) the task of relational verification. Notice that, thus far, the above embedding would still require one to reason that k_1 computes the factorial function, which is more work than truly necessary for non-interference. (This issue was highlighted in the seminal work that introduced the term 2-safety [Terauchi and Aiken 2005].) We can instead exploit that the left/right embeddings are homomorphisms, and exploit the algebra of the embedded KAT expressions, to find another equation that implies the above one. In Sect. 4 we will discuss the details that lead to the following:

$$[\mathtt{n} \doteq \mathtt{n}] \cdot \langle \mathtt{i:=n} \mid \mathtt{i:=n} \rangle \cdot \langle \mathtt{r:=1} \mid \mathtt{r:=1} \rangle \\ \cdot (\langle \mathtt{[i!=0]} \cdot \mathtt{r:=r*i} \cdot \mathtt{i:=i-1} \mid \mathtt{[i!=0]} \cdot \mathtt{r:=r*i} \cdot \mathtt{i:=i-1} \rangle)^* \cdot \langle \mathtt{[i==0]} \mid \mathtt{[i==0]} \rangle \cdot \neg[\mathtt{r} \doteq \mathtt{r}] \;=\; 0$$

This equation is derived from (1) using BiKAT laws, the fact that embedding distributes through the KAT operators, and the fact that left and right embeddings commute: $\langle k] \cdot [k'\rangle = [k'\rangle \cdot \langle k]$. We have now aligned the programs so that their loops iterate in "lock step". (We will see more complicated alignments below.) For this example, this alignment is sufficient because, using a few axioms that embody the semantics of primitive tests and assignments we can introduce relations within the *body* of the BiKAT term, using bitests $[\mathtt{i} \doteq \mathtt{i}]$, $[\mathtt{n} \doteq \mathtt{n}]$, $[\mathtt{r} \doteq \mathtt{r}]$ expressing agreement on $\mathtt{i}$, $\mathtt{n}$, $\mathtt{r}$. Overall this lets us conclude the original property without having to resort to reasoning about the full functional behavior of the programs involved.

Example 2.2. Aligning two different programs. Consider the programs from Shemer et al. [2019]:

$$D_1: \quad \mathtt{y:=0} \cdot \mathtt{z:=2*x} \cdot (\mathtt{[z>0]} \cdot \mathtt{z--} \cdot \mathtt{y:=y+x})^* \cdot \mathtt{[z<=0]} \\ D_2: \quad \mathtt{y:=0} \cdot \mathtt{z:=x} \cdot (\mathtt{[z>0]} \cdot \mathtt{z--} \cdot \mathtt{y:=y+x})^* \cdot \mathtt{[z<=0]} \cdot \mathtt{y:=y*2}$$

These programs compute $2\mathtt{x}^2$, but in different ways: D_1 iterates $2\mathtt{x}$ times, each time adding $\mathtt{x}$ to the result $\mathtt{y}$, whereas D_2 iterates only $\mathtt{x}$ times, but then multiplies by 2 at the end. This example has a flavor of a compiler optimization, where a program may be transformed to save on loop iterations. Here too we can formulate the property of equal outputs from equal inputs and, using the laws of BiKAT to rewrite the problem into this aligned form:

$$[\mathtt{x} \doteq \mathtt{x}] \cdot \langle \mathtt{y:=0} \mid \mathtt{y:=0} \rangle \cdot \langle \mathtt{z:=2*x} \mid \mathtt{z:=x} \rangle \cdot [\mathtt{z} \doteq 2\mathtt{z}] \\ \cdot ([\mathtt{z} \doteq 2\mathtt{z}][2\mathtt{y} \doteq \mathtt{y}]\langle \mathit{body} \cdot \mathit{body} \mid \mathit{body} \rangle)^* \\ \cdot [2\mathtt{y} \doteq \mathtt{y}] \cdot \langle \mathtt{[z==0]} \mid \mathtt{[z==0]} \cdot \mathtt{y:=y*2} \rangle \cdot [\mathtt{x} \doteq \mathtt{x}] \cdot \neg[\mathtt{x} \doteq \mathtt{x}] = 0$$

where $\mathit{body} \triangleq \mathtt{[z>0]\ z--;\ y:=y+x}$. (Throughout this paper, we often use juxtaposition or ; to mean the $\cdot$ KAT operator.) This example uses an alignment where two iterations of the loop in the first program are related to one iteration of the second one, to preserve the relational invariant that $\mathtt{z}$ in the left program is twice the value of $\mathtt{z}$ in the right program and that $\mathtt{y}$ in the right program is twice the value of $\mathtt{y}$ in the left program. To prove this we introduce agreement bitests in the body of the loop and use KAT-based inductive reasoning to show those bitests, including $[\mathtt{x} \doteq \mathtt{x}]$, are indeed invariant. With these simple relational bitests, we can prove the equivalence between D_1 and D_2.

Example 2.3. Mixing program algebra with alignment algebra. Now consider two programs that iterate over C-style arrays:

```
L1: x := 0; while (x < N * M){ a[x] := f(x); x++;}
L2: i := 0; while (i < N){ j := 0; while (j < M){ A[i,j] := f(i*M+j); j++; i++;}}
```

In L_1 there is a one-dimensional array, `a`, of size `N*M`, while L_2 uses an `N`-by-`M` two-dimensional array `A`. We prove that the two programs are equivalent modulo this change in data representation. This example is used in Barthe et al. [2013] to argue for alignment based on a control flow graph representation. A custom rewriting relation, with KAT-like rules, is used in Banerjee et al. [2016] to handle the example using syntactic RHL rules.

Aligning these programs is challenging because they have fundamentally different shapes to them (single versus nested loops). To address this example we exploit the fact that the programs can be represented as KAT terms which themselves can be algebraically manipulated, whilst embedded within the BiKAT. Our alignment (detailed in Sect. 5) first uses unary KAT laws to transform L_1 so that it consists of nested loops. We then align those nested loops and introduce loop alignment invariants including an agreement on indexing $[\texttt{x} \doteq \texttt{i} \times \texttt{M} + \texttt{j}]$ and an agreement on array values $[\texttt{a[x]} \doteq \texttt{A[i,j]}]$. In this way, the laws of BiKAT (and underlying KAT laws) allow one to use equational reasoning to simplify the task of deriving useful alignments, and then introduce simple relational invariants that suffice to prove the overall property.

Beyond the above examples, BiKAT can also be used to construct *data-dependent* alignment (e.g., [Shemer et al. 2019]). A single iteration in one program may need to be aligned with a varying number of iterations in a second program, dependent on the data. Example 6.3 will demonstrate BiKAT's capability to support such alignments.

Example 2.4. Expressing ∀∃ properties. Consider programs E_1 `: x:=any; y:=x` and E_2 `: t:=any; z:=t+1`. These simple programs choose a nondeterministic value (keyword `any`) and then use that value to make an assignment. A common specification pattern is that for any pair of pre-related states, and every execution of E_1, there are choices that E_2 can make to ensure that the post-states are related. The pattern is useful for program equivalence and also for information flow where it is known as *possibilistic non-interference* [Clarkson and Schneider 2010]. For this example, let us use `true` as the pre-relation and $[\texttt{y} \doteq \texttt{z}]$ as the post-relation.

In Sect. 7.5 we describe how BiKAT can be used to support ∀∃ properties such as the above. We again use BiKAT to derive alignments but for ∀∃ we also introduce bitests that intentionally restrict some of the behaviors of the programs. Let W be this "witness," consisting of the aligned programs, along with bitest restrictions. We reduce the problem to three conditions on W (Theorem 7.1):

(WC) The corresponding ∀∀ property holds of W.
(WO) W over-approximates the behavior of the left program.
(WU) W under-approximates the behavior of the right program.

For the above example, we can align the programs in lock-step, and use the bitest $[\texttt{x-1} \doteq \texttt{t}]$ so that, after the assignment `t:=any`, executions are then restricted to those where `t` is `x-1`. Doing so will ensure that the post-relation $[\texttt{y} \doteq \texttt{z}]$ holds (WC). Moreover, this bitest does not restrict the behaviors of the left program (WO), nor does it add behaviors on the right that are not already allowed by the right program (WU). In Sect. 7.4 we also provide deductive rules for such ∀∃ properties.

3 PRELIMINARIES

A Kleene Algebra with Tests (KAT) [Kozen 1997; Kozen and Smith 1996] is a two-sorted structure $(\mathbb{A}, \mathbb{B}, +, \cdot, *, \neg, 1, 0)$, where $\mathbb{B} \subseteq \mathbb{A}$, such that $\mathbb{B}$ is closed under the operations $+, \cdot, \neg$ and these satisfy the laws of Boolean algebra. Moreover $(\mathbb{A}, +, \cdot, *, 1, 0)$ is a Kleene algebra.

We use $a, b, c, \ldots$ for elements of $\mathbb{A}$ (the "actions") and $p, q \ldots$ for elements of $\mathbb{B}$ (the "tests"). We sometimes write $a \cdot b$ as $a; b$ or ab and let it bind tighter than $+$. The axioms of Kleene algebra are:

$$\begin{array}{rcl} a + (b + c) & = & (a + b) + c \\ a + b & = & b + a \\ a + 0 & = & a \\ a + a & = & a \\ a \cdot (b \cdot c) & = & (a \cdot b) \cdot c \\ 1 \cdot a & = & a \\ a \cdot 1 & = & a \\ a \cdot (b + c) & = & a \cdot b + a \cdot c \\ (a + b) \cdot c & = & a \cdot c + b \cdot c \\ 0 \cdot a & = & 0 \\ a \cdot 0 & = & 0 \end{array} \qquad \begin{array}{rcl} 1 + a \cdot a^* & = & a^* \\ 1 + a^* \cdot a & = & a^* \\ b + a \cdot c \leq c & \Rightarrow & a^* \cdot b \leq c \\ b + c \cdot a \leq c & \Rightarrow & b \cdot a^* \leq c \end{array}$$

where $\leq$ is the partial order defined by $a \leq b \Leftrightarrow a + b = b$. Some useful consequences are the ***sliding*** law $a \cdot (b \cdot a)^* = (a \cdot b)^* \cdot a$ and the ***invariance*** law $p \cdot a \leq p \cdot a \cdot p \Rightarrow p \cdot a^* \leq p \cdot a^* \cdot p$ (where p is a test). The correctness equation $p \cdot c \cdot \neg q = 0$ has useful equivalent forms: $p \cdot c = p \cdot c \cdot q$ and $p \cdot c \leq p \cdot c \cdot q$. Every test p satisfies $p \leq 1$.

Given some set Σ, a ***relational model*** is $(\mathbb{A}, \mathbb{B}, +, \cdot, *, \neg, 1, 0)$ where $\mathbb{A}$ is a set of relations[1] on Σ, where 1 is the identity relation id_Σ, 0 is the empty relation, $\cdot$ is relational composition, $*$ is reflexive-transitive closure, $+$ is union of relations, $\mathbb{B}$ is a set of ***sub-identities***, i.e., subsets of the identity relation 1, and $\neg b$ is the complement $1 \setminus b$. Note that $\leq$ is set inclusion.[2] Relational models can be obtained from big-step or denotational semantics of programs, with Σ the set of states.

A ***full*** relational model is one where $\mathbb{A}$ is all relations on Σ and $\mathbb{B}$ is all sub-identities. A full relational model has a ***top*** element, which we call ***havoc*** and write **hav**. Being top means $a \leq \mathbf{hav}$ for all a in $\mathbb{A}$.

A ***trace model*** [Kozen 2003, 2004] $(\mathbb{A}, \mathbb{B}, +, \cdot, *, \neg, 1, 0)$ is given in terms of some set of primitive actions and some set Σ of states. An element of $\mathbb{A}$ is a set of traces, where a trace is a nonempty alternating sequence of states and primitive actions, beginning and ending with a state. (The definitions are with respect to a given set of admissible traces, which one can consider as possible observations for a programming language of interest. For example, if each primitive action has an associated relation on states, then traces may be required to be consecutive with respect to those relations.) An element of $\mathbb{B}$ is a set of singleton traces (essentially a set of states) and negation is defined as complement with respect to Σ. The $+$ operation is union. For traces t and u, the ***coalesced catenation*** $t \diamond u$ is defined only if the last state of t is the first of u, in which case the sequences are catenated but omitting one copy of that state. Otherwise $t \diamond u$ is undefined. For a and b in $\mathbb{A}$, the set $a \cdot b$ is defined to be $\{t \diamond u \mid t \in a \wedge u \in b \wedge t \diamond u \text{ defined}\}$. The star operation is given by iterated catenation. A ***full*** trace model is one that contains all sets of admissible traces (so it has a top).

A KAT is ****-continuous*** [Kozen 1997] provided that for all x, y, z the supremum $\sup_{n \in \mathbb{N}}(x \cdot y^n \cdot z)$ (with respect to $\leq$) exists and $x \cdot y^* \cdot z = \sup_{n \in \mathbb{N}}(x \cdot y^n \cdot z)$. Here y^n is the iterate defined by $y^0 = 1$ and $y^{n+1} = y \cdot y^n$. Relational and trace models are *-continuous.

[1] We always mean binary relations.

[2] Also, $\mathbb{A}$ is closed under these operations, but need not be the full set $\wp(\Sigma \times \Sigma)$ of all relations; and $\mathbb{B}$ need not include all sub-identities.

4 BIKAT

4.1 Definitions and Basic Results

Definition 4.1. A ***BiKAT*** over the KAT $(\mathbb{A}, \mathbb{B}, +, \cdot, *, \neg, 1, 0)$ is a KAT $(\ddot{\mathbb{A}}, \ddot{\mathbb{B}}, \oplus, \fatsemi, \circledast, \ddot{\neg}, \ddot{1}, \ddot{0})$ with KAT homomorphisms $\langle\_] : \mathbb{A} \to \ddot{\mathbb{A}}$ and $[\_\rangle : \mathbb{A} \to \ddot{\mathbb{A}}$, which we call the *left* and *right embeddings*, that satisfy *left-right commutativity*:

$$(LRC) \qquad \langle x] \fatsemi [y\rangle = [y\rangle \fatsemi \langle x] \quad \text{for all } x, y \text{ in } \mathbb{A}.$$

We call $\mathbb{A}$ the ***underlying*** KAT. A BiKAT is ****-continuous*** if both $\mathbb{A}$ and $\ddot{\mathbb{A}}$ are.

For $\langle\_]$ to be a KAT homomorphism means the following:

- $\langle x] \in \ddot{\mathbb{B}}$ for all $x \in \mathbb{B}$
- $\langle 0] = \ddot{0}$, $\langle 1] = \ddot{1}$, and $\langle \neg x] = \ddot{\neg}\langle x]$ for all $x \in \mathbb{B}$
- $\langle x + y] = \langle x] \oplus \langle y]$, $\langle x \cdot y] = \langle x] \fatsemi \langle y]$, and $\langle x^*] = \langle x]^{\circledast}$ for all x, y in $\mathbb{A}$

Mutatis mutandis for $[\_\rangle$.

An obvious generalization is to consider two different underlying KATs (e.g., modeling source and target semantics, for compiler correctness). It is also possible to generalize to k-KATs; we return to this in Sect. 7.6. But our focus is on 2-trace properties. Specializing to 2 lets us streamline notation and use convenient "left/right" terminology.

Define $\langle\_ \mid \_\rangle$ by

$$\langle a \mid b\rangle \mathrel{\hat{=}} \langle a] \fatsemi [b\rangle$$

We call this the two-argument embedding. Note that $\langle a] = \langle a \mid 1\rangle$ and $[a\rangle = \langle 1 \mid a\rangle$ because $[1\rangle = \ddot{1}$ and $\ddot{1}$ is the identity element of $\fatsemi$. Another immediate consequence is that $\langle\_ \mid \_\rangle$ is a homomorphism to $\mathbb{A}$ from the product KAT $\mathbb{A} \times \mathbb{A}$, with the additional property that $\langle 0 \mid a\rangle = 0 = \langle a \mid 0\rangle$ for all $a \in \mathbb{A}$.[3] In case $\mathbb{A}$ is a relational model of KAT, the elements of $\mathbb{A} \times \mathbb{A}$ are pairs of relations on some set Σ. By contrast, the relational BiKAT over $\mathbb{A}$ will comprise relations on $\Sigma \times \Sigma$ (Sect. 4.2).

Being KAT homomorphisms, the embeddings send tests to tests. For tests p, q in $\mathbb{A}$ we have $\neg\langle p \mid q\rangle = \neg(\langle p] \fatsemi [q\rangle) = \langle \neg p] \oplus [\neg q\rangle$ by homomorphism and de Morgan.

We use identifiers $A, B, C \ldots$ for elements of $\ddot{\mathbb{A}}$ and $P, Q, R, S \ldots$ for ***bitests***, i.e., elements of $\ddot{\mathbb{B}}$. While we use fancy symbols $\ddot{1}, \oplus, \fatsemi, \ldots$ in Def. 4.1 to distinguish between the BiKAT's operations and those of the underlying KAT, we usually use the simpler $+, \cdot$ or $;$, $\neg$, etc., for both levels of a BiKAT since the types can be inferred from context.

The $\forall\forall$ judgment $c \mid c' : R \approx\!\!> S$ can be interpreted in any BiKAT, provided R and S are bitests and c and c' are actions in the underlying KAT. It is like the Hoare triple equation:

$$R; \langle c \mid c'\rangle; \neg S = 0 \tag{2}$$

Defining the judgment this way is justified in terms of the standard models (see Sect. 4.2 and Theorem 4.6).

