Logical Methods in Computer Science
Volume 21, Issue 4, 2025, pp. 21:1–21:68
https://lmcs.episciences.org/

Submitted Jul. 11, 2024
Published Nov. 10, 2025

ALIGNMENT COMPLETE RELATIONAL HOARE LOGICS FOR SOME AND ALL

RAMANA NAGASAMUDRAM [a], ANINDYA BANERJEE [b], AND DAVID A. NAUMANN [a]

[a] Stevens Institute of Technology, USA

[b] Dartmouth College, USA

Abstract. In relational verification, judicious alignment of computational steps facilitates proof of relations between programs using simple relational assertions. Relational Hoare logics (RHL) provide compositional rules that embody various alignments of executions. Seemingly more flexible alignments can be expressed in terms of product automata based on program transition relations. A single degenerate alignment rule (sequential composition), atop a complete Hoare logic, comprises a RHL for $\forall\forall$ properties that is complete in the sense of Cook. The notion of alignment completeness was previously proposed as an additional measure, and some rules were shown to be alignment complete with respect to a few ad hoc forms of alignment automata. This paper proves alignment completeness with respect to a general class of $\forall\forall$ alignment automata, for a RHL comprised of standard rules together with a rule of semantics-preserving rewrites based on Kleene algebra with tests. A new logic for $\forall\exists$ properties is introduced and shown to be sound and alignment complete for a new general class of automata. The $\forall\forall$ and $\forall\exists$ automata are shown to be semantically complete. Thus both logics are complete in the sense of Cook. The paper includes discussion of why alignment is not the only important principle for relational reasoning and proposes entailment completeness as further means to evaluate RHLs.

1. Introduction

A ubiquitous problem in programming is reasoning about relational properties, such as equivalence between two programs in the sense that from any given input they produce the same output. Relational properties are also of interest for a single program. For example, a basic notion in security is noninterference: any two executions with the same public inputs should result in the same public outputs, even if the secret inputs differ [SM03]. We write $c \mid c' : \mathcal{R} \approx\!> \mathcal{S}$ to say program c relates to program c' in the sense that for any pair of initial states related by $\mathcal{R}$, and terminated executions of c and c' from those states, the final states are related by $\mathcal{S}$. For example, letting $\mathbb{A}low$ say two states agree on the values of low-security variables, $c \mid c : \mathbb{A}low \approx\!> \mathbb{A}low$ specifies termination-insensitive noninterference for c.

Key words and phrases: logics of programs; relational properties; semantic completeness; relational Hoare logic; relative completeness; inductive assertion method.

Nagasamudram and Naumann were partially supported by NSF grants CNS 1718713 and CNS 2426414. Banerjee's research was based on work supported by the NSF, while working at the Foundation. Any opinions, findings, and conclusions or recommendations expressed in this article are those of the authors and do not necessarily reflect the views of the NSF.

LOGICAL METHODS IN COMPUTER SCIENCE
DOI:10.46298/LMCS-21(4:21)2025

Another kind of relational property is written $c \mid c' : \mathcal{R} \overset{\exists}{\approx\!>} \mathcal{S}$ and deals with nondeterminacy. It says that from any $\mathcal{R}$-related pair of initial states, and for any terminated run of c from the left state, there is a terminated run of c' from the right state such that $\mathcal{S}$ relates the final states. For example, a standard notion of refinement is obtained by taking $\mathcal{R}$ and $\mathcal{S}$ to be the identity. For data refinement [dRE98], let $\mathcal{R}$ connect two different data representations and let $\mathcal{S}$ be $\mathcal{R}$. For brevity we refer to the two kinds of relational properties as $\forall\forall$ and $\forall\exists$.

Often a pair of executions are similar, perhaps because the related programs are similar, or even the same as in the case of noninterference. So reasoning often relies on the *alignment* of similar steps —think of a programmer viewing two versions of a similar program, side by side on screen. Having chosen to align a pair of points in control flow, one can consider a relational assertion that holds whenever execution reaches those points.

Most work on formal reasoning about relational properties has focused on $\forall\forall$ properties which capture many requirements for deterministic programs. There are two main approaches. One is deductive systems inspired by Hoare logic (***HL***), which we call *relational Hoare logics* (***RHL***s) [Fra83, Ben04]. The other approach is to reason about program semantics in the form of an automaton (i.e., transition system), adapting the inductive assertion method (***IAM***) [Flo67] to use relational assertions at aligned pairs of control points. By representing the two programs as a product automaton one can easily express which control points are meant to be aligned. Alignments can be conditioned on the programs' data, with such conditions expressed using relational assertions. In case the program transition relation(s) can be expressed in a solvable fragment of first order logic, it is even possible to automatically infer alignment conditions and relational assertions [SGSV19, UTK21], using proof search in the constraint language instead of proof search in a logic of programs.

Deductive systems are important for several reasons. Because a deductive system applies to ordinary program syntax, it caters for human interaction which is essential for verification of programs involving dynamically allocated data and other features for which program semantics and/or specifications are not easily expressible in solvable fragments. Deductive proofs can also serve as independently checkable certificates to represent proofs that may be found by other means.

Finally, deductive rules can embody principles beyond IAM. The rule of conjunction decomposes a goal with a conjunctive postcondition into simpler subgoals which may be proved by IAM or deductively. Rules for procedure call and linking account for procedure modular verification, which in the IAM setting is deployed through use of specs as procedure summaries. These principles apply in both unary and relational reasoning, but there are further principles for relational reasoning. A well known relational principle is vertical —i.e., transitive— composition, which is sound for $\forall\exists$ but not $\forall\forall$. This makes $\forall\exists$ logic useful even for deterministic programs [HKLR13, DFD22].

Aside from soundness, the fundamental criterion for a deductive system is completeness: any true correctness judgment should be provable. For program logics one seeks Cook's *relative completeness* [Coo78, AdBO09, Win93], factoring out completeness and expressiveness of the assertion logic. The problem we address in this paper is that Cook completeness is unsatisfactory for RHLs.

Consider this sound rule: from $c \mid \mathsf{skip} : \mathcal{R} \approx\!> \mathcal{Q}$ and $\mathsf{skip} \mid c' : \mathcal{Q} \approx\!> \mathcal{S}$ infer $c \mid c' : \mathcal{R} \approx\!> \mathcal{S}$. The rule shows how deductive rules can embody alignments, in this case a *sequential alignment* wherein the final state of c is aligned with the initial state of c'. Under mild assumptions about encoding a pair of states as a single state, and thereby treating a relation

as a *unary* assertion, the premises of this rule can be expressed as partial correctness judgments in HL [Fra83, BDR04]. The sequential alignment rule, together with a complete set of unary rules, provides a Cook complete relational logic [Fra83, BDR04, Ber11]. Yet it is well known that sequential alignment is unsatisfactory. Consider proving $c \mid c : \mathcal{I} \approx\!\!> \mathcal{I}$ where $\mathcal{I}$ is a conjunction of agreements, one for each variable of c. Surely c is equivalent to itself as expressed by $\mathcal{I} \approx\!\!> \mathcal{I}$. By aligning c with itself step by step, the only intermediate assertion we need is $\mathcal{I}$ —whereas for reasoning about the sequential alignment, the intermediate assertion $\mathcal{Q}$ may have to "remember everything" [Fra83] about c. In particular this may require finding strong invariants for loops. Good alignment is essential for relational verification because it enables the use of relatively simple assertions [SGSV19]. As an example how different alignments can be formalized, here is a standard rule of RHL: from $c \mid c' : \mathcal{R} \approx\!\!> \mathcal{Q}$ and $d \mid d' : \mathcal{Q} \approx\!\!> \mathcal{S}$ infer $c; d \mid c'; d' : \mathcal{R} \approx\!\!> \mathcal{S}$. Another standard rule aligns loop iterations (see rAlgnDo in Figure 3). To be clear, sequential alignment has important uses, including cases where the two programs have different structure or simply differ in the order of atomic operations like assignments.

The notion of *alignment completeness* has been proposed as an additional criterion for RHLs [NN21]. Here's the idea: For any valid alignment-based proof there should be a proof of the same judgment in the RHL *using essentially the same assertions*. The approach of [NN21] uses automata to represent alignments and IAM proofs (as in e.g. [SGSV19, CPSA19, UTK21, ISV24]). In [NN21], alignment completeness results are only given for $\forall\forall$ properties and a few specialized kinds of automata for which there is a straightforward connection with specific proof rules. For example, for lockstep alignments of two programs with the same control structure one gets alignment completeness for the rules of Benton [Ben04] (our rules rAsgnAsgn, rSeq, rAlgnIf, rAlgnDo, rConseq in Figures 2 and 3).

In a nutshell, we make two contributions. First, we answer the challenge in [NN21] to obtain alignment completeness for a general class of $\forall\forall$ alignment automata. Second, we introduce a new logic for $\forall\exists$ properties and prove its soundness and alignment completeness. These achievements overcome several challenges.

Challenge: Find an interesting/useful general class of alignment automata. A key requirement is that the automaton be *adequate* in the sense of covering all pairs of executions. Adequacy has been worked out in several independent works (with terminology like "fair scheduler"), for $\forall\forall$ properties, but there is no standard theory, and there is little relevant work on $\forall\exists$. Our solutions are in sections 4 and 9. Alignment conditions and the choice of existential witnesses are specified by state relations as is done in many practical works on relational verification, like those cited above. For $\forall\exists$, our automata formulation (in section 9) shows that to prove a $\forall\exists$-spec $\mathcal{R} \overset{\exists}{\approx\!\!>} \mathcal{S}$, instead of explicitly constructing a positive witness of the existential [BCB+21, LS21] one can filter out the non-witnesses on the right, so what's left satisfies the corresponding $\forall\forall$-spec $\mathcal{R} \approx\!\!> \mathcal{S}$. All for some!

Challenge: Find a good set of RHL rules. In this paper we confine attention to simple imperative programs but, even so, a large number of rules can be found in the $\forall\forall$ RHL literature, and there is little prior work on $\forall\exists$ RHLs. For $\forall\forall$, several rules seem to be needed to account for the many possible program structures one may wish to align, e.g., relating a loop to a sequence of loops. The key is to include some form of semantics-preserving rewriting, so programs can be rewritten to have more similar control structure for which fewer rules are needed.

Challenge: Find a sensible notion of equivalence for use in rewriting. Prior works that appeal to rewriting in a deductive logic use ad hoc sets of rewrite rules (e.g., [BNN16, Appendix D],[BGHS17, BNNN22]), which raises a fresh question about completeness. Another question is whether equivalence should be a distinct judgment as opposed to a special case of the main relational judgment, and if so, should it involve preconditions.

Challenge: For alignment completeness, find a systematic way to obtain deductive proofs from automata-based ones.

Our answer to the latter two challenges is a key technical result: We show that every program can be rewritten, using laws of Kleene algebra with tests (***KAT***) [Koz97], into *automaton normal form* which mimics its transition system representation (section 5). As noted in the related work section 10, this is not too surprising but the specific form is new and plays a crucial role in proving the main results.

Technical contributions:

- We show that RHL+, a logic featuring a KAT-based rewrite rule (Figure 2), is alignment complete for $\forall\forall$ proofs based on a general class of alignment automata.
- We introduce a new general class of $\forall\exists$ alignment automata.
- We introduce ERHL+ and show it is sound and alignment complete for proofs of $\forall\exists$ properties.
- We show that both classes of automata are sound and semantically complete. Together with alignment completeness, this yields Cook completeness for both logics.

As a conceptual contribution, we show that for capturing IAM-style proofs, these core rules of RHL for imperative programs suffice: (a) one-sided rules for primitive commands, (b) same-structure rules for sequence, conditional, and (conditionally aligned) loop, and (c) rewriting a program to an unconditionally equivalent one. With these rules, many convenient rules found in the literature are derivable.

Although the IAM is clearly fundamental and widely used in relational verification (see section 10), it is not the only useful reasoning principle. Even for unary reasoning about simple imperative programs, it is common to augment the core complete set of rules (i.e., syntax-directed plus Consequence) with, for example, the rule of Conjunction which facilitates modular proofs [AdBO09]. For relational reasoning there are other natural principles such as transitivity which are not derivable from the core rules we use to obtain alignment completeness. As future work we highlight some reasoning challenges that motivate additional rules. This raises the question what are good criteria for RHLs in addition to alignment completeness, a question for which we propose a possible answer dubbed entailment completeness.

Outline. Section 2 provides an overview. Section 3 presents the $\forall\forall$ logic RHL+. Section 4 develops alignment products and their verification conditions for $\forall\forall$ properties. Section 5 develops the normal form. Section 6 gives the unary Floyd completeness result sketched in section 2, illuminating some ingredients of the proof of alignment completeness of RHL+ which appears in Section 7. Section 8 introduces the $\forall\exists$ logic ERHL+. Section 9 extends alignment products with filtering conditions and shows alignment completeness of ERHL+. Sections 7.2 and 9.3 revisit Cook completeness both based on alignment completeness and based on sequential alignment and unary logics. Related work is discussed in section 10 and future work in section 11. Section 12 concludes. Some details are in the appendix.

This article is meant to be self-contained and we include considerable details which lengthen the text but may shorten the reading. There is a longer document with additional technical details [NBN23].

2. Overview

Floyd's formulation of the IAM is based on annotation of control points, representing programs as automata. We use Dijkstra's guarded command syntax but with labelled control points. This somewhat silly command $c0$ will be a running example. Here mod means remainder and the superscripts $1 \ldots 5$ are labels.

$$c0: \quad x :=^1 y; \mathsf{do}^2 x > 0 \to \mathsf{if}^3 x \bmod 2 = 0 \to x :=^4 x - 1 \,[]\, x \bmod 2 \neq 0 \to x :=^5 x - 2 \;\mathsf{fi}\; \mathsf{od}$$

We aim to prove $c0 \mid c0 : \mathbb{A}y \approx\!\!> \mathbb{A}x$, which says any two runs of $c0$, from initial stores that agree on y, result in final stores that agree on x. (This expresses that the final value of x depends only on the initial value of y.) Pairs of runs can be represented by executions of a product automaton whose control is successively the same control point on left and right. An IAM-style proof annotates each control point (n, n) of the product with the relation $\mathbb{A}x$, except for $(1, 1)$ which is annotated by the precondition $\mathbb{A}y$. A deductive proof formulates such lockstep alignment using rules like rSeq (Figure 2) and rAsgnAsgn (Figure 3). By contrast, a sequential product automaton would run the left execution to completion and then the right, annotated with more complicated assertions than $\mathbb{A}x$. Such a proof can be presented using the sequential alignment rule mentioned in section 1 (cf. rLRseq in Figure 3).

The mentioned forms of automata can be made precise using conditions on control points to designate when left-side, right-side, or joint steps should be taken. Later we consider products where the alignment conditions can also depend on the stores.

Rewriting facilitates expression of alignment. A core element of our logic RHL+ is a KAT-based rule for rewriting of programs. It says that to show $c \mid c' : \mathcal{P} \approx\!\!> \mathcal{Q}$, it suffices to show $d \mid d' : \mathcal{P} \approx\!\!> \mathcal{Q}$ provided d and d' are equivalent to c and c' respectively. A minimal amount of equivalence-preserving rewriting using simple laws derived from KAT can open up opportunities for better alignments. KAT equality is sufficient for our purposes and it is well suited to serve as an auxiliary judgment in a program logic because it can be presented by a deductive system (equational logic) and even as a decidable premise of the rewrite rule as we note in passing later (Remark 5.5).

As an example we consider a variation on the loop tiling optimization example of Barthe et al [BCK13]. Both $c1$ and $c1'$ shown below compute the sum of all integers from 0 to $n \times m - 1$ for some $n, m > 0$. For this discussion we can omit labels on commands.

$$c1: \quad \mathsf{do}\ i < n \times m \to s := s + i;\ i{+}{+}\ \mathsf{od}$$
$$c1': \quad \mathsf{do}\ i < n \to j := 0;\ \mathsf{do}\ j < m \to (s := s + i \times m + j;\ j{+}{+})\ \mathsf{od};\ i{+}{+}\ \mathsf{od}$$

We aim to show equivalence under a precondition:

$$c1 \mid c1' : \mathbb{A}i \wedge \langle\!|i|\!\rangle = 0 \wedge \mathbb{A}s \wedge \langle\!|s|\!\rangle = 0 \wedge \mathbb{A}n \wedge \mathbb{A}m \approx\!\!> \mathbb{A}s \tag{2.1}$$

(Some notation: $\langle\!|i|\!\rangle$ is the value of i in the left state, and $|\!|i|\!\rangle$ is its value on the right, and $\mathbb{A}i$ is an abbreviation for $\langle\!|i|\!\rangle = |\!|i|\!\rangle$.) A naive alignment would require lining up the inner loop in $c1'$ with the body of the loop in $c1$. But this would require us to summarize the effect of the

inner loop in $c1'$ which is an avoidable complication. Instead we start by rewriting the two programs to the following.

$c2:$ $\mathsf{do}\ i < n \times m \to s := s + i; i{+}{+}; \mathsf{do}\ i < n \times m \wedge i \bmod m \neq 0 \to s := s + i; i{+}{+}\ \mathsf{od}\ \mathsf{od}$

$c2':$ $\mathsf{do}\ i < n \to j := 0;\ \mathsf{if}\ j < m\ \mathsf{then}\ s := i \times m + j;\ j{+}{+}\ \mathsf{else\ skip\ fi};$
$\mathsf{do}\ j < m \to (s := s + i \times m + j;\ j{+}{+})\ \mathsf{od};\ i{+}{+};\ \mathsf{od}$

We obtain $c2'$ by unrolling the inner loop in $c1'$ once, and $c2$ by using the fact that $(\mathsf{do}\ b \to c\ \mathsf{od})$ and $(\mathsf{do}\ b \to c; \mathsf{do}\ b \wedge b_0 \to c\ \mathsf{od}\ \mathsf{od})$ are equivalent for any choice of b, b_0 and c. Unconditional equivalences like these are purely about control structure, and encompassed by KAT's "propositional" view of programs [Koz97]. Now, by the rewriting rule of RHL+ it suffices to show $c2$ and $c2'$ satisfy the spec in (2.1). We can establish this by aligning the outer and inner loops in lockstep. The outer loops admit $\langle\!|i|\!\rangle = \flat i \times m\rangle \wedge \mathbb{A}s$ as relational invariant and this is sufficient to establish the postrelation. The inner loops admit $\langle\!|i|\!\rangle = \flat i \times m + j\rangle \wedge \mathbb{A}s$ as relational invariant.

Nondeterminacy and $\forall\exists$ judgments. While $\forall\forall$ specs capture a wide class of requirements, they do not express all interesting properties of nondeterministic programs. Consider the havoc command $\mathsf{hav}\ x$ which sets x to any integer. The $\forall\forall$ judgment $\mathsf{hav}\ x | \mathsf{hav}\ x : \mathit{true} \approx\!\!> \mathbb{A}x$ does not hold. However, for every execution of $\mathsf{hav}\ x$, there exists an execution of $\mathsf{hav}\ x$ such that $\mathbb{A}x$. This property is captured by the $\forall\exists$ judgment $\mathsf{hav}\ x \mid \mathsf{hav}\ x : \mathit{true} \overset{\exists}{\approx\!\!>} \mathbb{A}x$. We establish this judgment by picking an alignment and, in addition, a way of filtering out executions of the right program that violate the post-relation. One proof uses the sequential alignment that performs $\mathsf{hav}\ x$ on the left followed by $\mathsf{hav}\ x$ on the right. Filtering is achieved by *assuming* the relation $\mathbb{A}x$ after the $\mathsf{hav}\ x$ on the right. In other words, to establish $\mathbb{A}x$, we only consider executions of $\mathsf{hav}\ x$ on the right which match the left side value.

Filtering conditions are sound provided they permit all executions of the left program and at least one execution of the right program. In this example, the filtering condition $\mathbb{A}x$ is justified because every possible value of x that can be produced by $\mathsf{hav}\ x$ on the left can also be produced by $\mathsf{hav}\ x$ on the right. An alignment automaton that enforces the filter effectively reduces the $\forall\exists$ to a $\forall\forall$ property that can be proved by IAM. As another example, consider the invalid judgment

$$\mathsf{hav}\ x \mid \mathsf{hav}\ x; x := 2 \times x \ :\ \mathit{true} \overset{\exists}{\approx\!\!>} \mathbb{A}x \qquad \text{(invalid)}$$

It is not sound to assume $\mathbb{A}x$ because odd values of x on the left cannot be matched on the right, considering that x is an integer variable. In our $\forall\exists$ logic ERHL+, such "assumptions" are expressed by postconditions in $\overset{\exists}{\approx\!\!>}$ specs. We do not use verification exotica like assume statements or nonstandard program semantics.

Possibilistic Noninterference. For a more involved example with nondeterminism, consider the following program adapted from [UTK21]. It illustrates how alignment facilitates use of simple conditions for filtering.

$c3:$ $\mathsf{if}^1\ \mathit{high} \neq 0 \to \mathsf{hav}^2\ x;$
$\mathsf{if}^3\ x \geq \mathit{low} \to \mathsf{skip}^4\ [\!]\ x < \mathit{low} \to \mathsf{do}^5\ \mathit{true} \to \mathsf{skip}^6\ \mathsf{od}\ \mathsf{fi}$
$[\!]\ \mathit{high} = 0 \to x :=^7 \mathit{low};\ \mathsf{hav}^8\ b;$
$\mathsf{do}^9\ b \neq 0 \to x :=^{10} x + 1;\ \mathsf{hav}^{11}\ b\ \mathsf{od}\ \mathsf{fi}$

This satisfies a possibilistic noninterference property: from low-equivalent states, for any terminated execution of $c3$, there exists an execution of $c3$ such that the final values of x in both states are in agreement. This is expressed by the $\forall\exists$ judgment $c3 \mid c3 : \mathbb{A}low \overset{\exists}{\approx\!>} \mathbb{A}x$.

We verify this property of $c3$ by choosing a convenient alignment and a collection of filtering conditions associated with aligned points. The alignment is as follows. From a pair of initial states where both runs take the same branch of the conditional at label 1, consider the executions in lockstep. Otherwise, consider the left run up to termination, then reason about the right run, i.e., reason in terms of a left-first sequential alignment.

Suppose $high \neq 0$ in both initial states, so we reason in terms of a lockstep alignment: the points labelled (2,2) are aligned, as are (3,3). At (3,3) we assume $\mathbb{A}x$, to filter out executions where the aligned havocs disagree. With this filtering, it's easy to prove the post-relation $\mathbb{A}x$ holds, because the inner conditional takes the same branch on both sides. Notice that the diverging loop at label 5 does not falsify the noninterference property since we only consider pairs of executions in which $c3$ has terminated on the left. The case where $high = 0$ in both initial states is similar.

When $\langle\!|high|\!\rangle \neq 0$ and $|\!|high|\!| = 0$ holds in the initial states, we reason in terms of a left-first sequential alignment. Given that the left run of $c3$ is terminated, we have $\langle\!|x|\!\rangle \geq \langle\!|low|\!\rangle$ preceding the point we start to reason about the right run. This run first sets x on the right to low and then the loop at label 9 increments it some nondeterministically chosen number of times. We filter the right execution so that we maintain $|\!|b|\!| \geq 0 \wedge |\!|b|\!| = \langle\!|x|\!\rangle - |\!|x|\!|$ as an invariant of this loop. Intuitively, we only permit executions on the right in which $\mathsf{hav}^8\ b$ sets b to the difference between x on the two sides and $\mathsf{hav}^{11}\ b$ decrements b by 1 to maintain the invariant. Since the $\forall\exists$ judgment concerns existence of right executions, we must prove termination of the loop at label 9. We do so using the value of b as a variant of the loop. By the chosen filtering conditions, this quantity decreases in each iteration.

Left-first sequential alignment is also used for the case when $\langle\!|high|\!\rangle = 0$ and $|\!|high|\!| \neq 0$. The key idea here is to resolve $\mathsf{hav}^2\ x$ on the right so that $|\!|x|\!| \geq |\!|low|\!|$ holds, so we only consider right executions that don't diverge due to the loop at label 5. This is again done by filtering right executions so we maintain agreement on x.

On heuristics. The preceding proof sketch reflects some straightforward heuristics that are also applicable to $\forall\forall$ properties. To reason about executions of two programs with similar control structure, consider lockstep alignment following the control structure. For programs with different control structure, consider aligning the executions sequentially. In addition, for $\forall\exists$ judgments, align the left ($\forall$) execution before the right, so that filter conditions for right side nondeterminacy can refer to what happened on the left.

Data dependent alignment. Prior works showed the need for alignments to be conditioned on program state, which can be realized in some forms of product automata [SGSV19, CPSA19]. In a deductive proof of the $c3$ example, the case distinctions of the standard 4-way if-rule of RHLs directly enable the use of different alignments in different cases as we show later (Example 8.1). A deductive proof can also use the disjunction rule to introduce different cases in which different alignments can be used. For loops, a number of published RHLs offer only restricted alignment patterns, but Beringer [Ber11] formulated a loop rule that features auxiliary relations to designate the conditions under which iterations are aligned together or proceed on just one side. Our logics use this rule. Our results show that these ingredients suffice for deductive proofs to use any alignments expressed using automata.

Alignment conditions in product automata like those cited above are local in the sense of referring to the current (pair of) program states. It is conceivable to formulate alignment notions or other relational reasoning globally, in terms of entire program executions (see section 10), but this is beyond the assertional methods on which we focus in this paper.

Alignment completeness. The examples show the value of good alignment for both $\forall\forall$ and $\forall\exists$ reasoning. They also show that a deductive system must cater for relating both similar and different programs, to express various alignments. For $\forall\exists$, filtering must be expressed somehow, while disallowing unjustified assumptions. Our main results show that for a minimal set of rules it suffices to have one-sided rules for primitives (e.g., rAsgnSkip and rSkipAsgn in Figure 2) and same-structure rules for control structures (e.g., rSeq in Figure 2). Known examples show the need for alignment conditioned on data [SGSV19, CPSA19, BNNN22]; our main results show that it suffices to have conditional alignment in the loop rule. A key rule allows rewriting the programs into equivalent ones with different control structure.

In a human guided deductive proof one should rewrite the programs as little as possible, just enough to express a desired alignment as in the examples above. (A proof search procedure might select rewrites based on a similarity measure between programs to be related, but that is beyond the scope of this paper.) Rewriting can also be used to derive convenient general rules for common cases (Figure 3). Indeed, a few such derived rules suffice to capture the alignments described above for the $c3$ example. However, for the theoretical purpose of proving alignment completeness we go to the opposite extreme, rewriting the program into a normal form like a fetch-execute loop.[1] By "compiling" programs we make it possible to obtain a deductive proof from an IAM-style proof in the form of an aligned and annotated product automaton. Section 11 revisits minimal rewriting in connection with alignment completeness as a criterion for RHLs.

Our approach to proving alignment completeness can be explained most simply by considering unary correctness. Suppose we have an IAM proof that $c0$ satisfies an ordinary pre-post spec, so there is an inductive annotation of its control points. Instrument the program with a fresh variable pc serving as program counter. We write $!n$ to abbreviate $pc := n$ and $?n$ to abbreviate the expression $pc = n$ (to be precise, $!n$ is $pc :=^i n$ for some arbitrary i). The result looks as follows, using 6 as a final control point, which in IAM is annotated with the postcondition.

$$c0^+ : \quad !1; x :=^1 y; !2; \mathsf{do}^2 x > 0 \to !3; \mathsf{if}^3 x \bmod 2 = 0 \to !4; x :=^4 x - 1 \;[]\; x \bmod 2 \neq 0 \to !5; x :=^5 x - 2 \;\mathsf{fi}\; ; !2 \;\mathsf{od}\; ; \; !6$$

Using simple transformations, this can be rewritten into the equivalent form $!1; d0$ where

$$\begin{array}{lllll} d0 : & \mathsf{do} & ?1 \to x := y; !2 & []?3 \wedge x \bmod 2 = 0 \to !4 & []?4 \to x := x - 1; !2 \\ & & []?2 \wedge x > 0 \to !3 & []?3 \wedge x \bmod 2 \neq 0 \to !5 & []?5 \to x := x - 2; !2 \\ & & []?2 \wedge x \not> 0 \to !6 & & \mathsf{od} \end{array}$$

(When considering programs of this form, with explicit pc, the labels no longer matter and we omit them.) For the loop invariant we use[2] $1 \leq pc \leq 6 \wedge (\wedge i : 0 \leq i \leq 6 : pc = i \Rightarrow an(i))$ where $an(i)$ is the given annotation at control point i. To use the loop rule (Figure 1) there is a premise for each guarded command in $d0$, and *each of these corresponds closely to one*

[1] Please note that we are using the term "normal form" for a syntactic notion (like A-normal form used in compilation [FSDF93]), not in the sense of term rewriting systems [BN99].

[2] We use the notation (*operator variables* : *range* : *term*) of Dijkstra and Scholten [DS90].

of the VCs in the IAM proof. It turns out that this enables us to prove each premise using assignment, sequence, and consequence rules. Thus we get that $!1; d0$ satisfies the spec. Then, since $!1; d0$ is equivalent to $c0^+$, we have by the Rewrite rule that $c0^+$ satisfies the spec. But pc is fresh so the rule for ghost variables lets us erase the instrumentation. Erasing produces extraneous skips; these are removed by another application of Rewrite, completing the proof that $c0$ satisfies the spec. All assertions used in this proof are essentially boolean combinations of the assertions in the given IAM proof.

This argument works for any IAM proof of a unary spec, owing to our normal form Theorem 5.13 that says any program, instrumented with pc, is equivalent to its corresponding automaton normal form. The general result that one can obtain a deductive proof from an IAM proof is called Floyd completeness; we prove it for HL+ introduced later (Theorem 6.1).

The same approach works to prove alignment completeness for a relational logic. Suppose we are given an IAM-style proof using an alignment product. Compile each of the programs to its normal form. Instantiate the relational loop rule (rDo in Figure 2, eDo in Figure 15) using the automaton's alignment conditions. The side conditions of the loop rule follow from assumed adequacy conditions of the automaton-based proof. The premises of the loop rule correspond to the verification conditions of the automaton.

There are a number of technical challenges to work out these ideas in detail. Many of the details in section 4 for $\forall\forall$ alignment completeness are also used for the $\forall\exists$ alignment completeness result where there are additional complications about the filtering conditions and well-founded right-side alignment.

3. The programming language and its $\forall\forall$ relational Hoare logic

This section formalizes the programming language and presents RHL+, the $\forall\forall$ relational logic. Given that we will use KAT extensively, one might carry out the entire formal development using a KAT-like notation for programs (as is nicely done in [O'H20] for example). We make a different design decision, using a more conventional notation for programs. This may facilitate comparison with other logics and it enables direct application of the logics to examples. It does require a little extra work for translating programs to KAT terms, which is largely confined to section 5.

Guarded command language. The labelled guarded command syntax is defined as follows, where x ranges over a countable set Var of integer variables, e ranges over integer and boolean expressions, n ranges over integer literals.

$$\begin{array}{lcl} c & ::= & \mathsf{skip}^n \mid x :=^n e \mid \mathsf{hav}^n\ x \mid c; c \mid \mathsf{if}^n\ gcs\ \mathsf{fi} \mid \mathsf{do}^n\ gcs\ \mathsf{od} \\ gcs & ::= & e \to c \mid e \to c \,[]\, gcs \end{array}$$

We omit details about expressions except to note that boolean expressions are given from some primitives $bprim$ and the logical operators are written $\wedge, \vee, \neg$. A ***guarded command*** has the form $e \to c$ where e is a boolean expression. The category gcs is essentially non-empty lists of guarded commands and we sometimes treat it as such. The havoc command $\mathsf{hav}^n\ x$ nondeterministically assigns any integer to x; it can model inputs, randomization, and unknowns. The usual imperative control structures are special cases: $\mathsf{if}\ e \to c \,[]\, \neg e \to d\ \mathsf{fi}$ and $\mathsf{do}\ e \to c\ \mathsf{od}$.

Later we define a predicate, ok, on commands that says their labels are unique and positive. Labels play an important role in some results, for which the ok condition is needed.