In applications of KAT, one reasons under hypotheses that specify the interpretation of primitive tests and actions. For many applications it suffices to use hypotheses in the form of Hoare triples. In applications of BiKAT we can use relational Hoare triple hypotheses to specify the interpretation of primitive bitests, usually with respect to embedded unary actions. The key point is to leverage the use of such hypotheses by rewriting the given verification task into a conveniently aligned form. This can be done in ad hoc ways but several general patterns can be identified.

[3] An equivalent way to define BiKAT is to take $\langle\_ \mid \_\rangle$ as primitive and define $\langle a] = \langle a \mid 1\rangle$ and $[a\rangle = \langle 1 \mid a\rangle$. Then we get left-right commutativity because $\langle a]; [b\rangle = \langle a \mid 1\rangle; \langle 1 \mid b\rangle = \langle a; 1 \mid 1; b\rangle = \langle 1; a \mid b; 1\rangle = \langle 1 \mid b\rangle; \langle a \mid 1\rangle = [b\rangle; \langle a]$. For $\langle\_]$ and $[\_\rangle$ defined this way to be homomorphisms requires the property $\langle 0 \mid a\rangle = 0 = \langle a \mid 0\rangle$ since in $\mathbb{A} \times \mathbb{A}$ the values $(a, 0)$, $(0, 0)$, and $(0, a)$ are different in general. Since left-right commutativity is the key property we care about, we choose a formulation that highlights it.

The following general law is useful for reasoning about two loops while e do c and while e' do c'. They are aligned lockstep, under a common star, with a remainder to account for whichever may iterate longer.[4] Here e, e' are tests in the underlying KAT.

$$\langle (e;c)^* \mid (e';c')^* \rangle ; \langle \neg e \mid \neg e' \rangle = \langle e;c \mid e';c' \rangle^* ; (\langle e;c \mid \neg e' \rangle^* + \langle \neg e \mid e';c' \rangle^*) ; \langle \neg e \mid \neg e' \rangle \tag{3}$$

This holds in any BiKAT, i.e., it follows by equational reasoning from Def. 4.1. (see appendix of Antonopoulos et al. [2022a]). Note that $\langle (e;c)^* \mid (e';c')^* \rangle ; \langle \neg e \mid \neg e' \rangle = \langle (e;c)^* ; \neg e \mid (e';c')^* ; \neg e' \rangle$ by LRC.

The standard models of BiKAT have additional structure that we formalize as follows.

Definition 4.2. A ***BiKAT with projections*** is a BiKAT together with total functions $\swarrow : \ddot{\mathbb{A}} \to \mathbb{A}$ and $\searrow : \ddot{\mathbb{A}} \to \mathbb{A}$ to its underlying KAT, that satisfy the following for all a in $\mathbb{A}$ and A, B in $\ddot{\mathbb{A}}$.

(Inversion)	$\swarrow\langle a] = a$	$\searrow[a\rangle = a$
(Disjointness)	$a \neq 0 \Rightarrow \swarrow[a\rangle = \ddot{1}$	$a \neq 0 \Rightarrow \searrow\langle a] = \ddot{1}$
(Disjunctivity)	$\swarrow(A \oplus B) = \swarrow A + \swarrow B$	$\searrow(A \oplus B) = \searrow A + \searrow B$

Projections are not required to distribute over sequence or star. For most of our development, the projections are not needed. However, they exist in the standard models and they do serve a purpose that raises open problems discussed in Sect. 7.6.

For BiKATs with projections, we get the following easy consequences. Only the first condition is expressed using projection, but all of them are proved using projection properties.

Lemma 4.3. *In any BiKAT with projections, we have*

(Unit)	$\swarrow\ddot{1} = 1 = \searrow\ddot{1}$
(Separation)	$\langle a] = [b\rangle \wedge (a \neq 0 \vee b \neq 0) \Rightarrow a = b = 1$
Order Separation)	$\langle a] \geq [b\rangle \wedge a \neq 0 \neq b \Rightarrow a \geq 1 \wedge 1 \geq b$
(Injectivity)	$\langle a] = \langle b] \Rightarrow a = b$ and same for $[\_\rangle$
(Order-Injectivity)	$\langle a] \geq \langle b] \Rightarrow a \geq b$ and same for $[\_\rangle$

An attractive aspect of KAT is that implications $hypoth \Rightarrow c = d$ are decidable provided the hypotheses are of the form $b = 0$ which includes the KAT encoding of Hoare triples. On the other hand, implications with general commutativity hypotheses are undecidable [Kozen and Smith 1996]. One proof goes by reduction to the Post correspondence problem, and the proof idea can be adapted to BiKAT embeddings in such a way that the commutativity conditions are expressed by LRC (see appendix of Antonopoulos et al. [2022a]). It should be noted that the original argument is presented for $*$-continuous KATs and as such our proof is over $*$-continuous BiKATs.

Theorem 4.4. *It is undecidable whether a given identity holds in all $*$-continuous BiKATs.*

We do not need decidability in order to derive useful alignments for specific programs, and to derive general laws like (3). Consider a given verification problem $R; \langle c \mid c' \rangle ; \neg S = 0$. Alignment reasoning, using LRC and its consequences, transforms it to a problem of the form $R; B; \neg S = 0$ that is proved using unary and relational Hoare-triple hypotheses. For that matter, having used BiKAT to obtain well-aligned B, verification subtasks in B can be solved by whatever method you like.

4.2 Models of BiKAT

Elements of a relational or trace KAT can be seen as sets of observations, where an observation may be a proper trace or just a pair (initial,final) of states. Elements of a BiKAT are meant to be sets of observation pairs. We will describe trace models of BiKAT this way. For relational BiKATs,

[4] This is the most flexible loop alignment available in some systems, e.g., Cartesian Hoare logic [Sousa and Dillig 2016].

however, we use a slightly different representation that facilitates spelling out connections with relational Hoare logics etc.

First, some definitions. Let R and S be relations on some set Σ. Define $R \otimes S$ to be the relation on $\Sigma \times \Sigma$ such that $(\sigma, \sigma')(R \otimes S)(\tau, \tau')$ iff $\sigma R \tau$ and $\sigma' S \tau'$. If R and S are the denotations of two programs, $R \otimes S$ relates a pair (σ, σ') interpreted as initial states to a final pair (τ, τ'). (To strictly model the idea of observation pair, one would use $((\sigma, \tau), (\sigma', \tau'))$ instead of $((\sigma, \sigma'), (\tau, \tau'))$.)

For relation R on Σ, the relations $\langle R]$ and $[R\rangle$ on $\Sigma \times \Sigma$ are defined by $\langle R] = R \otimes id_\Sigma$ and $[R\rangle = id_\Sigma \otimes R$. (We write id or id_Σ for the identity relation on Σ.) In terms of states, we have

$$(\sigma, \sigma')\langle R](\tau, \tau') \;\Leftrightarrow\; \sigma R \tau \wedge \sigma' = \tau' \qquad (\sigma, \sigma')[R\rangle(\tau, \tau') \;\Leftrightarrow\; \sigma' R \tau' \wedge \sigma = \tau$$

For a relation $\mathcal{R}$ on $\Sigma \times \Sigma$, define relations $\swarrow\mathcal{R}$ and $\searrow\mathcal{R}$ on Σ by

$$\sigma(\swarrow\mathcal{R})\tau \;\Leftrightarrow\; \exists \sigma', \tau'.\, (\sigma, \sigma')\, \mathcal{R}\, (\tau, \tau') \qquad \sigma'(\searrow\mathcal{R})\tau' \;\Leftrightarrow\; \exists \sigma, \tau.\, (\sigma, \sigma')\, \mathcal{R}\, (\tau, \tau') \tag{4}$$

Equivalently, $\swarrow\mathcal{R} \mathrel{\hat{=}} fst^o; \mathcal{R}; fst$ and $\searrow\mathcal{R} \mathrel{\hat{=}} snd^o; \mathcal{R}; snd$ where $fst : \Sigma \times \Sigma \to \Sigma$ and $snd : \Sigma \times \Sigma \to \Sigma$ are the projection functions, treated as relations so we can use the converse operation (written ^{o} as in [Freyd and Scedrov 1990]) and relational composition (written ;).

A ***relational BiKAT*** over a relational model $(\mathbb{A}, \mathbb{B}, +, \cdot, *, \neg, 1, 0)$ is $(\ddot{\mathbb{A}}, \ddot{\mathbb{B}}, \oplus, \mathbin{⨟}, \circledast, \ddot{\neg}, \ddot{1}, \ddot{0})$ such that $\ddot{\mathbb{A}}$ is a set of relations on $\Sigma \times \Sigma$ and the structure on $\ddot{\mathbb{A}}$ is a relational model of KAT. Moreover, both $\langle R]$ and $[R\rangle$ are in $\ddot{\mathbb{A}}$, for any R in $\mathbb{A}$, and both $\swarrow\mathcal{R}$ and $\searrow\mathcal{R}$ are in $\mathbb{A}$, for any $\mathcal{R}$ in $\ddot{\mathbb{A}}$. Note that a relational BiKAT is a BiKAT with projections.

Given a relation R on states, we sometimes write $\dot{R}$ for the associated sub-identity in $\ddot{\mathbb{B}}$, i.e., $(\sigma, \sigma')\dot{R}(\tau, \tau')$ iff $\sigma R \sigma' \wedge \tau = \sigma \wedge \tau' = \sigma'$. Please note that $\dot{id_\Sigma}$ is different from $\ddot{1}$.

For an example, say a relation R is boundedly nondeterministic if for any σ there are finitely many τ with $\sigma R \tau$. Take $\mathbb{A}$ to be all boundedly nondeterministic relations on Σ and $\ddot{\mathbb{A}}$ to be all boundedly nondeterministic relations on $\Sigma \times \Sigma$.

A ***full*** relational BiKAT is one where the BiKAT and its underlying KAT are full relational models. The example of boundedly nondeterministic relations is not full; both the underlying KAT and the BiKAT lack a top.

For programs c, d (interpreted as relations on Σ) and relations R, S on Σ meant to be pre- and post-conditions, the $\forall\forall$ judgment $c \mid d : R \mathrel{\ddot{\approx}} S$ is pictured

$$\forall \quad \begin{array}{ccc} \sigma & \xmapsto{\;c\;} & \tau \\ {\scriptstyle R}\big\downarrow & & \\ \sigma' & \xmapsto{\;d\;} & \tau' \end{array} \quad \Longrightarrow \quad \begin{array}{c} \tau \\ \big\downarrow{\scriptstyle S} \\ \tau' \end{array} \tag{5}$$

This is meant to say $\forall \sigma, \sigma', \tau, \tau'.\ \sigma R \sigma' \wedge \sigma c \tau \wedge \sigma' d \tau' \Rightarrow \tau S \tau'$. By definitions we have the following.

Lemma 4.5 (adequacy for relational models). *In a relational BiKAT, for any c, d, the BiKAT term $\langle c \mid d\rangle$ denotes the set of all pairs of c, d computations.*[5]

Now we can confirm that defining the judgment $c \mid d : R \mathrel{\ddot{\approx}} S$ as the BiKAT equation (2) captures the semantic condition of (5). Moreover, BiKAT equality preserves adequacy.

Theorem 4.6 ($\forall\forall$ soundness for relational models). *Suppose that c, d are elements of the underlying KAT of a relational BiKAT, and suppose the sub-identity relations $\dot{R}, \dot{S}$ that represent R, S are among the BiKAT's tests. Then*

(a) *The condition (5) holds iff $\dot{R}; \langle c \mid d\rangle; \neg\dot{S} = 0$.*
(b) *For any B, if $\dot{R}; \langle c \mid d\rangle \leq \dot{R}; B$ and $\dot{R}; B; \neg\dot{S} = 0$ then (5) holds.*

[5]To be precise, the set of all $((\sigma, \sigma'), (\tau, \tau'))$ where $(\sigma, \tau) \in c$ and $(\sigma', \tau') \in d$.

Proof. For (a) the proof is by mutual implication. If (5) holds then $\dot{R};\langle c \mid d\rangle \le \dot{R};\langle c \mid d\rangle;\dot{S}$ by definitions. And $\dot{R};\langle c \mid d\rangle \le \dot{R};\langle c \mid d\rangle;\dot{S}$ is equivalent to $\dot{R};\langle c \mid d\rangle;\neg\dot{S} = 0$ as a fact about KATs. For the converse, assume $\dot{R};\langle c \mid d\rangle \le \dot{R};\langle c \mid d\rangle;\dot{S}$. To show (5), for any states $\sigma, \sigma', \tau, \tau'$ that satisfy the antecedent in (5), by adequacy Lemma 4.5 the executions are in $\dot{R};\langle c \mid d\rangle$, so by assumption they are in $\dot{R};\langle c \mid d\rangle;\dot{S}$. So by definitions the post states are related by S.

For Part (b), if $\dot{R};\langle c \mid d\rangle \le \dot{R};B$ then we have that $\dot{R};B;\neg\dot{S} \le 0$ implies $\dot{R};\langle c \mid d\rangle;\neg\dot{S} \le 0$ by KAT reasoning. This yields (5) using Part (a). □

Part (b) of Theorem 4.6 says that adequacy of B for proving $c \mid d : R \approx\!> S$ is expressed by the equation $\dot{R};\langle c \mid d\rangle \le \dot{R};B$ (called "R-adequacy" in Nagasamudram and Naumann [2021]). An adequacy proof can be interwoven with the correctness proof, in a form like $\dot{R};\langle c \mid d\rangle;\neg\dot{S} \le \ldots \le \dot{R};B;\neg\dot{S} = \ldots = 0$. Some of our examples have this form, using equality not $\le$.

A ***trace BiKAT*** over trace model $(\mathbb{A}, \mathbb{B}, +, \cdot, *, \neg, 1, 0)$ is $(\ddot{\mathbb{A}}, \ddot{\mathbb{B}}, \oplus, \fatsemi, \circledast, \ddot{\neg}, \ddot{1}, \ddot{0})$ where elements of $\ddot{\mathbb{A}}$ are sets of pairs of traces. To define $\fatsemi$, the coalesced catenation $\diamond$ is lifted to trace pairs and used as in ordinary trace models. That is, $A \fatsemi B = \{(t \diamond u, t' \diamond u') \mid (t,t') \in A, (u,u') \in B, t \diamond u \text{ and } t' \diamond u' \text{ defined}\}$. Sum and star are union and iterated $\fatsemi$. For trace set a, the left embedding $\langle a]$ is $\{(t,\sigma) \mid t \in a \wedge \sigma \in \Sigma\}$. For set A of trace pairs, the left projection $\swarrow A$ is $\{t \mid (t,u) \in A\}$, i.e., $\swarrow$ maps *fst* over A. As with relational BiKATs, we require that $\ddot{\mathbb{A}}$ contains all the images of $\langle\_]$ and $[\_\rangle$ on $\mathbb{A}$, and $\mathbb{A}$ contains the images of $\swarrow$ and $\searrow$ on $\ddot{\mathbb{A}}$.

To see why LRC holds in a trace BiKAT, let us write $\triangleleft t$ for the first state of trace t and $t \triangleright$ for the last. For trace sets c, c', and any $t \in c$ and $t' \in c'$, we have $(t,t') \in \langle c] \fatsemi [c'\rangle$ iff $(t, \triangleleft t') \in \langle c]$ and $(\triangleright t, t') \in [c'\rangle$. Note that $(t, \triangleleft t') \diamond (\triangleright t, t') = (t,t')$. Similarly, we have $(t,t') \in [c'\rangle \fatsemi \langle c]$. The upshot is that $\langle c] \fatsemi [c'\rangle = [c'\rangle \fatsemi \langle c]$.

A ***full trace BiKAT*** has all trace sets (relative to the given set of admissible traces), for the underlying KAT, and all pairs of traces for the BiKAT. Like full relational models, full trace models have a top.

Lemma 4.5 and Theorem 4.6 can be straightforwardly adapted to trace models.

5 USING BIKAT FOR ∀∀ RELATIONAL REASONING

Having introduced BiKAT we now demonstrate how it can be used to algebraically derive alignments and verify ∀∀ properties of a variety of examples that necessitate different kinds of alignment.

Simple Example. The following two programs compute the sum of integers up to some N.

$$C_1 \triangleq \quad \texttt{i:=0;}\ \ ([\texttt{i} \le \texttt{N}];\ \texttt{x:=x+i;}\ \texttt{i:=i+1})^*\ ;\ \neg[\texttt{i} \le \texttt{N}]$$
$$C_2 \triangleq \quad \texttt{i:=1;}\ \ ([\texttt{i} \le \texttt{N}];\ \texttt{x:=x+i;}\ \texttt{i:=i+1})^*\ ;\ \neg[\texttt{i} \le \texttt{N}]$$

To prove the ∀∀ relational judgment $C_1 \mid C_2 : \langle[\texttt{N} \ge 0]][\texttt{x} \ddot{=} \texttt{x}][\texttt{N} \ddot{=} \texttt{N}] \approx\!> [\texttt{x} \ddot{=} \texttt{x}]$ without resorting to functional correctness, we start from $\langle[\texttt{N} \ge 0]] \cdot [\texttt{x} \ddot{=} \texttt{x}] \cdot [\texttt{N} \ddot{=} \texttt{N}] \cdot \langle C_1 \mid C_2\rangle \cdot \neg[\texttt{x} \ddot{=} \texttt{x}] = 0$ and then manipulate the sequential composition $\langle C_1 \mid C_2\rangle$ into an alignment, where we can directly relate the programs' variables during loop iterations. In this case, there is a simple alignment: C_1 does one more iteration than C_2, so we unroll its loop once before aligning the two loop bodies in lockstep. Then, a simple relational loop invariant, $[\texttt{x} \ddot{=} \texttt{x}][\texttt{i} \ddot{=} \texttt{i}]$, suffices to establish equivalence.