But many definitions and results do not involve labels or require ok; for those definitions and results we omit labels, meaning that any labels are allowed.

Programs act on ***variable stores***, i.e., total functions $\mathsf{Var} \to \mathbb{Z}$. Later we consider automata with arbitrary sets of stores that need not be variable stores, and the term "state" will refer to a store together with a control point. In this section we say simply "store", meaning variable store. We assume expressions e are always defined. We write $[\![e]\!](s)$ for the value of expression e in store s. In case e is a boolean expression, the value is in $\{true, false\}$; otherwise it is in $\mathbb{Z}$. We use standard big-step semantics and write $[\![c]\!]\, s\, t$ to express that from initial store s the command c can terminate with final store t. (See Figure 23 in Appendix A.) We call $[\![c]\!]$ the ***denotation*** of c.

The standard semantics for guarded commands considers if gcs fi to fail if none of its guards is enabled [AdBO09]. Modeling failure would clutter the semantics without shedding any light, so we disallow such ifs, as follows. Define $\mathsf{enab}(gcs)$ as the disjunction of the guards. For example, $\mathsf{enab}(x > 0 \to y := 1 \,[\!]\, z < 1 \to y := 2)$ is the expression $x > 0 \vee z < 1$.

Definition 3.1. A command c is ***well formed*** if (a) c is typable in the sense that guards are boolean expressions and both integer and boolean operators are used sensibly, considering that all variables have type int; and (b) c satisfies the **total-if** condition which says: for every subprogram if gcs fi of c, $[\![\mathsf{enab}(gcs)]\!] = [\![true]\!]$. In the sequel we say ***command*** to mean well formed command.[3] We refer to the language of well formed commands as the ***guarded command language***, (***GCL***).

Program logic. Our relational logics are not based on unary HL, but for expository purposes we occasionally touch on HL. The partial correctness judgment is written $c : P \rightsquigarrow Q$ rather than the conventional $\{P\}c\{Q\}$. To focus on what's interesting we treat assertions as shallowly embedded:[4] so P and Q are sets of stores. But we use formula notation for clarity, e.g., $P \wedge Q$ means their intersection. A boolean expression e in a formula stands for $\{s \mid [\![e]\!](s) = true\}$, which we sometimes write as $[\![e]\!]$. For a set P of variable stores, we use substitution notation P^x_e with the standard meaning: s is in P^x_e iff the updated store $s[x \mapsto [\![e]\!](s)]$ is in P. (We write $s[x \mapsto v]$ for the store like s but with x mapped to v.) Quantifiers, as operators on store sets, are defined by $\dot\forall x.\, P \;\hat{=}\; \{s \mid \forall v \in \mathbb{Z}.\, s \in P^x_v\}$ and $\dot\exists x.\, P \;\hat{=}\; \{s \mid \exists v \in \mathbb{Z}.\, s \in P^x_v\}$. Define $\mathsf{indep}(x, P)$ to mean that $P = \dot\exists x.\, P$, that is, P is independent of x. Occasionally we restrict attention to assertions P that are ***finitely supported***, meaning that P is independent of all but finitely many variables. We use $\models$ to indicate a ***valid*** correctness judgment, defined as follows:

$$\models c : P \rightsquigarrow Q \;\hat{=}\; \text{For all } s, t \text{ such that } [\![c]\!]\, s\, t, \text{ if } s \in P \text{ then } t \in Q. \tag{3.1}$$

The proof system we call ***HL+*** comprises the standard rules of HL plus a rule for elimination of ghost[5] variables [FGP16] and a rule for rewriting programs (Figure 1). Define $\mathsf{ghost}(x, c)$ to mean that variable x occurs in c only in assignments to x and havocs of x. Define $\mathsf{erase}(x, c)$ to be c with every assignment to x, and havoc of x, replaced by skip.

[3] It is not difficult to enforce total-if syntactically. One way is given in [NBN23, Remark A.1].

[4] This approach is popular [Nip02, O'H20, PdAC+22] because it factors out the issue of expressiveness [Coo78, AdBO09, Win93].

[5] Also known as auxiliary variables, see [AdBO09].

$$\textsc{Rewrite}\ \frac{c : P \rightsquigarrow Q \qquad c \simeq d}{d : P \rightsquigarrow Q} \qquad \textsc{Ghost}\ \frac{c : P \rightsquigarrow Q \qquad \mathsf{ghost}(x,c) \qquad \mathsf{indep}(x,P) \qquad \mathsf{indep}(x,Q)}{\mathsf{erase}(x,c) : P \rightsquigarrow Q}$$

$$\textsc{Do}\ \frac{c : e \wedge P \rightsquigarrow P \text{ for every } e \to c \text{ in } gcs}{\mathsf{do}\ gcs\ \mathsf{od} : P \rightsquigarrow P \wedge \neg \mathsf{enab}(gcs)} \qquad \textsc{Asgn}\ x := e : P^x_e \rightsquigarrow P \qquad \textsc{Hav}\ \mathsf{hav}\ x : (\dot\forall x.\, P) \rightsquigarrow P$$

$$\textsc{Skip}\ \mathsf{skip} : P \rightsquigarrow P \qquad \textsc{Seq}\ \frac{c : P \rightsquigarrow R \qquad d : R \rightsquigarrow Q}{c;d : P \rightsquigarrow Q} \qquad \textsc{If}\ \frac{c : e \wedge P \rightsquigarrow Q \text{ for every } e \to c \text{ in } gcs}{\mathsf{if}\ gcs\ \mathsf{fi} : P \rightsquigarrow Q}$$

$$\textsc{Conseq}\ \frac{P \Rightarrow R \qquad c : R \rightsquigarrow S \qquad S \Rightarrow Q}{c : P \rightsquigarrow Q} \qquad \textsc{False}\ c : \mathit{false} \rightsquigarrow P$$

Figure 1: Proof rules of HL+.

Because we are using shallow embedding for assertions, HL+ does not need to be accompanied by a formal system for proving entailments between assertions. But the logic also uses command equality $c \simeq d$, in rule Rewrite. For this we rely on KAT as formalized in section 5. For soundness of rule rewrite we just need that the relation $\simeq$ implies equal denotations.

Relational specs and proof rules. For relational pre- and post-conditions we use (binary) relations on stores, by shallow embedding just like for unary predicates. To express that a unary predicate holds in the left store of a pair, we write $\{\!|P|\!\}$ for the set of pairs (s,t) where $s \in P$. Similarly, $|\!|P|\!|$ is the set of (s,t) where $t \in P$. Combined with our coercion of boolean expressions e to predicates, $\{\!|e|\!\}$ says that e is true in the left store. Notation: $\{\!|P \mid Q|\!|$ abbreviates $\{\!|P|\!\} \wedge |\!|Q|\!|$. Recall from section 2 that we write $\{\!|e|\!\}$ (resp. $|\!|e|\!|$) for the value of expression e in the left (resp. right) state, in formulas like $\{\!|e|\!\} \le |\!|e'|\!|$ which describes the set of (s,t) such that $[\![\, e \,]\!](s) \le [\![\, e' \,]\!](t)$. As an abbreviation, define $\mathbb{A}e \;\hat{=}\; \{\!|e|\!\} = |\!|e|\!|$.

For relation $\mathcal{R}$ on stores we write $\mathcal{R}^{x|}_{e|}$ for substitution of e for x in the left store. Similarly $\mathcal{R}^{|x}_{|e}$ substitutes on the right and $\mathcal{R}^{x|x'}_{e|e'}$ does both. We write $\dot\exists x|x'.\,\mathcal{R}$ for quantification over x on the left side and x' on the right. Specifically, $\dot\exists x|x'.\,\mathcal{R} \;\hat{=}\; \{(s,s') \mid \exists v, v'.\, (s,s') \in \mathcal{R}^{x|x'}_{v|v'}\}$. Define $\mathsf{indep}(x|x', \mathcal{R})$ iff $\mathcal{R} = \dot\exists x|x'.\,\mathcal{R}$. We also need one-sided quantifier forms, and note that $(\dot\forall x|x'.\,\mathcal{R}) = \dot\forall x|.\,(\dot\forall|x'.\,\mathcal{R}) = \dot\forall|x'.\,(\dot\forall x|.\,\mathcal{R})$. For store relations $\mathcal{R}, \mathcal{S}$, a relational spec is written $\mathcal{R} \approx\!\!> \mathcal{S}$. We use $\models$ to indicate valid judgment, and define $\models c \mid c' : \mathcal{R} \approx\!\!> \mathcal{S} \;\hat{=}$

> For all s, s', t, t' such that $[\![\, c \,]\!]\, s\, t$ and $[\![\, c' \,]\!]\, s'\, t'$,
> if $(s,s') \in \mathcal{R}$ then $(t,t') \in \mathcal{S}$.

We write $\forall\exists$ specs as $\mathcal{R} \overset{\exists}{\approx\!\!>} \mathcal{S}$ and define the valid judgments by $\models c \mid c' : \mathcal{R} \overset{\exists}{\approx\!\!>} \mathcal{S} \;\hat{=}$

> For all s, s', t such that $(s,s') \in \mathcal{R}$ and $[\![\, c \,]\!]\, s\, t$,
> there is t' such that $[\![\, c' \,]\!]\, s'\, t'$ and $(t,t') \in \mathcal{S}$. (3.2)

Figure 2 gives the proof rules of the $\forall\forall$ logic RHL+. The $\forall\exists$ logic appears in section 9; until then we focus on $\forall\forall$. We are not aware of a prior RHL that has been formulated

$$\textsc{rSkip}\quad \mathsf{skip} \mid \mathsf{skip} : \mathcal{R} \approx\!\!> \mathcal{R}$$

$$\textsc{rAsgnSkip}\quad x := e \mid \mathsf{skip} : \mathcal{R}^{x|}_{e|} \approx\!\!> \mathcal{R}$$

$$\textsc{rSkipAsgn}\quad \mathsf{skip} \mid x := e : \mathcal{R}^{|x}_{|e} \approx\!\!> \mathcal{R}$$

$$\textsc{rHavSkip}\quad \mathsf{hav}\ x \mid \mathsf{skip} : (\dot\forall x|.\ \mathcal{P}) \approx\!\!> \mathcal{P}$$

$$\textsc{rSkipHav}\quad \mathsf{skip} \mid \mathsf{hav}\ x : (\dot\forall |x.\ \mathcal{P}) \approx\!\!> \mathcal{P}$$

$$\textsc{rSeq}\quad \frac{c \mid c' : \mathcal{R} \approx\!\!> \mathcal{S} \qquad d \mid d' : \mathcal{S} \approx\!\!> \mathcal{T}}{c; d \mid c'; d' : \mathcal{R} \approx\!\!> \mathcal{T}}$$

$$\textsc{rIf}\quad \frac{c \mid c' : \mathcal{R} \wedge \langle\!|e|\!\rangle \wedge \langle\!|e'|\!\rangle \approx\!\!> \mathcal{S} \quad \text{for all } e \to c \text{ in } gcs \text{ and } e' \to c' \text{ in } gcs'}{\mathsf{if}\ gcs\ \mathsf{fi} \mid \mathsf{if}\ gcs'\ \mathsf{fi} : \mathcal{R} \approx\!\!> \mathcal{S}}$$

$$\textsc{rDo}\quad \frac{\begin{array}{l} c \mid \mathsf{skip} : \mathcal{Q} \wedge \langle\!|e|\!\rangle \wedge \mathcal{L} \approx\!\!> \mathcal{Q} \quad \text{for all } e \to c \text{ in } gcs \\ \mathsf{skip} \mid c' : \mathcal{Q} \wedge |\!\langle e'|\!\rangle \wedge \mathcal{R} \approx\!\!> \mathcal{Q} \quad \text{for all } e' \to c' \text{ in } gcs' \\ c \mid c' : \mathcal{Q} \wedge \langle\!|e|\!\rangle \wedge |\!\langle e'|\!\rangle \wedge \neg\mathcal{L} \wedge \neg\mathcal{R} \approx\!\!> \mathcal{Q} \quad \text{for all } e \to c \text{ in } gcs \text{ and } e' \to c' \text{ in } gcs' \\ \mathcal{Q} \Rightarrow (\langle\!|\mathsf{enab}(gcs)|\!\rangle = |\!\langle\mathsf{enab}(gcs')|\!\rangle) \vee (\mathcal{L} \wedge \langle\!|\mathsf{enab}(gcs)|\!\rangle) \vee (\mathcal{R} \wedge |\!\langle\mathsf{enab}(gcs')|\!\rangle)) \end{array}}{\mathsf{do}\ gcs\ \mathsf{od} \mid \mathsf{do}\ gcs'\ \mathsf{od} : \mathcal{Q} \approx\!\!> \mathcal{Q} \wedge \neg\langle\!|\mathsf{enab}(gcs)|\!\rangle \wedge \neg|\!\langle\mathsf{enab}(gcs')|\!\rangle}$$

$$\textsc{rRewrite}\quad \frac{c \mid c' : \mathcal{R} \approx\!\!> \mathcal{S} \qquad c \simeq d \qquad c' \simeq d'}{d \mid d' : \mathcal{R} \approx\!\!> \mathcal{S}}$$

$$\textsc{rConseq}\quad \frac{\mathcal{P} \Rightarrow \mathcal{R} \qquad c \mid c' : \mathcal{R} \approx\!\!> \mathcal{S} \qquad \mathcal{S} \Rightarrow \mathcal{Q}}{c \mid c' : \mathcal{P} \approx\!\!> \mathcal{Q}}$$

$$\textsc{rDisj}\quad \frac{c \mid c' : \mathcal{Q} \approx\!\!> \mathcal{S} \qquad c \mid c' : \mathcal{R} \approx\!\!> \mathcal{S}}{c \mid c' : \mathcal{Q} \vee \mathcal{R} \approx\!\!> \mathcal{S}}$$

$$\textsc{rFalse}\quad c \mid c' : \mathit{false} \approx\!\!> \mathcal{R}$$

$$\textsc{rGhost}\quad \frac{c \mid c' : \mathcal{R} \approx\!\!> \mathcal{S} \qquad \mathsf{ghost}(x, c) \qquad \mathsf{ghost}(x', c') \qquad \mathsf{indep}(x|x', \mathcal{R}) \qquad \mathsf{indep}(x|x', \mathcal{S})}{\mathsf{erase}(x, c) \mid \mathsf{erase}(x', c') : \mathcal{R} \approx\!\!> \mathcal{S}}$$

Figure 2: The rules of RHL+.

for GCL, but the rules are straightforward adaptations of rules found in prior work. In particular, rIf generalizes the standard 4-way if-rule: instead of 2×2 cases there are $i \times j$ where i and j are the number of branches left and right.

Rules such as the same-branch-if rule popular in RHLs [Ben04, Yan07] are easily derived, see rAlgnIf and rAlgnDo in Figure 3. It is also easy to derive rules relating different control structures, in particular the "one sided" rules like rSeqSkip and rSeqIf in Figure 3. These are useful for reasoning by sequential alignment which is embodied in rules rLRseq and rRLseq.

Rule rDoSkip is derived as follows. Observe that $\mathsf{skip} \simeq \mathsf{do}\ \mathit{false} \to \mathsf{skip}\ \mathsf{od}$, so we get the conclusion of rDoSkip by rRewrite from $\mathsf{do}\ gcs\ \mathsf{od} \mid \mathsf{do}\ \mathit{false} \to \mathsf{skip}\ \mathsf{od} : \mathcal{Q} \approx\!\!> \mathcal{Q} \wedge \neg|\!\langle\mathsf{enab}(gcs)|\!\rangle$. This we prove using rDo with $\mathcal{L}, \mathcal{R} := \mathit{true}, \mathit{false}$. The left-only premises have the form $c|\mathsf{skip} : \mathcal{Q} \wedge \langle\!|e|\!\rangle \wedge \mathit{true} \approx\!\!> \mathcal{Q}$. They follow from the premises of rDoSkip using rConseq to add the conjunct *true*.[6] The right-only and joint premises are proved by rFalse and rConseq. The side condition of this instantiation of rDo simplifies to $\mathcal{Q} \Rightarrow (\langle\!|\mathsf{enab}(gcs)|\!\rangle = |\!\langle\mathit{false}|\!\rangle) \vee \langle\!|\mathsf{enab}(gcs)|\!\rangle$ and the consequent simplifies to true.[7] Proofs of the other rules in Figure 3 are similar.

[6] By shallow embedding, rConseq is never needed to manipulate equivalent relations, but we mention it for clarity.

[7] The side condition in rDo uses notation $\langle\!|\mathsf{enab}(gcs)|\!\rangle = |\!\langle\mathsf{enab}(gcs')|\!\rangle$ for a store relation that some readers may prefer to write as $\langle\!|\mathsf{enab}(gcs)|\!\rangle \Leftrightarrow |\!\langle\mathsf{enab}(gcs')|\!\rangle$. Similarly for the side conditions of rAlignIf and rAlignDo.

$$\textsc{rLRseq}\ \frac{c \mid \mathsf{skip} : \mathcal{P} \approx\!\!> \mathcal{Q} \qquad \mathsf{skip} \mid c' : \mathcal{Q} \approx\!\!> \mathcal{R}}{c \mid c' : \mathcal{P} \approx\!\!> \mathcal{R}} \qquad \textsc{rRLseq}\ \frac{\mathsf{skip} \mid c' : \mathcal{P} \approx\!\!> \mathcal{Q} \qquad c \mid \mathsf{skip} : \mathcal{Q} \approx\!\!> \mathcal{R}}{c \mid c' : \mathcal{P} \approx\!\!> \mathcal{R}}$$

$$\textsc{rAsgnAsgn}\ x := e \mid x' := e' : \mathcal{R}^{x|x'}_{e|e'} \approx\!\!> \mathcal{R} \qquad \textsc{rHavHav}\ \mathsf{hav}\ x \mid \mathsf{hav}\ x' : (\dot\forall x|x'.\, \mathcal{P}) \approx\!\!> \mathcal{P}$$

$$\textsc{rAlgnIf}\ \frac{c \mid c' : \mathcal{R} \wedge \langle\!| e |\!\rangle \approx\!\!> \mathcal{S} \qquad d \mid d' : \mathcal{R} \wedge \langle\!| \neg e |\!\rangle \approx\!\!> \mathcal{S} \qquad \mathcal{R} \Rightarrow \langle\!| e |\!\rangle = |\!\rangle e' \langle\!|}{\mathsf{if}\ e \to c \,[]\, \neg e \to d\ \mathsf{fi} \mid \mathsf{if}\ e' \to c' \,[]\, \neg e' \to d'\ \mathsf{fi} : \mathcal{R} \approx\!\!> \mathcal{S}}$$

$$\textsc{rAlgnDo}\ \frac{c \mid c' : \mathcal{R} \wedge \langle\!| e |\!\rangle \wedge |\!\rangle e' \langle\!| \approx\!\!> \mathcal{R} \qquad \mathcal{R} \Rightarrow \langle\!| e |\!\rangle = |\!\rangle e' \langle\!|}{\mathsf{do}\ e \to c\ \mathsf{od} \mid \mathsf{do}\ e' \to c'\ \mathsf{od} : \mathcal{R} \approx\!\!> \mathcal{R} \wedge \neg\langle\!| e |\!\rangle \wedge \neg |\!\rangle e' \langle\!|} \qquad \textsc{rSeqSkip}\ \frac{c \mid \mathsf{skip} : \mathcal{R} \approx\!\!> \mathcal{Q} \qquad d \mid \mathsf{skip} : \mathcal{Q} \approx\!\!> \mathcal{S}}{c; d \mid \mathsf{skip} : \mathcal{R} \approx\!\!> \mathcal{S}}$$

$$\textsc{rIfSkip}\ \frac{c \mid \mathsf{skip} : \mathcal{R} \wedge \langle\!| e |\!\rangle \approx\!\!> \mathcal{S} \quad \text{for all } e \to c \text{ in } gcs}{\mathsf{if}\ gcs\ \mathsf{fi} \mid \mathsf{skip} : \mathcal{R} \approx\!\!> \mathcal{S}} \qquad \textsc{rDoSkip}\ \frac{c \mid \mathsf{skip} : \mathcal{Q} \wedge \langle\!| e |\!\rangle \approx\!\!> \mathcal{Q} \quad \text{for all } e \to c \text{ in } gcs}{\mathsf{do}\ gcs\ \mathsf{od} \mid \mathsf{skip} : \mathcal{Q} \approx\!\!> \mathcal{Q} \wedge \neg\langle\!| \mathsf{enab}(gcs) |\!\rangle}$$

$$\textsc{rDisjN}\ \frac{c \mid d : \mathcal{R}_i \approx\!\!> \mathcal{S} \text{ for all } i \in X \qquad X \text{ finite}}{c \mid d : (\vee i \,:\, i \in X \,:\, \mathcal{R}_i) \approx\!\!> \mathcal{S}}$$

Figure 3: Some derived rules of RHL+.

Unlike some RHLs in the literature, RHL+ does not make use of unary correctness judgments. However, unary correctness can be encoded as relational correctness, in the sense that validity of $c : P \rightsquigarrow Q$ is equivalent to validity of the judgment $c \mid \mathsf{skip} : \langle\!| P |\!\rangle \approx\!\!> \langle\!| Q |\!\rangle$ and also $\mathsf{skip} \mid c : |\!\rangle P \langle\!| \approx\!\!> |\!\rangle Q \langle\!|$. We return to this in subsection 7.2.

Example 3.2. Consider these programs adapted from [NN21] (and in turn from [Fra83]).

$c4:$ $y := x; z := 24; w := 0;$
$\mathsf{do}\ y \neq 4 \to \mathsf{if}\ w \bmod 2 = 0 \to z := z * y; y := y - 1 \,[]\, w \bmod 2 \neq 0 \to \mathsf{skip}\ \mathsf{fi}; w := w + 1\ \mathsf{od}$

$c5:$ $y := x; z := 16; w := 0;$
$\mathsf{do}\ y \neq 4 \to \mathsf{if}\ w \bmod 3 = 0 \to z := z * 2; y := y - 1 \,[]\, w \bmod 3 \neq 0 \to \mathsf{skip}\ \mathsf{fi}; w := w + 1\ \mathsf{od}$

For input x with $x \geq 4$ these compute factorial and exponent respectively. Without reasoning about those functions, however, one can show that factorial majorizes exponent for arguments at least 4. Formally: $c4 | c5 : \mathbb{A}x \wedge \langle\!| x |\!\rangle \geq 4 \approx\!\!> \langle\!| z |\!\rangle > |\!\rangle z \langle\!|$. The reader may enjoy to use the proof rules to prove the spec, using rule rDo with loop alignment conditions $\mathcal{L} := \langle\!| w \bmod 2 \neq 0 |\!\rangle$ and $\mathcal{R} := |\!\rangle w' \bmod 3 \neq 0 \langle\!|$. As a loop invariant, try $\mathbb{A}y \wedge \langle\!| y |\!\rangle \geq 4 \wedge \langle\!| z |\!\rangle > |\!\rangle z \langle\!| \wedge |\!\rangle z \langle\!| > 0$.

Another approach is to rewrite the loops, unfolding them 2 (resp. 3) times. This is not possible, however, if we replace literals 2 and 3 by an input variable v with precondition $\langle\!| v \geq 1 |\!\rangle \wedge |\!\rangle v \geq 1 \langle\!|$. Of course the example is contrived, but such data-dependent alignment patterns arise in settings such as equivalence checking for optimizing compilers. □

4. Alignment automata, adequacy, and verification conditions

This section lays groundwork for alignment completeness, defining alignment automata for $\forall\forall$ properties and the verification conditions for program alignment automata. The definitions are adapted to $\forall\exists$ properties in subsection 9.1.

4.1. **Automata, alignment automata and adequacy.** We adapt a number of technical definitions from [NN21], where automata are formulated in a way that can represent program semantics using, in essence, a finite control flow graph.

Definition 4.1. An ***automaton*** is a tuple $(Ctrl, Sto, init, fin, \mapsto)$ where Sto is a set (called the data stores), $Ctrl$ is a finite set (the control points) that contains distinct elements $init$ and fin, and $\mapsto \subseteq (Ctrl \times Sto) \times (Ctrl \times Sto)$ is the transition relation.[8] We require $(n, s) \mapsto (m, t)$ to imply $n \neq fin$ and $n \neq m$ and call these the ***finality*** and ***non-stuttering*** conditions respectively. Absence of stuttering loses no generality and facilitates definitions involving product automata. A ***state*** of the automaton is an element of $Ctrl \times Sto$.

We use the term "alignment product" informally, in reference to various constructions in the literature. We define a particular construction that we call alignment automaton.

Definition 4.2. Let $A = (Ctrl, Sto, init, fin, \mapsto)$ and $A' = (Ctrl', Sto', init', fin', \mapsto')$ be automata. Let L, R, and J be subsets of $(Ctrl \times Ctrl') \times (Sto \times Sto')$ that are ***live for*** A, A', meaning that:
$$\begin{array}{l} \forall((n,n'),(s,s')) \in L.\ (n,s) \in \mathsf{dom}(\mapsto) \\ \forall((n,n'),(s,s')) \in R.\ (n',s') \in \mathsf{dom}(\mapsto') \\ \forall((n,n'),(s,s')) \in J.\ (n,s) \in \mathsf{dom}(\mapsto) \wedge (n',s') \in \mathsf{dom}(\mapsto') \end{array} .$$
The ***alignment automaton*** $\prod(A, A', L, R, J)$ is the automaton

$$((Ctrl \times Ctrl'), (Sto \times Sto'), (init, init'), (fin, fin'), \Mapsto)$$

where $\Mapsto$ is defined by: $((n,n'),(s,s')) \Mapsto ((m,m'),(t,t'))$ iff one of these conditions holds:

LO : $((n,n'),(s,s')) \in L$ and $(n,s) \mapsto (m,t)$ and $(n',s') = (m',t')$
RO: $((n,n'),(s,s')) \in R$ and $(n,s) = (m,t)$ and $(n',s') \mapsto' (m',t')$
JO : $((n,n'),(s,s')) \in J$ and $(n,s) \mapsto (m,t)$ and $(n',s') \mapsto' (m',t')$

Notice that the states of $\prod(A, A', L, R, J)$ are $((Ctrl \times Ctrl') \times (Sto \times Sto'))$. So the ***alignment conditions*** L, R, and J are sets of alignment automaton states. We write $[n|n']$ for the set of states where control is at (n, n'), i.e., $[n|n'] \ \hat{=}\ \{((i,i'),(s,s')) \mid n = i \wedge n' = i'\}$. For example $[fin|fin']$ is the set of terminated states. Let $[n|*] \ \hat{=}\ \{((i,i'),(s,s')) \mid n = i\}$.

Roughly speaking, taking L and R to be false and J true, we obtain an automaton that runs A and A' in lockstep. Taking L, R, J all true, we obtain a nondeterministic alignment automaton that represents very many alignments. Taking L, R, J all false, we obtain an alignment automaton that represents no alignments whatsoever. Taking J to be false, L to be $[*|init']$, and R to be $[fin|*]$, we obtain an automaton that runs only A, unless it terminates, in which case it proceeds to run A' —the sequential alignment.[9]

Definition 4.3. Consider an alignment automaton $\prod(A, A', L, R, J)$ and relation $\mathcal{P} \subseteq Sto \times Sto'$. The alignment automaton is ***$\mathcal{P}$-adequate*** provided for all $(s, s') \in \mathcal{P}$ and t, t' with $(init, s) \mapsto^* (fin, t)$ and $(init', s') \mapsto'^* (fin', t')$, we have $((init, init'), (s, s')) \Mapsto^* ((fin, fin'), (t, t'))$.

[8]The symbol $\mapsto$ here has nothing to do with the store update notation, e.g., $s[x \mapsto v]$, where the square brackets should prevent any confusion.

[9]To be precise, one must ensure that L, R, J are live (per Definition 4.2). Later we construct automata from programs, and for such automata every state has a successor except when control is final (Lemma 4.12). Hence, for programs, an arbitrary triple (L, R, J) can be made live very simply, as $(L \setminus [fin|*], R \setminus [*|fin'], J \setminus [fin|fin'])$ where $\setminus$ is set subtraction.

An alignment automaton may be adequate for reasons that are specific to the underlying automata, for example it may not cover all traces but still cover all outcomes. We focus on alignment automata that are adequate in the sense that they cover all traces, which can be ensured as follows. Given a relation $\mathcal{P} \subseteq Sto \times Sto'$, an alignment automaton is ***manifestly $\mathcal{P}$-adequate*** provided that $L \vee R \vee J \vee [\mathit{fin}|\mathit{fin}']$ is ***$\mathcal{P}$-invariant***. This means $L \vee R \vee J \vee [\mathit{fin}|\mathit{fin}']$ holds at every state reachable from some $((\mathit{init}, \mathit{init}'), (s, s'))$ such that $(s, s') \in \mathcal{P}$.

Lemma 4.4. If alignment automaton $\prod(A, A', L, R, J)$ is manifestly $\mathcal{P}$-adequate then it is $\mathcal{P}$-adequate (in the sense of Definition 4.3).

4.2. **Correctness of automata, and the inductive assertion method.** Generalizing slightly from section 3, we consider specs $P \rightsquigarrow Q$ where P and Q are sets of automaton stores, not necessarily variable stores. Satisfaction of a spec by an automaton is written $A \models P \rightsquigarrow Q$ and defined to mean: For all s, t such that $(\mathit{init}, s) \mapsto^* (\mathit{fin}, t)$, if $s \in P$ then $t \in Q$.

Let A and A' be automata with store sets Sto and Sto' respectively. Generalizing slightly from section 3, we consider relational specs $\mathcal{R} \approx\!\!> \mathcal{S}$ where $\mathcal{R}$ and $\mathcal{S}$ are relations from Sto to Sto'. Satisfaction of the spec by the pair A, A' is written $A, A' \models \mathcal{R} \approx\!\!> \mathcal{S}$ and defined to mean, for any s, s', t, t':

$$\text{If } (\mathit{init}, s) \mapsto^* (\mathit{fin}, t) \text{ and } (\mathit{init}', s') \mapsto'^* (\mathit{fin}', t') \text{ and } (s, s') \in \mathcal{R} \text{ then } (t, t') \in \mathcal{S}.$$

A store relation $\mathcal{Q}$ for A, A' can be seen as a predicate on the stores of an alignment automaton $\prod(A, A', L, R, J)$ because the latter are pairs of stores. Hence, for relational spec $\mathcal{Q} \approx\!\!> \mathcal{S}$, the unary spec $\mathcal{Q} \rightsquigarrow \mathcal{S}$ makes sense for $\prod(A, A', L, R, J)$.

Lemma 4.5 (adequacy semantically sound and complete)**.** Suppose that $\prod(A, A', L, R, J)$ is $\mathcal{Q}$-adequate. Then $\prod(A, A', L, R, J) \models \mathcal{Q} \rightsquigarrow \mathcal{S}$ if and only if $A, A' \models \mathcal{Q} \approx\!\!> \mathcal{S}$.

Given automaton A and spec $P \rightsquigarrow Q$, an ***annotation*** is a function an from control points to store predicates such that $P \Rightarrow an(\mathit{init})$ and $an(\mathit{fin}) \Rightarrow Q$. The requirement $\mathit{init} \neq \mathit{fin}$ in Definition 4.1 ensures that annotations exist for any spec.[10] We lift an to a function $\hat{an}$ that yields states: $\hat{an}(n) = \{(m, s) \mid s \in an(n), m \in Ctrl\}$. Put differently:

$$(m, s) \in \hat{an}(n) \quad \text{iff} \quad s \in an(n) \qquad (\text{for all } m, n, s) \tag{4.1}$$

For each pair (n, m) of control points there is a ***verification condition*** (***VC***):

$$\mathsf{post}(\overset{n,m}{\longmapsto})(\hat{an}(n)) \subseteq \hat{an}(m) \tag{4.2}$$

Here post gives the direct image (i.e., strongest postcondition) of a relation,[11] and $\overset{n,m}{\longmapsto}$ is the ***fixed-control transition relation*** restricted to starting control point n and ending m, i.e.,

$$(i, s) \overset{n,m}{\longmapsto} (j, t) \;\text{ iff }\; i = n,\ j = m, \text{ and } (n, s) \mapsto (m, t) \tag{4.3}$$

[10] In Floyd's formulation, an annotation only needs to be defined on a subset of control points that cut every loop in the control flow graph. Such an annotation can always be extended to one for all control points.