Let us now see this step-by-step in the algebra of BiKAT. Unrolling C_1's loop once, we have that $\langle C_1 \mid C_2\rangle$ is equal to:

$$\langle \texttt{i:=0;}\ (1 + C; C^*)\ ; \neg e \mid \texttt{i:=1;} C^*; \neg e\rangle$$

where C is the program text $(e;\ \texttt{x:=x+i;i:=i+1})$ and e is $[\texttt{i} \le \texttt{N}]$. Distributing, we get

$$\langle \texttt{i:=0;} \neg e + \texttt{i:=0;} C; C^*; \neg e \mid \texttt{i:=1;} C^*; \neg e\rangle$$

For semantic reasons, $(\texttt{i:=0};\neg e)$ is infeasible. So we can assume hypothesis $(\texttt{i:=0};\neg e) = 0$ which lets us eliminate that term. For the other term, we calculate

$$\begin{array}{lll} & \langle \texttt{i:=0};C;C^*;\neg e \mid \texttt{i:=1};C^*;\neg e\rangle & \\ = & \langle \texttt{i:=0} \mid \texttt{i:=1}\rangle;\langle C\rangle;\langle C^* \mid C^*\rangle;\langle \neg e \mid \neg e\rangle & \text{embedding homomorphic} \\ = & \langle \texttt{i:=0}\rangle;[\texttt{i:=1}\rangle;\langle C\rangle;\langle C^*\rangle;[C^*\rangle;\langle \neg e \mid \neg e\rangle & \text{def two-argument embedding} \\ = & \langle \texttt{i:=0}\rangle;\langle C\rangle;[\texttt{i:=1}\rangle;\langle C^*\rangle;[C^*\rangle;\langle \neg e \mid \neg e\rangle & \text{LRC} \\ = & \langle \texttt{i:=0};C \mid \texttt{i:=1}\rangle;\langle C^* \mid C^*\rangle;\langle \neg e \mid \neg e\rangle & \text{embedding homomorphic} \end{array}$$

Using the expansion lemma (3), $\langle C^* \mid C^*\rangle$ can be rewritten into

$$(\langle [\texttt{i} \leq \texttt{N}];\ \texttt{x:=x+i};\ \texttt{i:=i+1} \mid [\texttt{i} \leq \texttt{N}];\ \texttt{x:=x+i};\ \texttt{i:=i+1}\rangle)^*$$

(Eliding terms that cancel out once we introduce the loop invariant.) By systematically using LRC along with the homomorphism property of embeddings, we align the loop bodies in lockstep:

$$(\langle [\texttt{i} \leq \texttt{N}] \mid [\texttt{i} \leq \texttt{N}]\rangle;\ \langle \texttt{x:=x+i} \mid \texttt{x:=x+i}\rangle;\ \langle \texttt{i:=i+1} \mid \texttt{i:=i+1}\rangle)^*$$

We next add the assumption that $[\texttt{x} \doteq \texttt{x}]$ initially, add relational loop invariant $[\texttt{i} \doteq \texttt{i}][\texttt{x} \doteq \texttt{x}]$, and conclude the post-relation $[\texttt{x} \doteq \texttt{x}]$ must hold beyond the loop.

Double Square (Example 2.2, Sect. 2). Recall the KAT expressions of the two programs:

$$\begin{array}{l} k_{D_1} \mathrel{\hat{=}} \texttt{y:=0;z:=2*x;}([\texttt{z>0}];\texttt{z:=z-1;y:=y+x})^*;\overline{[\texttt{z>0}]} \\ k_{D_2} \mathrel{\hat{=}} \texttt{y:=0;z:=x;}([\texttt{z>0}];\texttt{z:=z-1;y:=y+x})^*;\overline{[\texttt{z>0}]};\texttt{y:=2*y} \end{array}$$

Note we are using overline as alternate notation for negation.

In the preceding example, we did a unary unfolding of one iteration on the left, and then aligned the loops in lockstep. For this example we choose to align two iterations on the left with one iteration on the right. We first use KAT laws to rewrite k_{D_1} as:

$$k_{D_1} = \texttt{y:=0;z:=2*x;}([\texttt{z>0}];b;[\texttt{z>0}];b)^*;([\texttt{z>0}];b+1);\overline{[\texttt{z>0}]}$$

where $b \mathrel{\hat{=}} \texttt{z:=z-1;y:=y+x}$. We then align the two loops and apply the expansion law (3) as follows:

$$\begin{array}{ll} & \langle([\texttt{z>0}];b;[\texttt{z>0}];b)^* \mid ([\texttt{z>0}];b)^*\rangle \\ = & \langle [\texttt{z>0}];b;[\texttt{z>0}];b;[\texttt{z>0}];b \mid [\texttt{z>0}];b\rangle^*;(\langle [\texttt{z>0}];b;[\texttt{z>0}];b \mid \overline{[\texttt{z>0}]}\rangle^* + \langle \overline{[\texttt{z>0}]} \mid [\texttt{z>0}];b\rangle^*) \end{array}$$

Now it is easier to prove the relational loop invariant $\ddot{I} \mathrel{\hat{=}} [y \doteq 2y];[z \doteq 2z]$ by proving that it is preserved across the above expansion loops. In the derivation, we break the proof into sub-proofs over the embeddings of loop bodies:

$$\ddot{I};\langle([\texttt{z>0}];b;[\texttt{z>0}];b)^* \mid ([\texttt{z>0}];b)^*\rangle;\neg\ddot{I} = 0 \impliedby \begin{cases} \ddot{I};\langle [\texttt{z>0}];b;[\texttt{z>0}];b \mid [\texttt{z>0}];b\rangle;\neg\ddot{I} = 0 \wedge \\ \ddot{I};\langle [\texttt{z>0}];b;[\texttt{z>0}];b \mid \overline{[\texttt{z>0}]}\rangle;\neg\ddot{I} = 0 \wedge \\ \ddot{I};\langle \overline{[\texttt{z>0}]} \mid [\texttt{z>0}];b\rangle;\neg\ddot{I} = 0 \end{cases}$$

Loop Tiling (Example 2.3, Sect. 2). Here are the two programs L_1, L_2 as KAT expressions:

$$\begin{array}{l} k_{L_1} \mathrel{\hat{=}} \texttt{x:=0;}([\texttt{x<N*M}];\texttt{a[x]:=f(x);x:=x+1})^*;\overline{\texttt{x<N*M}} \\ k_{L_2} \mathrel{\hat{=}} \texttt{i:=0;}([\texttt{i<N}];\texttt{j:=0;}([\texttt{j<M}];\texttt{A[i,j]:=f(i*M+j);j:=j+1})^*;\overline{[\texttt{j<M}]};\texttt{i:=i+1})^*;\overline{[\texttt{i<N}]} \end{array}$$

For the alignment, we then transform these KAT expressions into two equivalent KAT expressions which have the same structure.

$$\begin{array}{l} k_{L_1} = \texttt{x:=0;}([\texttt{x<N*M}];b_1;([\texttt{x<N*M}];[\texttt{x\%M!=0}];b_1)^*;\overline{[\texttt{x<N*M}];[\texttt{x\%M!=0}]})^*;\overline{\texttt{x<N*M}} \\ k_{L_2} = \texttt{i:=0;}([\texttt{i<N}];\texttt{j:=0;}([\texttt{j<M}];b_2 + \overline{[\texttt{j<M}]});([\texttt{j<M}];b_2)^*;\overline{[\texttt{j<M}]};\texttt{i:=i+1})^*;\overline{[\texttt{i<N}]} \end{array}$$

where $b_1 \triangleq$ `a[x]:=f(x);x:=x+1` and $b_2 \triangleq$ `A[i,j]:=f(i*M+j);j:=j+1`. We next prove the judgment $\ddot{P};\langle k_{L_1} \mid k_{L_2}\rangle;\neg\ddot{Q} = 0$ with the pre-relation $\ddot{P} \triangleq [\mathtt{N} \ddot{=} \mathtt{N}];[\mathtt{M} \ddot{=} \mathtt{M}];[\mathtt{N} > 0];[\mathtt{M} > 0]$ and the post-relation $\ddot{Q} \triangleq [\mathcal{R}(\mathtt{N} \times \mathtt{M}, \mathtt{N}, \mathtt{M})]$, where $\mathcal{R}(x, i, j)$ is a predicate[6] that says every element of the array `a` up to the index x is equal to its corresponding element of the two-dimensional array `A` up to the index i, j. To prove the above judgment, we use the relational invariant $\ddot{\mathcal{I}} \triangleq [\mathtt{i} < \mathtt{N}];[\mathtt{j} \leq \mathtt{M}];[\mathtt{x} \ddot{=} \mathtt{i} \times \mathtt{M} + \mathtt{j}];[\mathcal{R}(\mathtt{x}, \mathtt{i}, \mathtt{j})]$ for the inner loops and the relational invariant $\ddot{\mathcal{J}} \triangleq [\mathtt{i} \leq \mathtt{N}];[\mathtt{x} \ddot{=} \mathtt{i} \times \mathtt{M}];[\mathcal{R}(\mathtt{x}, \mathtt{i}, 0)]$ for the outer loops.

Loop Summaries and Procedure Calls. KAT is a propositional theory of imperative control struture. In KAT-based systems, first order state variables, conditions, and assignments can be handled using hypotheses (typically, Hoare triples) that axiomatize their semantics.

```
arrayInsert (A, len, h) {
  i := 0;
  while (i<len && A[i]<h) i++;
  len := shift_array(A, i, 1);
  A[i] := h;
  while (i<len) i++;
  return i;
}
```

Here too, the KATs embedded in a BiKAT are parametric over alphabets of actions and tests and hypotheses about those tests. Consequently BiKAT inherently supports reasoning at coarser or finer granularies. Moreover, this parameterization allows BiKAT to be used in concert with other procedures (e.g. loop summarization, procedure specs/summaries, ghost states, prophecy variables, etc.) which could be applied beforehand and then incorporated into a BiKAT through the primitive actions and hypothesis. For example, consider the array insertion procedure[7] [Goyal et al. 2021; Shemer et al. 2019] shown to the right that will illustrate the use of externally-provided unary procedure summaries. The goal is to prove that, `arrayInsert` | `arrayInsert` : $[\mathtt{len} \ddot{=} \mathtt{len}][\mathtt{A} \ddot{=} \mathtt{A}] \lessapprox [\mathtt{i} \ddot{=} \mathtt{i}]$ i.e., the noninterference property that there is no leak on `h`.

Contrary to some examples above, this example does not require loop bodies to be lock-step aligned. It does, however, require alignment between intermediate points between the loops. Specifically, the programs must be aligned in three places: after both have completed their first loop, after both have completed the call to `shift_array` (incorporating that method's post-condition), and after both of completed their second loop. That is, the BiKAT alignment:

$$[\mathtt{len} \ddot{=} \mathtt{len}];\langle \mathtt{while(...)i++} \mid \mathtt{while(...)i++}\rangle;[\mathtt{len} \ddot{=} \mathtt{len}];\langle[\mathtt{i<=len}] \mid [\mathtt{i<=len}]\rangle;$$
$$\langle \mathtt{len := shift\_array(...)} \mid \mathtt{len := shift\_array(...)}\rangle;[\mathtt{len+1} \ddot{=} \mathtt{len+1}];$$
$$\langle \mathtt{while(i<len)i++};[\mathtt{i=len}]];[\mathtt{while(i<len)i++};[\mathtt{i=len}]\rangle;[\mathtt{i} \ddot{=} \mathtt{i}]$$

We first use the pre-relation $[\mathtt{len} \ddot{=} \mathtt{len}]$ and the postcondition of the first loop on both the left and right sides, aligning when both sides have completed to show that $[\mathtt{len} \ddot{=} \mathtt{len}]$ still holds, while $\mathtt{i} \leq \mathtt{len}$ (on the left) and $\mathtt{i}' \leq \mathtt{len}'$ (on the right).

At this point we have established the alignment necessary for this example. Completing the proof requires small semantic hypotheses (similar to those in earlier examples) and some strategy for establishing $[\mathtt{len} \ddot{=} \mathtt{len}]$ is preserved across the procedure call to `shift_array`. One option is to introduce relational hypothesis `shift_array(A,i,j)` | `shift_array(A,i,j)` : $[\mathtt{len} \ddot{=} \mathtt{len}][\mathtt{j} \ddot{=} \mathtt{j}] \lessapprox [\mathtt{len} \ddot{=} \mathtt{len}]$ which ensures agreement on `len` after the embedded procedures. Alternatively, one could employ a unary specifications of the form {}`shift_array(A,i,j)`{`len=old(len)+j`} through the use of KAT hypotheses of the form $pC\neg q = 0$ embedded on the left and the right. These unary post-conditions can be combined with the pre-call $[\mathtt{len} \ddot{=} \mathtt{len}]$ to add a post-call bitest $[\mathtt{len} \ddot{=} \mathtt{len}]$. Here there are some details that would be needed (*e.g.* ghost variables) to support post-condition

[6] $\mathcal{R}(x, i, j) \triangleq \forall l, r, c.\ 0 \leq l < x \wedge 0 \leq r < i \wedge (r < i-1 \wedge 0 \leq c < \mathtt{M} \vee r = i-1 \wedge 0 \leq c < j) \wedge l = r \times \mathtt{M} + c \Rightarrow [\mathtt{a}[l] \ddot{=} \mathtt{A}[r, c]]$

[7] The second loop sets `i` to `len` in a way that avoids a timing channel but we are not modeling timing here.

$$\textsc{dSeq}\ \frac{c \mid c' : P \mathrel{\approx\!\!>} R \qquad d \mid d' : R \mathrel{\approx\!\!>} Q}{c; d \mid c'; d' : P \mathrel{\approx\!\!>} Q}$$

$$\textsc{dIf}\ \frac{P \Rightarrow e \ddot{=} e' \qquad c \mid c' : P \wedge \langle e \rangle \wedge [e'\rangle \mathrel{\approx\!\!>} Q \qquad d \mid d' : P \wedge \neg\langle e \rangle \wedge \neg[e'\rangle \mathrel{\approx\!\!>} Q}{\text{if } e \text{ then } c \text{ else } d \mid \text{if } e' \text{ then } c' \text{ else } d' : P \mathrel{\approx\!\!>} Q}$$

$$\textsc{dWh}\ \frac{P \Rightarrow e \ddot{=} e' \qquad c \mid c' : P \wedge \langle e \rangle \wedge [e'\rangle \mathrel{\approx\!\!>} P}{\text{while } e \text{ do } c \mid \text{while } e' \text{ do } c' : P \mathrel{\approx\!\!>} P \wedge \neg\langle e \rangle \wedge \neg[e'\rangle} \qquad \textsc{rDisj}\ \frac{c \mid d : P \mathrel{\approx\!\!>} Q \qquad c \mid d : R \mathrel{\approx\!\!>} Q}{c \mid d : P \vee R \mathrel{\approx\!\!>} Q}$$

$$\textsc{SeqSk}\ \frac{c \mid \mathsf{skip} : P \mathrel{\approx\!\!>} R \qquad d \mid \mathsf{skip} : R \mathrel{\approx\!\!>} Q}{c; d \mid \mathsf{skip} : P \mathrel{\approx\!\!>} Q} \qquad \textsc{rConseq}\ \frac{R \Rightarrow P \qquad c \mid d : P \mathrel{\approx\!\!>} Q \qquad Q \Rightarrow S}{c \mid d : R \mathrel{\approx\!\!>} S}$$

Fig. 1. Selected inference rules of ∀∀ logic

tests that relate variables to pre-conditions. Finally, we use $[\texttt{len} \ddot{=} \texttt{len}]$ with the post-conditions of the last loops that $\texttt{i} = \texttt{len}$ on both sides to conclude that $[\texttt{i} \ddot{=} \texttt{i}]$.

6 RELATIONAL HOARE LOGIC IN BIKAT

In this section we show that relational Hoare logic rules can be derived in any BiKAT. Relational logics involve two programs, thus quadruples, sometimes written $\{P\}c \sim c'\{Q\}$ for commands c, c'. Benton [2004] writes $c \sim c' : P \Rightarrow Q$. We consider inference rules for the ∀∀ judgment form $c \mid c' : P \mathrel{\approx\!\!>} Q$ introduced in Sect. 1 and expressed in any BiKAT by the equation (2).

Deriving Rules of RHL. A number of publications have presented variations on relational Hoare logic. We consider a number of basic rules that can be found in Benton's influential paper [Benton 2004] and in Francez' less known paper [Francez 1983], and a number of subsequent works. There is not yet a standard set of rules, in part because until recently there was no satisfactory notion of completeness [Nagasamudram and Naumann 2021; Naumann 2020]. We consider a number of representative rules in Fig. 1.

In the rules we use suggestive syntax for formulas and programs, and we will not belabor the distinction between program syntax and its standard representation in KAT. We lift boolean expression e to a relation formula $\langle e]$ that says e is true in the left state, so its representation as a test in BiKAT will look the same.

We will show that all the rules are sound in any BiKAT. Recall that we interpret the judgment $c \mid c' : P \mathrel{\approx\!\!>} Q$ as the BiKAT equation $P; \langle c \mid c'\rangle; \neg Q = 0$. In the soundness proofs we use the equivalent form $P; \langle c \mid c'\rangle \leq P; \langle c \mid c'\rangle; Q$. This form is also used in Kozen's work deriving Hoare logic rules in KAT [Kozen 2000].

Several rules in Fig. 1 infer "diagonal" judgments relating two same-structured programs, e.g., dSeq, dIf, and dWh. The latter two cater for alignment whereby the same control path is followed, with a requirement of agreement on conditional tests. There is a rule dIf4, named after the number of its premises, does not require such agreement.

Some of the rules can be derived from others. Regarding rule SeqSk, from its premises one can use dSeq to obtain $c; d \mid \mathsf{skip}; \mathsf{skip} : P \mathrel{\approx\!\!>} Q$. But the inference rules provide no way to replace skip; skip by the equivalent skip. One of the benefits of working in KAT is free use of such equivalences, including more interesting ones like loop unrolling in the Tiling example.

Rule dIf can be derived from dIf4 using rule FalsePre and rConseq, but we prove dIf directly. The side condition $P \Rightarrow e \ddot{=} e'$ is most directly expressed as $P \leq e \ddot{=} e'$. The left-right equality $e \ddot{=} e'$ is

equivalent to $\langle e \mid e' \rangle + \langle \neg e \mid \neg e' \rangle$ so the side condition yields

$$P; \langle e \mid \neg e' \rangle = 0 \qquad P; \langle \neg e \mid e' \rangle = 0 \tag{6}$$

To prove soundness of DIF we calculate:

$$\begin{array}{lll}
 & P; \langle e; c + \neg e; d \mid e'; c' + \neg e'; d' \rangle & \\
= & P; \langle e \mid e' \rangle; \langle c \mid c' \rangle + P; \langle e \mid \neg e' \rangle; \langle c \mid d' \rangle & \text{emb homo, distrib} \\
 & + P; \langle \neg e \mid e' \rangle; \langle d \mid c' \rangle + P; \langle \neg e \mid \neg e' \rangle; \langle d \mid d' \rangle & \\
= & P; \langle e \mid e' \rangle; \langle c \mid c' \rangle + P; \langle \neg e \mid \neg e' \rangle; \langle d \mid d' \rangle & \text{using (6)} \\
\leq & P; \langle e \mid e' \rangle; \langle c \mid c' \rangle; Q + P; \langle \neg e \mid \neg e' \rangle; \langle d \mid d' \rangle; Q & \text{premises of DIF} \\
= & P; \langle e; c + \neg e; d \mid e'; c' + \neg e'; d' \rangle; Q & \text{reverse steps}
\end{array}$$

To prove DWH, first observe

$$\begin{array}{lll}
 & P; (\langle e; c \mid \neg e' \rangle^* + \langle \neg e \mid e'; c' \rangle^*) & \\
= & P; (1 + \langle e; c \mid \neg e' \rangle; \langle e; c \mid \neg e' \rangle^*) + P; (1 + \langle \neg e \mid \neg e'; c' \rangle; \langle \neg e \mid e'; c' \rangle^*) & \text{distrib, star unfold} \\
= & P + P; \langle e; c \mid \neg e' \rangle; \langle e; c \mid \neg e' \rangle^* + P + P; \langle \neg e \mid \neg e'; c' \rangle; \langle \neg e \mid e'; c' \rangle^* & \text{distrib} \\
= & P & \text{using (6)}
\end{array}$$

The last step uses that embedding is homomorphic, and the side condition (6) whence $P{+}0{+}P{+}0 = P$. Using the premise $P; \langle e \mid e' \rangle; \langle c \mid c' \rangle \leq P; \langle e \mid e' \rangle; \langle c \mid c' \rangle; P$, we get the conclusion of DWH by

$$\begin{array}{lll}
 & P; \langle (e; c)^*; \neg e \mid (e'; c')^*; \neg e' \rangle & \\
= & P; \langle (e; c)^* \mid (e'; c')^* \rangle; \langle \neg e \mid \neg e' \rangle & \text{emb homo} \\
= & P; \langle e; c \mid e'; c' \rangle^*; (\langle e; c \mid \neg e' \rangle^* + \langle \neg e \mid e'; c' \rangle^*); \langle \neg e \mid \neg e' \rangle & \text{expansion (3)} \\
\leq & P; \langle e; c \mid e'; c' \rangle^*; P; (\langle e; c \mid \neg e' \rangle^* + \langle \neg e \mid e'; c' \rangle^*); \langle \neg e \mid \neg e' \rangle & \text{premise, invariance} \\
= & P; \langle e; c \mid e'; c' \rangle^*; P; \langle \neg e \mid \neg e' \rangle & \text{observation above} \\
= & P; \langle e; c \mid e'; c' \rangle^*; P; \langle \neg e \mid \neg e' \rangle; P & \text{tests idem, tests commute} \\
= & P; \langle e; c \mid e'; c' \rangle^*; P; (\langle e; c \mid \neg e' \rangle^* + \langle \neg e \mid e'; c' \rangle^*); \langle \neg e \mid \neg e' \rangle; P & \text{observation, in reverse} \\
\leq & P; \langle e; c \mid e'; c' \rangle^*; (\langle e; c \mid \neg e' \rangle^* + \langle \neg e \mid e'; c' \rangle^*); \langle \neg e \mid \neg e' \rangle; P & P \leq 1 \text{ since } P \text{ is a test} \\
= & P; \langle (e; c)^*; \neg e \mid (e'; c')^*; \neg e' \rangle; P; \langle \neg e \mid \neg e' \rangle & \text{reverse steps using (3)}
\end{array}$$

THEOREM 6.1. *The rules of Fig. 1 are sound in any BiKAT.*

To cater for reasoning about related loops where data-dependent alignment is needed, the following rule CAWH for *conditionally aligned loops* has been shown sound for specific models in prior work [Banerjee et al. 2022; Beringer 2011; Nagasamudram and Naumann 2021]. The rule features relations Q (resp. R) as conditions under which an iteration on one side is aligned with doing nothing on the other side.

$$\textsc{caWh}\ \frac{\begin{array}{c} c \mid c' : P \wedge \langle e \rangle \wedge [e'\rangle \wedge \neg Q \wedge \neg R \lessapprox P \qquad c \mid \mathsf{skip} : P \wedge Q \wedge \langle e \rangle \lessapprox P \\ \mathsf{skip} \mid c' : P \wedge R \wedge [e'\rangle \lessapprox P \qquad P \Longrightarrow e \ddot{=} e' \vee (Q \wedge \langle e \rangle) \vee (R \wedge [e'\rangle) \end{array}}{\mathsf{while}\ e\ \mathsf{do}\ c \mid \mathsf{while}\ e'\ \mathsf{do}\ c' : P \lessapprox P \wedge \neg\langle e \rangle \wedge \neg[e'\rangle}$$

The rule can be used together with the one-side rules like SEQSK. To prove CAWH we use this expansion law:

$$\begin{array}{lll}
\langle e; c]^*; \langle \neg e]; [e'; c'\rangle^*; [\neg e'\rangle & = & (Q; \langle e; c] + R; [e'; c'\rangle + \neg Q; \neg R; \langle e; c \mid e'; c'\rangle \\
 & & \quad + \neg Q; \langle e; c \mid \neg e'\rangle + \neg R; \langle \neg e \mid e'; c'\rangle)^*; \langle \neg e \mid \neg e'\rangle
\end{array} \tag{7}$$

We do not know whether (7) holds in all BiKATs, but it holds in relational BiKATs and trace BiKATs.

THEOREM 6.2. *(a) Law (7) holds in any *-continuous BiKAT.*
(b) Rule CAWH is sound in any BiKAT that satisfies (7).

Example 6.3. Consider the following, adapted from Naumann [2020].

P_1 : `y:=x; z:=24; w:=0; while y>4 do if w%2=0 then z:=z*y; y:=y-1 fi; w:=w+1 od`
P_2 : `y:=x; z:=16; w:=0; while y>4 do if w%3=0 then z:=z*2; y:=y-1 fi; w:=w+1 od`

For $\mathtt{x} \geq 4$, P_1 computes $\mathtt{x}!$ in $\mathtt{z}$ and P_2, $2^{\mathtt{x}}$ in $\mathtt{z}$. We want to show that P_1 majorizes P_2, i.e., $P_1 \mid P_2 : \mathtt{x} \ddot{=} \mathtt{x}' \mathrel{\ggg} \mathtt{z} > \mathtt{z}'$ (using primed variables to refer to those in P_2). Notice that both programs take gratuitous steps, making it difficult to reason by a simple lockstep alignment of the two loops—the relational loop invariant would become needlessly complicated. Verification can be simplified by using the following data-dependent alignment: if $\langle \mathtt{w\%2} \neq \mathtt{0}]$, perform a left-only iteration; if $[\mathtt{w\%3} \neq \mathtt{0}\rangle$, perform a right-only iteration; otherwise, if $\langle \mathtt{w\%2} = \mathtt{0} \mid \mathtt{w\%3} = \mathtt{0}\rangle$, execute the loop bodies jointly. Then, $\mathtt{y} \ddot{=} \mathtt{y}' \wedge \mathtt{z} > \mathtt{z}' > 0$ is invariant and sufficient to establish the postrelation. This reasoning is done using rule CAWH in Nagasamudram and Naumann [2021]. An alternative is to reason in BiKAT with the following alignment:

$$\langle \mathtt{y:=x} \mid \mathtt{y:=x}\rangle; \langle \mathtt{z:=24} \mid \mathtt{z:=16}\rangle; \langle \mathtt{w:=0} \mid \mathtt{w:=0}\rangle;$$
$$\big(\langle [\mathtt{w\%2} \neq \mathtt{0}]; \mathtt{w:=w+1}] + [[\mathtt{w\%3} \neq \mathtt{0}]; \mathtt{w:=w+1}\rangle$$
$$+ \langle [\mathtt{w\%2=0}] \mid [\mathtt{w\%3=0}]\rangle; \langle \mathtt{z:=z*y; y:=y-1; w:=w+1} \mid \mathtt{z:=z*2; y:=y-1; w:=w+1}\rangle; \big)^*; \langle \neg[\mathtt{y>4}] \mid \neg[\mathtt{y>4}]\rangle.$$

It can be derived by starting with $[\mathtt{x} \ddot{=} \mathtt{x}']; \langle P_1 \mid P_2\rangle$, using (7) with $Q := \langle \mathtt{w\%2} \neq \mathtt{0}]$ and $R := [\mathtt{w\%2} \neq \mathtt{0}\rangle$, and then simplifying, relying on the loop invariant which cancels the $\neg Q$ and $\neg R$ cases in (7).

Self-Composition Rule. Researchers have repeatedly discovered that relational correctness can be encoded in (unary) Hoare logic, essentially because a pair of states can be represented by a single state, e.g., using renamed variables [Barthe et al. 2004; Francez 1983]. A state relation P can be expressed by a state predicate $\hat{P}$, and a command c can be renamed to $\hat{c}$ acting on the alternate variables. Then the judgment $c \mid d : P \mathrel{\ggg} Q$ is represented by the Hoare triple $c; \hat{d} : \hat{P} \rightsquigarrow \hat{Q}$. (We write $c : p \rightsquigarrow q$ for $\{p\}c\{q\}$.) A complete relational Hoare logic thereby comprises the single rule, "from $c; \hat{d} : \hat{P} \rightsquigarrow \hat{Q}$ infer $c \mid d : P \mathrel{\ggg} Q$", together with a complete (unary) Hoare logic. In terms of alignment, of course, this is the most degenerate form of reasoning. The notion of alignment completeness explains the need for other rules, in terms of alignment of automata; see Nagasamudram and Naumann [2021].

7 BEYOND 2-SAFETY: PROPERTIES AND LOGICS

Many relational requirements can be expressed as instances of the ∀∀ (2-safety) form depicted in (5), or other conditions that must hold for all pairs of behaviors. Some other frameworks, such as HyperLTL [Clarkson et al. 2014], can express properties with other patterns of quantification. In Sect. 7.1 we consider, two ∀∃ patterns based on pre- and post-relations the way (5) is. In Sect. 7.5 we consider ∃∀ and ∃∃ properties, for the sake of systematic exploration. The discussion focuses on relational models for concreteness, though the goal is model-independent algebraic formulations.

7.1 ∀∃ Properties

For programs that may be nondeterministic, the ∀∀ property depicted in (5) is often too strong. A range of interesting requirements such as possibilistic noninterference and data refinement are expressed in a ∀∃ form. We call it ***forward simulation*** and write $c \mid d : R \overset{\exists}{\mathrel{\ggg}} S$ for the following.

$$\forall \quad \begin{array}{ccc} \sigma & \xrightarrow{\ c\ } & \tau \\ {\scriptstyle R}\downarrow & & \\ \sigma' & & \end{array} \qquad \exists \quad \begin{array}{ccc} & & \tau \\ & & \downarrow{\scriptstyle S} \\ \sigma' & \xrightarrow{\ d\ } & \tau' \end{array} \tag{8}$$

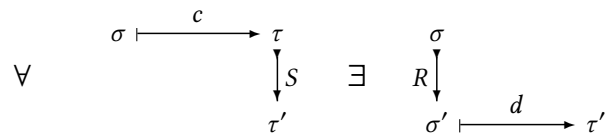
Here $\sigma, \sigma', \tau, \tau'$ are states and c, d, R, S are relations, so (8) says

$$\forall \sigma, \sigma', \tau, \tau'.\ \sigma R \sigma' \wedge \sigma c \tau \Rightarrow \exists \tau'.\ \sigma' d \tau' \wedge \tau S \tau'$$

An equivalent "point-free" formulation, using relation algebra, is $R^o; c \subseteq d; S^o$. (Recall from Sect. 4.2 that R^o means the converse of R.)

For possibilistic noninterference, $c = d$ and the relations express low indistinguishability: R expresses agreement on low inputs and S on low outputs. In the case of data refinement, $R = S$ and R captures some change of data representation.

For program refinement one sometimes needs the similar $\forall\exists$ property called ***backward simulation***, written $c \mid d : R \overset{\exists\leftarrow}{\approx\!>} S$.



(9)

This is expressed in relation algebra as $c; S \subseteq R; d$, and pointwise as

$$\forall \sigma, \tau, \tau'.\ \sigma c \tau \wedge \tau S \tau' \Rightarrow \exists \sigma'.\ \sigma R \sigma' \wedge \sigma' d \tau'$$

In Sect. 7.2 we give theorems that characterize the forward and backward simulation properties in terms of existence of BiKAT witnesses. Using those theorems, we derive (Sect. 7.4) rules for inferring judgments $c \mid d : R \overset{\exists}{\approx\!>} S$ and $c \mid d : R \overset{\exists\leftarrow}{\approx\!>} S$. In Sect. 7.6 we show how forward and backward simulation can be expressed in closed form, by generalizing BiKAT to a kind of 3-KAT.

As an aside, we note that the simulation properties subsume unary underapproximation. For unary tests p, q, the incorrectness logic [O'Hearn 2019] judgment "every state in q can be reached by a terminating execution of c from some state in p" is equivalent to $[c\rangle : [p\rangle \overset{\exists\leftarrow}{\approx\!>} [q\rangle$. The forward approximation condition "for every state in p there is a terminating execution of c that ends in q" is equivalent to $[c\rangle : [p\rangle \overset{\exists}{\approx\!>} [q\rangle$.

7.2 BiKAT Characterizations of Simulation

Alignment is well known to play a role in verifying $\forall\exists$ properties. Example 2.4 in Sect. 2 illustrates that one aligns the computations in a convenient way, in order to winnow out execution pairs in which the second execution makes undesirable nondeterministic choices. Bitests can serve as assume statements for this purpose.

Theorem 7.1 (forward witness soundness). *In a relational BiKAT over a KAT with top, we have forward simulation $c|d : R \overset{\exists}{\approx\!>} S$ if there is some BiKAT term W, called* **alignment witness***, that is* **f-valid***, which means:*

(WC)	$\dot{R}; W \leq \dot{R}; W; \dot{S}$	*(witness $\forall\forall$ correct)*
(WO)	$\dot{R}; \langle c] \leq W; [\mathbf{hav}\rangle$	*(witness overapproximates c)*
(WU)	$\dot{R}; W \leq \langle \mathbf{hav} \mid d\rangle$	*(witness underapproximates d)*

Proof. To prove $c \mid d : R \overset{\exists}{\approx\!>} S$, suppose $\sigma R \sigma'$ and $\sigma c \tau$. Thus $(\sigma, \sigma')\dot{R}; \langle c](\tau, \sigma')$. By (WO) there is τ' with $(\sigma, \sigma')W(\tau, \tau')[\mathbf{hav}\rangle(\tau, \sigma')$. Then by (WU) we have $\sigma' d \tau'$ and by (WC) we have $\tau S \tau'$. □

The theorem is not simply a reduction to $\forall\forall$-logic; it relies essentially on the use of inequalities (WO) and (WU) that are not $\forall\forall$ conditions of the form (5). It is no suprise that reasoning about the existential in a $\forall\exists$ property involves finding a witness—that is familiar in many settings. Here a

witness comprises an alignment of the programs, with embedded bitests that serve to select the witnessing executions —and all the correctness conditions are expressed equationally!