[11] Defined for any set X and relation R by $t \in \mathsf{post}(R)(X)$ iff $\exists s.\ (s, t) \in R \wedge s \in X$.

$$\langle \mathsf{hav}^n\ x,\ s\rangle \twoheadrightarrow \langle \mathsf{skip}^{-n},\ s[x \mapsto v]\rangle \qquad \langle \mathsf{skip}^n; c,\ s\rangle \twoheadrightarrow \langle c,\ s\rangle \qquad \frac{e \to c \text{ is in } gcs \qquad [\![e]\!](s) = \mathit{true}}{\langle \mathsf{if}^n\ gcs\ \mathsf{fi},\ s\rangle \twoheadrightarrow \langle c,\ s\rangle}$$

$$\frac{\mathsf{enab}(gcs)(s) = \mathit{false}}{\langle \mathsf{do}^n\ gcs\ \mathsf{od},\ s\rangle \twoheadrightarrow \langle \mathsf{skip}^{-n},\ s\rangle} \qquad \langle x :=^n e,\ s\rangle \twoheadrightarrow \langle \mathsf{skip}^{-n},\ s[x \mapsto [\![e]\!](s)]\rangle$$

$$\frac{e \to c \text{ is in } gcs \qquad [\![e]\!](s) = \mathit{true}}{\langle \mathsf{do}^n\ gcs\ \mathsf{od},\ s\rangle \twoheadrightarrow \langle c; \mathsf{do}^n\ gcs\ \mathsf{od},\ s\rangle} \qquad \frac{\langle c,\ s\rangle \twoheadrightarrow \langle d,\ t\rangle}{\langle c; b,\ s\rangle \twoheadrightarrow \langle d; b,\ t\rangle}$$

Figure 4: Transition semantics (with n and v ranging over $\mathbb{Z}$).

It is well known that (4.2) is equivalent to $\hat{an}(n) \subseteq \mathsf{wp}(\overset{n,m}{\longmapsto})(\hat{an}(m))$ using the universal preimage operator wp.[12]

The VC (4.2) says that for every transition from control point n and store $s \in an(n)$, if the step goes to control point m with store t, then t is in $an(m)$. Annotation an is ***valid*** if the VC is true for every pair (n, m) of control points. In most automata, including those we derive from programs, some pairs (n, m) have no transitions, in other words $\overset{n,m}{\longmapsto}$ is empty. In that case the VC (4.2) is true regardless of $an(n)$ and $an(m)$.

In case the stores of A are variable stores, a set S of A-states is ***finitely supported*** provided that for each control point n the set of stores $\{t \mid (n, t) \in S\}$ is finitely supported. Similarly for states of an alignment product. We say A is finitely supported if its transition relation acts on finitely many variables.[13] An annotation an is finitely supported provided $an(n)$ is, for all control points n.

Lemma 4.6 (semantic soundness and completeness of IAM [Flo67])**.** There is a valid annotation of A for $P \rightsquigarrow Q$ iff $A \models P \rightsquigarrow Q$. Moreover, in case A acts on variable stores and P, Q are finitely supported, $A \models P \rightsquigarrow Q$ implies there is a finitely supported valid annotation.

Corollary 4.7 (soundness of alignment automata)**.** Suppose that $\prod(A, A', L, R, J)$ is $\mathcal{Q}$-adequate and an is an annotation of $\prod(A, A', L, R, J)$ for $\mathcal{Q} \rightsquigarrow \mathcal{S}$. If an is valid then $A, A' \models \mathcal{Q} \approx\!\gg \mathcal{S}$.

Together, Corollary 4.7 and Lemma 4.4 provide a method to verify $A, A' \models \mathcal{Q} \approx\!\gg \mathcal{S}$: Find alignment conditions L, R, J and annotation an of $\prod(A, A', L, R, J)$ such that the annotation is valid and $\check{an}(i, j) \Rightarrow L \vee R \vee J \vee [\mathit{fin}|\mathit{fin}']$ for every i, j. This uses the abbreviation $\check{an}(n, n') \mathrel{\hat{=}} \hat{an}(n, n') \wedge [n|n']$. This implication ensures manifest adequacy.

Corollary 4.8 (semantic completeness of alignment automata)**.** Suppose $A, A' \models \mathcal{S} \approx\!\gg \mathcal{T}$. Then there are L, R, J and a valid annotation an of $\prod(A, A', L, R, J)$ for $\mathcal{S} \rightsquigarrow \mathcal{T}$ such that $\check{an}(i, j) \Rightarrow L \vee R \vee J \vee [\mathit{fin}|\mathit{fin}']$ for every i, j. Moreover, if A and A' act on variable stores, and $A, A', \mathcal{S}, \mathcal{T}$ are finitely supported, then so are L, R, J, and an.

4.3. **Automata from programs and their VCs.** Labels on commands serve as basis for defining the automaton for a program, to which end we make the following definitions

[12] Defined for any set X and relation R by $s \in \mathsf{wp}(R)(X)$ iff $\forall t.\ (s, t) \in R \Rightarrow t \in X$.

[13] This can be formalized as follows: $\mathsf{dom}(\mapsto)$ is finitely supported and for any x outside the support of $\mathsf{dom}(\mapsto)$ and any states (n, s) and (m, t), if $(n, s) \mapsto (m, t)$ then $(n, s[x \mapsto i]) \mapsto (m, t[x \mapsto i])$ for all $i \in \mathbb{N}$.

$$
\begin{array}{ll}
\mathsf{fsuc}(n, c; d, f) & \hat{=}\ \mathsf{fsuc}(n, c, \mathsf{lab}(d))\ \text{, if } n \in \mathsf{labs}(c) \\
& \hat{=}\ \mathsf{fsuc}(n, d, f)\ \text{, otherwise} \\
\mathsf{fsuc}(n, \mathsf{if}^n\ gcs\ \mathsf{fi}, f) & \hat{=}\ f \\
\mathsf{fsuc}(m, \mathsf{if}^n\ gcs\ \mathsf{fi}, f) & \hat{=}\ \mathsf{fsuc}(m, c, f)\ \text{, if } e \to c \in gcs \text{ and } m \in \mathsf{labs}(c) \\
\mathsf{fsuc}(n, \mathsf{do}^n\ gcs\ \mathsf{od}, f) & \hat{=}\ f \\
\mathsf{fsuc}(m, \mathsf{do}^n\ gcs\ \mathsf{od}, f) & \hat{=}\ \mathsf{fsuc}(m, c, n)\ \text{, if } e \to c \in gcs \text{ and } m \in \mathsf{labs}(c) \\
\mathsf{fsuc}(n, \mathsf{skip}^n, f) \ \hat{=}\ \mathsf{fsuc}(n, x :=^n e, f) \ \hat{=}\ \mathsf{fsuc}(n, \mathsf{hav}^n\ x, f) \ \hat{=}\ f &
\end{array}
$$

Figure 5: Following successor $\mathsf{fsuc}(n, c, f)$, assuming $\mathsf{ok}(c)$, $n \in \mathsf{labs}(c)$, and $f \notin \mathsf{labs}(c)$.

(adapted from [NN21]). Write $\mathsf{ok}(c)$ to say no label in c occurs more than once and all labels are positive. Write $\mathsf{lab}(c)$ for the label of c and $\mathsf{labs}(c)$ for the set of labels that occur in c. (The only non-obvious case is $\mathsf{lab}(c; d) \ \hat{=}\ \mathsf{lab}(c)$, see Figure 22.) For small-step semantics, we write $\langle c,\ s\rangle \rightarrowtriangle \langle d,\ t\rangle$ if command c with store s transitions to continuation command d and store t (see Figure 4).

Negative labels are used in the small-step semantics in a way that facilitates defining the automaton of a program. In a configuration reached from an ok command, the only negative labels are those introduced by the transition for assignment and the transition for termination of a loop. For every c, s, either $\langle c,\ s\rangle$ has a successor via $\rightarrowtriangle$ or c is skip^n for some $n \in \mathbb{Z}$. (This relies on $\mathsf{total\text{-}if}$ of Def. 3.1.) When a negative label is introduced, the configuration is either terminated or has the form $\langle \mathsf{skip}^{-n}; c,\ s\rangle$ in which case the next transition is $\langle \mathsf{skip}^{-n}; c,\ s\rangle \rightarrowtriangle \langle c,\ s\rangle$.

Lemma 4.9. For any c, s, t, we have $[\![\, c \,]\!]\, s\, t$ iff $\langle c,\ s\rangle \rightarrowtriangle^* \langle \mathsf{skip}^n,\ t\rangle$ for some n.

Write $\mathsf{sub}(n, c)$ for the sub-command of c with label n, if n is in $\mathsf{labs}(c)$. Let c and *fin* be such that $\mathsf{ok}(c)$ and *fin* $\notin \mathsf{labs}(c)$. We write $\mathsf{fsuc}(n, c, \mathit{fin})$ for the ***following successor*** of n in the control flow graph of c, i.e., the control successor of the subprogram at n, in the sense made precise in Figure 5. Note that *fin* serves as a final or exit label. The key case in the definition is for loops: the following successor is the control point after termination of the loop. For the running example, we have $\mathsf{fsuc}(2, c0, 6) = 6$ and $\mathsf{fsuc}(4, c0, 6) = 2$. Define $\mathsf{okf}(c, f)$ ("ok, fresh") to abbreviate the conjunction of $\mathsf{ok}(c)$ and $f \notin \mathsf{labs}(c)$.

If $\mathsf{okf}(c, f)$ then we define the ***automaton of*** c ***for*** f, written $\mathsf{aut}(c, f)$, with control set $\mathsf{labs}(c) \cup \{f\}$ and transitions in accord with the small-step semantics.

Definition 4.10. Suppose $\mathsf{okf}(c, f)$. The ***automaton of*** c ***for*** f, written $\mathsf{aut}(c, f)$, is $(\mathsf{labs}(c) \cup \{f\}, (\mathsf{Var} \to \mathbb{Z}), \mathsf{lab}(c), f, \mapsto)$ where $(n, s) \mapsto (m, t)$ iff either

- $\exists d.\ \langle \mathsf{sub}(n, c),\ s\rangle \rightarrowtriangle \langle d,\ t\rangle \wedge \mathsf{lab}(d) > 0 \wedge m = \mathsf{lab}(d)$, or
- $\exists d.\ \langle \mathsf{sub}(n, c),\ s\rangle \rightarrowtriangle \langle d,\ t\rangle \wedge \mathsf{lab}(d) < 0 \wedge m = \mathsf{fsuc}(n, c, f)$, or
- $\mathsf{sub}(n, c) = \mathsf{skip}^n \wedge m = \mathsf{fsuc}(n, c, f) \wedge t = s$

The first two cases use the semantics of Figure 4 for a sub-command on its own. The second case uses fsuc for a sub-command that takes a terminating step (either assignment, havoc, or loop). The third case handles skip, which on its own would be stuck but which should take a step when it occurs as part of a sequence.[14]

[14]For example, with $\rightarrowtriangle$ we have $\langle \mathsf{skip}^n; \mathsf{skip}^m,\ s\rangle \rightarrowtriangle \langle \mathsf{skip}^m,\ s\rangle$ but skip^n by itself has no transitions and the first two cases do not apply. Owing to the third case we have $(n, s) \mapsto (m, s) \mapsto (f, s)$. This also shows that the automaton steps are not in exact correspondence with those via $\rightarrowtriangle$ but this does not matter.

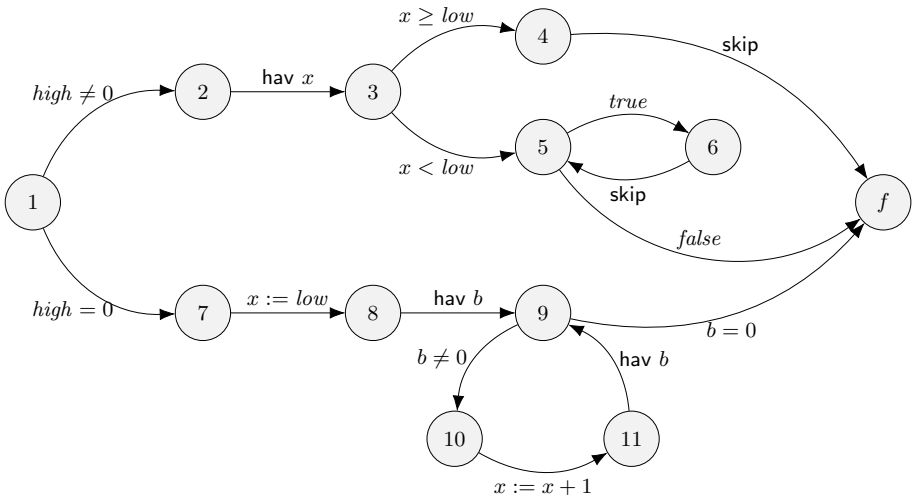

Figure 6: Automata for program $c3$ (in section 2): $\mathsf{aut}(c3, f)$ where $f = 12$.

if $\mathsf{sub}(n, c)$ is...	and m is...	then the VC for (n, m) is equivalent to...
skip^n	$\mathsf{fsuc}(n, c, f)$	$an(n) \Rightarrow an(m)$
$x :=^n e$	$\mathsf{fsuc}(n, c, f)$	$an(n) \Rightarrow an(m)^x_e$
$\mathsf{hav}^n\ x$	$\mathsf{fsuc}(n, c, f)$	$an(n) \Rightarrow \dot{\forall} x.\, an(m)$
$\mathsf{if}^n\ gcs\ \mathsf{fi}$	$\mathsf{lab}(d)$ where $e \to d$ is in gcs	$an(n) \wedge e \Rightarrow an(m)$
$\mathsf{do}^n\ gcs\ \mathsf{od}$	$\mathsf{lab}(d)$ where $e \to d$ is in gcs	$an(n) \wedge e \Rightarrow an(m)$
$\mathsf{do}^n\ gcs\ \mathsf{od}$	$\mathsf{fsuc}(n, c, f)$	$an(n) \wedge \neg\mathsf{enab}(gcs) \Rightarrow an(m)$

In all other cases, there are no transitions from n to m so the VC is $true$ by definition.

Figure 7: VCs for the automaton $aut(c, f)$ of ok program c and annotation an.

Example 4.11. The automaton for the command $c3$ on page 6 in section 2, using 12 as final label, is depicted in Figure 6. One can see an alignment automaton for $c3$ in Figure 17, ignoring the dashed boxes in the figure. □

Lemma 4.12. The only stuck states of $\mathsf{aut}(c, f)$ are terminated ones, i.e., where the control is f.

Lemma 4.13 (automaton consistency)**.** Suppose $\mathsf{okf}(c, f)$ and let $n = \mathsf{lab}(c)$. For any s, t we have $[\![c]\!]\, s\, t$ iff $(n, s) \mapsto^* (f, t)$ in $\mathsf{aut}(c, f)$. Hence $\models c : P \rightsquigarrow Q$ iff $\mathsf{aut}(c, f) \models P \rightsquigarrow Q$ for any P, Q. Also $\models c \,|\, c' : \mathcal{Q} \approx\!\!> \mathcal{S}$ iff $\mathsf{aut}(c, f), \mathsf{aut}(c', f') \models \mathcal{Q} \approx\!\!> \mathcal{S}$ for any $\mathcal{Q}, \mathcal{S}$ and $\mathsf{okf}(c', f')$. The same holds for the $\overset{\exists}{\approx\!\!>}$ judgment.

By inspection of the transition semantics in Figure 4, there are six kinds of transitions for $\rightarrowtriangle$ and also for the automaton relation $\mapsto$ derived from it. (There is a seventh rule for $\rightarrowtriangle$ that says a transition can occur for the first command in a sequence, but that is used together with one of the other six, and it is not relevant to Definition 4.10.) Thus there are six kinds of verification conditions, which can be derived from the semantic definitions.

Lemma 4.14 (VCs for programs)**.** Consider c and f such that $\mathsf{ok}(c, f)$, and let an be an annotation of $\mathsf{aut}(c, f)$. For each pair n, m of labels, the VC of equation (4.2) can be expressed as in Figure 7.

Proof. We give two cases. The other cases are similar.

Case skip^n with $m = \mathsf{fsuc}(n, c, f)$. To show: the VC is equivalent to $an(n) \Rightarrow an(m)$. The VC of (4.2) is the first line of the following calculation.

$$
\begin{array}{ll}
 & \mathsf{post}(\overset{n,m}{\longmapsto})(\hat{an}(n)) \subseteq \hat{an}(m) \\
\Leftrightarrow & \quad \text{def of } \mathsf{post} \\
 & \forall i, s, j, t.\ (i, s) \in \hat{an}(n) \wedge (i, s) \overset{n,m}{\longmapsto} (j, t) \ \Rightarrow\ (j, t) \in \hat{an}(m) \\
\Leftrightarrow & \quad \text{def } \overset{n,m}{\longmapsto} \text{ and one-point rule of predicate calculus} \\
 & \forall s, t.\ (n, s) \in \hat{an}(n) \wedge (n, s) \mapsto (m, t) \Rightarrow (m, t) \in \hat{an}(m) \\
\Leftrightarrow & \quad \text{def (4.1)} \\
 & \forall s, t.\ s \in an(n) \wedge (n, s) \mapsto (m, t) \Rightarrow t \in an(m) \\
\Leftrightarrow & \quad \text{def } \mapsto \text{ for the case of skip with successor } m \text{ (Definition 4.10)} \\
 & \forall s, t.\ s \in an(n) \wedge s = t \Rightarrow t \in an(m) \\
\Leftrightarrow & \quad \text{one-point rule} \\
 & \forall s.\ s \in an(n) \Rightarrow s \in an(m)
\end{array}
$$

The last line is equivalent to $an(n) \subseteq an(m)$ which we write as $an(n) \Rightarrow an(m)$.

Case $\mathsf{hav}^n\ x$ with $m = \mathsf{fsuc}(n, c, f)$. To show: the VC is equivalent to $an(n) \Rightarrow \dot{\forall} x. an(m)$. As in the previous case, the VC is equivalent to the first line of this calculation:

$$
\begin{array}{ll}
 & \forall s, t.\ s \in an(n) \wedge (n, s) \mapsto (m, t) \Rightarrow t \in an(m) \\
\Leftrightarrow & \quad \text{using def } \mapsto \text{ for case } \mathsf{hav}^n\ x \text{ with successor } m \\
 & \forall s, v.\ s \in an(n) \Rightarrow s[x \mapsto v] \in an(m) \\
\Leftrightarrow & \quad \text{using def of substitution} \\
 & \forall s, v.\ s \in an(n) \Rightarrow s \in an(m)^x_v \\
\Leftrightarrow & \quad \text{by predicate calculus} \\
 & \forall s.\ s \in an(n) \Rightarrow \forall v.\ s \in an(m)^x_v \\
\Leftrightarrow & \quad \text{by def} \\
 & \forall s.\ s \in an(n) \Rightarrow s \in \dot{\forall} x.\ an(m)
\end{array}
$$

which we write as $an(n) \Rightarrow \dot{\forall} x.\ an(m)$. □

4.4. **Relational VCs.** Consider an alignment automaton $\prod(A, A', L, R, J)$ where the underlying automata A and A' are obtained from programs by Definition 4.10. An annotation of $\prod(A, A', L, R, J)$ thus maps pairs of control points of A, A' to relations on variable stores. Verification conditions are associated with tuples $((n, n'), (m, m'))$ that represent edges in the control flow graph of $\prod(A, A', L, R, J)$, i.e., VCs are given by (4.2) instantiated with $n := (n, n')$ and $m := (m, m')$.

For a given pair $((n, n'), (m, m'))$ of alignment automaton control points, the transitions go only via LO, or only via RO, or only via JO in Definition 4.2. If $n = m$, i.e., control does not change on the left, the transitions must be via RO because the non-stuttering condition for automata (Definition 4.1) ensures there is no unary transition where control does not change. Similarly, if $n' = m'$ the transitions are via LO. If $n \neq m$ and $n' \neq m'$ the transitions only go via JO.

Figure 8 gives the six VCs for transitions that go by the LO condition. The VCs for RO are symmetric (and omitted). There are 36 combinations for JO transitions of an alignment automaton; some of their VCs are in Figure 9. By contrast with Figure 7 and Lemma 4.14 we do not eliminate lift (hat) notation in Figure 8 and Figure 9, because the alignment conditions L, R, J are sets of states, not sets of stores. We return to this later, using the *pc*

if $\mathsf{sub}(n,c)$ is...	and m is...	then the VC for $((n,n'),(m,n'))$ is equivalent to ...
skip^n	$\mathsf{fsuc}(n,c,f)$	$L \wedge \breve{an}(n,n') \Rightarrow \hat{an}(m,n')$
$x :=^n e$	$\mathsf{fsuc}(n,c,f)$	$L \wedge \breve{an}(n,n') \Rightarrow \hat{an}(m,n')^{x\vert}_{e\vert}$
$\mathsf{hav}^n\ x$	$\mathsf{fsuc}(n,c,f)$	$L \wedge \breve{an}(n,n') \Rightarrow \dot{\forall} x\vert.\ \hat{an}(m,n')$
$\mathsf{if}^n\ gcs\ \mathsf{fi}$	$\mathsf{lab}(d)$ where $e \to d$ is in gcs	$L \wedge \breve{an}(n,n') \wedge \{\!\vert e \vert\!\} \Rightarrow \hat{an}(m,n')$
$\mathsf{do}^n\ gcs\ \mathsf{od}$	$\mathsf{lab}(d)$ where $e \to d$ is in gcs	$L \wedge \breve{an}(n,n') \wedge \{\!\vert e \vert\!\} \Rightarrow \hat{an}(m,n')$
$\mathsf{do}^n\ gcs\ \mathsf{od}$	$\mathsf{fsuc}(n,c,f)$	$L \wedge \breve{an}(n,n') \wedge \neg\{\!\vert \mathsf{enab}(gcs) \vert\!\} \Rightarrow \hat{an}(m,n')$

In all other cases, there are no transitions from (n,n') to (m,n') so the VC is *true* by definition.

Figure 8: The left-only VCs for annotation an of $\prod(\mathsf{aut}(c,f),\mathsf{aut}(c',f'),L,R,J)$.

if $\mathsf{sub}(n,c)$ are... $\mathsf{sub}(n',c')$	and m are... m'	then the VC for $((n,n'),(m,m'))$ is equiv. to...
skip^n	$\mathsf{fsuc}(n,c,f)$	$J \wedge \breve{an}(n,n') \Rightarrow \hat{an}(m,m')$
$\mathsf{skip}^{n'}$	$\mathsf{fsuc}(n',c',f')$	
$x :=^n e$	$\mathsf{fsuc}(n,c,f)$	$J \wedge \breve{an}(n,n') \Rightarrow \hat{an}(m,m')^{x\vert x'}_{e\vert e'}$
$x' :=^{n'} e'$	$\mathsf{fsuc}(n',c',f')$	
$\mathsf{hav}^n\ x$	$\mathsf{fsuc}(n,c,f)$	$J \wedge \breve{an}(n,n') \Rightarrow \forall x\vert x'.\ \hat{an}(m,m')$
$\mathsf{hav}^{n'}\ x'$	$\mathsf{fsuc}(n',c',f')$	
$\mathsf{if}^n\ gcs\ \mathsf{fi}$	$\mathsf{lab}(d)$ where $e \to d$ is in gcs	$J \wedge \breve{an}(n,n') \wedge \{\!\vert e \vert\!\} \wedge [\!\vert e' \vert\!] \Rightarrow \hat{an}(m,m')$
$\mathsf{if}^{n'}\ gcs'\ \mathsf{fi}$	$\mathsf{lab}(d')$ where $e' \to d'$ is in gcs'	
$\mathsf{do}^n\ gcs\ \mathsf{od}$	$\mathsf{lab}(d)$ where $e \to d$ is in gcs	$J \wedge \breve{an}(n,n') \wedge \{\!\vert e \vert\!\} \wedge [\!\vert e' \vert\!] \Rightarrow \hat{an}(m,m')$
$\mathsf{do}^{n'}\ gcs'\ \mathsf{od}$	$\mathsf{lab}(d')$ where $e' \to d'$ is in gcs'	
$\mathsf{do}^n\ gcs\ \mathsf{od}$	$\mathsf{fsuc}(n,c,f)$	$J \wedge \breve{an}(n,n') \wedge \neg\{\!\vert \mathsf{enab}(gcs) \vert\!\} \wedge \neg[\!\vert \mathsf{enab}(gcs') \vert\!]$
$\mathsf{do}^{n'}\ gcs'\ \mathsf{od}$	$\mathsf{fsuc}(n',c',f')$	$\Rightarrow \hat{an}(m,m')$
$\mathsf{do}^n\ gcs\ \mathsf{od}$	$\mathsf{lab}(d)$ where $e \to d$ is in gcs	$J \wedge \breve{an}(n,n') \wedge \{\!\vert e \vert\!\} \wedge \neg[\!\vert \mathsf{enab}(gcs') \vert\!] \Rightarrow \hat{an}(m,m')$
$\mathsf{do}^{n'}\ gcs'\ \mathsf{od}$	$\mathsf{fsuc}(n',c',f')$	
$\mathsf{do}^n\ gcs\ \mathsf{od}$	$\mathsf{fsuc}(n,c,f)$	$J \wedge \breve{an}(n,n') \wedge \neg\{\!\vert \mathsf{enab}(gcs) \vert\!\} \wedge [\!\vert e' \vert\!] \Rightarrow \hat{an}(m,m')$
$\mathsf{do}^{n'}\ gcs'\ \mathsf{od}$	$\mathsf{lab}(d)$ where $e \to d$ is in gcs	
$x :=^n e$	$\mathsf{fsuc}(n,c,f)$	$J \wedge \breve{an}(n,n') \Rightarrow \hat{an}(m,m')^{x\vert}_{e\vert}$
$\mathsf{skip}^{n'}$	$\mathsf{fsuc}(n',c',f')$	
$x :=^n e$	$\mathsf{fsuc}(n,c,f)$	$J \wedge \breve{an}(n,n') \wedge [\!\vert e' \vert\!] \Rightarrow \hat{an}(m,m')^{x\vert}_{e\vert}$
$\mathsf{if}^{n'}\ gcs'\ \mathsf{fi}$	$\mathsf{lab}(d')$ where $e' \to d'$ is in gcs'	
$x :=^n e$	$\mathsf{fsuc}(n,c,f)$	$J \wedge \breve{an}(n,n') \wedge [\!\vert e' \vert\!] \Rightarrow \hat{an}(n,m')^{x\vert}_{e\vert}$
$\mathsf{do}^{n'}\ gcs'\ \mathsf{od}$	$\mathsf{lab}(d')$ where $e' \to d'$ is in gcs'	
$x :=^n e$	$\mathsf{fsuc}(n,c,f)$	$J \wedge \breve{an}(n,n') \wedge \neg[\!\vert e' \vert\!] \Rightarrow \hat{an}(n,m')^{x\vert}_{e\vert}$
$\mathsf{do}^{n'}\ gcs'\ \mathsf{od}$	$\mathsf{fsuc}(n',c',f')$	

Omitted: the other 24 cases with nontrivial VCs.

Figure 9: Selected joint VCs for annotation an of $\prod(\mathsf{aut}(c,f),\mathsf{aut}(c',f'),L,R,J)$.

variable to encode the control part of the state. In Figure 8 and Figure 9, the substitution notation is lifted from store relations to state relations and likewise for $\{\!\vert e \vert\!\}$ and $[\!\vert e \vert\!]$.

Lemma 4.15 (VCs for program alignment automata)**.** Consider programs c and c', with alignment automaton $\prod(\mathsf{aut}(c,f),\mathsf{aut}(c',f'),L,R,J)$. Let an be an annotation. For each pair $(n,n'),(m,m')$ of alignment automaton control points, the VC can be expressed as in Figure 9, Figure 8, and in the omitted RO cases that are symmetric to Figure 8.

Proof. First, for any (n, n') and (m, m') the VC is equivalent to the first line of this calculation:

$$
\begin{array}{ll}
 & \mathsf{post}(\overset{(n,n'),(m,m')}{\Rightarrow})(\hat{an}(n,n')) \subseteq \hat{an}(m,m') \\
\Leftrightarrow & \quad \text{definitions} \\
 & \forall i, i', s, s', j, j', t, t'.\ ((i,i'),(s,s')) \in \hat{an}(n,n') \wedge ((i,i'),(s,s')) \\
 & \qquad \overset{(n,n'),(m,m')}{\Rightarrow} ((j,j'),(t,t')) \\
 & \qquad \Rightarrow ((j,j'),(t,t')) \in \hat{an}(m,m') \\
\Leftrightarrow & \quad \text{using def (4.1) of hat} \\
 & \forall i, i', s, s', j, j', t, t'.\ (s,s') \in an(n,n') \wedge ((i,i'),(s,s')) \overset{(n,n'),(m,n')}{\Rightarrow} ((j,j'),(t,t')) \\
 & \qquad \Rightarrow (t,t') \in an(m,m') \\
\Leftrightarrow & \quad \text{using def (4.3)} \\
 & \forall i, i', s, s', j, j', t, t'.\ (s,s') \in an(n,n') \wedge (i,i') = (n,n') \wedge (j,j') = (m,m') \\
 & \qquad \wedge ((i,i'),(s,s')) \Rrightarrow ((j,j'),(t,t')) \Rightarrow (t,t') \in an(m,m') \\
\Leftrightarrow & \quad \text{predicate calculus (the one-point rule)} \\
 & \forall s, s', t, t'.\ (s,s') \in an(n,n') \wedge ((n,n'),(s,s')) \Rrightarrow ((m,m'),(t,t')) \qquad (4.4) \\
 & \qquad \Rightarrow (t,t') \in an(m,m')
\end{array}
$$

From here we proceed by distinguishing the three cases discussed preceeding this lemma in Sec. 4.4, depending on whether $n = m$, $n' = m'$, or neither equality holds.

In case $n' = m'$, the LO case applies and unfolding the definition of $\Rrightarrow$ in (4.4) we get

$$
\begin{array}{ll}
\forall s, s', t.\ & (s,s') \in an(n,n') \wedge ((n,n'),(s,s')) \in L \wedge (n,s) \mapsto (m,t) \qquad (4.5) \\
 & \Rightarrow (t,s') \in an(m,n')
\end{array}
$$

In case $n = m$, the RO case applies, which is similar. In case $n \neq m$ and $n' \neq m'$, the JO case applies and unfolding $\Rrightarrow$ in (4.4) we get

$$
\begin{array}{ll}
\forall s, s', t, t'.\ & (s,s') \in an(n,n') \wedge ((n,n'),(s,s')) \in J \wedge (n,s) \mapsto (m,t) \wedge (n',s') \mapsto (m',t') \\
 & \Rightarrow (t,t') \in an(m,m')
\end{array}
$$

From here, we must consider for each LO case the six possible commands $\mathsf{sub}(n, c)$ and successors m, for each RO case the possible commands $\mathsf{sub}(n', c')$ and successors m', and for each JO all 36 combinations.

Here is one case: LO where $\mathsf{sub}(n, c)$ is $x :=^n e$, $m = \mathsf{fsuc}(n, c, f)$, so the VC is for $((n, n'), (m, n'))$. By definition of $\mapsto$, (4.5) is equivalent to

$$
\forall s, s', t.\ \ (s,s') \in an(n,n') \wedge ((n,n'),(s,s')) \in L \wedge t = s[x \mapsto [\![e]\!](s)] \Rightarrow (t,s') \in an(m,n')
$$

which equivales (by def of semantic substitution and one-point rule)

$$
\forall s, s'.\ \ (s,s') \in an(n,n') \wedge ((n,n'),(s,s')) \in L \Rightarrow (s,s') \in an(m,n')^{x|}_{e|}
$$

which, using def (4.1) of lift and def of the control predicate $[n|n']$, is equivalent to

$$
\begin{array}{ll}
\forall i, i', s, s'.\ & ((i,i'),(s,s')) \in \hat{an}(n,n') \wedge ((i,i'),(s,s')) \in [n|n'] \wedge ((i,i'),(s,s')) \in L \Rightarrow \\
 & ((i,i'),(s,s')) \in \hat{an}(m,n')^{x|}_{e|}
\end{array}
$$

Now that the condition is phrased uniformly in terms of state sets, we can write it as $L \wedge \hat{an}(n, n') \wedge [n|n'] \Rightarrow \hat{an}(m, n')^x_e$, and even more succinctly as $L \wedge \breve{an}(n, n') \Rightarrow \hat{an}(m, n')^{x|}_{e|}$ using the $\breve{an}$ abbreviation. This concludes the proof for the left assignment case in Figure 8. (Note: It is for the sake of this last step that we did not eagerly simplify using the one-point

if $\mathsf{sub}(n,c)$ is...	and m is...	then the encoded VC for $((n,n'),(m,n'))$ is...
skip^n	$\mathsf{fsuc}(n,c,f)$	$\tilde{L} \wedge \{\!\{?n \mid ?n'\}\!\} \wedge an(n,n') \Rightarrow an(m,n')$
$x :=^n e$	$\mathsf{fsuc}(n,c,f)$	$\tilde{L} \wedge \{\!\{?n \mid ?n'\}\!\} \wedge an(n,n') \Rightarrow an(m,n')^{x\vert}_{e\vert}$
$\mathsf{hav}^n\ x$	$\mathsf{fsuc}(n,c,f)$	$\tilde{L} \wedge \{\!\{?n \mid ?n'\}\!\} \wedge an(n,n') \Rightarrow \dot{\forall} x\vert.\ an(m,n')$
$\mathsf{if}^n\ gcs\ \mathsf{fi}$	$\mathsf{lab}(d)$ where $e \to d$ is in gcs	$\tilde{L} \wedge \{\!\{?n \mid ?n'\}\!\} \wedge an(n,n') \wedge \{\!\{e\}\!\} \Rightarrow an(m,n')$
$\mathsf{do}^n\ gcs\ \mathsf{od}$	$\mathsf{lab}(d)$ where $e \to d$ is in gcs	$\tilde{L} \wedge \{\!\{?n \mid ?n'\}\!\} \wedge an(n,n') \wedge \{\!\{e\}\!\} \Rightarrow an(m,n')$
$\mathsf{do}^n\ gcs\ \mathsf{od}$	$\mathsf{fsuc}(n,c,f)$	$\tilde{L} \wedge \{\!\{?n \mid ?n'\}\!\} \wedge an(n,n') \wedge \neg\{\!\{\mathsf{enab}(gcs)\}\!\} \Rightarrow an(m,n')$

Figure 10: The pc-encoded left-only VCs for an and $\prod(\mathsf{aut}(c,f),\mathsf{aut}(c',f'),L,R,J)$.

rule with $(i,i') = (n,n')$. For the quantifier-free notation to make sense, everything needs to be a predicate of the same type.)

All the other cases are proved similarly. □

Encoding relational VCs in terms of store relations. The unary VCs are given in terms of store predicates (Figure 7). For relational VCs, although the annotation comprises store relations, the relational VCs involve alignment conditions (L,R,J) which are sets of alignment automaton states. So the relational VCs are given in terms of states, using $a\hat{n}$ and $a\check{n}$ in Figs. 8 and 9. For use in RHL+ proofs about programs in automaton normal form, we will encode the VCs in terms of relations on stores that use a variable pc (for "program counter") to encode control information. (The normal form is sketched in section 2 and developed in section 5.)

We often use the following abbreviations for assignments and tests of the chosen pc variable.

$$!n \text{ for } pc := n \qquad ?n \text{ for } pc = n \qquad \text{for literal } n \in \mathbb{N} \tag{4.6}$$

To be precise, we define $!n$ to be $pc :=^0 n$, using 0 arbitrarily as the label. Our uses of $!n$ will be in contexts where labels are irrelevant, in commands that are not required to be ok. Note that the notations $!n$ and $?n$ depend, implicitly, on the choice of the variable pc.

Definition 4.16. Let R be a set of states of a program alignment automaton. Let pc be a variable such that $\mathsf{indep}(pc|pc,R)$. Define the ***pc*-encoded** R to be a relation on stores as follows: $\tilde{R} \;\hat{=}\; \{(s,s') \mid ((s(pc),s'(pc)),(s,s')) \in R\}$.

Lemma 4.17. Let R be set of alignment automaton states such that $\mathsf{indep}(pc|pc,R)$. If $(s,s') \in \tilde{R} \wedge \{\!\{?n \mid ?n'\}\!\}$ then $((n,n'),(s,s')) \in R$.

Proof. Suppose $(s,s') \in \tilde{R} \wedge \{\!\{?n \mid ?n'\}\!\}$. Then $s(pc) = n$ and $s'(pc) = n'$ by definition of $\{\!\{?n \mid ?n'\}\!\}$, hence $((n,n'),(s,s')) \in R$ by definition of $\tilde{R}$. □

The encoding makes it possible to derive pc-encoded forms of the verification conditions in terms of store relations, enabling their use in deductive proofs.

Lemma 4.18 (pc-encoded VCs for alignment automata)**.** Let an be an annotation of an alignment automaton $\prod(\mathsf{aut}(c,f),\mathsf{aut}(c',f'),L,R,J)$ for commands c,c'. Suppose pc is a fresh variable in the sense that it does not occur in c or c', and all of L,R,J and any $an(i,j)$ are independent from pc on both sides. Then each left-only relational VC of Figure 8 implies the corresponding condition on store relations in Figure 10. Each joint relational VC in Figure 9 implies a corresponding condition (see Figure 11). Similarly for right-only VCs.

if $\mathsf{sub}(n,c)$ are... $\mathsf{sub}(n',c')$	and m are... m'	then the encoded VC for $((n,n'),(m,m'))$ is...
skip^n	$\mathsf{fsuc}(n,c,f)$	$\tilde{J} \wedge \{\!\{?n \mid ?n'\}\!\} \wedge an(n,n') \Rightarrow an(m,m')$
$\mathsf{skip}^{n'}$	$\mathsf{fsuc}(n',c',f')$	
$x :=^n e$	$\mathsf{fsuc}(n,c,f)$	$\tilde{J} \wedge \{\!\{?n \mid ?n'\}\!\} \wedge an(n,n') \Rightarrow an(m,m')^{x\mid x'}_{e\mid e'}$
$x' :=^{n'} e'$	$\mathsf{fsuc}(n',c',f')$	
if^n gcs fi	$\mathsf{lab}(d)$ where $e \to d$ is in gcs	$\tilde{J} \wedge \{\!\{?n \mid ?n'\}\!\} \wedge an(n,n') \wedge \{\!\{e \mid e'\}\!\} \Rightarrow an(m,m')$
$\mathsf{if}^{n'}$ gcs' fi	$\mathsf{lab}(d')$ where $e' \to d'$ is in gcs'	

Figure 11: Selected *pc*-encoded joint VCs for an and $\prod(\mathsf{aut}(c,f),\mathsf{aut}(c',f'),L,R,J)$.

Proof. Consider the left-only skip case. The primary VC is $L \wedge \check{an}(n,n') \Rightarrow \hat{an}(m,n')$ (first row of Figure 8), which abbreviates

$$L \wedge [n|n'] \wedge \hat{an}(n,n') \Rightarrow \hat{an}(m,n') \tag{4.7}$$

We show this implication entails the first line in Figure 10, i.e., $\tilde{L} \wedge \{\!\{?n \mid ?n'\}\!\} \wedge an(n,n') \Rightarrow an(m,n')$. To prove the latter implication, observe for any (s,s')

	$(s,s') \in \tilde{L} \wedge (s,s') \in \{\!\{?n \mid ?n'\}\!\} \wedge (s,s') \in an(n,n')$	
$\Rightarrow$	$((n,n'),(s,s')) \in L \wedge (s,s') \in an(n,n')$	Lemma 4.17
$\Leftrightarrow$	$((n,n'),(s,s')) \in L \wedge ((n,n'),(s,s')) \in [n\|n'] \wedge (s,s') \in an(n,n')$	def $[n\|n']$
$\Leftrightarrow$	$((n,n'),(s,s')) \in L \wedge ((n,n'),(s,s')) \in [n\|n'] \wedge ((n,n'),(s,s')) \in \hat{an}(n,n')$	def (4.1)
$\Rightarrow$	$((n,n'),(s,s')) \in \hat{an}(m,n')$	primary VC (4.7)
$\Leftrightarrow$	$(s,s') \in an(m,n')$	def (4.1)

The argument is essentially the same for all cases in Figs. 8 and 9. □

5. Automaton normal form reduction and KAT

5.1. Command equivalence and KAT.

Definition 5.1. A ***Kleene algebra with tests*** [Koz97] (***KAT***) is a structure $(\mathbb{K},\mathbb{B},+,;,*,\neg,1,0)$ such that (a) $\mathbb{K}$ is a set and $\mathbb{B} \subseteq \mathbb{K}$ (elements of $\mathbb{B}$ are called ***tests***). (b) $\mathbb{B}$ contains 1 and 0, and is closed under the operations $+,;,\neg$, and these satisfy the laws of Boolean algebra, with 1 as true and ; as conjunction. (c) $\mathbb{K}$ is an idempotent semiring and the following hold for all x,y,z in $\mathbb{K}$.

$$1 + x\,;x^* = x^* \qquad y + x\,;z \le z \;\Rightarrow\; x^*\,;y \le z$$
$$1 + x^*\,;x = x^* \qquad y + z\,;x \le z \;\Rightarrow\; y\,;x^* \le z$$

The ordering $\le$ is defined by $x \le y$ iff $x + y = y$. The operator ; binds tighter than $+$.

In a ***relational model*** [KS96], $\mathbb{K}$ is some set of relations on some set Σ, with 0 the empty relation, 1 the identity relation on Σ, $+$ union of relations, ; relational composition, and $*$ reflexive-transitive closure. Moreover $\mathbb{B}$ is a set of ***coreflexives***, i.e., subsets of the identity relation 1, and $\neg$ is complement with respect to 1. In a relational model, $\le$ is set inclusion. The ***relational model for GCL***, denoted $\mathfrak{L}$, is the relational model comprising all relations on Σ where Σ is the set of variable stores.[15]

[15]KAT is complete for relational models and there are other completeness results for KAT [KS96]. We make no use of those results. All we need about KAT is that it is sufficient for our Theorem 5.13. That result is like [Koz97] which uses KAT to prove that every command is simulated by one with a single loop.

Representing programs in KAT. In this paper we work with $\mathfrak{L}$ and also with equational hypotheses formalized in terms of KAT expressions which are usually defined with respect to given finite sets of primitive tests and actions [KS96, Sect. 2.3]. We only need the special case where the actions are primitive commands and the tests are primitive boolean expressions.[16] Define the ***KAT expressions*** as follows.[17] Recall that *bprim* stands for primitive boolean expressions in GCL.

$$\begin{array}{lcl} KB & ::= & \underline{bprim} \mid KB + KB \mid KB\,;KB \mid \neg KB \mid 1 \mid 0 \\ KE & ::= & \underline{x := e} \mid \underline{\mathsf{hav}\ x} \mid KE + KE \mid KE\,;KE \mid KE^* \mid KB \end{array}$$

We refer to terms given by KB as KAT boolean expressions and by KE as KAT expressions.

Given any model, and mappings of the primitive expressions to elements in the model, one obtains an interpretation of all KAT expressions in the model [KS96, Sect. 2.3]. To be precise, let $\mathfrak{M}$ be a model $(\mathbb{K}^{\mathfrak{M}}, \mathbb{B}^{\mathfrak{M}}, +^{\mathfrak{M}}, ;^{\mathfrak{M}}, *^{\mathfrak{M}}, \neg^{\mathfrak{M}}, 1^{\mathfrak{M}}, 0^{\mathfrak{M}})$. Given interpretations $\underline{bprim}^{\mathfrak{M}}$, $\underline{x := e}^{\mathfrak{M}}$, and $\underline{\mathsf{hav}\ x}^{\mathfrak{M}}$ in $\mathfrak{M}$ for each primitive KAT expression, we get an interpretation $KE^{\mathfrak{M}}$ for every KE, defined homomorphically as usual. For example: $(KE_0 + KE_1)^{\mathfrak{M}} \mathrel{\hat{=}} KE_0^{\mathfrak{M}} +^{\mathfrak{M}} KE_1^{\mathfrak{M}}$. Henceforth we never write $+^{\mathfrak{M}}$ because the reader can infer whether the symbol $+$ is meant as syntax in a KAT expression or as an operation of a particular KAT.

The ***GCL-to-KAT translation*** $(\!|-|\!)$ maps GCL commands (resp. boolean expressions) to KAT expressions (KE) (resp. KAT boolean expressions (KB)). For boolean expressions it is defined by structural recursion as follows:

$$(\!|\mathit{bprim}|\!) \mathrel{\hat{=}} \underline{\mathit{bprim}} \qquad (\!|\neg e|\!) \mathrel{\hat{=}} \neg(\!|e|\!) \qquad (\!|e \wedge e'|\!) \mathrel{\hat{=}} (\!|e|\!)\,;(\!|e'|\!) \qquad (\!|e \vee e'|\!) \mathrel{\hat{=}} (\!|e|\!) + (\!|e'|\!)$$

For commands the translation is defined by mutual recursion with the definition of $(\!|\mathit{gcs}|\!)$:

$$\begin{array}{lcllcl} (\!|x := e|\!) & \mathrel{\hat{=}} & \underline{x := e} & (\!|c;d|\!) & \mathrel{\hat{=}} & (\!|c|\!)\,;(\!|d|\!) \\ (\!|\mathsf{hav}\ x|\!) & \mathrel{\hat{=}} & \underline{\mathsf{hav}\ x} & (\!|\mathsf{if}\ \mathit{gcs}\ \mathsf{fi}|\!) & \mathrel{\hat{=}} & (\!|\mathit{gcs}|\!) \\ (\!|\mathsf{skip}|\!) & \mathrel{\hat{=}} & 1 & (\!|\mathsf{do}\ \mathit{gcs}\ \mathsf{od}|\!) & \mathrel{\hat{=}} & (\!|\mathit{gcs}|\!)^*\,;\neg(\!|\mathsf{enab}(\mathit{gcs})|\!) \\ (\!|\mathit{gcs}|\!) & \mathrel{\hat{=}} & (+e, c\ :\ (e \to c) \in \mathit{gcs}\ :\ (\!|e|\!)\,;(\!|c|\!)) & & & \end{array} \tag{5.1}$$

This is an adaptation of the well known translation of imperative programs into KAT [Koz97].

We define interpretations in $\mathfrak{L}$ for the primitive expressions, as follows:

$$\begin{array}{lcl} \underline{x := e}^{\mathfrak{L}} & \mathrel{\hat{=}} & [\![\, x := e \,]\!] \\ \underline{\mathsf{hav}\ x}^{\mathfrak{L}} & \mathrel{\hat{=}} & [\![\, \mathsf{hav}\ x \,]\!] \end{array} \qquad \begin{array}{lcl} \underline{\mathit{bprim}}^{\mathfrak{L}} & \mathrel{\hat{=}} & \{(s,s) \mid [\![\, \mathit{bprim} \,]\!](s) = \mathit{true}\} \end{array}$$

As with an interpretation in any model, this induces an interpretation $KE^{\mathfrak{L}}$ for any KAT expression KE.

Lemma 5.2. For all commands c we have $(\!|c|\!)^{\mathfrak{L}} = [\![\, c \,]\!]$. For all boolean expressions e we have $(\!|e|\!)^{\mathfrak{L}} = \{(s,s) \mid [\![\, e \,]\!](s) = \mathit{true}\}$.

Proof. The proof is by induction on e and then on c. The base cases are by definition. A secondary induction is used for the loop case. □

[16] Unlike in [KS96], we have infinitely many primitives, because the GCL grammar generates infinitely many boolean and arithemetic expressions, but this causes no problems.

[17] The primitives $\underline{\mathit{bprim}}$, $\underline{x := e}$, and $\underline{\mathsf{hav}\ x}$ are written this way to avoid confusion with their counterparts in GCL syntax.

Command equivalence and soundness of HL+ and RHL+. For specifications where the pre- and post-condition are expressible as tests in the KAT, one can express correctness judgments [Koz00]. In our setting, for boolean expressions e_0 and e_1 we have that $\models c : e_0 \rightsquigarrow e_1$ is equivalent to the equation $(\!|\, e_0 \,|\!)\,;(\!|\, c \,|\!)\,;\neg(\!|\, e_1 \,|\!) = 0$ being true in $\mathfrak{L}$. In this paper we do not use the KAT formulation for correctness judgments in general, but we do use it in a limited way.

Let H be a set of equations between KAT expressions. We write $H \vdash KE_0 = KE_1$ to say $KE_0 = KE_1$ is provable by equational reasoning from hypotheses H plus the axioms of KAT (Definition 5.1).

The idea for the equivalence condition $c \simeq d$ in rules Rewrite (Figure 1) and rRewrite (Figure 2) is that it should mean $H \vdash (\!|\, c \,|\!) = (\!|\, d \,|\!)$ for a suitable set of hypotheses that axiomatize the semantics of some primitive boolean expressions and commands. To prove our main results we only need a few axioms (as detailed in Appendix B). For the sake of a straightforward presentation we formulate equivalence in terms of a larger set of axioms.

Definition 5.3. Define Hyp to be the set of equations given by: (a) the equation $(\!|\, e \,|\!) = 0$ for every boolean expression e such that $e \Rightarrow \mathit{false}$ is valid; (b) the equation $(\!|\, e_0 \,|\!);(\!|\, x := e \,|\!);\neg(\!|\, e_1 \,|\!) = 0$ for all assignments $x := e$ and boolean expressions e_0, e_1 such that $e_0 \Rightarrow e_1{}^{x}_{e}$ is valid; (c) the equation $(\!|\, e_0 \,|\!);(\!|\, \mathsf{hav}\ x \,|\!);\neg(\!|\, e_1 \,|\!) = 0$ for x, e_0, e_1 such that $e_0 \Rightarrow \dot{\forall} x.\, e_1$ is valid.

Definition 5.4 (command equivalence)**.** Define $\simeq$ by $c \simeq d \;\hat{=}\; \mathsf{Hyp} \vdash (\!|\, c \,|\!) = (\!|\, d \,|\!)$.

An example is the equivalence $\mathsf{do}\ e_0 \to c\ \mathsf{od} \;\simeq\; \mathsf{do}\ e_0 \to c; \mathsf{do}\ e_0 \wedge e_1 \to c\ \mathsf{od}\ \mathsf{od}$ which holds for any c, e_0, e_1. It is used in the loop tiling example, see (2.1).

Remark 5.5. It is straightforward to present a deductive system for $\simeq$, based on Hyp, the axioms of KAT, and the rules of equational logic. For practical purposes an alternative is to leverage the fact that the hypotheses are all equations of the form $KE = 0$. For any finite set H of such equations, entailments $H \vdash KE_0 = KE_1$ are decidable in PSPACE [CKS96]. Many practical cases of $\simeq$ require no hypotheses at all. For the equivalences used in our normal form theorem, the requisite hypotheses are a finite set of KAT-consequences of Hyp, syntactically determined by the relevant command c as detailed in Appendix B. □

Having defined $\simeq$, the key ingredient of the rules Rewrite and rRewrite, we have completed the definition of HL+ and RHL+.

Lemma 5.6. Every equation in Hyp holds in $\mathfrak{L}$.

This is an easy consequence of the definition of Hyp.

Theorem 5.7. All the rules of HL+ are sound.

Proof. The proofs are straightforward using the definitions, and induction for the loop rule. The only rule which is not standard is Rewrite which we prove as follows. Suppose $c \simeq d$ holds. That means $\mathsf{Hyp} \vdash (\!|\, c \,|\!) = (\!|\, d \,|\!)$ so $(\!|\, c \,|\!)^{\mathfrak{M}} = (\!|\, d \,|\!)^{\mathfrak{M}}$ in any model $\mathfrak{M}$ that satisfies the equations Hyp. So by Lemma 5.6 we have $(\!|\, c \,|\!)^{\mathfrak{L}} = (\!|\, d \,|\!)^{\mathfrak{L}}$. Hence $[\![\, c \,]\!] = [\![\, d \,]\!]$ by Lemma 5.2. Suppose the premise of Rewrite holds, i.e., $\models c : P \rightsquigarrow Q$. This is a condition on $[\![\, c \,]\!]$ —see (3.1)— so we have $\models d : P \rightsquigarrow Q$. □

Theorem 5.8. All the rules of RHL+ (Figure 2) are sound.

Proof. The proofs are straightforward using the definitions, and induction for the loop rule. (Similar loop rules are proved sound in [Ber11, BNNN22] and in the long version of [NN21].) The proof of rRewrite is similar to the proof of Rewrite for Theorem 5.7. □

$$
\begin{array}{lcl}
\mathsf{add}^{pc}(\mathsf{skip}^n) & \hat{=} & !n\,;\mathsf{skip}^n \\
\mathsf{add}^{pc}(x :=^n e) & \hat{=} & !n\,;x :=^n e \\
\mathsf{add}^{pc}(\mathsf{hav}^n\ x) & \hat{=} & !n\,;\mathsf{hav}^n\ x \\
\mathsf{add}^{pc}(c;d) & \hat{=} & \mathsf{add}^{pc}(c)\,;\mathsf{add}^{pc}(d) \\
\mathsf{add}^{pc}(\mathsf{if}^n\ gcs\ \mathsf{fi}) & \hat{=} & !n\,;\mathsf{if}^n\ (\mathsf{add}_0^{pc}(gcs))\ \mathsf{fi} \\
\mathsf{add}^{pc}(\mathsf{do}^n\ gcs\ \mathsf{od}) & \hat{=} & !n\,;\mathsf{do}^n\ (\mathsf{add}_1^{pc}(n, gcs))\ \mathsf{od} \\
\mathsf{add}_0^{pc}(e \to c) & \hat{=} & e \to \mathsf{add}^{pc}(c) \\
\mathsf{add}_0^{pc}(e \to c \,[\!]\, gcs) & \hat{=} & e \to \mathsf{add}^{pc}(c) \,[\!]\, \mathsf{add}_0^{pc}(gcs) \\
\mathsf{add}_1^{pc}(n, e \to c) & \hat{=} & e \to \mathsf{add}^{pc}(c); !n \\
\mathsf{add}_1^{pc}(n, e \to c \,[\!]\, gcs) & \hat{=} & e \to \mathsf{add}^{pc}(c); !n \,[\!]\, \mathsf{add}_1^{pc}(n, gcs)
\end{array}
$$

Figure 12: Definition of add^{pc}.

$$
\mathsf{skip}^n \,/\, f \hookrightarrow (?n \to !f) \qquad x :=^n e \,/\, f \hookrightarrow (?n \to x := e; !f) \qquad \mathsf{hav}^n\ x \,/\, f \hookrightarrow (?n \to \mathsf{hav}\ x; !f)
$$

$$
\frac{c_0 \,/\, f \hookrightarrow gcs_0 \qquad c_1 \,/\, f \hookrightarrow gcs_1}{\mathsf{if}^n\ e_0 \to c_0 \,[\!]\, e_1 \to c_1\ \mathsf{fi} \,/\, f \hookrightarrow ?n \wedge e_0 \to !\mathsf{lab}(c_0) \,[\!]\, ?n \wedge e_1 \to !\mathsf{lab}(c_1) \,[\!]\, gcs_0 \,[\!]\, gcs_1}
$$

$$
\frac{c \,/\, \mathsf{lab}(d) \hookrightarrow gcs_0 \qquad d \,/\, f \hookrightarrow gcs_1}{c; d \,/\, f \hookrightarrow gcs_0 \,[\!]\, gcs_1} \qquad \frac{c \,/\, n \hookrightarrow gcs}{\mathsf{do}^n\ e \to c\ \mathsf{od} \,/\, f \hookrightarrow ?n \wedge e \to !\mathsf{lab}(c) \,[\!]\, ?n \wedge \neg e \to !f \,[\!]\, gcs}
$$

$$
\frac{c_0 \,/\, n \hookrightarrow gcs_0 \qquad c_1 \,/\, n \hookrightarrow gcs_1}{\mathsf{do}^n\ e_0 \to c_0 \,[\!]\, e_1 \to c_1\ \mathsf{od} \,/\, f \hookrightarrow ?n \wedge e_0 \to !\mathsf{lab}(c_0) \,[\!]\, ?n \wedge e_1 \to !\mathsf{lab}(c_1) \,[\!]\, ?n \wedge \neg(e_0 \vee e_1) \to !f \,[\!]\, gcs_0 \,[\!]\, gcs_1}
$$

Figure 13: Normal form bodies.

5.2. **Automaton normal form.** Choose a variable name pc. Figure 12 defines add^{pc}, a function from commands to commands that adds pc as in the example $c0^+$ on page 8. The definition of add^{pc} is by structural recursion, with mutually recursive helpers add_0^{pc} and add_1^{pc}, and is written using the abbreviations in (4.6). For if commands, add_0^{pc} maps add^{pc} over the guarded commands, and for do, add_1^{pc} additionally adds a trailing assignment to set pc to the loop label.

Lemma 5.9. If pc does not occur in c then $\mathsf{erase}(pc, \mathsf{add}^{pc}(c)) \simeq c$ and $\mathsf{ghost}(pc, \mathsf{add}^{pc}(c))$ holds.

Proof. To show $\mathsf{erase}(pc, \mathsf{add}^{pc}(c)) \simeq c$, we must show $\mathsf{Hyp} \vdash (\!|\, \mathsf{erase}(pc, \mathsf{add}^{pc}(c)) \,|\!) = (\!|\, c \,|\!)$. In fact we can show $\vdash (\!|\, \mathsf{erase}(pc, \mathsf{add}^{pc}(c)) \,|\!) = (\!|\, c \,|\!)$. To do so, go by induction on c, using that $(\!|\, \mathsf{skip} \,|\!) = 1$ and 1 is the unit of sequence. The proof of $\mathsf{ghost}(pc, \mathsf{add}^{pc}(c))$ is also a straightforward induction on c. □

Figure 13 defines the ternary relation $c \,/\, m \hookrightarrow gcs$ to relate a command and a label to the gcs that will be the body of its normal form. To be precise, the relation depends on a choice of pc variable but we leave this implicit, as it is in the notations $?\_$ and $!\_$ of (4.6). We can write "$c \,/\, m \hookrightarrow gcs$ (for pc)" to make the choice explicit. For readability, Figure 13 gives special cases for if and do. The general form is relegated to Appendix C.

An example is $x :=^4 x - 1 \,/\, 2 \hookrightarrow (?4 \to x := x - 1; !2)$. For the running example, we have $c0 \,/\, 6 \hookrightarrow gcs$ where gcs is the body of example $d0$ on page 8.

There are six kinds of transition in the small step semantics. Each kind has a corresponding form of guarded command in the normal form. We spell them out for later reference.

Lemma 5.10 (guarded commands of a normal form)**.** Suppose $\mathsf{okf}(c, f)$ and $c \,/\, f \hookrightarrow gcs$. Every guarded command in gcs has one of these six forms:

- $?k \to !m$, for some k, m such that $\mathsf{sub}(k, c)$ is skip^k and $m = \mathsf{fsuc}(k, c, f)$.
- $?k \to x := e; !m$, for some k, m, x, e such that $\mathsf{sub}(k, c)$ is $x :=^k e$ and $m = \mathsf{fsuc}(k, c, f)$.
- $?k \to \mathsf{hav}\ x; !m$, for some k, m, x such that $\mathsf{sub}(k, c)$ is $\mathsf{hav}^k\ x$ and $m = \mathsf{fsuc}(k, c, f)$.
- $?k \wedge e \to !m$, for some k, m, e, d, gcs_0 such that $\mathsf{sub}(k, c)$ is $\mathsf{if}^k\ gcs_0\ \mathsf{fi}$ and $m = \mathsf{lab}(d)$ where $e \to d$ is in gcs_0.
- $?k \wedge e \to !m$, for some k, m, e, d, gcs_0 such that $\mathsf{sub}(k, c)$ is $\mathsf{do}^k\ gcs_0\ \mathsf{od}$ and $m = \mathsf{lab}(d)$ where $e \to d$ is in gcs_0.
- $?k \wedge \neg\mathsf{enab}(gcs_0) \to !m$, for some k, m, gcs_0 such that $\mathsf{sub}(k, c)$ is $\mathsf{do}^k\ gcs_0\ \mathsf{od}$ and $m = \mathsf{fsuc}(k, c, f)$.

Lemma 5.11. For all c and f, there is some gcs with $c \,/\, f \hookrightarrow gcs$.

Proof. Straightforward structural induction on c. □

In fact gcs is uniquely determined by c, m, and the chosen variable pc. But none of our results depend on uniqueness.

Definition 5.12. An ***automaton normal form*** of a command c with $\mathsf{lab}(c) = n$ and label $f \notin \mathsf{labs}(c)$, for chosen variable pc, is the command $!n; \mathsf{do}\ gcs\ \mathsf{od}$ where $c \,/\, f \hookrightarrow gcs$.

This does not require c to be ok, but the normal form is only useful if $\mathsf{okf}(c, f)$.

As noted in section 1, we are not using the term "normal form" in the sense of term rewriting systems [BN99]. It is not the case that semantically equal programs have identical automaton normal forms. The important property is the theorem to follow.

Normal form equivalence theorem. Finally we are ready for the main result of section 5, which loosely speaking says every command is equivalent to one in automaton normal form.

Theorem 5.13. If $\mathsf{okf}(c, f)$, $pc \notin vars(c)$, and $c \,/\, f \hookrightarrow gcs$ (for pc) then

$$!n; \mathsf{do}\ gcs\ \mathsf{od} \simeq \mathsf{add}^{pc}(c); !f \quad \text{where } n = \mathsf{lab}(c). \tag{5.2}$$

Proof. By definition of $\simeq$ we must prove $\mathsf{Hyp} \vdash (\!|\, !n; \mathsf{do}\ gcs\ \mathsf{od} \,|\!) = (\!|\, \mathsf{add}^{pc}(c); !f \,|\!)$ using laws of KAT. In fact we do not need all of Hyp; the proof shows that $\mathsf{nfax}(pc, c, f) \vdash (\!|\, !n; \mathsf{do}\ gcs\ \mathsf{od} \,|\!) = (\!|\, \mathsf{add}^{pc}(c); !f \,|\!)$ where $\mathsf{nfax}(pc, c, f)$ is the relevant finite set of axioms for c (detailed in Appendix B).[18] These include the following:

(setTest) $(\!|\, !i \,|\!)\,;(\!|\, ?i \,|\!) = (\!|\, !i \,|\!)$ for i in $\mathsf{labs}(c) \cup \{f\}$.

(diffTest) $(\!|\, ?i \,|\!)\,;(\!|\, ?j \,|\!) = 0$ for i and j in $\mathsf{labs}(c) \cup \{f\}$ such that $i \neq j$.

By equational reasoning these yield consequences such as

(diffTestNeg) $(\!|\, ?i \,|\!) = (\!|\, ?i \,|\!); \neg(\!|\, ?j \,|\!)$ for $i \neq j$ with i, j in $\mathsf{labs}(c) \cup \{f\}$.

(nf-enab-labs) $(\!|\, \mathsf{enab}(gcs) \,|\!) = (\!|\, (\vee i \,:\, i \in \mathsf{labs}(c) \,:\, ?i) \,|\!)$.

The theorem's proof goes by rule induction on $c \,/\, f \hookrightarrow gcs$. There is one normal form rule per command form, so we proceed by cases on those forms. In each case, aside from unfolding definitions of add^{pc}, $(\!|\, - \,|\!)$, etc., we use only KAT reasoning and the nfax hypotheses, together with induction hypotheses for subprograms. The lengthy details can be found in [NBN23, Appendix D]; here we show just one case.