Theorem 7.2 (backward witness soundness). *In a relational BiKAT over a KAT with top, we have* $c \mid d : R \overset{\exists\leftarrow}{\approx\!>} S$ *if there is alignment witness* W *that is* **b-valid**, *meaning:*

$$\begin{array}{lll} \textit{(WCb)} & W;\dot{S} \le \dot{R};W;\dot{S} & \textit{(witness reverse } \forall\forall \textit{ correct)} \\ \textit{(WOb)} & \langle c];\dot{S} \le [\mathbf{hav}\rangle;W & \textit{(witness overapproximates } c\textit{)} \\ \textit{(WUb)} & W;\dot{S} \le \langle \mathbf{hav} \mid d\rangle & \textit{(witness underapproximates } d\textit{)} \end{array}$$

Proof. To prove $c \mid d : R \overset{\exists\leftarrow}{\approx\!>} S$, suppose $\sigma c \tau$ and $\tau S \tau'$. By (WOb) there is σ' with $(\sigma,\sigma')W(\tau,\tau')$. By (WUb) we have $\sigma' d \tau'$. By (WCb) we have $\sigma R \sigma'$. □

Theorem 7.3 (witness completeness). *In a full relational BiKAT, a forward (resp. backward) simulation judgment holds if and only if it has an f-valid (resp. b-valid) alignment witness.*

Proof. To prove completeness, one defines a witness relation; restricting to full models ensures that this relation is in the model (as is **hav**). The completeness direction for forward simulation is proved as follows. Suppose $c|d : R \overset{\exists}{\approx\!>} S$ holds. Using suggestive identifiers for bound variables, define the predicate $\mathcal{P}(\sigma,\sigma',\tau) \mathrel{\hat{=}} \sigma R \sigma' \wedge \sigma c \tau$ and the set $\mathcal{X}(\sigma,\sigma',\tau) \mathrel{\hat{=}} \{\tau' \mid \sigma' d \tau' \wedge \tau S \tau'\}$. For any (σ,σ',τ) that satisfy $\mathcal{P}$ we have $\mathcal{X}(\sigma,\sigma',\tau) \neq \emptyset$, owing to $c \mid d : R \overset{\exists}{\approx\!>} S$. So define $\mathcal{Y}(\sigma,\sigma',\tau)$ to be a chosen element of $\mathcal{X}(\sigma,\sigma',\tau)$ if $\mathcal{P}(\sigma,\sigma',\tau)$, and undefined otherwise. Define $W \mathrel{\hat{=}} \{((\sigma,\sigma'),(\tau,\tau')) \mid \mathcal{P}(\sigma,\sigma',\tau) \wedge \tau' = \mathcal{Y}(\sigma,\sigma',\tau)\}$. By fullness, W is in the BiKAT. We have (WC) because $W = \dot{R};W;\dot{S}$. We have (WU) using the definition of W. We have (WO) also by definitions: If $(\sigma,\sigma')\dot{R};\langle c](\tau,\sigma')$ then $\mathcal{P}(\sigma,\sigma',\tau)$ so let $\tau' = \mathcal{Y}(\sigma,\sigma',\tau)$; we get $(\sigma,\sigma')W(\tau,\tau')$ so $(\sigma,\sigma')W;[\mathbf{hav}\rangle(\tau,\sigma')$.

For backward simulation, completeness is proved similarly. □

For clarity we defined the judgment forms $c \mid d : R \overset{\exists}{\approx\!>} S$ and $c \mid d : R \overset{\exists\leftarrow}{\approx\!>} S$ for relational models, but it is straightforward to interpret them in trace models. Essentially the horizontal arrows in (8) and (9) are interpreted as sequences of zero or more steps. Then the soundness Theorems 7.1 and 7.2 extend to trace models and we get a witness completeness theorem for full trace models, by an argument like our proof of Theorem 7.3.

Theorem 7.3 says witnesses exist in the model; it does not say witnesses are definable in BiKAT from a particular set of primitives. Our examples show some witnesses are definable, and the results in Sect. 7.4 establish that witnesses are BiKAT definable for many judgments.

7.3 Examples Proving ∀∃ with Witnesses

Example 2.4 in Sect. 2 considers the programs E_1 : `x:=any; y:=x` and E_2 : `t:=any; z:=t+1`. To establish $E_1 \mid E_2$: `true` $\overset{\exists}{\approx\!>}$ `y` $\ddot{=}$ `z` we choose witness $W \mathrel{\hat{=}} \langle$`x:=any` $\mid$ `t:=any`$\rangle;[$`x-1` $\ddot{=}$ `t`$];\langle$`y:=x` $\mid$ `z:=t+1`$\rangle$. The bitest [`x-1` $\ddot{=}$ `t`] winnows execution pairs so choices made by E_2 match favorably the nondeterministic assignment made by E_1. Condition (WO) ensures that all executions of E_1 are still covered. The three conditions are proved using axioms to express semantics of the primitives, e.g., the bitest [`x-1` $\ddot{=}$ `t`] commutes with $\langle$`y:=x`$]$. We also use a condition which expresses the left-totality of the bitest [`x-1` $\ddot{=}$ `t`] as discussed later in connection with rule enAss.

Example 7.4 (∀∃ path alignment). When considering ∀∃ properties, sometimes the choices made by the witness determine which paths are taken in the program, rather than merely values taken for variables. Consider this example adapted from Beutner and Finkbeiner [2022].

```
a1 := any; a2 := any; if (h > l) o := l + a1;              // k1
                      else { x := a2; if(x > l) o := x;    // k2
                                      else      o := l; } // k3
```

Here we are interested in the possibilistic non-interference property $C \mid C : [\mathtt{l} \doteq \mathtt{l}] \overset{\exists}{\approx\!>} [\mathtt{o} \doteq \mathtt{o}]$. This example has multiple cases to consider depending on (i) how the inputs to an execution impact the conditional and (ii) how those choices may differ from one execution to another. Consequently, for any path taken in the left program, our choices for the `any`s in the second may involve taking different paths than were taken in the first program. The following is the KAT representation of the three paths and the whole program ($k = k_1 + k_2 + k_3$):

$$
\begin{array}{lcl}
k_1 & \hat{=} & \texttt{a1:=any; a2:=any; [h>l]; o:=l+a1} \\
k_2 & \hat{=} & \texttt{a1:=any; a2:=any; [h<=l]; x:=a2; [x>l]; o:=x} \\
k_3 & \hat{=} & \texttt{a1:=any; a2:=any; [h<=l]; x:=a2; [x<=l]; o:=l}
\end{array}
$$

We will refer to the right program as k', having primed variables. We use the following witness

$$
\begin{array}{lcl}
W & \hat{=} & \langle \texttt{a1:=any; a2:=any;} \mid \texttt{a1':=any; a2':=any;} \rangle; \\
& & (\ [\texttt{h>l; h'>l'; a1'=a1}] \qquad\qquad + [\texttt{h>l; h'<=l'; a2'>l'; a2'=l+a1}] \\
& & + [\texttt{h<=l; h'>l'; a2>l; a1'=a2-l'}] \qquad + [\texttt{h<=l; h'>l'; a2<=l; a1'=0}] \\
& & + [\texttt{h<=l; h'<=l'; a2>l; a2'>l'; a2'=a2}] \quad + [\texttt{h<=l; h'<=l'; a2<=l; a2'<=l'}])\ ;\ \langle c \mid c' \rangle
\end{array}
$$

where c and c' are the remainders of k and k', respectively, after the nondeterministic choices. (So $c \equiv$ `([h>l]; o:=l+a1 + [h<=l]; x:=a2; ([x>l]; o:=x + [x<=l]; o:=l;))`.) The witness W comprises six cases covering *all* preconditions of the input `l, h` and `l', h'` of the two programs. The witness can be rewritten into $W = W_1 + W_2 + \ldots + W_6$ where each W_i corresponds to a case in W. For example, the witness W_2 corresponding to the precondition $\mathtt{h} > \mathtt{l} \wedge \mathtt{h'} \leq \mathtt{l'}$ can be simplified into

$$
\begin{array}{lcl}
W_2 & \hat{=} & \langle \texttt{a1:=any; a2:=any;} \mid \texttt{a1':=any; a2':=any;} \rangle; \\
& & [\texttt{h>l; h'<=l'; a2'>l'; a2'=l+a1}]; \langle c \mid c' \rangle \\
& = & \langle \texttt{a1:=any; a2:=any;} \mid \texttt{a1':=any; a2':=any;} \rangle; \\
& & [\texttt{h>l; h'<=l'; a2'>l'; a2'=l+a1}]; \\
& & \langle \texttt{[h>l]; o:=l+a1} \mid \texttt{[h'<=l']; x':=a2'; ([x'>l']; o':=x' + [x'<=l']; o':=l';)} \rangle \\
& = & \langle \texttt{a1:=any; a2:=any;} \mid \texttt{a1':=any; a2':=any;} \rangle; \\
& & [\texttt{h>l; h'<=l'; a2'>l'; a2'=l+a1}]; \langle \texttt{o:=l+a1} \mid \texttt{x':=a2'; o':=x'} \rangle \quad \text{(distrib, cancel)}
\end{array}
$$

In the above W_2, infeasible paths in the left and right program under the precondition $\mathtt{h} > \mathtt{l} \wedge \mathtt{h'} \leq \mathtt{l'}$ and the chooser $\mathtt{a2'} > \mathtt{l'}$ have been pruned out. Under this precondition, path k_1 in the left program can be aligned with path k'_2 and path k'_3 in the right program. However, the alignment between the left k_1 and the right k'_3 is invalid w.r.t the $\forall\exists$ property because it requires that the left `a1` must be always 0, which is infeasible since under the $\forall$ quantifier, we have to consider every execution of the left program. The condition $\mathtt{a2'} > \mathtt{l'}$ in W_2 then chooses the right k'_2 to align with the left k_1 and the condition $\mathtt{a2'} = \mathtt{l} + \mathtt{a1}$ shows that there exists an execution under that alignment to achieve agreement on `o`. With that intuition, the (WC) condition for the witness W_2, and all the conditions of Theorem 7.1 for the witness terms, are straightforward to prove.

Example 7.5 (Backward simulation). Consider the following, where `x`, `t`, `s`, and `z` range over the natural numbers:

$$
\begin{array}{ll}
C_1: & \texttt{while x>n do x:=x-1 od; t:=any+x; z:=x+t} \\
C_2: & \texttt{s:=any; while x>n do x:=x-1 od; z:=x+s}
\end{array}
$$

We want to show their possibilistic equivalence, which could be expressed as $C_1 \mid C_2 : R \overset{\exists}{\approx\!>} S$, where $R \hat{=} \mathtt{x} \doteq \mathtt{x} \wedge \mathtt{n} \doteq \mathtt{n}$ and $S \hat{=} \mathtt{z} \doteq \mathtt{z}$. To prove this it would be convenient to align the two loops, to enable use of simple relational invariants $\mathtt{x} \doteq \mathtt{x}$ etc. But then the nondeterministic assignment to `t` is aligned far after the assignment to `s` that needs to match it. There are two well known ways to deal with such situations: introduce a prophecy variable [Abadi and Lamport 1988] or (equivalently an auxiliary variable [Morgan 1988]) use backward simulation. We can prove the

$$\textsc{eWhL}\ \frac{P \Rightarrow ([e'\rangle \Rightarrow \langle e])\qquad c \mid c' : P \wedge \langle e] \wedge [e'\rangle \overset{\exists}{\approx\!>} P \qquad c \mid \mathsf{skip} : P \wedge \langle e] \overset{\exists}{\approx\!>} P}{\mathsf{while}\ e\ \mathsf{do}\ c\ \mid\ \mathsf{while}\ e'\ \mathsf{do}\ c' : P \overset{\exists}{\approx\!>} P \wedge \neg\langle e] \wedge \neg[e'\rangle}$$

$$\textsc{eHav}\ \frac{\ddot{1} \leq [\mathbf{hav}\rangle; \dot{R}; [\mathbf{hav}\rangle}{\langle \mathbf{hav} \mid \mathbf{hav}\rangle : \mathit{true} \overset{\exists}{\approx\!>} R} \qquad \textsc{enAss}\ \frac{\ddot{1} \leq [y := \mathrm{any}\rangle; \dot{R}; [y := \mathrm{any}\rangle}{\langle x := \mathrm{any} \mid y := \mathrm{any}\rangle : \mathit{true} \overset{\exists}{\approx\!>} R} \qquad \textsc{eDisj}\ \frac{c \mid d : P \overset{\exists}{\approx\!>} Q \qquad c \mid d : R \overset{\exists}{\approx\!>} Q}{c \mid d : P \vee R \overset{\exists}{\approx\!>} Q}$$

Fig. 2. Selected rules for ∀∃ forward simulation correctness. **Please note:** there are also rules eConseq, eSeq, eIf, eWh, which look the same as rConseq, dSeq, dIf, dWh but using $- \overset{\exists}{\approx\!>} -$.

following:[8] $C_1 \mid C_2 : R \overset{\exists\leftarrow}{\approx\!>} T$, where $T \mathrel{\hat{=}} \mathtt{x} \ddot{=} \mathtt{x} \wedge \mathtt{n} \ddot{=} \mathtt{n} \wedge \mathtt{z} \ddot{=} \mathtt{z} \wedge \mathtt{t} \ddot{=} \mathtt{s}$. To prove this using the witness technique of Theorem 7.2, we choose the witness to be:

$$W \mathrel{\hat{=}} [\mathtt{s:=any}\rangle; \langle X \mid X\rangle^*; \langle \neg[\mathtt{x>n}]]; \langle \mathtt{t:=any+x}]; \langle \mathtt{z:=x+t} \mid \mathtt{z:=x+s}\rangle; T$$

where $X \mathrel{\hat{=}} [\mathtt{x>n}]; \mathtt{x:=x-1}$. Notice that W ends with the postrelation T and we do not need to introduce additional bitests. The three conditions the witness must satisfy are easily proved. As usual we rely on axioms for the semantics of primitives. Interestingly, these include ones about backward preservation of bitests, e.g., $\langle X \mid X\rangle; [\mathtt{x} \ddot{=} \mathtt{x}] = [\mathtt{x} \ddot{=} \mathtt{x}]; \langle X \mid X\rangle; [\mathtt{x} \ddot{=} \mathtt{x}]$.

Example 7.6 (Forward simulation and prophecy). As a variation on Example 7.5 we can prove $C_3 \mid C_2 : R \overset{\exists}{\approx\!>} S$ for a modified version of C_1 that uses variable p to "prophesize" the value for t.

$$C_3 : \quad \mathtt{p:=any;\ while\ x>n\ do\ x:=x-1\ od;\ t:=p+x;\ z:=x+t}$$

To show $C_3 \mid C_2 : R \overset{\exists}{\approx\!>} S$ using the witness technique in Theorem 7.1, choose witness W to be:

$$W \mathrel{\hat{=}} \langle \mathtt{p:=any} \mid \mathtt{s:=any}\rangle; B; \langle X \mid X\rangle^*; \langle \neg[\mathtt{x>n}]]; \langle \mathtt{t:=p+x;\ z:=x+t} \mid \mathtt{z:=x+s}\rangle$$

where $X \mathrel{\hat{=}} [\mathtt{x>n}]; \mathtt{x:=x-1}$ and $B \mathrel{\hat{=}} [\mathtt{p+min(x,n)} \ddot{=} s]$. As in the previous forward simulation examples, the witness aligns the two nondeterministic assignments together, the loops in lockstep, and introduces a bitest B that filters executions of C_2 to only those that ensure agreement on z upon termination. The three witness conditions for W are straightforward to check.

In the next section we introduce deductive rules for forward simulation. With these, the judgment about C_3 and C_2 is proved in a way that implicitly follows the alignment W. The chooser bitest B is introduced as a postcondition, by a rule for aligned nondeterministic assignments, and it is manipulated in intermediate assertions rather than being inlined like it is in W. (See the appendix of the extended version [Antonopoulos et al. 2022a].)

7.4 Logics of Forward and Backward Simulation

7.4.1 Forward Simulation Logic. Theorem 7.1 gives a way to prove forward simulation judgments, by direct reasoning in a BiKAT. There are also inference rules for forward simulation. In fact several of the inference rules for ∀∀ judgments (Fig. 1) are also sound for forward simulation. Fig. 2 gives some rules for the $- \overset{\exists}{\approx\!>} -$ judgment.

Consider eWh, which is simply dWh but for the $- \overset{\exists}{\approx\!>} -$ judgment. Informally, eWh is sound because any terminating execution of the left program can be matched by one that terminates on the right, owing to the side condition that says the loop tests agree. Rule eWhL has a similarly simple side condition that suffices to ensure relative termination: in terms of alignment, if the right

[8] A stronger postcondition is needed for this backwards property, for similar reasons to what happens in incorrectness logic [O'Hearn 2019].

loop can continue to iterate then so can the left, and their joint iterations can be aligned in lockstep. An additional premise handles the situation where the left loop has more iterations than the right.

One can consider two other situations. One is where the iterations can be aligned in lockstep, but the right loop may need more iterations. Consider this rule:

$$\textsc{eWhR}\ \frac{P \Rightarrow (\neg\langle e\rangle \vee [e'\rangle) \wedge [f \geq 0\rangle \qquad \mathsf{skip} \mid c' : P \wedge [e' \wedge f = n\rangle \overset{\exists}{\approx\!>} P \wedge [f < n\rangle \qquad c \mid c' : P \wedge \langle e] \wedge [e'\rangle \overset{\exists}{\approx\!>} P}{\mathsf{while}\ e\ \mathsf{do}\ c \ \mid\ \mathsf{while}\ e'\ \mathsf{do}\ c' : P \overset{\exists}{\approx\!>} P \wedge \neg\langle e] \wedge \neg[e'\rangle}$$

It uses a variant expression f to establish termination on the right side (like in total correctness Hoare logic). A related but different idea is the general rule to infer $c \mid c' : P \overset{\exists}{\approx\!>} R$ from $c \mid c' : P \approx\!> R$ together with termination of c' from states in the codomain of P. But KAT does not support direct expression of termination, and we refrain from formulating the requisite notations for a BiKAT encoding of these rules.

The other situation for loops is where lockstep alignment is not sufficient. We conjecture that a rule similar to rule caWh can be devised, but that is beyond the scope of this paper.

To relate two nondeterministic assignments, this axiom is sound in relational and trace models: $\langle x := \mathsf{any} \mid y := \mathsf{any}\rangle : (\forall \overleftarrow{x} \,.\, \exists \overrightarrow{y} \,.\, R) \overset{\exists}{\approx\!>} R$. It uses suggestive informal notation for quantification over the left and right states. The precondition ensures that for any value assigned to x there is some value for y such that R holds. But in this paper we refrain from formalizing formulas for relations. Instead we consider a rule for judgments of the form $\langle x := \mathsf{any} \mid y := \mathsf{any}\rangle : \mathit{true} \overset{\exists}{\approx\!>} R$. This holds provided that, in any pair of states, for every value of x there is some value for y making R true. As a step towards an algebraic formulation for that condition, first consider the fully nondeterministic action **hav**.

For $\langle \mathbf{hav} \mid \mathbf{hav}\rangle : \mathit{true} \overset{\exists}{\approx\!>} \dot{R}$ to hold, R must be a domain-total relation. In terms of relations this can be expressed by the equation $R; \mathbf{hav} = \mathbf{hav}$ but we prefer to express the condition in terms of the bitest $\dot{R}$ for R. The condition in rule eHav (Fig. 2), i.e., BiKAT equation $\ddot{1} \leq [\mathbf{hav}\rangle; \dot{R}; [\mathbf{hav}\rangle$, holds (in a relational model) just if the relation R is domain-total.

For two nondeterministic assignments to satisfy the judgment $\langle x := \mathsf{any} \mid y := \mathsf{any}\rangle : \mathit{true} \overset{\exists}{\approx\!>} R$, the condition that we wrote as $\forall \overleftarrow{x} \,.\, \exists \overrightarrow{y} \,.\, R$ should be valid. This is equivalent to the condition $\ddot{1} \leq [y := \mathsf{any}\rangle; \dot{R}; [y := \mathsf{any}\rangle$ of rule enAss. An informal reading is that for any pair of states, there is some value for y on the right that makes R hold. (The trailing assignment to y can restore the initial value of y.)

Lemma 7.7. *In any BiKAT, and for any rule in Fig. 2, given an f-valid witnesses for the premises, there is an f-valid witness for the conclusion.*

Theorem 7.8. *The rules in Fig. 2 are sound in any full relational model.*

Proof. For each rule, if its premises are true then by Theorem 7.3 there are f-valid witnesses for the premises. So by Lemma 7.7 we obtain an f-valid witness for the conclusion, so by Theorem 7.1 the conclusion is true. □

We consider illustrative cases in the proof of Lemma 7.7.

Proof of eHav. The rule has no premise judgment, only the antecedent $\ddot{1} \leq [\mathbf{hav}\rangle; \dot{R}; [\mathbf{hav}\rangle$. To prove the conclusion we take witness W to be $\langle \mathbf{hav} \mid \mathbf{hav}\rangle; \dot{R}$. Using the BiKAT notation $\ddot{1}$ for the pre-relation *true*, the f-validity conditions are:

(WC) $\ddot{1}; \langle \mathbf{hav} \mid \mathbf{hav}\rangle; \dot{R} \leq \ddot{1}; \langle \mathbf{hav} \mid \mathbf{hav}\rangle; \dot{R}; \dot{R}$

(WU) $\ddot{1}; \langle \mathbf{hav} \mid \mathbf{hav}\rangle; \dot{R} \leq \langle \mathbf{hav} \mid \mathbf{hav}\rangle$

(WO) $\ddot{1}; \langle \mathbf{hav}] \leq \langle \mathbf{hav} \mid \mathbf{hav}\rangle; \dot{R}; [\mathbf{hav}\rangle$

We have (WC) by idempotence of the test $\dot{R}$, and (WU) using $\dot{R} \leq \ddot{1}$. The condition (WO), expressing existence, is proved using the antecedent condition for $\dot{R}$. $\ddot{1}; \langle \mathbf{hav}] = \ddot{1}; \langle \mathbf{hav}]; \ddot{1} \leq \ddot{1}; \langle \mathbf{hav}]; [\mathbf{hav}\rangle; \dot{R}; [\mathbf{hav}\rangle = \langle \mathbf{hav} \mid \mathbf{hav}\rangle; \dot{R}; [\mathbf{hav}\rangle$. For rule enAss the proof is similar.

Proof of eSeq. Suppose for premise $c \mid c' : P \overset{\exists}{\approx\!>} R$ we have witness Z and for $d \mid d' : R \overset{\exists}{\approx\!>} Q$ we have witness W, so the conditions are

(WCZ)	$\dot{P}; Z \leq \dot{P}; Z; \dot{R}$	(WCW)	$\dot{R}; W \leq \dot{R}; W; \dot{Q}$
(WUZ)	$\dot{P}; Z \leq \langle \mathbf{hav} \mid c' \rangle$	(WUW)	$\dot{R}; W \leq \langle \mathbf{hav} \mid d' \rangle$
(WOZ)	$\dot{P}; \langle c] \leq Z; [\mathbf{hav}\rangle$	(WOW)	$\dot{R}; \langle d] \leq W; [\mathbf{hav}\rangle$

To prove $c; d \mid c'; d' : P \overset{\exists}{\approx\!>} Q$ we use $Z; W$ as witness.
• (WC) To show $\dot{P}; Z; W \leq \dot{P}; Z; W; \dot{Q}$ we have $\dot{P}; Z; W \leq \dot{P}; Z; \dot{R}; W \leq \dot{P}; Z; \dot{R}; W; \dot{Q} \leq \dot{P}; Z; W; \dot{Q}$ using (WCZ), (WCW), and $\dot{R} \leq \ddot{1}$.

• (WU) We have $\dot{P}; Z; W \leq \dot{P}; Z; \dot{R}; W \leq \langle \mathbf{hav} \mid c' \rangle; \langle \mathbf{hav} \mid d' \rangle = \langle \mathbf{hav} \mid c'; d' \rangle$ using (WCZ), (WUZ), (WUW), embedding homomorphic, and idempotence of **hav**.
• (WO)

	$\dot{P}; \langle c; d]$	
$=$	$\dot{P}; \langle c]; \langle d]$	emb homo
$\leq$	$\dot{P}; Z; [\mathbf{hav}\rangle; \langle d]$	(WOZ)
$=$	$\dot{P}; Z; \langle d]; [\mathbf{hav}\rangle$	(LRC)
$=$	$\dot{P}; Z; \dot{R}; \langle d]; [\mathbf{hav}\rangle$	(WCZ)
$\leq$	$\dot{P}; Z; W; [\mathbf{hav}\rangle; [\mathbf{hav}\rangle$	(WOW)
$=$	$\dot{P}; Z; W; [\mathbf{hav}\rangle$	**hav** idem, emb homo
$\leq$	$Z; W; [\mathbf{hav}\rangle$	$P \leq \ddot{1}$

7.4.2 Backward Simulation Logic. The backward simulation judgment has a number of inference rules, with some interesting differences from the ∀∀ and forward simulation rules. The rules are derivable in BiKAT.

Hoare's assignment axiom based on weakest precondition, using substitution in the precondition, is often called "backwards". The relational generalization works for ∀∀ and forward simulation:

dAss
$$v := e \mid v' := e' : P^{v|v'}_{e|e'} \approx\!> P$$

eAss
$$v := e \mid v' := e' : P^{v|v'}_{e|e'} \overset{\exists}{\approx\!>} P$$

Variables and substitution are not part of KAT/BiKAT but the rules are sound in models. But a judgment of this form is unsound for backward simulation, for the same reason as in incorrectness logic [O'Hearn 2019]: with an arbitrary postcondition, there can be final states that are not in the image of the assignment. The unary Floyd [1967] axiom for assignment is $v := e : q \rightsquigarrow \exists u. q^v_u \wedge v = e^v_u$ which suggests the following:

$$v := e \mid v' := e' \;:\; P \overset{\exists\leftarrow}{\approx\!>} \exists u, u'.\, P^{v|v'}_{u|u'} \wedge \langle v = e^v_u] \wedge [v' = e'^{v'}_{u'}\rangle$$

Consider τ, τ' that satisfy the postcondition and let σ be the left initial state. Let $\hat{u}, \hat{u}'$ be values that witness the existential and observe that τ is σ with v updated to $\hat{u}$. Choose σ' be a state such that τ' is σ' with v updated to $\hat{u'}$. Conclude σ, σ' satisfy the precondition, and further, that executing $v' := e'$ in σ' yields τ'; hence the judgment holds.

Unlike the simple law that can be used to express semantics of assignments in a proof of eAss for forward simulation, a BiKAT formulation of the Floyd rule would need a more complicated way to reason about assignment and postcondition formula. It can be done by choosing some type, say integers, for data, and then treating the existential as an integer-indexed sum of tests, but we leave this to the reader.

$$\textsc{bIf}\ \frac{c \mid c' : P \wedge \langle e\rangle \wedge [e'\rangle \overset{\exists\leftarrow}{\approx\!>} Q \qquad d \mid d' : P \wedge \neg\langle e\rangle \wedge \neg[e'\rangle \overset{\exists\leftarrow}{\approx\!>} Q}{\text{if } e \text{ then } c \text{ else } d \mid \text{if } e' \text{ then } c' \text{ else } d' : P \overset{\exists\leftarrow}{\approx\!>} Q}$$

$$\textsc{bWh}\ \frac{c \mid c' : P \wedge \langle e\rangle \wedge [e'\rangle \overset{\exists\leftarrow}{\approx\!>} P}{\text{while } e \text{ do } c \mid \text{while } e' \text{ do } c' : P \overset{\exists\leftarrow}{\approx\!>} P \wedge \neg\langle e\rangle \wedge \neg[e'\rangle} \qquad \textsc{bSeq}\ \frac{c \mid c' : P \overset{\exists\leftarrow}{\approx\!>} R \qquad d \mid d' : R \overset{\exists\leftarrow}{\approx\!>} Q}{c; d \mid c'; d' : P \overset{\exists\leftarrow}{\approx\!>} Q}$$

$$\textsc{bnAss}\ \frac{\ddot{1} \le [y := \text{any}\rangle; \dot{R}; [y := \text{any}\rangle}{\langle x := \text{any} \mid y := \text{any}\rangle : R \overset{\exists\leftarrow}{\approx\!>} \mathit{true}} \qquad \textsc{bConseq}\ \frac{R \Rightarrow P \qquad c \mid d : R \overset{\exists\leftarrow}{\approx\!>} S \qquad Q \Rightarrow S}{c \mid d : P \overset{\exists\leftarrow}{\approx\!>} Q}$$

$$\textsc{bDisj}\ \frac{c \mid d : P \overset{\exists\leftarrow}{\approx\!>} Q \qquad c \mid d : P \overset{\exists\leftarrow}{\approx\!>} R}{c \mid d : P \overset{\exists\leftarrow}{\approx\!>} Q \vee R}$$

Fig. 3. Selected rules for ∀∃ backward simulation correctness.

Another resemblance to incorrectness logic is that the consequence rule is reversed from the one for forward simulation and ∀∀. Suppose W is a witness for $c \mid d : P \overset{\exists\leftarrow}{\approx\!>} Q$, and moreover $P \Rightarrow R$ and $S \Rightarrow Q$. Then W is also a witness for $c|d : R \overset{\exists\leftarrow}{\approx\!>} S$ To show (WCb) for the latter, we have $W; \dot{S} \le W; \dot{Q} \le \dot{P}; W; \dot{Q} \le \dot{R}; W; \dot{Q}$ (using (WCd) for W, i.e., $W; Q \le P; W; Q$). And $W; \dot{S} \le \dot{R}; W; \dot{Q}$ iff $W; \dot{S} \le \dot{R}; W; \dot{S}$ using $S \Rightarrow Q$.

The rule for sequence looks just like dSeq (and eSeq), and is proved by a calculation similar to the one proving eSeq. For conditional and loop, the pattern of rules dIf and dWh (also eIf and eWh) can be retained but the side conditions are not needed! The conclusion of rule bWh in Fig. 3 says that given final states (τ, τ') related by P in which the loop tests are false, and a terminating execution from some initial σ with (σ, σ') related by P, there is an execution from σ' ending in τ'. Informally, the conclusion follows because we can repeatedly invoke the premise, starting from the last iteration, to obtain the requisite right execution. If σ_i is the state reached after the ith iteration on the left, and τ_i the corresponding right state given by our induction hypothesis, and there is a preceding iteration on the left from σ_{i-1}, the premise yields some τ_{i-1} with a matching right iteration, and moreover e' holds in τ_{i-1} so the loop does take this iteration.

Theorem 7.9. *The rules in Fig. 3 are sound in any full relational model.*

The proof uses Theorem 7.3. Thorough investigation of loop rules for backward simulation is beyond the scope of this paper.

7.5 On ∃∀ and ∃∃ Properties

We have looked at various forms of properties involving 2 executions, with pre/post-conditions playing a different roles. So far we have focused on some ∀∀ properties (Sect. 6), and some ∀∃ ones (Sects. 7.1–7.4). Many of the relational properties can be expressed in these forms, but the classes that correspond to the duals of those properties are also interesting. Consider the property

$$\exists\ \begin{array}{ccc} \sigma & \overset{c}{\longmapsto} & \tau \\ {\scriptstyle R}\downarrow & & \\ \sigma' & & \end{array} \qquad \forall\ \begin{array}{ccc} & & \tau \\ & & \downarrow{\scriptstyle S} \\ \sigma' & \overset{d}{\longmapsto} & \tau' \end{array} \qquad \Rightarrow \qquad \begin{array}{c} \tau \\ \downarrow{\scriptstyle S} \\ \tau' \end{array} \tag{10}$$

that says $\exists\sigma, \sigma', \tau.\ \sigma R\sigma' \wedge \sigma c\tau \wedge \forall\tau'(\sigma' d\tau' \Rightarrow \tau S\tau')$. This property is the dual of forward simulation with a negated postcondition, or in other words (10) is equivalent to $\neg(c \mid d : R \overset{\exists}{\approx\!>} \neg S)$. Using Theorem 7.3, to prove this property, we can check whether the following holds: for any BiKAT term Z at least one of the conditions (WO), (WU) or (WC) fails.

Although not many properties of the form $\exists\forall$ naturally occur in the literature on relational verification, they are nonetheless important, and not only as duals of the more frequently occurring $\forall\exists$ ones. The reason is that with the former, it suffices to find one single execution of the left program that captures the required behavior against as traces of the right one. Checking whether such an execution or trace exists is often hard, and a proof system of $\exists\forall$ properties will be of value.

Arguing similarly we can use BiKAT reasoning to explore $\exists\exists$ properties on two executions. A prime example of such properties is "definite non-determinism", or in other words the existence of an input, and two executions on that input that produce different output. Consider

$$\exists\sigma, \sigma', \tau, \tau'.\ \sigma R\sigma' \wedge \sigma c\tau \wedge \sigma' d\tau' \wedge \neg\tau S\tau'$$

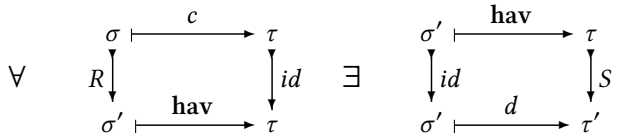
as a general form of $\exists\exists$ properties. This formulation is the dual of the main 2-safety property we study in the previous sections (see Equation (5)). Non-determinism can then be expressed by setting both programs c and d to be the same, and setting R and S to be relations expressing agreement on all variables.

7.6 TriKAT

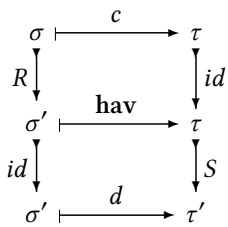
One might hope that the existential quantifications of (8) and (9) could be expressed algebraically using BiKAT projection (Def. 4.2). Unfortunately, projection existentially quantifies both initial and final state of the second execution (see (4)). So projection does not directly capture (8), where the second execution's initial state is universally quantified, nor (9) where its final state is universally quantified. We sketch a way to use projection that merits further investigation but is not used in the rest of the paper.

Possibilistic noninterference can be expressed in HyperLTL, a temporal logic with explicit quantifiers that range over the traces of a fixed program [Clarkson et al. 2014]. The formula says that for all traces π, π' with initial states related by R, there is a trace π'' with the same initial state as π', and R relates the final states of π and π''. The reason three traces are needed is that two initial states are universally quantified in (8).

To express $c \mid d : R \overset{\exists}{\approx\!>} S$ we consider three executions, using in place of (8) the following pattern.



We define a notion of TriKAT so these ingredients can be described in the form of the following diagram. It depicts a relation on state triples represented by the displayed TriKAT term using notation to be explained.



represented by TriKAT term
$\langle \dot{R} \parallel \dot{id} \rangle; \langle c \mid \mathbf{hav} \mid d \rangle; \langle \dot{id} \parallel \dot{S} \rangle$ (11)

The existential quantification will then be expressed by projection, as we proceed to show.