[18] As remarked following Def. 5.4, owing to the form of our hypotheses any instance is decidable (in PSPACE). But this does not help prove the theorem, where we have infinitely many instances to prove.

$$
\begin{array}{ll}
 & (\!| !n; \mathsf{do}\ ?n \to x :=^n e; !f\ \mathsf{od} |\!) \\
= & \quad \text{def } (\!| - |\!), \text{ see (5.1), and def } \mathsf{enab} \\
 & (\!| !n |\!); ((\!| ?n |\!); (\!| x :=^n e |\!); (\!| !f |\!))^*; \neg (\!| ?n |\!) \\
= & \quad \text{star unfold, distrib} \\
 & (\!| !n |\!); \neg (\!| ?n |\!) + (\!| !n |\!); (\!| ?n |\!); (\!| x :=^n e |\!); (\!| !f |\!); ((\!| ?n |\!); (\!| x :=^n e |\!); (\!| !f |\!))^*; \neg (\!| ?n |\!) \\
= & \quad \text{left term is 0, using axiom (setTest) and lemma (diffTestNeg)} \\
 & (\!| !n |\!); (\!| ?n |\!); (\!| x :=^n e |\!); (\!| !f |\!); ((\!| ?n |\!); (\!| x :=^n e |\!); (\!| !f |\!))^*; \neg (\!| ?n |\!) \\
= & \quad \text{(setTest) for } n \\
 & (\!| !n |\!); (\!| x :=^n e |\!); (\!| !f |\!); ((\!| ?n |\!); (\!| x :=^n e |\!); (\!| !f |\!))^*; \neg (\!| ?n |\!) \\
= & \quad \text{(setTest) for } f \\
 & (\!| !n |\!); (\!| x :=^n e |\!); (\!| !f |\!); (\!| ?f |\!); ((\!| ?n |\!); (\!| x :=^n e |\!); (\!| !f |\!))^*; \neg (\!| ?n |\!) \\
= & \quad \text{(diffTest) with } f \neq n \text{ from } \mathsf{okf}(c, f); \\
 & \quad \text{KAT fact } p; (q; a)^* = p \text{ if } p;\ q = 0 \text{ (any } p, q, a) \\
 & (\!| !n |\!); (\!| x :=^n e |\!); (\!| !f |\!); (\!| ?f |\!); \neg (\!| ?n |\!) \\
= & \quad \text{(diffTestNeg), } f \neq n \text{ by } \mathsf{okf}(c, f) \\
 & (\!| !n |\!); (\!| x :=^n e |\!); (\!| !f |\!); (\!| ?f |\!) \\
= & \quad \text{(setTest)} \\
 & (\!| !n |\!); (\!| x :=^n e |\!); (\!| !f |\!) \\
= & \quad \text{def } (\!| - |\!) \\
 & (\!| !n; x :=^n e; !f |\!) \\
= & \quad \text{def } \mathsf{add}^{pc} \\
 & (\!| \mathsf{add}^{pc}(x :=^n e); !f |\!)
\end{array}
$$

Figure 14: Proof of assignment case for Theorem 5.13.

Case c **is** $x :=^n e$. We have $x :=^n e \,/\, f \hookrightarrow (?n \to x :=^n e; !f)$. Operationally, the loop $\mathsf{do}\ ?n \to x :=^n e; !f\ \mathsf{od}$ iterates exactly once. This is reflected in our proof of (5.2) for this case, in which we unroll the loop once; see the calculation in Figure 14. Note: in hints we do not mention associativity, unit law for 1, etc. □

Lemma 5.14. If $\mathsf{okf}(c, f)$ and $c \,/\, f \hookrightarrow gcs$ then $[\![\, \mathsf{enab}(gcs) \,]\!] = [\![\, (\vee i \ :\ i \in \mathsf{labs}(c) \ :\ ?i) \,]\!]$.

Proof. This follows from the provable equation (nf-enab-labs) mentioned in the proof of Theorem 5.13, together with Lemma 5.2. □

6. Floyd completeness

In this section we put the normal form equivalence theorem to work showing that any IAM proof of a unary correctness judgment can be translated to one in HL+. This sets a pattern that guides the proofs of alignment completeness. It also gives a way to prove completeness of HL+ in the sense of Cook.

Theorem 6.1. Suppose $\mathsf{okf}(c, f)$. Suppose an is a valid annotation of $\mathsf{aut}(c, f)$ for $P \rightsquigarrow Q$ and P, Q, an are finitely supported. Then $c : P \rightsquigarrow Q$ can be proved in HL+, using only assertions derived from an.

The phrase "derived from an" is deliberately vague, as is the similar result of Nagasamudram and Naumann [NN21]. As sketched by example in section 2, our HL+ proof uses only a single instance of the Do rule and no instance of the If rule. More importantly, the judgments use only assertions derived from those of an in simple ways. In particular, we use boolean combinations of the following: assertions $an(i)$, boolean expressions that occur in c, and equality tests $pc = n$ of the program counter variable and numeric literals; and we use substitution instances $an(i)^x_e$ for assignments $x := e$ that occur in c. The assumption about finite support is a technicality. It holds for assertions expressed by formulas in any usual assertion language.

Proof. Choose variable pc that is fresh with respect to c, P, Q, and an (i.e., for all i, $an(i)$ is independent from pc). Existence of such a variable is ensured by the assumption of finite support. By Lemma 5.11 we have some gcs with $c \,/\, f \hookrightarrow gcs$. By Theorem 5.13 we have

$$!n; \mathsf{do}\ gcs\ \mathsf{od} \ \simeq\ \mathsf{add}^{pc}(c); !f \quad \text{where } n = \mathsf{lab}(c). \tag{6.1}$$

For a loop invariant to reason about the normal form, with an eye on the example we might try this formula: $1 \le pc \le f \wedge (\wedge i \ :\ 0 \le i \le f \ :\ pc = i \Rightarrow an(i))$. But this only makes sense if the labels form a contiguous sequence, which we do not require. There is no need to reason arithmetically about labels. We define the invariant I as follows:

$$I: \quad (\vee i \ :\ i \in \mathsf{labs}(c) \cup \{f\} \ :\ ?i) \wedge (\wedge i \ :\ i \in \mathsf{labs}(c) \cup \{f\} \ :\ ?i \Rightarrow an(i))$$

The disjunction says the current value of pc is in $\mathsf{labs}(c) \cup \{f\}$.

The next step is to obtain proofs of

$$b : I \wedge e \rightsquigarrow I \qquad \text{for each } e \to b \text{ in } gcs \tag{6.2}$$

To do so, first note that we have for any m that

$$!m : an(m) \rightsquigarrow an(m) \wedge ?m \tag{6.3}$$

using rules Asgn and Conseq, because by freshness pc is not in $an(m)$. (We are not writing explicit $\vdash$ for provability of correctness judgments.) Second, note that by definition of I we have valid implications

$$I \wedge ?n \Rightarrow an(n) \qquad \text{and} \qquad an(n) \wedge ?n \Rightarrow I \qquad \text{for any } n \text{ in } \mathsf{labs}(c) \cup \{f\} \tag{6.4}$$

Now go by the possible cases of b in (6.2), which are given by Lemma 5.10.

- b has the form $?n \to x := e; !m$, where $\mathsf{sub}(n, c)$ is $x :=^n e$ and $m = \mathsf{fsuc}(n, c, f)$.
  To show: $x := e; !m : I \wedge ?n \rightsquigarrow I$. By rule Asgn we have $x := e : an(m)^x_e \rightsquigarrow an(m)$. By the VC in Figure 7 we have $an(n) \Rightarrow an(m)^x_e$, so using Conseq we get $x := e : an(n) \rightsquigarrow an(m)$. By fact (6.3) we have $!m : an(m) \rightsquigarrow an(m) \wedge ?m$, so by rule Seq we have $x := e; !m : an(n) \rightsquigarrow an(m) \wedge ?m$. So by Conseq using both implications in fact (6.4) we get $x := e; !m : I \wedge ?n \rightsquigarrow I$.

The other cases are similar (see [NBN23, Appendix D.4]).

Having established the premises of rule Do, we get its conclusion:

$$\mathsf{do}\ gcs\ \mathsf{od} : I \rightsquigarrow I \wedge \neg\mathsf{enab}(gcs)$$

By Lemma 5.14 and definition of I, $I \wedge \neg\mathsf{enab}(gcs)$ is equivalent to $I \wedge ?f$, so by consequence we get $\mathsf{do}\ gcs\ \mathsf{od} : I \wedge ?n \rightsquigarrow I \wedge ?f$. Now $I \wedge ?n$ is equivalent to $an(n) \wedge ?n$. We have $P \Rightarrow an(n)$ and $an(f) \Rightarrow Q$ because an is an annotation for the spec $P \rightsquigarrow Q$. So by

consequence we get $\mathsf{do}\ gcs\ \mathsf{od} : P \wedge ?n \rightsquigarrow Q$. By the assignment rule and consequence using that pc is fresh for P we get $!n : P \rightsquigarrow P \wedge ?n$, so using the sequence rule we get

$$!n; \mathsf{do}\ gcs\ \mathsf{od} : P \rightsquigarrow Q$$

Then Rewrite using (6.1) yields $\mathsf{add}^{pc}(c); !f : P \rightsquigarrow Q$. By Lemma 5.9 and freshness of pc we have that pc is ghost in $\mathsf{add}^{pc}(c); !f$. Also, P and Q are independent from pc. So by rule Ghost we get $\mathsf{erase}(pc, \mathsf{add}^{pc}(c); !f) : P \rightsquigarrow Q$. Now using Lemma 5.9 together with the general law $c; \mathsf{skip} \simeq c$ and transitivity of $\simeq$,[19] we have that $\mathsf{erase}(pc, \mathsf{add}^{pc}(c); !f) \simeq c$, so by Rewrite we get $c : P \rightsquigarrow Q$. □

Corollary 6.2 (HL+ is Cook complete)**.** If $\models c : P \rightsquigarrow Q$ and P, Q are finitely supported then there is a proof of $c : P \rightsquigarrow Q$ in HL+.

Proof. Suppose $\models c : P \rightsquigarrow Q$. Without loss of generality assume c has ok labels and $\mathsf{okf}(c, f)$ for some f. By completeness of IAM (Lemma 4.6) there is a valid and finitely supported annotation of $\mathsf{aut}(c, f)$ for P, Q. So by Theorem 6.1 there is a proof in HL+ of $c : P \rightsquigarrow Q$. □

7. RHL+ is alignment complete and Cook complete

In subsection 7.1 we prove alignment completeness. In subsection 7.2 we prove Cook completeness and consider connections with unary logic, in particular the well known use of sequential alignment to obtain Cook completeness from a complete unary logic.

7.1. Alignment completeness of RHL+.

Our main result for $\forall\forall$ properties says that given any IAM-style proof for a program alignment automaton, one can construct an RHL+ proof. Conditions (b) and (c) in the theorem say there is an IAM-style proof.

Theorem 7.1. Suppose we have the following.
(a) $\mathsf{okf}(c, f)$ and $\mathsf{okf}(c', f')$.
(b) an is a valid annotation of $\prod(\mathsf{aut}(c, f), \mathsf{aut}(c', f'), L, R, J)$ for $\mathcal{S} \rightsquigarrow \mathcal{T}$.
(c) $\breve{an}(i, j) \Rightarrow L \vee R \vee J \vee [\mathit{fin} | \mathit{fin}']$ for all control points (i, j) of $\prod(\mathsf{aut}(c, f), \mathsf{aut}(c', f'), L, R, J)$.
(d) an, $\mathcal{S}$, $\mathcal{T}$, L, R, and J all have finite support.
Then the judgment $c \mid c' : \mathcal{S} \approx\!\!> \mathcal{T}$ has a proof in RHL+.

The proof of the Theorem yields a deductive proof that uses only relational assertions derived in simple ways from the relations $an(i, j)$ of the annotation together with L, R, and J. Specifically, the proof uses boolean combinations of the annotation's assertions, conjunctions with conditional tests in the code (and with L and R), and substitutions for expressions in assignment commands.

Restriction (a) in Theorem 7.1 is just a technicality. The okf condition says labels of c are unique and do not include f. Labels have no effect on program semantics so they can always be chosen to satisfy the condition. Restriction (d) certainly holds when specs are given by formulas in some assertion language; it is a technicality to ensure that a fresh pc variable can be chosen for application of Theorem 5.13.

[19] Both of which are easily derived using the definition of $\simeq$.

Proof. Suppose $\mathcal{S}, \mathcal{T}, c, c', f, f', L, R, J$ and an satisfy the hypotheses (a)–(d) of the theorem. Choose variable pc that is fresh with respect to $\mathcal{S}, \mathcal{T}, c, c', an, L, R, J$. To be precise: $\mathsf{indep}(pc|pc, \mathcal{S})$, $\mathsf{indep}(pc|pc, \mathcal{T})$, pc does not occur in c or c', and L, R, J are independent from pc on both sides, as is $an(i, j)$ for all i, j. Existence of such a variable is ensured by hypothesis (d) of finite support.

By Lemma 5.11 there are gcs and gcs' such that $c \,/\, f \hookrightarrow gcs$ and $c' \,/\, f' \hookrightarrow gcs'$. Let $n = \mathsf{lab}(c)$ and $n' = \mathsf{lab}(c')$. By Theorem 5.13 we have

$$\begin{array}{l} !n; \mathsf{do}\ gcs\ \mathsf{od} \ \simeq\ \mathsf{add}^{pc}(c); !f \\ !n'; \mathsf{do}\ gcs'\ \mathsf{od} \ \simeq\ \mathsf{add}^{pc}(c'); !f' \end{array} \tag{7.1}$$

Define store relation $\mathcal{Q}$ to be $\mathcal{Q}_{an} \wedge \mathcal{Q}_{pc}$ where

$$\begin{array}{ll} \mathcal{Q}_{an}: & (\wedge i, j \ :\ i \in \mathsf{labs}(c) \cup \{f\} \wedge j \in \mathsf{labs}(c') \cup \{f'\} \ :\ \langle\!\langle ?i \mid ?j \rangle\!\rangle \ \Rightarrow an(i, j)) \\ \mathcal{Q}_{pc}: & (\vee i, j \ :\ i \in \mathsf{labs}(c) \cup \{f\} \wedge j \in \mathsf{labs}(c') \cup \{f'\} \ :\ \langle\!\langle ?i \mid ?j \rangle\!\rangle\ ) \end{array}$$

We will derive

$$\mathsf{do}\ gcs\ \mathsf{od} \mid \mathsf{do}\ gcs'\ \mathsf{od} : \mathcal{Q} \approx\!\!> \mathcal{Q} \wedge \neg\langle\!\langle \mathsf{enab}(gcs) \rangle\!\rangle \wedge \neg\rangle\!\langle \mathsf{enab}(gcs') \rangle\!\langle \tag{7.2}$$

using rule rDo instantiated with $\mathcal{Q} := \mathcal{Q}$, $\mathcal{L} := \tilde{L}$, and $\mathcal{R} := \tilde{R}$. The side condition of rDo is

$$\mathcal{Q} \Rightarrow (\langle\!\langle \mathsf{enab}(gcs) \rangle\!\rangle = \rangle\!\langle \mathsf{enab}(gcs') \rangle\!\langle) \vee (\tilde{L} \wedge \langle\!\langle \mathsf{enab}(gcs) \rangle\!\rangle) \vee (\tilde{R} \wedge \rangle\!\langle \mathsf{enab}(gcs') \rangle\!\langle) \tag{7.3}$$

To prove (7.3), first rewrite $\mathcal{Q}$ using distributivity and renaming dummies, to the equivalent form

$$(\vee i, i' \ ::\ \langle\!\langle ?i \mid ?i' \rangle\!\rangle \ \wedge (\wedge k, k' \ ::\ (\langle\!\langle ?k \mid ?k' \rangle\!\rangle \ \Rightarrow an(k, k'))))$$

where we omit that i, k range over $\mathsf{labs}(c) \cup \{f\}$ and i', k' range over $\mathsf{labs}(c') \cup \{f'\}$. This implies

$$(\vee i, i' \ ::\ \langle\!\langle ?i \mid ?i' \rangle\!\rangle \ \wedge an(i, i'))$$

Thus by hypothesis (c), any $\mathcal{Q}$-state satisfies $\tilde{L} \vee \tilde{R} \vee \tilde{J} \vee \langle\!\langle ?f \mid ?f' \rangle\!\rangle$. We show each of these disjuncts implies the right side of (7.3).

- $\tilde{L}$ implies $\tilde{L} \wedge \langle\!\langle \mathsf{enab}(gcs) \rangle\!\rangle$ because (i) L is a set of states of the alignment automaton, with control on the left ranging over $\mathsf{labs}(c) \cup \{f\}$, (ii) by liveness (Def. 4.2), L allows transitions by $\mathsf{aut}(c, f)$, and so excludes control being at f, and (iii) $\langle\!\langle \mathsf{enab}(gcs) \rangle\!\rangle$ means control on the left is in $\mathsf{labs}(c)$, by Lemma 5.14.
- $\tilde{R}$ implies $\tilde{R} \wedge \rangle\!\langle \mathsf{enab}(gcs') \rangle\!\langle$ for reasons symmetric to the L case
- $\tilde{J}$ implies $\langle\!\langle \mathsf{enab}(gcs) \rangle\!\rangle$ and $\rangle\!\langle \mathsf{enab}(gcs') \rangle\!\langle$ are both true (using liveness and Lemma 5.14 again), so $\langle\!\langle \mathsf{enab}(gcs) \rangle\!\rangle = \rangle\!\langle \mathsf{enab}(gcs') \rangle\!\langle$
- $\langle\!\langle ?f \mid ?f' \rangle\!\rangle$ implies both $\langle\!\langle \mathsf{enab}(gcs) \rangle\!\rangle$ and $\rangle\!\langle \mathsf{enab}(gcs') \rangle\!\langle$ are false (again using Lemma 5.14) so $\langle\!\langle \mathsf{enab}(gcs) \rangle\!\rangle = \rangle\!\langle \mathsf{enab}(gcs') \rangle\!\langle$

So the side condition (7.3) of rDo is proved. Before proceeding to prove the premises for rDo, note that we can prove

$$!m \mid !m' : \mathcal{P} \approx\!\!> \mathcal{P} \wedge \langle\!\langle ?m \rangle\!\rangle \wedge \rangle\!\langle ?m' \rangle\!\langle \tag{7.4}$$

for any $\mathcal{P}$ in which pc does not occur, and any m, m', using rules rAsgn and rConseq. Also, by definition of $\mathcal{Q}$ we have valid implications

$$\begin{array}{l} \mathcal{Q} \wedge \langle\!\langle ?m \mid ?m' \rangle\!\rangle \ \Rightarrow an(m, m') \qquad \text{and} \qquad an(m, m') \wedge \langle\!\langle ?m \mid ?m' \rangle\!\rangle \ \Rightarrow \mathcal{Q} \\ \text{for any } m \text{ in } \mathsf{labs}(c) \cup \{f\} \text{ and } m' \text{ in } \mathsf{labs}(c') \cup \{f'\} \end{array} \tag{7.5}$$

From condition (c) of the theorem we get $an(i,j) \wedge \{\!|?i \mid ?j|\!\} \Rightarrow \tilde{L} \vee \tilde{R} \vee \tilde{J} \vee \{\!|?fin \mid ?fin'|\!\}$ for all i, j (by definitions), and hence

$$an(i,j) \wedge \{\!|?i \mid ?j|\!\} \wedge \neg\tilde{L} \wedge \neg\tilde{R} \Rightarrow \tilde{J} \quad \text{for all } i,j \text{ with } i \neq fin \text{ or } j \neq fin' \tag{7.6}$$

There are three sets of premises of rDo for the loops in (7.2), with these forms:

(left-only) $b \mid \mathsf{skip} : \mathcal{Q} \wedge \{\!|e|\!\} \wedge \tilde{L} \approx\!\!> \mathcal{Q}$ for each $e \to b$ in gcs

(right-only) $\mathsf{skip} \mid b' : \mathcal{Q} \wedge |\!\{e'\}\!| \wedge \tilde{R} \approx\!\!> \mathcal{Q}$ for each $e' \to b'$ in gcs'

(joint) $b \mid b' : \mathcal{Q} \wedge \{\!|e \mid e'|\!\} \wedge \neg\tilde{L} \wedge \neg\tilde{R} \approx\!\!> \mathcal{Q}$ for each $e \to b$ in gcs and $e' \to b'$ in gcs'

By Lemma 5.10, the guarded commands in gcs and gcs' have six possible forms, so there are six left-only cases to consider, six right-only, and 36 joint ones. We start with the latter.

Joint cases. For each of the six possibilities for $e \to b$ in gcs for c, we must consider it with each of the six possibilities for $e' \to b'$ in gcs' for c'. We give the argument for one case, with skip on both sides.

- $!m \mid !m' :\ \mathcal{Q} \wedge \{\!|?k \mid ?k'|\!\} \wedge \neg\tilde{L} \wedge \neg\tilde{R} \approx\!\!> \mathcal{Q}$, where
  $\mathsf{sub}(k,c) = \mathsf{skip}^k$, $m = \mathsf{fsuc}(k,c,f)$, $\mathsf{sub}(k',c') = \mathsf{skip}^{k'}$, $m = \mathsf{fsuc}(k',c',f')$.
  By (7.4) (and freshness of pc) we have $!m \mid !m' : an(m,m') \approx\!\!> an(m,m') \wedge \{\!|?m \mid ?m'|\!\}$.
  So by rConseq using the second implication in (7.5) we have

$$!m \mid !m' : an(m,m') \approx\!\!> \mathcal{Q} \tag{7.7}$$

  By the first implication in (7.5) we have $\mathcal{Q} \wedge \{\!|?k \mid ?k'|\!\} \Rightarrow an(k,k')$. So using (7.6) we get $\mathcal{Q} \wedge \{\!|?k \mid ?k'|\!\} \wedge \neg\tilde{L} \wedge \neg\tilde{R} \Rightarrow \tilde{J}$. By validity of the annotation, we have the VC in the first row of Figure 9, i.e., $J \wedge \breve{an}(k,k') \Rightarrow \hat{an}(m,m')$. Then by Lemma 4.18 we get the pc-encoded form $\tilde{J} \wedge \{\!|?k \mid ?k'|\!\} \wedge an(k,k') \Rightarrow an(m,m')$. So this is valid:

$$\mathcal{Q} \wedge \{\!|?k \mid ?k'|\!\} \wedge \neg\tilde{L} \wedge \neg\tilde{R} \Rightarrow an(m,m')$$

  Using this with rConseq and (7.7) yields $!m \mid !m' :\ \mathcal{Q} \wedge \{\!|?k \mid ?k'|\!\} \wedge \neg\tilde{L} \wedge \neg\tilde{R} \approx\!\!> \mathcal{Q}$.

We refrain from spelling out details of the remaining joint cases (more can be found in [NBN23]). The arguments are all similar: every case uses a VC and rule rConseq, together with (7.4). Cases that involve an assignment in the original program c or c' also use rules rAsgnAsgn, rSkipAsgn, or rAsgnSkip. Cases that involve havoc use the corresponding rules.

Left-only cases. These cases are proved using the same rules as the joint cases, plus one additional rule: rDisj, in the form rDisjN derived from it (see Figure 3). This is needed due to the following complication. The joint cases determine a starting pair and ending pair of control points, which determines which VC to appeal to. The left-only cases do not determine a control point on the right side; instead we have VCs for each possible point on the right (Figure 10). So we go by cases on the possible control points on the right, for which purpose we make the following observation. In virtue of the conjunct $\mathcal{Q}_{pc}$ of $\mathcal{Q}$, we have that $\mathcal{Q}$ is equivalent to this disjunction over control points:

$$(\vee\, i,j \ :\ i \in \mathsf{labs}(c) \cup \{f\} \wedge j \in \mathsf{labs}(c') \cup \{f'\} \ :\ \mathcal{Q}^{i,j})$$

where $\mathcal{Q}^{i,j}$ is defined to say control is at those points: $\mathcal{Q}^{i,j} \ \hat{=}\ \mathcal{Q} \wedge \{\!|?i \mid ?j|\!\}$. Now we have the equivalence

$$\mathcal{Q} \wedge \{\!|?i|\!\} \Leftrightarrow (\vee\, j \ :\ j \in \mathsf{labs}(c') \cup \{f'\} \ :\ \mathcal{Q}^{i,j}) \quad \text{for any } i$$

With this we can proceed to prove the left-only premises. We give only the assignment case.

- $x := e; !m \mid \mathsf{skip} : \mathcal{Q} \wedge \langle\!|?k|\!\rangle \wedge \tilde{L} \approx\!\!> \mathcal{Q}$, where $\mathsf{sub}(k, c) = x :=^k e$ and $m = \mathsf{fsuc}(k, c, f)$.

  In accord with the discussion above, we have that $\mathcal{Q} \wedge \langle\!|?k|\!\rangle$ is equivalent to $(\vee\, j \,::\, \mathcal{Q}^{k,j})$ (omitting the range $j \in \mathsf{labs}(c') \cup \{f'\}$), so the goal can be obtained by rConseq from

  $$x := e; !m \mid \mathsf{skip} : (\vee\, j \,::\, \mathcal{Q}^{k,j}) \wedge \tilde{L} \approx\!\!> \mathcal{Q}$$

  In turn, this can be obtained by derived rule rDisjN from judgments

  $$x := e; !m \mid \mathsf{skip} : \mathcal{Q}^{k,j} \wedge \tilde{L} \approx\!\!> \mathcal{Q} \quad \text{for every } j. \tag{7.8}$$

  for all j (in range $j \in \mathsf{labs}(c') \cup \{f'\}$ that we continue to omit). It remains to prove (7.8) for arbitrary j.

  By rAsgnSkip and rConseq (using that $\langle\!|?j|\!\rangle$ is independent from pc on the left) we get

  $$!m \mid \mathsf{skip} : an(m, j) \wedge \langle\!|?j|\!\rangle \approx\!\!> an(m, j) \wedge \langle\!|?m \mid ?j|\!\rangle$$

  from which using (7.5) we get

  $$!m \mid \mathsf{skip} : an(m, j) \wedge \langle\!|?j|\!\rangle \approx\!\!> \mathcal{Q}$$

  By rAsgnSkip, using that $\langle\!|?j|\!\rangle$ is independent from x because pc is fresh, we can prove

  $$x := e \mid \mathsf{skip} : an(m, j)^{x|}_{e|} \wedge \langle\!|?j|\!\rangle \approx\!\!> an(m, j) \wedge \langle\!|?j|\!\rangle$$

  By derived rule rSeqSkip (Figure 3), from the above we get

  $$x := e; !m \mid \mathsf{skip} : an(m, j)^{x|}_{e|} \wedge \langle\!|?j|\!\rangle \approx\!\!> \mathcal{Q}$$

  The disjunction over j was introduced so that we can appeal to a VC, specifically the lifted VC for $((k, j), (m, j))$. It is an instance of the second line in Figure 10 and it says this is valid: $\tilde{L} \wedge \langle\!|?k \mid ?j|\!\rangle \wedge an(k, j) \Rightarrow an(m, j)^{x|}_{e|}$, so by rConseq we get

  $$x := e; !m \mid \mathsf{skip} : \tilde{L} \wedge \langle\!|?k \mid ?j|\!\rangle \wedge an(k, j) \approx\!\!> \mathcal{Q}$$

  By definitions we have $\mathcal{Q}^{k,j} \wedge \tilde{L} \Rightarrow \tilde{L} \wedge \langle\!|?k \mid ?j|\!\rangle \wedge an(k, j)$. Using this with rConseq yields (7.8) and we are done with this case.

The other left-only cases are similar. The right-only cases are symmetric with the left-only cases. We omit them all and proceed.

Finishing the proof. Having proved the premises and the side condition (7.3), rule rDo yields (7.2). The remaining steps are similar to corresponding steps in the proof of the Floyd completeness Theorem 6.1 and we spell them out.

Using Lemma 5.14 twice, and the definition of $\mathcal{Q}$, we have

$$\mathcal{Q} \wedge \neg\langle\!|\mathsf{enab}(gcs)|\!\rangle \wedge \neg\langle\!|\mathsf{enab}(gcs')|\!\rangle \Rightarrow \mathcal{Q} \wedge \langle\!|?f \mid ?f'|\!\rangle$$

So, using the first implication in (7.5) and assumption (b) of the theorem (which says $an(f, f') \Rightarrow \mathcal{T}$ since an is an annotation for $\mathcal{S} \approx\!\!> \mathcal{T}$), we can use rConseq with (7.2) to get

$$\mathsf{do}\ gcs\ \mathsf{od} \mid \mathsf{do}\ gcs'\ \mathsf{od} : \mathcal{Q} \approx\!\!> \mathcal{T}$$

Using the second implication in (7.5) and assumption (b) (which says $\mathcal{S} \Rightarrow an(n, n')$ since n, n' are the initial control points), we have $\langle\!|?n \mid ?n'|\!\rangle \wedge \mathcal{S} \Rightarrow \mathcal{Q}$, so by rConseq we get

$$\mathsf{do}\ gcs\ \mathsf{od} \mid \mathsf{do}\ gcs'\ \mathsf{od} : \langle\!|?n \mid ?n'|\!\rangle \wedge \mathcal{S} \approx\!\!> \mathcal{T}$$

By freshness assumption for pc, by (7.4) we have a proof of $!n \mid !n' : \mathcal{S} \approx\!\!> \langle\!|?n \mid ?n'|\!\rangle \wedge \mathcal{S}$. So by rSeq we get

$$!n; \mathsf{do}\ gcs\ \mathsf{od} \mid !n'; \mathsf{do}\ gcs'\ \mathsf{od} : \mathcal{S} \approx\!\!> \mathcal{T}$$

Now using rule rRewrite with the equivalences (7.1) we get

$$\mathsf{add}^{pc}(c);!f \mid \mathsf{add}^{pc}(c');!f' : \mathcal{S} \approx\!\!> \mathcal{T}$$

By freshness of pc and Lemma 5.9, it has the ghost property for both $\mathsf{add}^{pc}(c);!f$ and $\mathsf{add}^{pc}(c');!f'$, and does not occur in $\mathcal{S}$ or $\mathcal{T}$, so by rule rGhost we get

$$\mathsf{erase}(pc, \mathsf{add}^{pc}(c);!f) \mid \mathsf{erase}(\mathsf{add}^{pc}(c');!f') : \mathcal{S} \approx\!\!> \mathcal{T} \tag{7.9}$$

By definition of erase, we have $\mathsf{erase}(pc, \mathsf{add}^{pc}(c);!f) = \mathsf{erase}(pc, \mathsf{add}^{pc}(c)); \mathsf{skip}$ and also $\mathsf{erase}(\mathsf{add}^{pc}(c');!f') = \mathsf{erase}(pc, \mathsf{add}^{pc}(c')); \mathsf{skip}$. So using Lemma 5.9 together with the general law $c; \mathsf{skip} \simeq c$ and transitivity of $\simeq$, we have

$$\mathsf{erase}(pc, \mathsf{add}^{pc}(c);!f) \simeq c \qquad \text{and} \qquad \mathsf{erase}(pc, \mathsf{add}^{pc}(c');!f') \simeq c'$$

Using these equivalences with rRewrite, from (7.9) we obtain $c \mid c' : \mathcal{S} \approx\!\!> \mathcal{T}$. □

7.2. **Cook completeness revisited.** Cook completeness can be proved as a consequence of alignment completeness.

Theorem 7.2 (Cook completeness of RHL+)**.** Suppose $\models c \mid c' : \mathcal{S} \approx\!\!> \mathcal{T}$ and $\mathcal{S}, \mathcal{T}$ are finitely supported. Then there is a proof of $c \mid c' : \mathcal{S} \approx\!\!> \mathcal{T}$ in RHL+.

Proof. Suppose $\models c \mid c' : \mathcal{S} \approx\!\!> \mathcal{T}$, and assume wlog that $\mathsf{okf}(c, f)$ and $\mathsf{okf}(c', f')$. Using Lemma 4.13 we have $\mathsf{aut}(c, f), \mathsf{aut}(c', f') \models \mathcal{S} \approx\!\!> \mathcal{T}$. By Corollary 4.8 there are L, R, J, an such that an is a valid annotation of $\prod(\mathsf{aut}(c, f), \mathsf{aut}(c', f'), L, R, J)$ for $\mathcal{S} \rightsquigarrow \mathcal{T}$ and for any i, j we have that $\breve{an}(i, j) \Rightarrow L \vee R \vee J \vee [\mathit{fin} | \mathit{fin}']$ which is condition (c) of Theorem 7.1. Moreover these are finitely supported as required by condition (d). We have conditions (a) and (b) as well, so by the theorem we get a proof of $c \mid c' : \mathcal{S} \approx\!\!> \mathcal{T}$ in RHL+. □

Prior Cook completeness results for RHLs were based on a left-first sequential alignment rule like our rLRseq in Figure 3, together with unary HL and a way to represent or interpret the one-sided judgments $c \mid \mathsf{skip} : \mathcal{P} \approx\!\!> \mathcal{Q}$ and $\mathsf{skip} \mid c' : \mathcal{Q} \approx\!\!> \mathcal{R}$ as unary judgments in HL. It is instructive to consider that approach in our setting. But first we indulge in little detour.

Our formulation of RHL+ does not include unary judgments but it subsumes HL+ in the following sense. A unary judgment $c : P \rightsquigarrow Q$ can be represented by the relational judgment $c \mid \mathsf{skip} : \{\!|P|\!\} \approx\!\!> \{\!|Q|\!\}$, as well as by $\mathsf{skip} \mid c : |\!\{P\}\!| \approx\!\!> |\!\{Q\}\!|$. That is, the judgments express the same semantic property of c. So if the judgment $c : P \rightsquigarrow Q$ is valid then we have $\models c \mid \mathsf{skip} : \{\!|P|\!\} \approx\!\!> \{\!|Q|\!\}$, hence by Theorem 7.2 one can prove $c \mid \mathsf{skip} : \{\!|P|\!\} \approx\!\!> \{\!|Q|\!\}$ in RHL+. Furthermore, any proof in HL+ gives rise to a proof in RHL+ with the same structure and intermediate assertions. This is because every rule in HL+, when translated to the representation $c \mid \mathsf{skip} : \{\!|P|\!\} \approx\!\!> \{\!|Q|\!\}$, is a derivable rule in RHL+. Indeed, they are instances of the more general one-sided rules rAsgnSkip, rIfSkip, etc. in Figure 2 and Figure 3.

Having completed the detour we return to the reduction of relational reasoning to unary. To this end we need to interpret judgments of the forms $c \mid \mathsf{skip} : \mathcal{P} \approx\!\!> \mathcal{Q}$ and $\mathsf{skip} \mid c' : \mathcal{Q} \approx\!\!> \mathcal{R}$ —which are one-sided in terms of the code, but which involve general relational formulas— as unary judgments. Such an interpretation is easier to formulate in terms of syntactic formulas. By renaming the variables of c and c' to be disjoint, and renaming the variables of the relational formulas accordingly, one can consider that $\mathcal{Q}$ simply *is* a unary formula.[20]

[20]Renaming can be minimized by assuming at the outset that the considered programs c, c' are acting on separable parts of the store [BDR04, BDR11], but some complications are inevitable for programs acting on the heap [Nau06, Ber11].

In the present context, using shallow embedding of relations and assertions, let us postulate the following: Any store relation $\mathcal{Q}$ has an encoding ${}^\bullet\mathcal{Q}$ as a predicate on variable stores where the variables on which the considered program acts are "on the left", and an encoding $\mathcal{Q}^\bullet$ where those variables are "on the right".[21] That is, the encodings satisfy the following, for all $c, \mathcal{P}, \mathcal{Q}$.

$$\begin{array}{lll} \models c \mid \mathsf{skip} : \mathcal{P} \approx\!\!> \mathcal{Q} & \text{iff} & \models c : {}^\bullet\mathcal{P} \rightsquigarrow {}^\bullet\mathcal{Q} \\ \models \mathsf{skip} \mid c : \mathcal{P} \approx\!\!> \mathcal{Q} & \text{iff} & \models c : \mathcal{P}^\bullet \rightsquigarrow \mathcal{Q}^\bullet \end{array} \tag{7.10}$$

To put these tedious details to work, we need the following which can be proved straightforwardly using semantic weakest preconditions or strongest postconditions. For later reference we introduce notation for weakest preconditions of program pairs: $\mathsf{wp}(c \mid c')(\mathcal{R}) \mathrel{\hat{=}} \{(s,s') \mid \forall t,t'.\, [\![c]\!]\, s\, t \wedge [\![c']\!]\, s'\, t' \Rightarrow (t,t') \in \mathcal{R}\}$. Note that $\models c \mid c' : \mathcal{P} \approx\!\!> \mathcal{R}$ iff $\mathcal{P} \Rightarrow \mathsf{wp}(c \mid c')(\mathcal{R})$.

Lemma 7.3. For any $c, c', \mathcal{P}$ and $\mathcal{R}$, the following are equivalent:
(1) $\models c \mid c' : \mathcal{P} \approx\!\!> \mathcal{R}$
(2) There is a $\mathcal{Q}$ such that $\models c \mid \mathsf{skip} : \mathcal{P} \approx\!\!> \mathcal{Q}$ and $\models \mathsf{skip} \mid c' : \mathcal{Q} \approx\!\!> \mathcal{R}$.
(3) There is a $\mathcal{Q}$ such that $\models \mathsf{skip} \mid c' : \mathcal{P} \approx\!\!> \mathcal{Q}$ and $\models c \mid \mathsf{skip} : \mathcal{Q} \approx\!\!> \mathcal{R}$.

A direct consequence is the soundness of the following rules, which are analogous to rLRseq and rRLseq.

$$\textsc{uLRseq}\ \frac{c : {}^\bullet\mathcal{P} \rightsquigarrow {}^\bullet\mathcal{Q} \qquad c' : \mathcal{Q}^\bullet \rightsquigarrow \mathcal{R}^\bullet}{c \mid c' : \mathcal{P} \approx\!\!> \mathcal{R}} \qquad \textsc{uRLseq}\ \frac{c' : \mathcal{P}^\bullet \rightsquigarrow \mathcal{Q}^\bullet \qquad c : {}^\bullet\mathcal{Q} \rightsquigarrow {}^\bullet\mathcal{R}}{c \mid c' : \mathcal{P} \approx\!\!> \mathcal{R}}$$

Now we obtain a completeness result akin to those in the literature: rule uLRseq, together with HL, comprises a Cook complete logic for relational judgments. The proof is as follows. Suppose $\models c \mid c' : \mathcal{P} \approx\!\!> \mathcal{R}$ is valid. By Lemma 7.3, there is a $\mathcal{Q}$ such that $\models c \mid \mathsf{skip} : \mathcal{P} \approx\!\!> \mathcal{Q}$ and $\models \mathsf{skip} \mid c' : \mathcal{Q} \approx\!\!> \mathcal{R}$. By (7.10) we have $\models c : {}^\bullet\mathcal{P} \rightsquigarrow {}^\bullet\mathcal{Q}$ and $\models c' : \mathcal{Q}^\bullet \rightsquigarrow \mathcal{R}^\bullet$. By completeness of HL these judgments are provable. Application of rule rLRseq proves $c \mid c' : \mathcal{P} \approx\!\!> \mathcal{R}$. A symmetric proof establishes Cook completeness of uRLseq plus HL.

In accord with the detour about embedding HL in RHL+, the preceding considerations lead to an alternate proof of Theorem 7.2 along the following lines. First show that RHL+ is complete for one-sided judgments (since HL is). Second, the rules rSeq and rRewrite suffice to derive rRLseq (in fact using only command equivalences of the form $c; \mathsf{skip} \simeq c$ and $\mathsf{skip}; c \simeq c$). Finally, proceed by an argument similar to the preceding paragraph.

8. The ∀∃ logic ERHL+

In this section, we consider a standalone deductive system for the $\overset{\exists}{\approx\!\!>}$ judgment, called ERHL+, which involves only the $\overset{\exists}{\approx\!\!>}$ judgment together with assertion validity and command equivalence $\simeq$ just like RHL+. Unary correctness is subsumed because $c : P \rightsquigarrow Q$ is valid iff

[21] Such encodings are slightly tricky to formalize in our setting with stores as total maps on all variables. We sketch the idea in terms of a simpler setting where stores are finite maps, written like $\{x{:}1, y{:}7, z{:}12\}$. Suppose that for the relevant variables for the considered programs and specs are $x, y, \ldots$. Assume some bijection to a disjoint set of variables $\hat{x}, \hat{y}, \ldots$. So a pair of stores, say $(\{x{:}1, y{:}2, z{:}3\}, \{x{:}4, y{:}5, z{:}6\})$ has encodings ${}^\bullet(\{x{:}1, y{:}2, z{:}3\}, \{x{:}4, y{:}5, z{:}6\}) \mathrel{\hat{=}} \{x{:}1, y{:}2, z{:}3, \hat{x}{:}4, \hat{y}{:}5, \hat{z}{:}6\}$ and $(\{x{:}1, y{:}2, z{:}3\}, \{x{:}4, y{:}5, z{:}6\})^\bullet \mathrel{\hat{=}} \{\hat{x}{:}1, \hat{y}{:}2, \hat{z}{:}3, x{:}4, y{:}5, z{:}6\}$. Then ${}^\bullet\mathcal{P} \mathrel{\hat{=}} \{{}^\bullet(s,t) \mid (s,t) \in \mathcal{P}\}$. To do something similar for total map stores, one should restrict to finitely supported relations $\mathcal{P}$ so the definitions of ${}^\bullet\mathcal{P}$ and $\mathcal{P}^\bullet$ can exploit unused variables.

$$\textsc{eSkipHav}\quad \mathsf{skip} \mid \mathsf{hav}\ x : (\dot{\exists}|x.\ \mathcal{P}) \overset{\exists}{\approx\!>} \mathcal{P} \qquad \textsc{eHavSkip}\quad \mathsf{hav}\ x \mid \mathsf{skip} : (\dot{\forall}x|.\ \mathcal{P}) \overset{\exists}{\approx\!>} \mathcal{P}$$

$$\textsc{eDo}\quad \frac{\begin{array}{l} c \mid \mathsf{skip} : \mathcal{Q} \wedge \{\!|e|\!\} \wedge \mathcal{L} \overset{\exists}{\approx\!>} \mathcal{Q} \quad \text{for all } e \to c \text{ in } gcs \\ \mathsf{skip} \mid c' : \mathcal{Q} \wedge |\!\{e'\}\!| \wedge \mathcal{R} \wedge V = v \overset{\exists}{\approx\!>} \mathcal{Q} \wedge V \prec v \quad \text{for all } e' \to c' \text{ in } gcs' \text{ and all } v \in D \\ c \mid c' : \mathcal{Q} \wedge \{\!|e|\!\} \wedge |\!\{e'\}\!| \wedge \neg\mathcal{L} \wedge \neg\mathcal{R} \overset{\exists}{\approx\!>} \mathcal{Q} \quad \text{for all } e \to c \text{ in } gcs \text{ and } e' \to c' \text{ in } gcs' \\ \mathcal{Q} \Rightarrow (\{\!|\mathsf{enab}(gcs)|\!\} = |\!\{\mathsf{enab}(gcs')\}\!| \vee (\mathcal{L} \wedge \{\!|\mathsf{enab}(gcs)|\!\}) \vee (\mathcal{R} \wedge |\!\{\mathsf{enab}(gcs')\}\!|)) \end{array}}{\mathsf{do}\ gcs\ \mathsf{od} \mid \mathsf{do}\ gcs'\ \mathsf{od} : \mathcal{Q} \overset{\exists}{\approx\!>} \mathcal{Q} \wedge \neg\{\!|\mathsf{enab}(gcs)|\!\} \wedge \neg|\!\{\mathsf{enab}(gcs')\}\!|}$$

Figure 15: Rules of ERHL+. In eDo, $(D, \prec)$ is well-ordered and V is a total function $(\mathsf{Var} \to \mathbb{Z}) \times (\mathsf{Var} \to \mathbb{Z}) \to D$. Rules eRewrite, eGhost, eSkip, eSkipAss, eAssSkip, eSeq, eIf, eConseq, eDisj, and eFalse are the same as those with corresponding names in Figure 2 but for $\overset{\exists}{\approx\!>}$.

$c \mid \mathsf{skip} : \{\!|P|\!\} \overset{\exists}{\approx\!>} \{\!|Q|\!\}$ is valid. On the other side, $\mathsf{skip} \mid c : |\!\{P\}\!| \overset{\exists}{\approx\!>} |\!\{Q\}\!|$ is valid iff the ***forward underapproximation*** judgment $c : P \overset{\exists}{\rightsquigarrow} Q$ is valid. This is defined by

$$\models c : P \overset{\exists}{\rightsquigarrow} Q \text{ iff for any } s \in P \text{ there exists } t \in Q \text{ such that } [\![c]\!]\, s\, t. \tag{8.1}$$

This has been called *possible correctness* [Hoa78] and more recently the *existential* Hoare triple of [DYZD22] and the *sufficient incorrectness* triple of [ABGL24].[22]

The rules of ERHL+ appear in Figure 15. Most have the same form as corresponding rules in Figure 2, although the soundness proofs are different in detail. If d is deterministic then $\models c \mid d : \mathcal{P} \overset{\exists}{\approx\!>} \mathcal{Q}$ implies $\models c \mid d : \mathcal{P} \approx\!> \mathcal{Q}$. If d denotes a domain-total[23] relation then $\models c \mid d : \mathcal{P} \approx\!> \mathcal{Q}$ implies $\models c \mid d : \mathcal{P} \overset{\exists}{\approx\!>} \mathcal{Q}$. This hints at why some proof rules for $\overset{\exists}{\approx\!>}$ are the same as those in Figure 2 for the $\approx\!>$ judgment.

The rule eSkipHav reflects the $\forall\exists$ nature of the $\overset{\exists}{\approx\!>}$ judgment, existentially quantifying x on the right state whereas eHavSkip universally quantifies it on the left. Note that eIf and eDo do *not* existentially quantify over guarded commands on the right, as one might guess at first. This would be unsound, because property (3.2) universally quantifies over all pairs of initial states; we return to this later. A distinguishing feature of the $\overset{\exists}{\approx\!>}$ judgment is that it does not validate the rule of conjunction which is sound for the $\forall\forall$ judgment. For example, both $\mathsf{skip} \mid \mathsf{hav}\ x : \mathit{true} \overset{\exists}{\approx\!>} |\!\{x < 0\}\!|$ and $\mathsf{skip} \mid \mathsf{hav}\ x : \mathit{true} \overset{\exists}{\approx\!>} |\!\{x > 0\}\!|$ are valid but not $\mathsf{skip} \mid \mathsf{hav}\ x : \mathit{true} \overset{\exists}{\approx\!>} |\!\{x < 0 \wedge x > 0\}\!|$.

Besides eSkipHav, another noticeable difference from RHL+ is eDo. Like the rule rDo in RHL+, eDo captures a conditional alignment of iterations directed by the relations $\mathcal{L}$ and $\mathcal{R}$. The side condition connects the invariant $\mathcal{Q}$ with $\mathcal{L}$ and $\mathcal{R}$ in a way that ensures adequacy in the sense of covering all iterations. The rule differs from rDo in its treatment of right-only iterations. The $\forall\exists$ judgment requires termination on the right, and the rule

[22] O'Hearn's incorrectness judgment [O'H20] —backwards underapproximation— is similarly obtained by a backwards version of (3.2), as noted in [AKL+23].

[23] Domain-totality means the command may terminate from any state, not that it must. For example, if $\mathit{true} \to \mathit{diverge}$ [] $\mathit{true} \to \mathsf{skip}$ fi is domain-total.

$$\textsc{eHavHav}\quad \mathsf{hav}\ x \mid \mathsf{hav}\ x' : (\dot{\forall} x|.\ (\dot{\exists} |x'.\ \mathcal{P})) \overset{\exists}{\approx\!>} \mathcal{P} \qquad \textsc{eAsgnAsgn}\quad x := e \mid x' := e' : \mathcal{R}^{x|x'}_{e|e'} \overset{\exists}{\approx\!>} \mathcal{R}$$

$$\textsc{eSkipDo}\quad \frac{\mathsf{skip} \mid c : \mathcal{Q} \wedge \llparenthesis e \rrparenthesis \wedge V = v \overset{\exists}{\approx\!>} \mathcal{Q} \wedge V \prec v \quad \text{for all } e \to c \text{ in } gcs \text{ and all } v \in D}{\mathsf{skip} \mid \mathsf{do}\ gcs\ \mathsf{od} : \mathcal{Q} \overset{\exists}{\approx\!>} \mathcal{Q} \wedge \neg \llparenthesis \mathsf{enab}(gcs) \rrparenthesis}$$

Figure 16: Some derived rules of ERHL+. Rules eLRseq, eRLseq, eAlgnIf, eSeqSkip, eIfSkip, eDoSkip, and eDisjN are the same as those with corresponding names in Figure 3 but for $\overset{\exists}{\approx\!>}$.

relies on the standard approach of showing a variant decreases.[24] The variant V maps pairs of stores to some well-ordered set D. In accord with the shallow embedding of relational assertions, for any value v in D we write $V = v$ to denote the relation $\{(s, s') \mid V(s, s') = v\}$. Likewise for $V \prec v$.

Formally the premise for $\mathsf{skip} \mid c'$ in eDo is a D-indexed set of premises, but this is just an artifact of shallow embedding. Think of v as a logical variable in a single premise, universally quantified over the judgment.

Some derived rules are in Figure 16. Rule eHavHav is proposed in [AKL+23]. It expresses that $\mathcal{P}$ in the initial state must be total as a relation from x on the left to x' on the right. It can be derived as follows. Instantiate eHavSkip as $\mathsf{hav}\ x \mid \mathsf{skip} : (\dot{\forall} x|.\ (\dot{\exists} |x'.\ \mathcal{P})) \overset{\exists}{\approx\!>} \dot{\exists} |x'.\ \mathcal{P}$. Then use eSeq with $\mathsf{skip} \mid \mathsf{hav}\ x : \dot{\exists} |x.\ P \overset{\exists}{\approx\!>} P$ (from eSkipHav) to get

$$\mathsf{hav}\ x; \mathsf{skip} \mid \mathsf{skip}; \mathsf{hav}\ x' : (\dot{\forall} x|.\ (\dot{\exists} |x'.\ \mathcal{P})) \overset{\exists}{\approx\!>} \mathcal{P}$$

Now obtain eHavHav by eRewrite using skip unit laws.

Aside from eSkipDo, the other derived rules in Figure 16 are like those in Figure 3 and can be derived using eRewrite. For instance, eAsgnAsgn can be derived using eSkipAsgn, eAsgnSkip, eSeq and eRewrite. The $\forall\exists$ versions of rLRseq and rRLseq are both derivable, but eRLseq is less useful than eLRseq because the premises of eRLseq amount to a strong $\exists\forall$ property. In subsection 9.3 we show that eLRseq is the basis of a Cook complete logic whereas eRLseq is not.

Rule eSkipDo is derived as follows. Observe that $\mathsf{skip} \simeq \mathsf{do}\ \mathit{false} \to \mathsf{skip}\ \mathsf{od}$. So we get the conclusion of eSkipDo by eRewrite from $\mathsf{do}\ \mathit{false} \to \mathsf{skip}\ \mathsf{od} \mid \mathsf{do}\ gcs\ \mathsf{od} : \mathcal{Q} \overset{\exists}{\approx\!>} \mathcal{Q} \wedge \neg \llparenthesis \mathsf{enab}(gcs) \rrparenthesis$. This we prove using eDo with $\mathcal{L}, \mathcal{R} := \mathit{false}, \mathit{true}$. The right-only premises have the form $\mathsf{skip} \mid c : \mathcal{Q} \wedge \llparenthesis e \rrparenthesis \wedge \mathit{true} \wedge (V = d) \overset{\exists}{\approx\!>} \mathcal{Q} \wedge (V \prec d)$. They follow from the premises of eSkipDo using eConseq to add the conjunct *true*. The left-only and joint premises are proved by eFalse. The side condition of this instantiation of eDo is $\mathcal{Q} \Rightarrow (\{\!|\mathit{false}|\!\} = \llparenthesis \mathsf{enab}(gcs) \rrparenthesis) \vee \llparenthesis \mathsf{enab}(gcs) \rrparenthesis$ and the consequent simplifies to true.

Example 8.1. Recall the example adapted from Unno et al. [UTK21], described on page 6 in section 2. The judgment $c3 \mid c3 : \mathbb{A} \mathit{low} \overset{\exists}{\approx\!>} \mathbb{A} x$ specifies possibilistic noninterference. We construct a deductive proof in ERHL+ following heuristics mentioned in section 2.

[24] As an alternative to universally quantifying over $v \in D$ in the metalanguage, it is possible to formulate rule eDo using a fresh program variable to snapshot the initial value of V, with side condition $\mathcal{Q} \Rightarrow V \geq 0$. This is sufficient because for completeness it suffices to take D to be $\mathbb{N}$; this is shown in the proof of Theorem 9.7.

The proof is presented in a goal directed style. It starts by an application of eIf, yielding four obligations, corresponding to the four combinations of the guards $high \neq 0$ and $high = 0$.

(1) $\mathsf{hav}^2\ x; \mathsf{if}^3\ \ldots\ \mathsf{fi} \mid \mathsf{hav}^2\ x; \mathsf{if}^3\ \ldots\ \mathsf{fi} : \mathbb{A}low \wedge \langle\!|high \neq 0 \mid high \neq 0|\!\rangle \overset{\exists}{\approx\!>} \mathbb{A}x$.
We prove this using a lockstep alignment. Rule eSeq is instantiated with $\mathbb{A}low \wedge \mathbb{A}x$ at the intermediate point. The judgment $\mathsf{hav}^2\ x \mid \mathsf{hav}^2\ x : \mathbb{A}low \wedge \langle\!|high \neq 0 \mid high \neq 0|\!\rangle \overset{\exists}{\approx\!>} \mathbb{A}low \wedge \mathbb{A}x$ is obtained using eHavHav and then eConseq with the valid implication $\mathbb{A}low \Rightarrow (\dot{\forall} x|.\ (\dot{\exists}|x.\ \mathbb{A}low \wedge \mathbb{A}x))$. The postcondition $\mathbb{A}x$ is chosen to serve as filtering condition. The judgment $\mathsf{if}^3\ \ldots\ \mathsf{fi} \mid \mathsf{if}^3\ \ldots\ \mathsf{fi} : \mathbb{A}low \wedge \mathbb{A}x \overset{\exists}{\approx\!>} \mathbb{A}x$ is proved using the lockstep rules eAlgnIf and eAlgnDo with loop invariant $\mathbb{A}x$.

(2) $x :=^7 low; \mathsf{hav}^8\ b; \mathsf{do}^9\ b \neq 0 \to x :=^{11} x + 1 \ldots\ \mathsf{od} \mid x :=^7 low; \mathsf{hav}^8\ b; \mathsf{do}^9\ b \neq 0 \to x :=^{11} x + 1 \ldots\ \mathsf{od} : \mathbb{A}low \wedge \langle\!|high = 0 \mid high = 0|\!\rangle \overset{\exists}{\approx\!>} \mathbb{A}x$
This judgment, too, is proved using a lockstep alignment. Key to the proof is having $\mathbb{A}x \wedge \mathbb{A}b$ as the relational invariant for the two loops at control point 9. In this derivation eHavHav is applied twice with a post-relation that includes $\mathbb{A}b$ (again, for filtering).

(3) $\mathsf{hav}^2\ x; \mathsf{if}^3\ \ldots\ \mathsf{fi} \mid x :=^7 low; \ldots : \mathbb{A}low \wedge \langle\!|high \neq 0 \mid high = 0|\!\rangle \overset{\exists}{\approx\!>} \mathbb{A}x$.
We prove this using the left-first sequential alignment rule eLRseq with intermediate assertion $\mathbb{A}low \wedge \langle\!|x \geq low|\!\rangle$, for which the premises are:
   (a) $\mathsf{hav}^2\ x; \mathsf{if}^3\ \ldots\ \mathsf{fi} \mid \mathsf{skip} : \mathbb{A}low \wedge \langle\!|high \neq 0 \mid high = 0|\!\rangle \overset{\exists}{\approx\!>} \mathbb{A}low \wedge \langle\!|x \geq low|\!\rangle$
   This judgment is derived using the left-side rules eSeqSkip, eHavSkip, eIfSkip, and eDoSkip with invariant $\mathbb{A}low \wedge \langle\!|x < low|\!\rangle$.
   (b) $\mathsf{skip} \mid x :=^7 low; \mathsf{hav}^8\ b; \mathsf{do}^{10}\ b \neq 0 \to \ldots\ \mathsf{od} : \mathbb{A}low \wedge \langle\!|x \geq low|\!\rangle \overset{\exists}{\approx\!>} \mathbb{A}x$
   This is proved using eSkipSeq twice, to compose the following three judgments. First in the sequence is $\mathsf{skip} \mid x :=^7 low : \mathbb{A}low \wedge \langle\!|x \geq low|\!\rangle \overset{\exists}{\approx\!>} \mathbb{A}low \wedge \langle\!|x \geq low \mid x = low|\!\rangle$, proved using eSkipAssg and eConseq. Second is $\mathsf{skip} \mid \mathsf{hav}^8\ b : \mathbb{A}low \wedge \langle\!|x \geq low \mid x = low|\!\rangle \overset{\exists}{\approx\!>} |\!|b \geq 0|\!\rangle \wedge (|\!|b|\!| = \langle\!|x|\!\rangle - |\!|x|\!|)$, proved by eSkipHav and eConseq, noting that the precondition implies $(\dot{\exists}|b.\ |\!|b \geq 0|\!\rangle \wedge (|\!|b|\!| = \langle\!|x|\!\rangle - |\!|x|\!|))$. Third is $\mathsf{skip} \mid \mathsf{do}^9\ b \neq 0 \to x :=^{10} x + 1; \mathsf{hav}^{11}\ b\ \mathsf{od} : |\!|b \geq 0|\!\rangle \wedge (|\!|b|\!| = \langle\!|x|\!\rangle - |\!|x|\!|) \overset{\exists}{\approx\!>} \mathbb{A}x$, proved using eSkipDo with invariant $\mathcal{Q} := |\!|b \geq 0|\!\rangle \wedge (|\!|b|\!| = \langle\!|x|\!\rangle - |\!|x|\!|)$ and variant $V := abs(|\!|b|\!|)$. The loop body is proved using eSkipSeq, eSkipAssg, and eSkipHav; eSkipHav is instantiated with post-relation $|\!|b|\!| = \langle\!|x|\!\rangle - |\!|x|\!|$.

(4) $x :=^7 low; \mathsf{hav}^8\ b; \mathsf{do}^9\ \ldots\ \mathsf{od} \mid \mathsf{hav}^2\ x; \mathsf{if}^3\ \ldots\ \mathsf{fi} : \mathbb{A}low \wedge \langle\!|high = 0 \mid high \neq 0|\!\rangle \overset{\exists}{\approx\!>} \mathbb{A}x$.
We prove this by left-first sequential alignment again, instantiating eLRseq with intermediate assertion $\mathbb{A}low \wedge \langle\!|x \geq low|\!\rangle$.

Theorem 8.2. All the rules of ERHL+ (Figure 15) are sound.

Proof. Soundness proofs of most ERHL+ rules are straightforward. Soundness of eRewrite holds for the same reason Rewrite and rRewrite are sound: the property (3.2) is about program semantics which is preserved by $\simeq$. The soundness proof for eGhost relies on the fact that the ghost condition ensures erasure of a ghost variable does not influence termination.

For eIf, assume we have the premises: $c \mid c' : \mathcal{R} \wedge \langle\!|e|\!\rangle \wedge |\!|e'|\!\rangle \approx\!> \mathcal{S}$ for all $e \to c$ in gcs and $e' \to c'$ in gcs'. To show the conclusion $\mathsf{if}\ gcs\ \mathsf{fi} \mid \mathsf{if}\ gcs'\ \mathsf{fi} : \mathcal{R} \approx\!> \mathcal{S}$, consider states s, s', t such that $(s, s') \in \mathcal{R}$ and $[\![\,\mathsf{if}\ gcs\ \mathsf{fi}\,]\!]\, s\, t$. We must show there is t' such that $[\![\,\mathsf{if}\ gcs'\ \mathsf{fi}\,]\!]\, s'\, t'$ and $(t, t') \in \mathcal{S}$. By semantics there is $e \to c$ in gcs such that $s \in [\![\, e \,]\!]$ and $[\![\, c \,]\!]\, s\, t$. By the total-if

condition for $\mathsf{if}\ gcs'\ \mathsf{fi}$, there is some $e' \to c'$ in gcs' such that $s' \in [\![e']\!]$. By the premise $c \mid c' : \mathcal{R} \wedge \langle\!| e |\!\rangle \wedge |\!\langle e' \rangle\!| \approx\!\!> \mathcal{S}$ there is some t' with $[\![c']\!]\, s'\, t'$ and $(t, t') \in \mathcal{S}$.

For EDo we sketch the argument for loops that have a single guarded command; a detailed proof is given in [NBN23]. The goal is to show $\models \mathsf{do}\ e \to c\ \mathsf{od} \mid \mathsf{do}\ e' \to c'\ \mathsf{od} : \mathcal{Q} \overset{\exists}{\approx\!\!>} \mathcal{Q} \wedge \neg\langle\!| e |\!\rangle \wedge \neg |\!\langle e' \rangle\!|$ given the side condition $\mathcal{Q} \Rightarrow (\langle\!| e |\!\rangle = |\!\langle e' \rangle\!| \vee (\mathcal{L} \wedge \langle\!| e |\!\rangle \vee (\mathcal{R} \wedge |\!\langle e' \rangle\!|))$ and premises $\models c \mid \mathsf{skip} : \mathcal{Q} \wedge \langle\!| e |\!\rangle \wedge \mathcal{L} \overset{\exists}{\approx\!\!>} \mathcal{Q}$, $\models \mathsf{skip} \mid c' : \mathcal{Q} \wedge |\!\langle e' \rangle\!| \wedge \mathcal{R} \wedge (V = v) \overset{\exists}{\approx\!\!>} \mathcal{Q} \wedge (V \prec v)$ (for all v), and $\models c \mid c' : \mathcal{Q} \wedge \langle\!| e |\!\rangle \wedge |\!\langle e' \rangle\!| \wedge \neg\mathcal{L} \wedge \neg\mathcal{R} \overset{\exists}{\approx\!\!>} \mathcal{Q}$. Consider states s, s', t such that $(s, s') \in \mathcal{Q}$ and $[\![\mathsf{do}\ e \to c\ \mathsf{od}]\!]\, s\, t$. We must show there exists a t' such that $[\![\mathsf{do}\ e' \to c'\ \mathsf{od}]\!]\, s'\, t'$ and $(t, t') \in \mathcal{Q} \wedge \neg\langle\!| e |\!\rangle \wedge \neg |\!\langle e' \rangle\!|$. We proceed by rule induction on $[\![\mathsf{do}\ e \to c\ \mathsf{od}]\!]\, s\, t$, keeping s' arbitrary. In the base case, the run is already terminated: $s \in \neg\langle\!| e |\!\rangle$ and $s = t$. Existence of the required t' is proved by well-founded induction on $V(s, s')$, as follows. If $s' \in \neg |\!\langle e' \rangle\!|$, we are done by letting $t' := s'$. Otherwise, using $(s, s') \in \mathcal{Q}$, $s \in \neg\langle\!| e |\!\rangle$, and the side condition we get the precondition of the right-side premise for c'. Applying that premise yields some t'' with $(s, t'') \in \mathcal{Q}$ and $V(s, t'') \prec V(s, s')$ and then the inner induction hypothesis yields the required t'.