One can easily generalize BiKAT with a three-argument embedding we will write as $\langle \_ \mid \_ \mid \_ \rangle$. But we also need BiKAT elements to encode the relations R, S, id. So we define a *two*-argument embedding $\langle \_ \,[\!]\, \_ \rangle$ with the following meaning: if A, B are BiKAT elements, thus denoting pairs of executions, then $\langle A \,[\!]\, B \rangle$ denotes triples of executions, comprising pairs from A and B that agree on the middle one. Specifically, for elements A and B of a relational BiKAT, define $\langle A \,[\!]\, B \rangle$ to be this relation on $\Sigma \times \Sigma \times \Sigma$.

$$(\sigma, \sigma', \sigma'')\langle A \,[\!]\, B\rangle(\tau, \tau', \tau'') \mathrel{\hat{=}} (\sigma, \sigma')A(\tau, \tau') \wedge (\sigma', \sigma'')B(\tau', \tau'')$$

For the special case of BiKAT elements embedded from the underlying KAT, define the abbreviation

$$\langle a \mid b \mid c \rangle \mathrel{\hat{=}} \langle \langle a \,|\, b \rangle \,[\!]\, \langle b \,|\, c \rangle \rangle$$

Observe that $(\sigma, \sigma', \sigma'')\langle a \mid b \mid c\rangle(\tau, \tau', \tau'')$ iff $\sigma a \tau \wedge \sigma' b \tau' \wedge \sigma'' c \tau''$. Now one can check that the relation on state triples depicted by the diagram in (11) is denoted by the term on the right in (11).

Among the available projections we need the one that projects the left two of three. This can be considered as a projection from the "TriKAT" to the left underlying BiKAT. For any relation X on $\Sigma \times \Sigma \times \Sigma$, define the relation $\swarrow\!\!\swarrow X$ on $\Sigma \times \Sigma$ by

$$(\sigma, \sigma') \swarrow\!\!\swarrow X(\tau, \tau') \quad \text{iff} \quad \exists \sigma'', \tau''.\ (\sigma, \sigma', \sigma'')X(\tau, \tau', \tau'')$$

Lemma 7.10. *In a relational model, $c \mid d : R \overset{\exists}{\approx\!>} S$, i.e., (8), is equivalent to this BiKAT equation:*

$$\dot{R}; \langle c \,|\, \mathbf{hav} \rangle; \dot{id} \;\leq\; \swarrow\!\!\swarrow(\langle \dot{R} \,[\!]\, \dot{id} \rangle; \langle c \mid \mathbf{hav} \mid d \rangle; \langle \dot{id} \,[\!]\, \dot{S} \rangle) \tag{12}$$

and $c \mid d : R \overset{\exists\leftarrow}{\approx\!>} S$, i.e., (9) is equivalent to

$$\dot{id}; \langle c \,|\, \mathbf{hav} \rangle; \dot{S} \;\leq\; \swarrow\!\!\swarrow(\langle \dot{id} \,[\!]\, \dot{R} \rangle; \langle c \mid \mathbf{hav} \mid d \rangle; \langle \dot{S} \,[\!]\, \dot{id} \rangle) \tag{13}$$

The proofs are by unfolding definitions. The whole story works also for trace models. We hoped that (12) could be used directly to prove simulations, but were unsuccessful. The reader may check that in relational or trace models of BiKAT, projection distributes only weakly over sequence, as $\swarrow(A; B) \leq \swarrow A; \swarrow B$, and $\swarrow\!\!\swarrow(X; Y) \leq \swarrow\!\!\swarrow X; \swarrow\!\!\swarrow Y$, which is not helpful for proving equations like (12). Nonetheless, we can use (12) and (13) to derive some rules including eDisj in Fig. 2.

The operation $\langle - \mid - \mid - \rangle$ used above does distribute homomorphically over sequence and the other operators. As a result, BiKAT generalizes straightforwardly to n-KAT, which encompasses $\forall^n$ properties of program n-tuples, as in Cartesian Hoare logic [Sousa and Dillig 2016]. However, it is an open question how to axiomatize projections and the bi-to-tri embedding $\langle - \,[\!]\, - \rangle$ in a way that is useful for $\forall\exists$, so we do not formally define TriKAT.

8 DISCUSSION

We have described BiKAT, a theory for equational reasoning about alignment for relational program verification. In this section we consider implications for automated reasoning, connections with other work including product programs and deductive verification, and problems for future work.

8.1 Automation of Alignment

Recent years have seen advances in automation of relational verification, as summarized in Sect. 8.3. To our knowledge, none of these techniques algebraically derive an alignment. As we developed the algebraic theory of BiKAT, along the way we found possible approaches for automation such as integrating with existing automated techniques for discovering alignments or relational invariants. We save automation for future work, but summarize some general approaches here.

Solving BiKAT equivalence queries. A BiKAT is itself a KAT and we can, therefore exploit a wide range of KAT-based tools such as symbolic equality reasoning [Pous 2015], Coq tactics [Braibant

and Pous 2010], abstract interpretation with KAT [Antonopoulos et al. 2019], and methods for constructing and deciding equality in concrete KATs [Greenberg et al. 2022]. Existing KAT tools do not have a built-in way of representing our particular kinds of KATs. However, we can encode a BiKAT, minus LRC, as a KAT through suitable variable renaming. We did this manually for a simple, concrete BiKAT and confirmed that KMT [Greenberg et al. 2022] was able to verify some equivalence queries. Of course we interactively use LRC to obtain the desired alignment and do not give KMT instances of the LRC axiom, owing to undecidability.

Constraint-based relational verification. Unno et al. [2021] reduced k-safety and possibilisitic non-interference to a constraint-satisfaction problem. Although these reductions are relatively complete, the approach does not scale well, as it searches for possible alignments. One possible path forward is to use BiKAT reasoning to algebraically derive alignments at a coarse-grained level, and then employ a constraint-satisfaction problem to solve the fine-grained subproblems.

Semi-automation. Automated solving (e.g. via KAT tools or constraint-satisfaction) could also be used as part of a larger semi-automated reasoning framework. Our Coq development already provides the basic BiKAT laws/lemmas and could be extended to include forward/backward simulation rules, which would then be used interactively on given problem to derive an alignment. Along the way, automated solvers could be integrated and used to discharge smaller semantic queries.

Numerous other works discuss automation of relational verification. Pick et al. [2018] describe a technique that aligns conditional blocks (in addition to loops) and exploits symmetries to reduce the verification burden. Farzan and Vandikas [2019] describe an approach to hypersafety verification of unary programs by discovering representative executions of a product program, whose correctness proofs are sufficient to prove the overall property. Unno et al. [2021] work with transition systems and use a constraint-solving approach to automatically discover a "scheduler"—a form of alignment described as a function that directs which element of the k-tuple (product of transition systems) should take the next step. Badihi et al. [2020] describe ARDiff, using a combination of abstraction and refinement for automatically proving program equivalence. Mordvinov and Fedyukovich [2019] work in the context of CHCs, and infer relational invariants.

8.2 Product Programs and Expressibility

BiKAT serves as notation for alignment products which in turn represent alignments of executions. We have already shown examples of some alignments in the literature, including the $2x^2$ example of Shemer et al. [2019] (our Example 2.2) and the array insertion example of Shemer *et al.* (in Sect. 5). In this section we consider what alignments can be represented in BiKAT, compared with product programs and other representations in the literature. We also consider how adequacy is established in various works, compared with our Theorem 4.6.

In some works, products are literally programs (e.g., [Barthe et al. 2011a, 2004; Eilers et al. 2018]). In others, products are represented in some form of control-flow automata (e.g., [Churchill et al. 2019]) or transition system (e.g., [Shemer et al. 2019]). In all cases, products represent alignments between corresponding points in execution pairs (or k-tuples), for reasoning based on assertions at the aligned points.

Representation of products as programs has the advantages (and disadvantages) of syntactic representation. For products as programs, one approach to ensuring adequacy is developed by Barthe et al. [2011a, 2016] who develop a ternary judgment that connects two programs to a third that represents an aligned product of them. Assert commands are used to ensure adequacy. For example, in the case of if-else, the guard-agreement side condition of rule IF (in Fig. 1) can be added to the product program as an initial assertion. If the assertion holds, the product is adequate. For if e then c else d and if e' then c' else d', one might try to write their product as this BiKAT term: $e \doteqdot e'; (\langle e]; \langle c \mid c' \rangle + \neg\langle e]; \langle d \mid d' \rangle)$. However, this treats the initial agreement as an assumption,

whereas for adequacy it must be a consequence of the precondition. KAT can be extended with failures (FailKAT) in order to express assertions [Mamouras 2017].

The approach of Barthe *et al.* has the advantage of making a close connection with Hoare logic, but the disadvantage of lacking means to leverage left-right-commutativity as such. Banerjee et al. [2022, 2016] handle products using custom syntax for what they call *biprograms*. The semantics of their bi-if is essentially like having the guard-agreement assertion. Their bi-command form $(c|c')$ serves as a product with no designated intermediate alignment. Relational judgments apply to biprograms and a verification problem is posed in the form $(c|c')$. There is an auxiliary relation on biprograms, called weaving, that effectively performs left-right-commutings in a way that preserves adequacy. Whereas Barthe *et al.* reduce relational verification to unary Hoare logic, Banerjee *et al.* use a custom proof system for biprograms. In both of these lines of work, adequacy is proved as a general result about the system. By inspection of our examples and Theorems 6.1 and 6.2, one can see that BiKAT can represent the alignments achievable in these systems, when they are adequate, but (as noted above) cannot directly encode adequacy checks expressed as assertions. We conjecture that the systems can be encoded in an extension of BiKAT based on FailKAT [Mamouras 2017]. Such an extension would also serve another purpose, namely to encode the fault-sensitive variation of $\forall\forall$ used in practical verification systems and logics including Banerjee et al. [2022, 2016]; Yang [2007].

As mentioned earlier, the tiling example is handled in Banerjee et al. [2016] using a custom rewriting relation with KAT-like rules. The probabilistic relational Hoare logic of Barthe et al. [2017] has a similar rule, called structural equivalence (as well as a rule for conditionally aligned loops). The logic's relational correctness judgment connects the related programs to a product program that witnesses a probabilistic coupling.

As we discuss in Sect. 5, procedure calls can be treated as primitives in BiKAT, which can thereby express alignment of calls for use with relational specs as hypotheses. However, BiKAT has no mean to express patterns of alignments involving nested procedure calls as in the work of Godlin and Strichman [2008]. Nor does BiKAT provide for directly expressing alignment defined by code overlay as in the ghost monitors of Clochard et al. [2020]. However, history-sensitive alignment can be expressed in BiKAT and other systems using ghost code.

Many prior works use products based on the representation of programs, and products, as transition systems. We sketch how BiKAT can express such products quite generally. A transition system can be presented as rules of the form $g \to a$ where the guard g is a state condition, and a is an assignment command (or basic block). So the program can be written in Dijkstra's guarded command notation [Apt et al. 2009] as do $g_0 \to a_0 \;[\!]\; g_1 \to a_1 \;[\!]\; \ldots$ od. Regardless of the form of the commands a_i, this has a simple representation in KAT, as $(g_0; a_0 + g_1; a_1 \ldots)^*; \neg(\Sigma_i\, g_i)$. For clarity in the following discussion we ignore the negated condition and simply write $(g_0; a_0 + g_1; a_1 \ldots)^*$. So the following BiKAT term represents a product of two transition systems.

$$\langle (g_0; a_0 + g_1; a_1 \ldots)^* \mid (g_0'; a_0' + g_1'; a_1' \ldots)^* \rangle$$

Alignment products in the literature constrain executions of the underlying programs by some conditions L, R, J on state-pairs, that designate whether to take a left-only step, right-only step, or joint step. (More generally, which of k copies, as in Shemer et al. [2019], Eilers et al. [2018].) In a BiKAT, assuming the conditions L, R, J are expressible as bitests, the alignment is expressible using finite sums as

$$((\Sigma_{g,a,g',a'}\; J; \langle g; a \mid g'; a' \rangle) + (\Sigma_{g,a}\; L; \langle g; a \mid 1 \rangle) + (\Sigma_{g',a'}\; R; \langle 1 \mid g'; a' \rangle))^* \tag{14}$$

In these sums, g, a range over the guarded actions $g \to a$ of the left program and g', a' range over the right.

So much for expressing alignments. What about proving adequacy? One can formulate general conditions under which a product is adequate. Roughly, the idea is that $L \vee R \vee J \vee TRM$ must be invariant, where TRM stands for "both sides terminated". Shemer et al. [2019] give an adequacy result of this form. (Their term is "fairness". There is no standard term; we take "adequacy" from Nagasamudram and Naumann [2021].) Churchill et al. [2019] formulate adequacy as verification conditions involving their alignment invariants. In our setting, adequacy is proved equationally (Theorem 4.6), raising the question whether the term (14) can be derived from $\langle (g_0; a_0 + g_1; a_1 \ldots)^* \mid (g_0'; a_0' + g_1'; a_1' \ldots)^* \rangle$. Invariance of $L \vee R \vee J \vee TRM$ is analogous to the side condition of the proof rule CAWH, and the equality of terms $\langle (g_0; a_0 + g_1; a_1 \ldots)^* \mid (g_0'; a_0' + g_1'; a_1' \ldots)^* \rangle$ and (14) is analogous to how the expansion law (7) is used to prove CAWH. We conjecture that the general equality can be proved, in *-continuous BiKATs, just as we have done for (7).

The L, R, J form discussed above is very general. In implementations, the conditions L, R, J tend to be restricted to constraints supported by an efficient solver, and the same restrictions would be applicable in uses of BiKAT. Ignoring such restrictions, that general form seems as expressive as BiKAT. We are not aware of patterns that can be expressed in BiKAT but not by products represented as transition systems.

8.3 Other Related Work

We have covered many of the most related works; we now mention a few others.

In recent work D'Osualdo et al. [2022] describe a logic for hyper-triple composition (LHC) based on weakest preconditions that can decompose a hypersafety proof along the boundary of hyper tuples, offering ways of combining multiple k-safety proofs with differing ks. In contrast with BiKAT, LHC is a calculus based on weakest pre-condition rather than an equational system, and supports only $\forall\forall$ pre/post k-safety properties. There may be a connection between LHC-style decomposition and our proposed work on TriKAT discussed in Sect. 7.6. Both permit ways to combine two relational proofs that share some common program terms into an overall proof by correlating the common terms. However, we leave this investigation to future work.

Barthe et al. [2019] discuss relational verification in a first order predicate logic in which program variables are represented as functions $v(i, tr)$ over a time step i and trace identified by tr. The authors' encoding can express highly non-local relationships between traces such as equating the value of v at the beginning of one trace with the value of v at the end of another trace. This approach does not involve deriving an alignment, but rather enables correlating of arbitrary computation steps. The encoding in FO exploits quantifiers available in first-order provers. Although, in principle, the traces could be quantified existentially, the authors only discuss $\forall\forall$ properties non-interference and sensitivity without mention of quantifier alternation over traces.

Murray [2020] introduces a relational incorrectness logic for imperative programs, inspired by incorrectness logic of O'Hearn [2019]. Using relational semantics, the judgment relates c to d for spec R, S iff $\forall \tau, \tau'. \tau R \tau' \implies \exists \sigma, \sigma'. \sigma R \sigma' \wedge \sigma c \tau \wedge \sigma' d \tau'$. So the postcondition is an underapproximation of the reachable pairs. In a BiKAT over a KAT with havoc, this can be expressed as $\langle \mathbf{hav} \mid \mathbf{hav} \rangle; \dot{S} \leq \langle \mathbf{hav} \mid \mathbf{hav} \rangle; \dot{R}; \langle c \mid d \rangle$, generalizing O'Hearn's KAT formulation of incorrectness [O'Hearn 2019, Sect. 5.3]. Zhang et al. [2022] investigated KATs with top for (unary) incorrectness logic.

Although KAT equations under commutativity hypotheses are undecidable, there are recent positive results for other classes of hypotheses [Doumane et al. 2019; Pous et al. 2021]. Synchronous KAT [Wagemaker et al. 2019] has models based on strings-of-sets which can be interpreted as multiple simultaneous actions; this could perhaps be used to model the step-by-step alignments of Kovács et al. [2013] and Banerjee et al. [2022].

Apropos $\forall\forall$ properties, Terauchi and Aiken [2005] introduce the term 2-safety and describe a type system-based alignment used for secure information flow. Their rules (e.g. their Fig. 8)

can be formulated so that the alignments are expressed in BiKAT, leading to a more equational algebraic derivation strategy of the non-interference property. Sousa and Dillig [2016] describe Cartesian Hoare Logic for reasoning about k-safety of individual programs by alignment without explicit representation of a product program. Eilers et al. [2018] describe a k-way lock-step product encoding for single programs, that facilitates the use of k-safety procedure specifications.

Apropos $\forall\exists$ properties, Lamport and Schneider [2021] use TLA+ as a logic for deductive reasoning about such properties in the setting of temporal logic. Clochard et al. [2020] manually encode alignment products as programs in the Why3 deductive verification tool, including resolution of nondeterminacy to prove $\forall\exists$ properties. Barthe et al. [2013] define left products for proving forward simulation, in a formulation based on control flow graphs. Hawblitzel et al. [2013] use the forward simulation property (calling it *relative termination*), in translation validation using relational summaries and verification conditions. Example 7.4 was drawn from Beutner and Finkbeiner [2022] who discuss a more generalized possibilistic non-interference $\forall^k\exists^l$.