In the inductive case, we have $s \in \langle\!| e |\!\rangle$, $[\![c]\!]\, s\, u$, and $[\![\mathsf{do}\ e \to c\ \mathsf{od}]\!]\, u\, t$ for some u. The inductive hypothesis says that for any u', if $(u, u') \in \mathcal{Q}$, then there is a t' such that $[\![\mathsf{do}\ e' \to c'\ \mathsf{od}]\!] u' t'$ and $(t, t') \in \mathcal{Q} \wedge \neg\langle\!| e |\!\rangle \wedge \neg |\!\langle e' \rangle\!|$. If (s, s') satisfy the precondition for the left-only or joint premise then applying the premise yields t' (reached from s' by zero or one iterations of c') such that $(u, t') \in \mathcal{Q}$ whence the inductive hypothesis yields our goal. Otherwise, by $(s, s') \in \mathcal{Q}$ and the side condition, (s, s') satisfies the precondition for the right-only premise. But that does not immediately enable use of the inductive hypothesis. So we show by well founded induction on $V(s, s')$ that there exists some t'', reached by some number of iterations of c', with $(s, t'') \in \mathcal{Q}$ and (s, t'') satisfies either the left-only or joint premise. Either of those premises, together with the main induction hypothesis and reachability of t'', yields the goal. □

8.1. **Digression on control determinacy.** Our GCL has two forms of nondeterminacy. The language has havoc, which makes an unboundedly nondeterministic choice of a value. This can serve to model randomization as well as input data. It also has nondeterminacy in terms of control. This does not necessarily lead to nondeterministic outcomes. For example, the command $\mathsf{if}^1\ true \to x :=^2 y \,[]\, true \to x :=^3 y\ \mathsf{fi}$ satisfies $\langle\!| y |\!\rangle = |\!\langle y \rangle\!| \approx\!\!> \langle\!| x |\!\rangle = |\!\langle x \rangle\!|$ but its control flows nondeterministically to either the point labelled 2 or the point labelled 3. On the other hand, $d3'$ below has nondeterministic outcomes.

$$d3 \mathrel{\hat{=}} \mathsf{if}\ true \to x := 0\ \mathsf{fi} \qquad d3' \mathrel{\hat{=}} \mathsf{if}\ true \to x := 0 \,[]\, true \to x := 1\ \mathsf{fi}$$

A command c is called ***control deterministic*** provided that the guards are mutually exclusive, for every if- and do-command in c. A control deterministic $\mathsf{if}\ e_0 \to c_0 \,[]\, e_1 \to c_1 \,[]\, \ldots\ \mathsf{fi}$ is equivalent to the nested if-else $\mathsf{if}\ e_0\ \mathsf{then}\ c_0\ \mathsf{else}\ \mathsf{if}\ e_1\ \mathsf{then}\ c_1\ \mathsf{else}\ \ldots\ \mathsf{fi}\ \mathsf{fi}$. A control deterministic do-command can be represented similarly.

For a command $\mathsf{if}\ e_0 \to c_0 \,[]\, e_1 \to c_1 \ldots\ \mathsf{fi}$ that is not control deterministic, one can make it so in the form $\mathsf{if}\ e_0 \to c_0 \,[]\, e_1 \wedge \neg e_0 \to c_1 \ldots\ \mathsf{fi}$ but this may eliminate some behaviors, as is the case for $d3'$ above. To retain all behaviors one can use havoc with an extra variable that serves to prophesize the choice, as in

$$d3'' \mathrel{\hat{=}} \mathsf{hav}\ z; \mathsf{if}\ true \wedge z = 0 \to x := 0 \,[]\, true \wedge z \neq 0 \to x := 1\ \mathsf{fi}$$

where z is fresh. Conventional if/else and while commands are control deterministic, and the normal form construction of Definition 5.12 preserves control determinacy. So there is little reason to dwell on programs that are not control deterministic. Nonetheless we briefly consider the following limitation of ERHL+ for such programs.

This $\forall\exists$ judgment is valid: $d3 \mid d3' : \mathit{true} \stackrel{\exists}{\approx\!>} \mathbb{A}x$. But rule eIf is not directly applicable, because one of the premises would be $x := 0 \mid x := 1 : \mathit{true} \wedge \mathit{true} \wedge \mathit{true} \stackrel{\exists}{\approx\!>} \mathbb{A}x$ which is false. One might guess to change the rule simply by existentially quantifying the right side guarded commands but this is unsound. For example, consider

$$d4 \mathrel{\hat{=}} \mathsf{if}\ \mathit{true} \to x := 0\ \mathsf{fi} \qquad d4' \mathrel{\hat{=}} \mathsf{if}\ x \geq 0 \to x := 0 \,[]\, x \leq 0 \to x := 1\ \mathsf{fi}.$$

For every guarded command on the left, there is one on the right that relates according to the spec $\mathit{true} \wedge \llparenthesis e \rrparenthesis \stackrel{\exists}{\approx\!>} \mathbb{A}x$ where e is the guard. In particular: $x := 0 | x := 0 : \mathit{true} \wedge \llparenthesis x \geq 0 \rrparenthesis \stackrel{\exists}{\approx\!>} \mathbb{A}x$. But it is not the case that $d4 \mid d4'$ satisfies $\mathit{true} \stackrel{\exists}{\approx\!>} \mathbb{A}x$. What is needed is to existentially quantify a subset of guarded commands on the right that covers all cases.

$$\frac{\begin{array}{c}\text{For all } e \to c \text{ in } \mathit{gcs} \text{ there is some } g \subseteq \mathit{gcs}' \text{ such that } \mathcal{R} \wedge \{\!|e|\!\} \Rightarrow \mathsf{enab}(g) \text{ and} \\ c \mid c' : \mathcal{R} \wedge \{\!|e|\!\} \wedge \llparenthesis e' \rrparenthesis \stackrel{\exists}{\approx\!>} \mathcal{S} \text{ for all } e' \to c' \text{ in } g\end{array}}{\mathsf{if}\ \mathit{gcs}\ \mathsf{fi} \mid \mathsf{if}\ \mathit{gcs}'\ \mathsf{fi} : \mathcal{R} \stackrel{\exists}{\approx\!>} \mathcal{S}}\ \textsc{eIfX}$$

The side condition $\mathcal{R} \wedge \{\!|e|\!\} \Rightarrow \mathsf{enab}(g)$ ensures the set g covers all cases. Note that there is no such set that can be used to show the invalid judgment $d4 \mid d4' : \mathit{true} \stackrel{\exists}{\approx\!>} \mathbb{A}x$. Rule eIfX does yield $d3 \mid d3' : \mathit{true} \stackrel{\exists}{\approx\!>} \mathbb{A}x$. A similar issue arises for the joint premises in rule eDo, and one can address it in the same way as eIfX. The rule eDoX replaces the joint premise of eDo with the following:

$$\begin{array}{c}\text{For all } e \to c \text{ in } \mathit{gcs} \text{ there is some } g \subseteq \mathit{gcs}' \text{ such that } \mathcal{Q} \wedge \{\!|e|\!\} \wedge \neg\mathcal{L} \wedge \neg\mathcal{R} \Rightarrow \mathsf{enab}(g) \\ \text{and for all } e' \to c' \text{ in } g \text{ we have } c \mid c' : \mathcal{Q} \wedge \{\!|e|\!\} \wedge \llparenthesis e' \rrparenthesis \wedge \neg\mathcal{L} \wedge \neg\mathcal{R} \stackrel{\exists}{\approx\!>} \mathcal{Q}\end{array}$$

We choose the simpler rules rIf and rDo to streamline the presentation, at the cost of restricting to control deterministic programs when necessary, specifically Theorem 9.10(a). We conjecture the restriction can be dropped, using rules eIfX and eDoX. There is little practical motivation because conventional control structures (if/else and while) are control deterministic.

9. Filtered alignment automata and alignment completeness of ERHL+

This section introduces a form of alignment automaton suited to $\forall\exists$ properties. The logic ERHL+ is shown, in subsection 9.2, to be alignment complete with respect to these automata.

9.1. Filtered alignment automata.

For $\forall\exists$ reasoning, the form of alignment given by Definition 4.2 is unsatisfactory: if the right program is nondeterministic, there needs to be a way to keep some but not all its transitions for right-only and joint steps of the product. An example is $c3$ in section 2, which we explore further in the sequel. First we adapt Definition 4.2 of alignment product to include an additional state relation which serves to filter product executions.

Definition 9.1. Suppose $\prod(A, A', L, R, J)$ is an alignment automata as in Definition 4.2. Let ***keep set*** K be a set of states, i.e., $K \subseteq (Ctrl \times Ctrl') \times (Sto \times Sto')$. The ***filtered alignment automaton*** $\prod(A, A', L, R, J, K)$ is $((Ctrl \times Ctrl'), (Sto \times Sto'), (init, init'), (fin, fin'), \Mapsto_K)$ where $\Mapsto_K$ is defined by: $\sigma \Mapsto_K \tau$ iff $\tau \in K$ and $\sigma \Mapsto \tau$ (with σ, τ ranging over states).

Here the un-subscripted $\Mapsto$ refers to the relation in Definition 4.2. Note that K is used in $\Mapsto_K$ to filter states that the product steps to. As a consequence, $[init \mid init'] \vee K$ is always a true-invariant of $\Mapsto_K$. Apart from this use of K, the transition relation is the same as the relation $\Mapsto$ for unfiltered alignment automata (Definition 4.2).

Example 9.2. Fig. 17 shows a filtered product for $c3$, the running example on $\forall\exists$ properties adapted from [UTK21]— see page 6 in section 2, and Figure 6. Fig. 17 depicts possible transitions of the product, given L, R, J conditions that capture the following alignment: lockstep if the two executions agree on the test $high \neq 0$ (abbreviated as h in Fig. 17); left-first sequential otherwise. This is similar to the deductive proof in Example 8.1. Automata edges are labeled with tests or commands. We write $\langle\!|c|\!\rangle$, $\langle\!|c|\!|$, and $|\!|c|\!\rangle$ to mean that c takes place on both sides, on the left, and on the right, respectively. The label f abbreviates the final control point 12. Filter K is *true* everywhere except at control points marked in the figure with green dashed boxes. For example, the step from $(2, 2)$ to $(3, 3)$ which havocs x on both sides is filtered by $\mathbb{A}x$. The step to control point $(f, 9)$ is filtered so that the value of $|\!|b|\!\rangle$ is the same as the difference between the values of x on both sides.

The figure does not depict transitions that cannot take place. In particular, vertices for control points $(f, 5)$ and $(f, 6)$ are missing. These vertices correspond to the diverging loop in the example taking place on the right. The filter $\mathbb{A}x$ on $(f, 3)$, in conjunction with invariant $[f|3] \Rightarrow \langle\!|x \geq low|\!|$ ensures that the transition from $(f, 4)$ to $(f, 5)$ which would be guarded by $|\!|x < low|\!\rangle$ cannot occur. □

Adequacy of filtered alignment automata.

Definition 9.3. Let $\prod(A, A', L, R, J, K)$ be a filtered alignment automata and let $\mathcal{Q} \subseteq (Sto \times Sto')$. The filtered automata is ***$\mathcal{Q}$-adequate in the $\forall\exists$ sense*** provided for all $(s, s') \in \mathcal{Q}$ and t with $(init, s) \mapsto^* (fin, t)$, there exists a t' such that $((init, init'), (s, s')) \Mapsto_K^* ((fin, fin'), (t, t'))$.

Lemma 9.4. Given automata A,A' and store relations $\mathcal{Q}$, $\mathcal{S}$, if $\prod(A, A', L, R, J, K)$ is $\mathcal{Q}$-adequate in the $\forall\exists$ sense and $\prod(A, A', L, R, J, K) \models \mathcal{Q} \rightsquigarrow \mathcal{S}$, then $A, A' \models \mathcal{Q} \overset{\exists}{\approx\!>} \mathcal{S}$.

Proof. To prove $A, A' \models \mathcal{Q} \overset{\exists}{\approx\!>} \mathcal{S}$, consider any s, s', t such that $(init, s) \mapsto^* (fin, t)$ and $(s, s') \in \mathcal{Q}$. By $\mathcal{Q}$-adequacy there is a state t' such that $((init, init'), (s, s')) \Mapsto_K^* ((fin, fin'), (t, t'))$. By taking the right projection of this trace, and destuttering, we obtain a trace $(init', s') \mapsto'^* (fin', t')$ of A'. (Stuttering steps can arise from left-only steps of the product.) Now from $\prod(A, A', L, R, J, K) \models \mathcal{Q} \rightsquigarrow \mathcal{S}$ we have $(t, t') \in \mathcal{S}$. □

The lemma suggests an approach for verifying $\forall\exists$ properties: to show $A, A' \models \mathcal{Q} \overset{\exists}{\approx\!>} \mathcal{S}$, construct a filtered product, prove it is adequate and satisfies the partial correctness spec $\mathcal{Q} \rightsquigarrow \mathcal{S}$. The latter amounts to proving a $\forall\forall$ property of the set of trace pairs represented by the product. As in the $\forall\forall$ setting, a proof method needs to connect adequacy with annotations and alignment conditions, in a way that can be checked modularly. To this end, we state the key definition and then explain its elements.

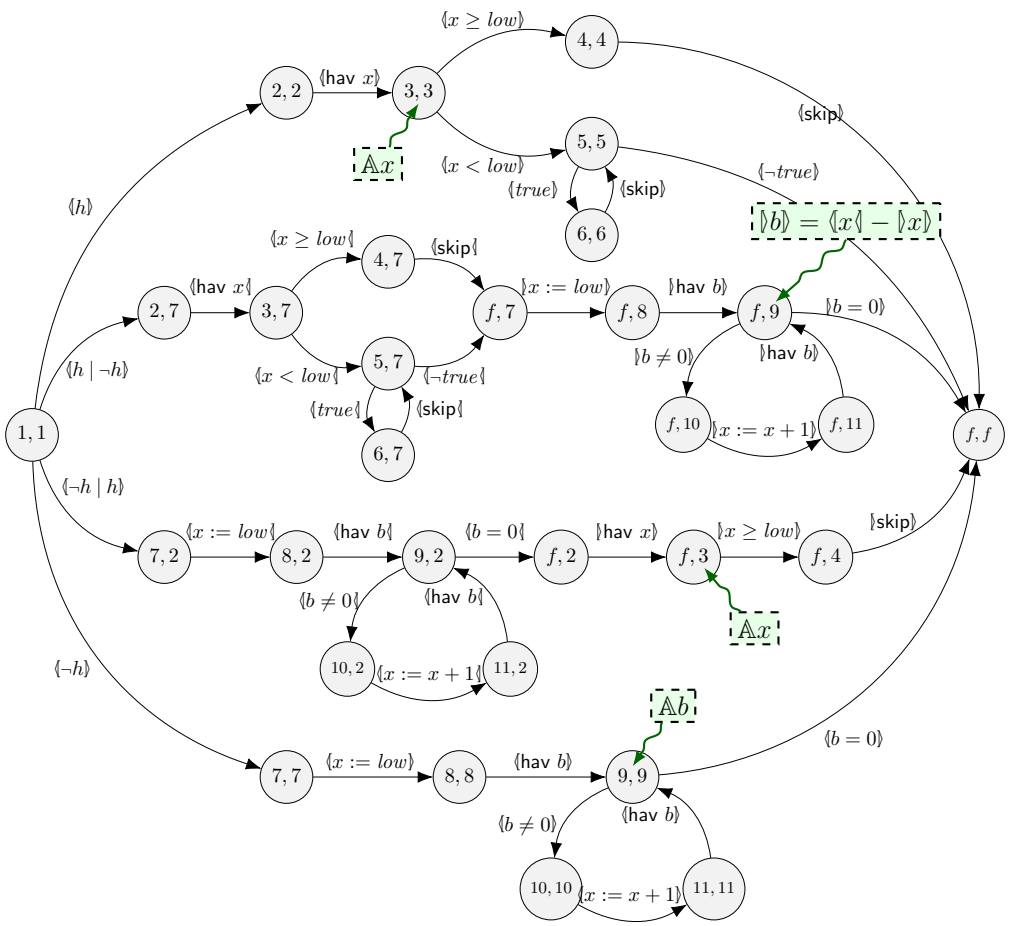

Figure 17: Filtered product for example $c3$. Non-trivial filter conditions are shown in green dashed boxes; the filter is *true* at all other control points. Notation $\langle\!\langle e \rangle\!\rangle$ abbreviates $\langle\!\langle e \mid e \rangle\!\rangle$, and we use similar notation for commands.

Definition 9.5. Given $\prod(A, A', L, R, J, K)$ with annotation an, we say the automaton has ***adequate filtering for*** an iff there exists a function $V : (Ctrl \times Ctrl') \times (Sto \times Sto') \to D$ where $(D, \prec)$ is a well-ordered set, such that the following four conditions hold. (Where $\mapsto, \mapsto'$ are the transition relations of A, A'.)

***Left-permissive*:** For any n, n', s, s', m, t, if $(s, s') \in an(n, n')$ and $((n, n'), (s, s')) \in L$ and $(n, s) \mapsto (m, t)$ then $((m, n'), (t, s')) \in K$.

***Joint-productive*:** For any n, n', s, s', m, t, if $(s, s') \in an(n, n')$ and $((n, n'), (s, s')) \in J$ and $(n, s) \mapsto (m, t)$ then there are m', t' such that $(n', s') \mapsto' (m', t')$ and $((m, m'), (t, t')) \in K$.

***Right-productive*:** For any n, n', s, s', if $(s, s') \in an(n, n')$ and $((n, n'), (s, s')) \in R$ then there are m', t' such that $(n', s') \mapsto' (m', t')$ and $((n, m'), (s, t')) \in K$ and $V((n, m'), (s, t')) \prec V((n, n'), (s, s'))$.

***Enabled*:** For any n, n', the implication $\breve{an}(n, n') \Rightarrow L \vee R \vee J \vee [fin|fin']$ is valid.

We sometimes say "K is an ***adequate filtering for*** an" to emphasize the key role of K. We briefly explain why these conditions ensure adequacy in the $\forall\exists$ sense, if the annotation

an is valid. First, the *enabled* condition ensures the underlying L, R, J-product can always perform left-only, right-only, or joint steps; this is the same as in section 7 for $\forall\forall$. But transitions by $\Mapsto_K$ are filtered by the keep set K, so we need to ensure that it keeps enough. Transitions of A can only be covered by the alignment product using LO or JO steps. The *left-permissive* condition applies to states $((n, n'), (s, s'))$ where the product is poised to perform an LO step, and requires that it be allowed by K. The *joint-productive* condition applies if the product is ready to take a JO step. It requires that for any A transition, there is some A' transition such that the joint step is allowed by K. Finally, the *right-productive* condition ensures that for RO steps, K is not just keeping divergent traces of A'. When the product is poised to take a RO step, K must allow some transition that decreases the value of the variant V.

The following results parallel Lemma 4.4, Corollary 4.7, and Corollary 4.8.

Lemma 9.6. Let *an* be a valid annotation of $\prod(A, A', L, R, J, K)$ for $\mathcal{P} \rightsquigarrow \mathcal{Q}$ and suppose K is an adequate filtering for *an*. Then $\prod(A, A', L, R, J, K)$ is $\mathcal{P}$-adequate in the $\forall\exists$ sense.

Proof. First we make a simple observation that pertains to any automaton A and annotation *an* for some spec $P \rightsquigarrow Q$. If *an* is valid, s is in P, and $(\mathit{init}, s) \mapsto^* (m, t)$, then $t \in \mathit{an}(m)$. Thus for the product $\prod(A, A', L, R, J, K)$, the condition $\mathit{an}(n, n')$ holds whenever the product is at control point (n, n'). Hence $L \vee R \vee J \vee [\mathit{fin}|\mathit{fin}']$ is an invariant of the product, owing to the enabled condition in Definition 9.5 of adequate filtering.

To show that $\prod(A, A', L, R, J, K)$ is $\mathcal{P}$-adequate in the $\forall\exists$ sense, suppose $(s, s') \in \mathcal{P}$ and suppose $(\mathit{init}, s) \mapsto^* (\mathit{fin}, t)$, i.e., there is a terminated trace τ of A from s to t. We must show that the product can simulate τ, without getting stuck, until it reaches $(\mathit{fin}, \mathit{fin}')$. Because $L \vee R \vee J \vee [\mathit{fin}|\mathit{fin}']$ is invariant, and using the liveness of L, R, and J as required by Definition 4.2 (via Definition 9.1), the product can keep taking steps until it reaches $(\mathit{fin}, \mathit{fin}')$. The LO and JO steps move forward simulating τ, thus eventually matching all of τ, unless the product diverges taking only RO steps. Right-productivity ensures that only finitely many RO steps can happen before $L \vee J \vee [\mathit{fin}|\mathit{fin}']$ holds.

Making this precise requires a slightly intricate induction hypothesis. A detailed proof can be found in [NBN23]. □

Theorem 9.7 (semantic soundness and completeness of filtered automata)**.** We have $A, A' \models \mathcal{P} \overset{\exists}{\approx\!>} \mathcal{Q}$ iff there are L, R, J, K and a valid annotation *an* of $\prod(A, A', L, R, J, K)$ for $\mathcal{P} \rightsquigarrow \mathcal{Q}$ such that K is an adequate filtering for *an*. Moreover, if A, A' act on variable stores, and $A, A', \mathcal{P}, \mathcal{Q}$ are finitely supported, then there are finitely supported such L, R, J, K, an (and witness V).

Proof (Sketch). The proof is by mutual implication. For right-implies-left the argument is easy. If *an* is a valid annotation of $\prod(A, A', L, R, J, K)$ for $\mathcal{P} \overset{\exists}{\approx\!>} \mathcal{Q}$ and K is an adequate filtering for *an*, then by Lemma 9.6 the product is $\mathcal{P}$-adequate. Since *an* is valid for $\mathcal{P} \rightsquigarrow \mathcal{Q}$ we have $\prod(A, A', L, R, J, K) \models \mathcal{P} \rightsquigarrow \mathcal{Q}$ by Lemma 4.6. So by Lemma 9.4 we have $A, A' \models \mathcal{P} \overset{\exists}{\approx\!>} \mathcal{Q}$.

For left-implies-right, suppose $A, A' \models \mathcal{P} \overset{\exists}{\approx\!>} \mathcal{Q}$. The idea is to use a left-first sequential alignment automaton such that, once the left-only execution has finished, with final store t, the filter condition only keeps states that are in some terminating run of A' that ends in a store t' such that $(t, t') \in \mathcal{Q}$. The variant is defined in terms of shortest terminating runs

if $\mathsf{sub}(n', c')$ is...	and m' is...	then the VC for $((n, n'), (n, m'))$ is equivalent to ...
$\mathsf{skip}^{n'}$	$\mathsf{fsuc}(n', c', f')$	$R \wedge \breve{an}(n, n') \wedge K(n, m') \Rightarrow \hat{an}(n, m')$
$x' :=^{n'} e'$	$\mathsf{fsuc}(n', c', f')$	$R \wedge \breve{an}(n, n') \wedge K(n, m')\vert^{x'}_{\vert e'} \Rightarrow \hat{an}(n, m')\vert^{x'}_{\vert e'}$
$\mathsf{hav}^{n'}\ x'$	$\mathsf{fsuc}(n', c', f')$	$\forall v' \in \mathbb{Z}.\ R \wedge \breve{an}(n, n') \wedge K(n, m')\vert^{x'}_{\vert v'} \Rightarrow \hat{an}(n, m')\vert^{x'}_{\vert v'}$
$\mathsf{if}^{n'}\ gcs'\ \mathsf{fi}$	$\mathsf{lab}(d')$ where $e' \to d'$ in gcs'	$R \wedge \breve{an}(n, n') \wedge [\![e']\!] \wedge K(n, m') \Rightarrow \hat{an}(n, m')$
$\mathsf{do}^{n'}\ gcs'\ \mathsf{od}$	$\mathsf{lab}(d')$ where $e' \to d'$ in gcs'	$R \wedge \breve{an}(n, n') \wedge [\![e']\!] \wedge K(n, m') \Rightarrow \hat{an}(n, m')$
$\mathsf{do}^{n'}\ gcs'\ \mathsf{od}$	$\mathsf{fsuc}(n', c', f')$	$R \wedge \breve{an}(n, n') \wedge \neg[\![\mathsf{enab}(gcs')]\!] \wedge K(n, m') \Rightarrow \hat{an}(n, m')$

In all other cases, there are no transitions from (n, n') to (n, m') so the VC is *true* by definition.

Figure 18: The right-only VCs for annotation an of $\prod(\mathsf{aut}(c, f), \mathsf{aut}(c', f'), L, R, J, K)$.

if $\mathsf{sub}(n, c)$ are... and $\mathsf{sub}(n', c')$	m m' are...	then the VC for $((n, n'), (m, m'))$ is equivalent to...
$x :=^{n} e$	$\mathsf{fsuc}(n, c, f)$	$J \wedge \breve{an}(n, n') \wedge [\![e']\!] \wedge K(m, m')^{x\vert}_{e\vert} \Rightarrow \hat{an}(m, m')^{x\vert}_{e\vert}$
$\mathsf{if}^{n'}\ gcs'\ \mathsf{fi}$	$\mathsf{lab}(d')$ for $e' \to d'$ in gcs'	
$\mathsf{hav}^{n}\ x$	$\mathsf{fsuc}(n, c, f)$	$\forall v \in \mathbb{Z}.\ J \wedge \breve{an}(n, n') \wedge K(m, m')^{x\vert x'}_{v\vert e'} \Rightarrow \hat{an}(m, m')^{x\vert x'}_{v\vert e'}$
$x' :=^{n'} e'$	$\mathsf{fsuc}(n', c', f')$	
$\mathsf{hav}^{n}\ x$	$\mathsf{fsuc}(n, c, f)$	$\forall v \in \mathbb{Z}.\ J \wedge \breve{an}(n, n') \wedge [\![e']\!] \wedge K(m, m')^{x\vert}_{v\vert} \Rightarrow \hat{an}(m, m')^{x\vert}_{v\vert}$
$\mathsf{do}^{n'}\ gcs'\ \mathsf{od}$	$\mathsf{lab}(d')$ for $e' \to d'$ in gcs'	
$\mathsf{hav}^{n}\ x$	$\mathsf{fsuc}(n, c, f)$	$\forall v, v' \in \mathbb{Z}.\ J \wedge \breve{an}(n, n') \wedge K(m, m')^{x\vert x'}_{v\vert v'} \Rightarrow \hat{an}(m, m')^{x\vert x'}_{v\vert v'}$
$\mathsf{hav}^{n'}\ x'$	$\mathsf{fsuc}(n', c', f')$	

Omitted: the other 32 cases with nontrivial VCs.

Figure 19: Selected joint VCs for annotation an of $\prod(\mathsf{aut}(c, f), \mathsf{aut}(c', f'), L, R, J, K)$.

that end in $\mathcal{Q}$. The details are somewhat complicated and relegated to an appendix for the interested reader (Appendix D). □

Verification conditions for filtered alignment automata. Let an be an annotation for the filtered alignment automaton $\prod(A, A', L, R, J, K)$ for $\mathcal{S} \rightsquigarrow \mathcal{T}$. For each pair of control points (n, n'), (m, m') of the product, the corresponding VC is given by definition (4.2) instantiated by $\Mapsto_K$. Restricting attention to filtered alignment automata of programs, the VCs can be expressed in a form similar to those for (unfiltered) alignment automata as given in Figure 8 and Figure 9. The key difference is in how the keep set K is handled. To facilitate the definitions, we define for any control points (n, n') the state set $K(n, n') \mathrel{\hat{=}} \{((i, i'), (s, s')) \mid ((n, n'), (s, s')) \in K\}$.

Figure 18 lists right-only VCs for the filtered product $\prod(\mathsf{aut}(c, f), \mathsf{aut}(c', f'), L, R, J, K)$. As an example, consider the VC for right-only $\mathsf{hav}^{n'}\ x'$. It says that for any value of x', if R and $\breve{an}(n, n')$ hold, and that value of x' will be "kept" by K when control reaches (n, m'), then the annotation at (n, m') holds for that value of x'. Left-only VCs are similar. Figure 19 lists a selected set of joint VCs.

For the alignment completeness result for ERHL+, we rely on encoded VCs that use pc-encoded versions of L, R, J, and a convenient encoding of K. For the keep set encoding, define

$$K^{\downarrow}(m, m') = \{(s, s') \mid ((m, m'), (s, s')) \in K\}$$

Corresponding to Lemma 4.18, we have the following for filtered alignment automata.

if $\mathsf{sub}(n', c')$ is...	and m' is...	then the encoded VC for $((n, n'), (n, m'))$ is ...
$\mathsf{skip}^{n'}$	$\mathsf{fsuc}(n', c', f')$	$\tilde{R} \wedge \{\!\{?n \mid ?n'\}\!\} \wedge an(n, n') \wedge K^{\downarrow}(n, m') \Rightarrow an(n, m')$
$x' :=^{n'} e'$	$\mathsf{fsuc}(n', c', f')$	$\tilde{R} \wedge \{\!\{?n \mid ?n'\}\!\} \wedge an(n, n') \wedge K^{\downarrow}(n, m')\big\vert^{x'}_{e'} \Rightarrow an(n, m')\big\vert^{x'}_{e'}$
$\mathsf{hav}^{n'}\ x'$	$\mathsf{fsuc}(n', c', f')$	$\forall v' \in \mathbb{Z}.\ \tilde{R} \wedge \{\!\{?n \mid ?n'\}\!\} \wedge an(n, n') \wedge K^{\downarrow}(n, m')\big\vert^{x'}_{v'} \Rightarrow an(n, m')\big\vert^{x'}_{v'}$
$\mathsf{if}^{n'}\ gcs'\ \mathsf{fi}$	$\mathsf{lab}(d')$ for $e' \to d'$ in gcs'	$\tilde{R} \wedge \{\!\{?n \mid ?n'\}\!\} \wedge an(n, n') \wedge [\![e]\!] \wedge K^{\downarrow}(n, m') \Rightarrow an(n, m')$
$\mathsf{do}^{n'}\ gcs'\ \mathsf{od}$	$\mathsf{lab}(d')$ for $e' \to d'$ in gcs'	$\tilde{R} \wedge \{\!\{?n \mid ?n'\}\!\} \wedge an(n, n') \wedge [\![e]\!] \wedge K^{\downarrow}(n, m') \Rightarrow an(n, m')$
$\mathsf{do}^{n'}\ gcs'\ \mathsf{od}$	$\mathsf{fsuc}(n', c', f')$	$\tilde{R} \wedge \{\!\{?n \mid ?n'\}\!\} \wedge an(n, n') \wedge \neg[\![\mathsf{enab}(gcs')]\!] \wedge K^{\downarrow}(n, m') \Rightarrow an(n, m')$

Figure 20: The right-only *pc*-encoded VCs for an and $\prod(\mathsf{aut}(c, f), \mathsf{aut}(c', f'), L, R, J, K)$.

if $\mathsf{sub}(n, c)$ are... $\mathsf{sub}(n', c')$	and m are... m'	then the encoded VC for $((n, n'), (m, m'))$ is ...
skip^{n}	$\mathsf{fsuc}(n, c, f)$	$\tilde{J} \wedge \{\!\{?n \mid ?n'\}\!\} \wedge an(n, n') \wedge K^{\downarrow}(m, m') \Rightarrow an(m, m')$
$\mathsf{skip}^{n'}$	$\mathsf{fsuc}(n', c', f')$	
$x :=^{n} e$	$\mathsf{fsuc}(n, c, f)$	$\tilde{J} \wedge \{\!\{?n \mid ?n'\}\!\} \wedge an(n, n') \wedge K^{\downarrow}(m, m')^{x\vert x'}_{e\vert e'} \Rightarrow an(m, m')^{x\vert x'}_{e\vert e'}$
$x' :=^{n'} e'$	$\mathsf{fsuc}(n', c', f')$	
$\mathsf{hav}^{n}\ x$	$\mathsf{fsuc}(n, c, f)$	$\forall v \in \mathbb{Z}.\ \tilde{J} \wedge \{\!\{?n \mid ?n'\}\!\} \wedge an(n, n') \wedge K^{\downarrow}(m, m')^{x\vert x'}_{v\vert e'} \Rightarrow an(m, m')^{x\vert x'}_{v\vert e'}$
$x' :=^{n'} e'$	$\mathsf{fsuc}(n', c', f')$	

Figure 21: Selected *pc*-encoded joint VCs for an and $\prod(\mathsf{aut}(c, f), \mathsf{aut}(c', f'), L, R, J, K)$.