None of the preceding works provide inference rules for $\forall\exists$ judgments. The Iris-based relational logic [Frumin et al. 2018; Gäher et al. 2022] does so, in particular a judgment for contextual refinement, in forward simulation form. The logic has complex features catering for higher order concurrent programs. Prophecy variables have been added [Frumin et al. 2020] so backward simulation reasoning is available in some form. It is unclear whether the simple forward and backward rules applicable to sequential programs can be extracted from this framework. Maillard et al. [2020] develop a relational dependent type theory that accounts for core RHL rules encompassing a range of computational effects. It is not clear that the framework facilitates manipulation of intricate data-dependent alignments together with simple first-order relational assertions as used in automated relational verification.

8.4 Future Work

BiKAT provides a foundation for several interesting directions for future work in addition to questions raised in Sects. 6 and 7. As already detailed above, we are optimistic about the outlook for automation.

There are several open questions about completeness. We have proved soundness for a number of forward and backward simulation rules, but have not investigated their completeness. Completeness in the sense of "true implies provable" is problematic owing to the possibility to reduce the problem to unary Hoare logic via the self-composition rule. A more relevant notion is alignment completeness, which so far was formulated only for 2-safety [Nagasamudram and Naumann 2021]. One may think that BiKAT provides a notion of "regular alignment", which could provide a yardstick to evalute alignment completeness of other systems. However, in an applied BiKAT the interpretation of tests, in combination with conditional alignments like Eqn. (7), expresses more than regular patterns.

What about completeness of the BiKAT axioms with respect to models? For KAT, Kozen and Smith [1996] obtain completeness for relational models from a completeness result for a language model called guarded strings. A *guarded string* model is given by sets of primitive actions and primitive tests. An *atom* is a boolean valuation of primitive tests. The model is the trace model where Σ is the set of atoms and all sequences are admissible. The guarded string model has a canonical interpretation of its primitive tests and actions. Being a trace model, it gives rise to a trace BiKAT. One can then interpret primitive bitests as relations on atoms. However, the trace BiKAT for guarded strings does not determine a canonical interpretation of bitests; nor does it validate LRC. Several problems remain open: Is BiKAT, or BiKAT with projections, complete for relational models? Is there a class of BiKATs where the underlying KAT is a guarded string model, for which BiKAT is complete?

General study of algebras of programs has a long history [Hoare et al. 1987] and remains active [Höfner et al. 2019]. Because KAT is about control structure it seems particularly suited to describing alignment. For reasoning, however, other algebraic structures are relevant, e.g., Concurrent Kleene Algebra [Hoare et al. 2016] has an operator that can model spatial separation and it would be interesting to explore some notion of spatial locality in connection with alignment and BiKAT. We have highlighted the relevance of relation algebra, as also done in the Coq library of Damien Pous for KAT [Pous [n. d.]]. Such settings may be helpful for further exploration of ∀∃ properties along the lines of TriKAT, and for relational properties of nonterminating programs (e.g., using ω-algebra [Cohen 2000]). One specific question is whether some formulation of TriKAT can be used to obtain an algebraic proof to generalize Theorem 7.3.

ACKNOWLEDGMENTS

The authors would like to thank Anindya Banerjee, Lennart Beringer and Michael Greenberg for helpful discussions and the anonymous reviewers for their valuable feedback.

Authors Antonopoulos, Koskinen, Le, and Naumann were supported in part by the Office of Naval Research under Grant No. N00014-17-1-2787. Antonopoulos, Koskinen and Le were supported in part by NSF award CCF-2106845. Antonopoulos was supported in part by NSF award CCF-2131476. Naumann, Nagasamudram, and Ngo were supported in part by NSF award CNS-1718713.

REFERENCES

Martín Abadi and Leslie Lamport. 1988. The Existence of Refinement Mappings. In *Proceedings of LICS*.

Timos Antonopoulos, Eric Koskinen, and Ton Chanh Le. 2019. Specification and inference of trace refinement relations. *Proceedings of the ACM on Programming Languages* 3, OOPSLA (2019), 1–30.

Timos Antonopoulos, Eric Koskinen, Ton Chanh Le, Ramana Nagasamudram, David A. Naumann, and Minh Ngo. 2022a. An algebra of alignment for relational verification (extended version). *CoRR* abs/2202.04278 (2022). arXiv:2202.04278 https://arxiv.org/abs/2202.04278

Timos Antonopoulos, Eric Koskinen, Ton Chanh Le, Ramana Nagasamudram, David A. Naumann, and Minh Ngo. 2022b. *An algebra of alignment for relational verification (artifact).* https://doi.org/10.5281/zenodo.7144067

Krzysztof R. Apt, Frank S. de Boer, and Ernst-Rüdiger Olderog. 2009. *Verification of Sequential and Concurrent Programs* (3 ed.). Springer. https://doi.org/10.1007/978-1-84882-745-5

Sahar Badihi, Faridah Akinotcho, Yi Li, and Julia Rubin. 2020. ARDiff: Scaling Program Equivalence Checking via Iterative Abstraction and Refinement of Common Code. In *Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 13–24. https://doi.org/10.1145/3368089.3409757

Anindya Banerjee, Ramana Nagasamudram, Mohammad Nikouei, and David A. Naumann. 2022. A Relational Program Logic with Data Abstraction and Dynamic Framing. *ACM Transactions on Programming Languages and Systems* (2022). Accepted for publication. Available as http://arxiv.org/abs/1910.14560.

Anindya Banerjee, David A. Naumann, and Mohammad Nikouei. 2016. Relational Logic with Framing and Hypotheses. In *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science.* Long version at http://arxiv.org/abs/1611.08992.

Gilles Barthe, Juan Manuel Crespo, and César Kunz. 2011a. Relational Verification Using Product Programs. In *Formal Methods*.

Gilles Barthe, Juan Manuel Crespo, and César Kunz. 2013. Beyond 2-Safety: Asymmetric Product Programs for Relational Program Verification. In *Logical Foundations of Computer Science (LFCS) (LNCS, Vol. 7734)*. 29–43.

Gilles Barthe, Juan Manuel Crespo, and César Kunz. 2016. Product Programs and Relational Program Logics. *J. Logical and Algebraic Methods in Programming* 85, 5 (2016), 847–859.

Gilles Barthe, Pedro R. D'Argenio, and Tamara Rezk. 2004. Secure Information Flow by Self-Composition. In *IEEE CSFW*. See extended version [Barthe et al. 2011b].

Gilles Barthe, Pedro R. D'Argenio, and Tamara Rezk. 2011b. Secure information flow by self-composition. *Math. Struct. Comput. Sci.* 21, 6 (2011).

Gilles Barthe, Renate Eilers, Pamina Georgiou, Bernhard Gleiss, Laura Kovács, and Matteo Maffei. 2019. Verifying relational properties using trace logic. In *2019 Formal Methods in Computer Aided Design (FMCAD)*. 170–178.

Gilles Barthe, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. 2017. Coupling proofs are probabilistic product programs. In *ACM Symposium on Principles of Programming Languages*. 161–174. https://doi.org/10.1145/3009837.3009896

Bernhard Beckert and Mattias Ulbrich. 2018. Trends in relational program verification. In *Principled Software Development*. Springer, 41–58.

N. Benton. 2004. Simple Relational Correctness Proofs for Static Analyses and Program Transformations. In *POPL*. 14–25.

Lennart Beringer. 2011. Relational Decomposition. In *Interactive Theorem Proving (ITP) (LNCS, Vol. 6898)*.

Raven Beutner and Bernd Finkbeiner. 2022. Software Verification of Hyperproperties Beyond k-Safety. In *Computer Aided Verification*. 341–362. https://doi.org/10.1007/978-3-031-13185-1_17

Thomas Braibant and Damien Pous. 2010. An efficient Coq tactic for deciding Kleene algebras. In *International Conference on Interactive Theorem Proving*. 163–178.

Berkeley R. Churchill, Oded Padon, Rahul Sharma, and Alex Aiken. 2019. Semantic program alignment for equivalence checking. In *PLDI*.

Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. 2014. Temporal Logics for Hyperproperties. In *Principles of Security and Trust (POST) (LNCS, Vol. 8414)*. 265–284.

Michael R. Clarkson and Fred B. Schneider. 2010. Hyperproperties. *Journal of Computer Security* 18, 6 (2010), 1157–1210.

Martin Clochard, Claude Marché, and Andrei Paskevich. 2020. Deductive Verification with Ghost Monitors. *Proc. ACM Program. Lang.* 4, POPL (2020).

Ernie Cohen. 2000. Separation and Reduction. In *Mathematics of Program Construction (LNCS, Vol. 1837)*. 45–59. https://doi.org/10.1007/10722010_4

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

Willem-Paul de Roever and Kai Engelhardt. 1998. *Data Refinement: Model-Oriented Proof Methods and their Comparison*. Cambridge University Press.

Amina Doumane, Denis Kuperberg, Damien Pous, and Pierre Pradic. 2019. Kleene Algebra with Hypotheses. In *Foundations of Software Science and Computation Structures (FOSSACS) (LNCS, Vol. 11425)*. 207–223.

Emanuele D'Osualdo, Azadeh Farzan, and Derek Dreyer. 2022. Proving Hypersafety Compositionally. *Proc. ACM Program. Lang.* 6, OOPSLA2, Article 135 (2022), 26 pages. https://doi.org/10.1145/3563298

Marco Eilers, Peter Müller, and Samuel Hitz. 2018. Modular Product Programs. In *European Symposium on Programming*.

Azadeh Farzan and Anthony Vandikas. 2019. Automated hypersafety verification. In *Computer Aided Verification*. 200–218.

Robert Floyd. 1967. Assigning Meaning to Programs. In *Symp. on Applied Math. 19, Math. Aspects of Comp. Sci.* Amer. Math. Soc., 19–32.

Nissim Francez. 1983. Product Properties and Their Direct Verification. *Acta Informatica* 20 (1983), 329–344.

Peter J. Freyd and Andre Scedrov. 1990. *Categories, Allegories*. North-Holland.

Dan Frumin, Robbert Krebbers, and Lars Birkedal. 2018. ReLoC: A Mechanised Relational Logic for Fine-Grained Concurrency. In *IEEE Symp. on Logic in Computer Science*. 442–451.

Dan Frumin, Robbert Krebbers, and Lars Birkedal. 2020. ReLoC Reloaded: A Mechanized Relational Logic for Fine-Grained Concurrency and Logical Atomicity. *CoRR* abs/2006.13635 (2020). arXiv:2006.13635 https://arxiv.org/abs/2006.13635

Benny Godlin and Ofer Strichman. 2008. Inference rules for proving the equivalence of recursive procedures. *Acta Inf.* 45, 6 (2008), 403–439.

Manish Goyal, Muqsit Azeem, Kumar Madhukar, and R. Venkatesh. 2021. Direct Construction of Program Alignment Automata for Equivalence Checking. https://doi.org/10.48550/ARXIV.2109.01864

Michael Greenberg, Ryan Beckett, and Eric Hayden Campbell. 2022. Kleene algebra modulo theories: a framework for concrete KATs. In *PLDI*. 594–608. https://doi.org/10.1145/3519939.3523722

Lennard Gäher, Michael Sammler, Simon Spies, Ralf Jung, Hoang-Hai Dang, Robbert Krebbers, Jeehoon Kang, and Derek Dreyer. 2022. Simuliris: a separation logic framework for verifying concurrent program optimizations. *Proc. ACM Program. Lang.* 6, POPL (2022).

Chris Hawblitzel, Ming Kawaguchi, Shuvendu K. Lahiri, and Henrique Rebêlo. 2013. Towards Modularly Comparing Programs Using Automated Theorem Provers. In *CADE*. 282–299.

C. A. R. Hoare, I.J. Hayes, He Jifeng, C.C. Morgan, A.W. Roscoe, J.W. Sanders, I.H. Sorensen, J.M. Spivey, and B.A. Sufrin. 1987. Laws of Programming. *Commun. ACM* 30 (1987), 672–686,770.

Tony Hoare, Stephan van Staden, Bernhard Möller, Georg Struth, and Huibiao Zhu. 2016. Developments in concurrent Kleene algebra. *J. Log. Algebraic Methods Program.* 85, 4 (2016), 617–636. https://doi.org/10.1016/j.jlamp.2015.09.012

Peter Höfner, Damien Pous, and Georg Struth. 2019. Relational and algebraic methods in computer science. *J. Log. Algebraic Methods Program.* 106 (2019), 198–199. https://doi.org/10.1016/j.jlamp.2019.05.005

Máté Kovács, Helmut Seidl, and Bernd Finkbeiner. 2013. Relational abstract interpretation for the verification of 2-hypersafety properties. In *ACM Computer and Communications Security*.

Dexter Kozen. 1997. Kleene algebra with tests. *ACM Transactions on Programming Languages and Systems* 19, 3 (1997), 427–443.

Dexter Kozen. 2000. On Hoare logic and Kleene algebra with tests. *ACM Trans. Comput. Log.* 1, 1 (2000), 60–76.

Dexter Kozen. 2003. *Kleene algebra with tests and the static analysis of programs.* Technical Report. Cornell University.
Dexter Kozen. 2004. Some results in dynamic model theory. *Sci. Comput. Program.* 51, 1-2 (2004), 3–22.
Dexter Kozen and Frederick Smith. 1996. Kleene algebra with tests: Completeness and decidability. In *International Workshop on Computer Science Logic*. Springer, 244–259.
Leslie Lamport and Fred B. Schneider. 2021. Verifying Hyperproperties With TLA. In *IEEE Computer Security Foundations Symposium (CSF)*. 1–16.
Kenji Maillard, Cătălin Hriţcu, Exequiel Rivas, and Antoine Van Muylder. 2020. The Next 700 Relational Program Logics. *Proc. ACM Program. Lang.* 4, POPL (2020), 4:1–4:33.
Konstantinos Mamouras. 2017. Equational Theories of Abnormal Termination Based on Kleene Algebra. In *FoSSaCS*. 88–105.
Dmitry Mordvinov and Grigory Fedyukovich. 2019. Property Directed Inference of Relational Invariants. In *Formal Methods in Computer Aided Design*. 152–160. https://doi.org/10.23919/FMCAD.2019.8894274
Carroll Morgan. 1988. Auxiliary Variables in Data Refinement. *Inform. Process. Lett.* 29, 6 (1988), 293–296.
Toby Murray. 2020. An Under-Approximate Relational Logic: Heralding Logics of Insecurity, Incorrect Implementation & More. *CoRR* abs/2003.04791 (2020). arXiv:2003.04791 https://arxiv.org/abs/2003.04791
Ramana Nagasamudram and David A. Naumann. 2021. Alignment Completeness for Relational Hoare Logics. In *IEEE Symp. on Logic in Computer Science*. 1–13. Extended version at https://arxiv.org/abs/2101.11730..
David A. Naumann. 2020. Thirty-Seven Years of Relational Hoare Logic: Remarks on Its Principles and History. In *9th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*. 93–116. https://doi.org/10.1007/978-3-030-61470-6_7 Extended version at https://arxiv.org/abs/2007.06421..
Peter W O'Hearn. 2019. Incorrectness logic. *Proceedings of the ACM on Programming Languages* 4, POPL (2019), 1–32.
Lauren Pick, Grigory Fedyukovich, and Aarti Gupta. 2018. Exploiting Synchrony and Symmetry in Relational Verification. In *Computer Aided Verification*. 164–182.
Damien Pous. [n. d.]. Relation Algebra and KAT in Coq. http://perso.ens-lyon.fr/damien.pous/ra/ Coq library, accessed July 2022.
Damien Pous. 2015. Symbolic algorithms for language equivalence and Kleene algebra with tests. In *ACM Symposium on Principles of Programming Languages*. 357–368.
Damien Pous, Jurriaan Rot, and Jana Wagemaker. 2021. On Tools for Completeness of Kleene Algebra with Hypotheses. In *Relational and Algebraic Methods in Computer Science (RAMiCS) (LNCS, Vol. 13027)*. 378–395.
Andrei Sabelfeld and Andrew C. Myers. 2003. Language-Based Information-Flow Security. *IEEE J. Selected Areas in Communications* 21, 1 (Jan. 2003), 5–19.
Ron Shemer, Arie Gurfinkel, Sharon Shoham, and Yakir Vizel. 2019. Property directed self composition. In *International Conference on Computer Aided Verification*. Springer, 161–179.
Marcelo Sousa and Isil Dillig. 2016. Cartesian Hoare logic for verifying k-safety properties. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 57–69.
Tachio Terauchi and Alex Aiken. 2005. Secure information flow as a safety problem. In *International Static Analysis Symposium*. Springer, 352–367.
Hiroshi Unno, Tachio Terauchi, and Eric Koskinen. 2021. Constraint-Based Relational Verification. In *Computer Aided Verification*. 742–766.
Jana Wagemaker, Marcello M. Bonsangue, Tobias Kappé, Jurriaan Rot, and Alexandra Silva. 2019. Completeness and Incompleteness of Synchronous Kleene Algebra. In *Mathematics of Program Construction (LNCS, Vol. 11825)*. 385–413.
Hongseok Yang. 2007. Relational Separation Logic. *Theo. Comp. Sci.* 375 (2007).
Cheng Zhang, Arthur Azevedo de Amorim, and Marco Gaboardi. 2022. On incorrectness logic and Kleene algebra with top and tests. *Proc. ACM Program. Lang.* 6, POPL (2022), 1–30. https://doi.org/10.1145/3498690

Received 2022-07-07; accepted 2022-11-07