Lemma 9.8 (*pc*-encoded VCs for filtered alignment automata)**.** Let an be an annotation of $\prod(\mathsf{aut}(c, f), \mathsf{aut}(c', f'), L, R, J, K)$ for commands c, c'. Suppose pc is a fresh variable in the sense that it does not occur in c or c', and all of L, R, J and any $an(i, j)$ are independent from pc on both sides. Then each of the right-only relational VC of Figure 18 implies the corresponding condition on store relations in Figure 20. Each joint VC in Figure 19 implies the corresponding condition in Figure 21. Similarly for left-only VCs.

The conditions in Figure 20 are obtained using the *pc*-encoded $\tilde{R} \wedge \{\!\{?n \mid ?n'\}\!\}$ in place of $R \wedge [n|n']$, an in place of $\hat{an}$, and $K^{\downarrow}$ in place of K. The conditions in Figure 21 are obtained similarly. Note that Figure 21 lists only a few *pc*-encoded joint VCs but all 36 VC have encodings. The proof of Lemma 9.8 is similar to that of Lemma 4.18.

9.2. **Alignment completeness of ERHL+.** Almost all the ground work has been laid to state and prove alignment completeness of ERHL+ with respect to filtered alignment automata. It remains to address the issue of control determinacy of programs that occur on the right in $\forall\exists$ judgments, as discussed in subsection 8.1. A straightforward argument connects the property to automata as follows.[25]

Lemma 9.9. If c is control deterministic then $\mathsf{aut}(c, f)$ is control deterministic in the sense that for any states (n, s), (m_0, t_0), and (m_1, t_1), if $(n, s) \mapsto (m_0, t_0)$ and $(n, s) \mapsto (m_1, t_1)$ then $m_0 = m_1$.

The alignment completeness theorem restricts the right-side program to be control deterministic. To this end, define $\mathsf{okfd}(c, f)$ to mean c is control deterministic and $\mathsf{okf}(c, f)$.

[25]If A and A' are control deterministic then so is any $\prod(A, A', \dots)$, but we do not need this fact.

It is used in assumption (a) of the theorem below. Assumptions (b) and (c) constitute an IAM-style proof. Assumption (d) is a technicality to ensure that a fresh *pc* variable can be chosen for application of Theorem 5.13.

Theorem 9.10. Suppose we have the following.
(a) $\mathsf{okf}(c, f)$ and $\mathsf{okfd}(c', f')$.
(b) *an* is a valid annotation of $\prod(\mathsf{aut}(c, f), \mathsf{aut}(c', f'), L, R, J, K)$ for $\mathcal{S} \rightsquigarrow \mathcal{T}$.
(c) K is an adequate filtering for *an* and $\prod(\mathsf{aut}(c, f), \mathsf{aut}(c', f'), L, R, J, K)$.
(d) *an*, $\mathcal{S}$, $\mathcal{T}$, L, R, J, and the witness V for adequate filtering, all have finite support.
Then the judgment $c \mid c' : \mathcal{S} \overset{\exists}{\approx\!>} \mathcal{T}$ has a proof in ERHL+.

As in the case of Theorem 7.1, the only relational assertions used in the proof are those derived from $an(i, j)$ L, R, J, and K.

Proof (Sketch). The lengthy proof is similar to that of Theorem 7.1 for RHL+. We start by choosing a fresh variable *pc*, transform c, c' to their automata normal forms, and apply rule eDo. The loop invariant $\mathcal{Q}$ is identical to the one defined in the proof of Theorem 7.1. The main difference between the two proofs is in how each of the premises of the loop rule are established. As in the proof for RHL+, we derive premises of eDo by the assignment and sequence rules, and eConseq, using VCs provided by *an*. However, VCs of filtered automata have antecedents involving K, see Fig 18 and Figure 19. So to exploit the VCs, in applications of eConseq, we rely on the fact that K is an adequate filtering. Proofs of left-only and right-only premises use eDisj in the same way rDisj is used in the proof of Theorem 7.1. For right-only premises, we additionally need to reason about the variant, for which we rely on the right-productivity condition of Definition 9.5. Owing to the assumption that c' is control deterministic, the automaton $\mathsf{aut}(c', f')$ is control deterministic (Lemma 9.9). This is important because when appealing to right-productivity or joint-productivity there is a unique control point witnessing the existential in those properties.

Finally, having used eDo to show that the automata normal forms of c and c' satisfy $\mathcal{S} \overset{\exists}{\approx\!>} \mathcal{T}$, we finish the proof using eRewrite (with Theorem 5.13) and eGhost to conclude the same judgment for c, c', as in the proof of Theorem 7.1. □

9.3. **Cook completeness revisited for ERHL+.**

Theorem 9.11. The logic ERHL+ is Cook complete, for control deterministic programs.

Proof. The proof is like Cook completeness for RHL+. If $\models c \mid c' : \mathcal{S} \overset{\exists}{\approx\!>} \mathcal{T}$ and $\mathcal{S}, \mathcal{T}$ are finitely supported then there is a valid annotation and adequate filtered alignment automaton for the automata of c and c', by Theorem 9.7. Suppose c' is control deterministic. Then by Theorem 9.10 there is a proof in ERHL+. □

By inspection of the proof of Theorem 9.7, that theorem still holds if the well-ordered set D in Definition 9.5 is restricted to be $\mathbb{N}$ with the usual order. So Cook completeness holds even if rule eDo of ERHL+ is restricted to use $\mathbb{N}$. Cognoscenti may note that the presence of unbounded nondeterminacy (as with our havoc command) necessitates use of ordinals beyond ω for proving always-termination [AP86]; but the $\forall\exists$ judgment is about existence of terminating runs.

In parallel to the discussion in subsection 7.2, the reader can check that, using the representation $c \mid \mathsf{skip} : \langle\!|P|\!\rangle \overset{\exists}{\approx\!>} \langle\!|Q|\!\rangle$ for $c : P \rightsquigarrow Q$, the rules of HL+ can be derived in ERHL+.

Furthermore, rules of forward underapproximation logic (e.g., [DYZD22, ABGL24]) can be derived using the representation $\mathsf{skip} \mid c : \mathbb{I}P\mathbb{I} \overset{\exists}{\approx\!>} \mathbb{I}Q\mathbb{I}$ for $c : P \overset{\exists}{\rightsquigarrow} Q$ (see for example eSkipDo in Figure 16). The difference between $\forall\forall$ and $\forall\exists$ judgments is evident when we consider proving Cook completeness using sequential alignment. First, compared with Lemma 7.3 the following result is only for left-before-right.

Lemma 9.12. For any $c, c', \mathcal{P}$ and $\mathcal{R}$, we have $\models c \mid c' : \mathcal{P} \overset{\exists}{\approx\!>} \mathcal{R}$ iff there is a $\mathcal{Q}$ such that $\models c \mid \mathsf{skip} : \mathcal{P} \overset{\exists}{\approx\!>} \mathcal{Q}$ and $\models \mathsf{skip} \mid c' : \mathcal{Q} \overset{\exists}{\approx\!>} \mathcal{R}$.

Proof. For the if direction, consider s, s', t with $(s, s') \in \mathcal{P}$ and $[\![c]\!]\, s\, t$. By the assumption $\models c \mid \mathsf{skip} : \mathcal{P} \overset{\exists}{\approx\!>} \mathcal{Q}$ we have $(t, s') \in \mathcal{Q}$. Thus, by the assumption about $\mathsf{skip} \mid c'$, there is a t' such that $(t, t') \in \mathcal{R}$ and we are done.

For the only-if direction, let $\mathcal{Q} := \{(t, s') \mid \exists s.\ (s, s') \in \mathcal{P} \wedge [\![c]\!]\, s\, t\}$. This $\mathcal{Q}$ is the strongest relation that holds after executing c on the left from $\mathcal{P}$-related states. It is straightforward to show $\models c \mid \mathsf{skip} : \mathcal{P} \overset{\exists}{\approx\!>} \mathcal{Q}$. To show $\models \mathsf{skip} \mid c' : \mathcal{Q} \overset{\exists}{\approx\!>} \mathcal{R}$, consider s, s', t with $(s, s') \in \mathcal{Q}$ and $[\![\mathsf{skip}]\!]\, s\, t$ which implies $s = t$ by semantics. By definition of $\mathcal{Q}$ there is u such that $[\![c]\!]\, u\, t$ and $(u, s') \in \mathcal{P}$. By the assumption $\models c \mid c' : \mathcal{P} \overset{\exists}{\approx\!>} \mathcal{R}$, there is a t' such that $[\![c']\!]\, s'\, t'$ and $(t, t') \in \mathcal{R}$. Thus, $\models \mathsf{skip} \mid c' : \mathcal{Q} \overset{\exists}{\approx\!>} \mathcal{R}$. □

Analogous to (7.10) we can connect with the unary over- and under-approximate judgments using relations encoded as predicates on variable stores:

$$\begin{array}{lll} \models c \mid \mathsf{skip} : \mathcal{P} \overset{\exists}{\approx\!>} \mathcal{Q} & \text{iff} & \models c : {}^{\bullet}\mathcal{P} \rightsquigarrow {}^{\bullet}\mathcal{Q} \\ \models \mathsf{skip} \mid c : \mathcal{P} \overset{\exists}{\approx\!>} \mathcal{Q} & \text{iff} & \models c : \mathcal{P}^{\bullet} \overset{\exists}{\rightsquigarrow} \mathcal{Q}^{\bullet} \end{array} \tag{9.1}$$

Consider these rules that correspond to the derivable rules eLRseq and eRLseq, which reduce a relational property to unary properties.

$$\textsc{ueLRseq}\ \frac{c : {}^{\bullet}\mathcal{P} \rightsquigarrow {}^{\bullet}\mathcal{Q} \qquad c' : \mathcal{Q}^{\bullet} \overset{\exists}{\rightsquigarrow} \mathcal{R}^{\bullet}}{c \mid c' : \mathcal{P} \overset{\exists}{\approx\!>} \mathcal{R}} \qquad \textsc{ueRLseq}\ \frac{c' : \mathcal{P}^{\bullet} \rightsquigarrow \mathcal{Q}^{\bullet} \qquad c : {}^{\bullet}\mathcal{Q} \overset{\exists}{\rightsquigarrow} {}^{\bullet}\mathcal{R}}{c \mid c' : \mathcal{P} \overset{\exists}{\approx\!>} \mathcal{R}}$$

Soundness of ueLRseq follows from Lemma 9.12. In fact ueRLseq is also sound, but by contrast with Lemma 9.12 there is not an equivalence but only an implication.

Example 9.13. In general, $\models c \mid c' : \mathcal{P} \overset{\exists}{\approx\!>} \mathcal{R}$ does not imply that there exists $\mathcal{Q}$ such that both $\models \mathsf{skip} \mid c' : \mathcal{P} \overset{\exists}{\approx\!>} \mathcal{Q}$ and $\models c \mid \mathsf{skip} : \mathcal{Q} \overset{\exists}{\approx\!>} \mathcal{R}$. Here is a counter-example. We have $\models \mathsf{hav}\ x \mid \mathsf{hav}\ x : \mathit{true} \overset{\exists}{\approx\!>} \mathbb{A}x$. Note that, because skip is domain total and deterministic (per comments at start of section 9), we have for any $c, \mathcal{P}, \mathcal{R}$ that $c \mid \mathsf{skip} : \mathcal{P} \overset{\exists}{\approx\!>} \mathcal{R}$ iff $\mathcal{P} \Rightarrow \mathsf{wp}(c \mid \mathsf{skip})(\mathcal{R})$. Now suppose there is a $\mathcal{Q}$ such that (i) $\models \mathsf{skip} \mid \mathsf{hav}\ x : \mathit{true} \overset{\exists}{\approx\!>} \mathcal{Q}$ and (ii) $\models \mathsf{hav}\ x \mid \mathsf{skip} : \mathcal{Q} \overset{\exists}{\approx\!>} \mathbb{A}x$. By (ii), $\mathcal{Q} \Rightarrow \mathsf{wp}(\mathsf{hav}\ x \mid \mathsf{skip})(\mathbb{A}x)$. But $\mathsf{wp}(\mathsf{hav}\ x \mid \mathsf{skip})(\mathbb{A}x) \Leftrightarrow (\forall v.\ \mathbb{A}x^{x|}_{v|}) \Leftrightarrow (\forall v.\ v = \mathbb{I}x\mathbb{I}) \Leftrightarrow \mathit{false}$. Thus $\mathcal{Q} = \mathit{false}$. Note that by definition of the $\forall\exists$ judgment there are no $c, d, \mathcal{P}$ such that $\models c \mid d : \mathcal{P} \overset{\exists}{\approx\!>} \mathit{false}$ (unless c has no executions from $\mathcal{P}$, which is not the case in the counter-example). So $\mathcal{Q} = \mathit{false}$ contradicts (i). □

Now we can show Cook completeness, for $\forall\exists$ properties, with the single ueLRseq rule. Assume we have a complete logic for $\overset{\exists}{\rightsquigarrow}$ judgments.[26] Suppose $\models c \mid c' : \mathcal{P} \overset{\exists}{\approx\!>} \mathcal{R}$ holds. By

[26]Such logics do exist. For example, sufficient incorrectness logic [ABGL24] is shown to be complete.

Lemma 9.12 there is a $\mathcal{Q}$ such that $\models c \mid \mathsf{skip} : \mathcal{P} \overset{\exists}{\approx\!>} \mathcal{Q}$ and $\models \mathsf{skip} \mid c' : \mathcal{Q} \overset{\exists}{\approx\!>} \mathcal{R}$. So we have $\models c : {}^{\bullet}\mathcal{P} \rightsquigarrow {}^{\bullet}\mathcal{Q}$ and $\models c' : \mathcal{Q}^{\bullet} \overset{\exists}{\rightsquigarrow} \mathcal{R}^{\bullet}$. By completeness of the unary logics, these specs are provable. One application of ueLRseq yields $c \mid c' : \mathcal{P} \overset{\exists}{\approx\!>} \mathcal{R}$.

By contrast with the situation for $\forall\forall$, completeness does not hold for ueRLseq plus unary $(\rightsquigarrow, \overset{\exists}{\rightsquigarrow})$ rules. Consider the counterexample $\mathsf{hav}\ x \mid \mathsf{hav}\ x : \mathit{true} \overset{\exists}{\approx\!>} \mathbb{A}x$ in Example 9.13. It cannot be proved using ueRLseq because there is no $\mathcal{Q}$ with which to instantiate the premises.

For the record, Cook completeness of ERHL+ (Theorem 9.11) can be proved for $\forall\exists$ judgments without restriction to judgments where the right side program is control deterministic. The details are in [NBN23].

10. Related work

Trace logic [BEG+19] reasons about $\forall\forall$ properties by way of constraints between arbitrary different points in the traces rather than restricting to points that progress in the manner of a schedule–what we call alignment. In principle, alignments may be defined by arbitrary strategy functions and the like [KSF13, BCB+21, CMP20]. In the following we confine attention to assertion-oriented work rather than works using global conditions on traces.

Cook's completeness result [Coo78] is in terms of a formal language of assertions. The result is relative to provability of assertion entailments, and depends on expressivity of the assertion language (good explanations can be found in [AdBO09] and [Win93]). In recent years, and especially in the context of machine-checked theories, it has become common to sidestep these issues by way of shallow embedding, as we have done. What is left is exactly the standard notion of completeness for a logic; we use the term "Cook completeness" for contrast with alignment completeness.

Cook completeness has been proved for several $\forall\forall$ relational Hoare logics (e.g., [Ber11, SD16, BGHS17, WDLC18]) using in each case the semantic completeness of sequential alignment and relying on completeness of HL along the lines we sketch in subsection 7.2. For alignment completeness, we are not aware of prior results besides those of Nagasamudram and Naumann [NN21]. They give several $\forall\forall$ alignment completeness results for very specialized automata forms that account for alignments given by particular proof rules. Their proofs of alignment completeness are quite different from ours: owing to the specialized structure of the automata considered, they are able to construct deductive proofs that follow the structure of the source programs without any rewriting. The alignment completeness result for RHL+ was presented in our unpublished preprint [BNN22], which is superceded by the present article (and [NBN23]).

Nagasamudram and Naumann [NN21] introduce the term Floyd completeness and use it for a result like our Theorem 6.1. The practical importance of both their and our Floyd completeness results is that the deductive proof does not require more expressive assertions than used in the IAM proof.

The conditionally aligned loop rule (our rDo in Figure 2) appears first in Beringer's work [Ber11]. Variations appear in Barthe et al [BGHS17], in the full version of Banerjee et al [BNN16], and in [BNNN22]. Our adaptation for $\forall\exists$ (eDo in Figure 15) is new. Several recently published logics only support lockstep alignment of loops, or lockstep until one terminates (even [DYZD22] published subsequent to [BGHS17]). Beutner's [Beu24] loop rule for $\forall\exists$, named *loop-counting*, generalizes the lockstep pattern, catering for alignments

that relate n unrollings of a loop with m unrollings of the other, for fixed n and m. The rule requires loops being related to terminate simultaneously, disallowing alignments that reason in terms of lockstep iterations up to the point where one loop terminates and then reason about the remaining iterations of the other. Due to the side-condition on termination, the rule doesn't require a variant. Although fixing n and m independent of data can handle some examples, it is too restrictive for others as discussed in Example 3.2. A primary motivation for Beutner's work is automated verification, for which the loop-counting rule is shown to be effective.

We are not aware of unary HLs that feature a rewriting rule, but verification tools often use correctness-preserving rewriting. The RHLs of [BGHS17] and [BNN16] each feature a rewriting rule and a custom set of rules for command equivalence (with a relational precondition, in [BGHS17]). (The judgments of [BGHS17] are probabilistic but alignment is still central.) Rewriting by command equivalence is combined with relational reasoning in an extension of KAT called BiKAT [AKL$^+$23]. Rewriting (unfoldings) is used by Strichman and Veitsman [SV16] to improve alignment in regression verification of recursive functions. Verifiable C [CBG$^+$18] includes a proof rule that reassociates sequences, as does [Beu24].

The $\forall\forall$ property is also known as 2-safety [TA05]. Cartesian Hoare logic [SD16, PFG18] reasons about k-safety, for k that is fixed throughout a proof. D'Osualdo et al [DFD22] develop a logic for k-safety that features rules for combining judgments with varying k; they have a Cook completeness result based essentially on sequential alignment as explained in [DFD22, appendix C]. The system of *op. cit.* includes a rewriting rule, but based on semantic refinement (i.e., refinement is not formalized in a deductive system but rather must be proved in the metalogic). Our results should generalize to $k > 2$, but the case of 2 admits simpler notations and suffices to illuminate the issues we address.

Except as noted, none of the preceding works consider $\forall\exists$ properties. Hawblitzel et al [HKLR13] use the term relative termination for $\forall\exists$ judgments. The logic of Benton [Ben04] is for cotermination of deterministic programs, which can be expressed by a pair of $\forall\exists$ judgments of the form $c \mid d : \mathcal{P} \overset{\exists}{\approx\!>} \mathcal{Q}$ and $d \mid c : \mathcal{P}^\circ \overset{\exists}{\approx\!>} \mathcal{Q}^\circ$ (where $\mathcal{P}^\circ$ is the converse of $\mathcal{P}$). Rinard gives a logical formulation of verification conditions for a $\forall\exists$ relation, to prove correctness of compiler transformations acting on control flow graphs [RM99, Rin99].

Antonopoulos et al [AKL$^+$23] derive some deductive rules for $\forall\exists$ (and also the variation known as backward simulation), in a KAT-based algebraic framework. Their "witness" technique for $\forall\exists$ reasoning is akin to our filtering K condition, obtaining existence by filtering behaviors of a $\forall\forall$ automaton, by contrast with works that literally construct witness executions [LS21, UTK21]. An early formulation of relational verification using automata is [BCK13] which introduces a notion of asymmetric product for verifying $\forall\exists$ properties.

To our knowledge the first published deductive system for general pre-post $\forall\exists$ properties is RHLE [DYZD22]. It is based on HL together with unary rules for forward underapproximation —the judgment we write as $c : P \overset{\exists}{\rightsquigarrow} Q$, see (8.1). The core rules of RHLE reduce relational reasoning to sequential unary reasoning. Soundness is proved. The system includes rules for modular reasoning using unary procedure specs, and uses over- and under-approximate semantics of procedure calls. Loops are handled using the mostly lockstep rules like in Sousa and Dillig's work [SD16]. The use of unary procedure specs gives rise to nondeterminacy, motivating the use of $\forall\exists$ judgements. By contrast, Eilers et al [EMH18] and Banerjee et al [BNNN22] use relational procedure specs for $\forall\forall$ properties. Subsequent to developing our results we became aware of Beutner's Forall-Exists Hoare logic (FEHL) for $\forall\exists$ properties [Beu24]. FEHL features rules similar to RHLE, and includes the more general

loop rule (loop-counting) mentioned earlier in this section. It also includes a couple of rules for rewriting with sequence associativity and skip unit law. Like in RHLE, the rules in FEHL rely on HL and a unary logic for forward underapproximation ($\overset{\exists}{\rightsquigarrow}$). Cook completeness is obtained for FEHL via Cook completeness of HL and a logic for forward underapproximation, using a variation of our rule ueLRseq. Owing to the restrictive treatment of loops it is unlikely that FEHL or RHLE is alignment complete. The purpose of both FEHL and RHLE is for use in automated search and both works provide a search algorithm and experimental results.

ReLoC [FKB18] is a logic for contextual refinement of higher order concurrent programs, a specific $\forall\exists$ property. It is not a freestanding deductive system but rather it is shallow embedded in Iris [JKJ+18] (which in turn is implemented in the Coq proof assistant). So one can, e.g., negate the refinement judgment and express some forms of conditional refinement.

Turning to IAM-style verification, the work of [CPSA19, SGSV19, UTK21] can be seen as various techniques for finding adequate alignment conditions (L, R, J) and annotations expressible in SMT-supported assertion languages. Churchill et al also use testing to evaluate whether a candidate (L, R, J) is manifestly adequate. Our example $c3$ (section 2) is from [UTK21]; they formulate adequacy conditions for $\forall\exists$ properties, whereas the others cited only address $\forall\forall$. Unno et al address the right-only progress condition for $\forall\exists$ by finding a well-founded relation between transition states; this is not directly representable in a logic of pre/post relations (though it may be in a logic like RHTT [NBG13] where postconditions constrain two initial + two final states), so we need variant function V and a fresh snapshot variable in rule eDo. Instead of filtering, Unno et al require that choices on right be given by a function of the left state together with prophecy variables about the final state on the left. Our formalization enables use of prophecy for both final and intermediate values. More recently, Itzhaky et al. [ISV24] develop a method for verifying $\forall\exists$ properties via reduction to constrained Horn clauses (CHCs). Their technique solves simultaneously for alignments, relational invariants, and witness functions for right side executions. The natural encoding of the L, R, J adequacy condition as a first-order formula is not Horn, so Itzhaky et al. use multiple steps to transform it to a set of equi-satisfiable CHCs. This enables using a single CHC solver query when searching for solutions, as opposed to prior approaches [UTK21, SGSV19] that require multiple queries or the use of specialized solvers.

Results like our adequacy Proposition 4.5 have been proved for several notions of alignment product that are similar to ours. Our L, R, J corresponds to the "alignment predicate" in Churchill et al [CPSA19], the "composition function" in Shemer et al [SGSV19], and "scheduler" in Unno et al [UTK21]. These works focus on automated search for good alignments and annotations using solvers for restricted assertion languages.

Sequential alignment has been used to prove completeness results for product automata [Fra83, UTK21] as in Theorem 9.7.

Beutner and Finkbeiner [BF22b] formulate temporal $\forall\exists$ properties in terms of games, so a proof involves a strategy whereby the $\exists$ player produces a witnessing execution. They represent programs by transition systems, and combine the search for a strategy with search for an alignment (called reduction, cf. [FV19]) and filtering conditions (called restrictions), also represented as a game. Itzhaky et al's approach [ISV24] to $\forall\exists$ verification using CHCs is shown to be sound, and complete with respect to this game semantics for transition systems with bounded nondeterminism. However, this game semantics itself is incomplete. Beutner and Finkbeiner show [BF22a] that with the inclusion of prophecies, a game based method is complete for finite state systems and specifications in synchronous HyperLTL.

Our normal form is like that of Hoare et al [HHS93] which uses an explicit program counter variable. They prove every program can be reduced to normal form using a set of refinement laws, demonstrating a kind of completeness of the laws. Kozen [Koz97] uses KAT to prove that every command is simulated by one with a single loop. That result is refined in [GKM14] using KAT augmented with finite mutable state; in principle this would provide an alternate way to define $\simeq$, cf. Remark 5.5.

11. Future work

Because our development makes no commitment or unusual demands on the assertion language, the results should be applicable to richer data types including dynamically allocated pointer structures, and deductive systems including separation logic. Our program equivalence $\simeq$ makes a minimal demand on reasoning about data: the ability to specify that primitive expressions and commands do not interfere with a fresh ghost variable.

For languages with procedures, alignment of calls is important to facilitate use of relational specs [GS08, EMH20, BNNN22]. A theory of alignment completeness of such languages would involve the complications of automata representation for programs and some notion of modular IAM.

A key question is what are good criteria for relational logics. Cook completeness is clearly important. Alignment completeness is an additional criterion that accounts for widely used rules of RHL by connection with the fundamental IAM. For unary HL, the corresponding notion is Floyd completeness. These notions are not the only sensible criteria.

One obvious criterion is essentially aesthetic: the core set of rules should be general and orthogonal or minimal in some sense. The model is HL for simple imperative programs. There is one rule for each program construct. In addition there are so-called *structural* rules. To capture IAM reasoning it suffices to have a single structural rule, Conseq; this is formalized in the Floyd completeness result of [NN21]. Principles beyond IAM, such as framing, conjunctive splitting and auxiliary variables, which facilitate modular reasoning, are embodied by additional rules often included in HL. For example, the rule of conjunction enables to prove correctness of quicksort by first proving the permutation property and then proving sortedness (see [AdBO09, chapter 5]). History and prophecy variables extend the expressive power of assertions: Although an assertion pertains to the store at a particular control point (or aligned pair thereof), auxiliary variables are used to express relationships with computation steps at other control points.

For our main results we focus on sets of rules (RHL+ and ERHL+) with only the structural rules needed for alignment completeness —rewriting, ghost elimination, and disjunction. For our main results one could also severely restrict the other rules: Inspecting the proofs of alignment completeness one may see we could drop the rIf/eIf rules and others, and rely on a specialized form of rDo/eDo that only applies to the patterns that appear in the normal form (Lemma 5.10). Of course the aesthetic criterion guides us to include a single, general rule for each program construct.

Adopting highly specialized rules tailored to alignment completeness is a bad idea because it would force all proofs to go by maximal rewriting to normal form, and all proofs to be IAM style, largely abandoning the syntax-oriented benefits of Hoare logic. This suggests another criterion: a good logic should support proofs that are modular and natural. One way to make "natural" more precise is to connect it with alignment completeness. We pose this open problem: *Prove alignment completeness by some technique that minimizes the use of*

rewriting and instead preserves the original program structure as much as possible. Although definitions 4.2 and 9.1 admit strange and gratuitously complicated alignment conditions, as may arise using automated inference techniques, examples suggest that for deductive proofs it should usually suffice to rewrite just enough to make the control structures similar. For that matter, if an automaton has one of the forms in [NN21] then little rewriting should be needed as shown in that article. To derive rules for common patterns, such as those in Figure 3 and Figure 16, it suffices to do rewrites by very minimal laws such as the unit law for skip. Finding a notion of minimality for rewriting, and proving alignment-completeness with minimal rewrites, might have practical value to combine automata-based and deductive methods.

A key criterion that can be made precise is known as ***adaptation completeness*** [Kle99, AO19]. For unary correctness it says that if the spec $P \rightsquigarrow Q$ is semantically entailed by the spec $R \rightsquigarrow S$, in the sense that $\models c : R \rightsquigarrow S$ implies $\models c : P \rightsquigarrow Q$ for any c, then the proof rules are sufficient to derive $c : P \rightsquigarrow Q$ from $c : R \rightsquigarrow S$. The Conseq rule is useful for this purpose but not adaptation complete, nor is the Adaptation rule of [Hoa71] but complete rules are known [AO19].

A generalization of adaptation completeness emerges in the context of relational reasoning. As an example, D'Osualdo et al [DFD22] consider the following hypotheses about unknown commands (or command variables) c and d. First, $c \mid c : \mathbb{A}x \approx\!\!> \mathbb{A}x$ and $d \mid d : \mathbb{A}x \approx\!\!> \mathbb{A}x$, i.e., they are deterministic with respect to variable x. Second, they commute in the sense that $(c;d) \mid (d;c) : \mathbb{A}x \approx\!\!> \mathbb{A}x$. Third, they may terminate from any state, i.e., $c : \mathit{true} \overset{\exists}{\rightsquigarrow} \mathit{true}$ and $d : \mathit{true} \overset{\exists}{\rightsquigarrow} \mathit{true}$. Equivalently, $\mathsf{skip} \mid c : \mathit{true} \overset{\exists}{\approx\!\!>} \mathit{true}$ and $\mathsf{skip} \mid d : \mathit{true} \overset{\exists}{\approx\!\!>} \mathit{true}$. It follows semantically that $(c;d;d) \mid (d;d;c) : \mathbb{A}x \approx\!\!> \mathbb{A}x$. One expects to show this by transitively composing the judgments $(c;d;d) \mid (d;c;d) : \mathbb{A}x \approx\!\!> \mathbb{A}x$ and $(d;c;d) \mid (d;d;c) : \mathbb{A}x \approx\!\!> \mathbb{A}x$. D'Osualdo et al posit that this is beyond the reach of RHLs and introduce a somewhat different deductive system that handles the example. As noted earlier, $\forall\forall$ judgments do not transitively compose in general, owing to the possibility that the middle program diverges. So D'Osualdo et al introduce a special judgment for termination. Another approach is to leverage $\forall\exists$ judgments as in this sound rule.

$$\frac{c \mid d : \mathcal{P} \approx\!\!> \mathcal{Q} \qquad \mathsf{skip} \mid d : \mathcal{P} \overset{\exists}{\approx\!\!>} \mathit{true} \qquad d \mid c' : \mathcal{R} \approx\!\!> \mathcal{S}}{c \mid c' : \mathcal{P};\mathcal{R} \approx\!\!> \mathcal{Q};\mathcal{S}}\ \textsc{rTrans}$$

Here we write $\mathcal{P};\mathcal{R}$ for composition of relations and note that $\mathbb{A}x;\mathbb{A}x$ is $\mathbb{A}x$.

In [NBN23] we extend RHL+ and ERHL+ with this rule and a few others adapted from D'Osualdo et al, which suffice to prove the examples. This includes reasoning about idempotence where one execution of a command is related to a sequence of its executions, which is beyond the conventional notion of alignment. Although D'Osualdo et al show their system works nicely on a range of examples, they neither state nor establish the general property which we dub ***entailment completeness***. Meaning: If a judgment follows semantically from a set H of judgments, then it can be derived from H in the logic. To our knowledge the only entailment completeness result for a program logic is that of Kozen and Tiuryn [KT01] for propositional Hoare logic.[27] For relational correctness, we suspect that this open problem can be solved for some system that combines $\forall\forall$ and $\forall\exists$ judgments.

[27]In more powerful systems like dynamic logic or embeddings in higher order logic, entailment can expressed by a formula so entailment completeness reduces to ordinary completeness. This can be done in KAT [Koz00] and in propositional dynamic logic [HKT00, Thm. 7.7].

To see how entailment completeness motivates some well known structural rules, here is an example which in particular shows the need for prophecy variables to prove $\forall\exists$ judgments. Assuming that $c \mid c : \mathbb{A}w \overset{\exists}{\approx\!>} \mathbb{A}w$, and c neither reads nor writes y, this is valid:

$$c; \mathsf{hav}\ x \mid \mathsf{hav}\ y; c : \mathbb{A}w \overset{\exists}{\approx\!>} \mathbb{A}w \wedge (\!|x|\!) = (\!|y|\!) \tag{11.1}$$

Typically, as in Example 8.1, we align right-side havocs together with, or following, left-side havocs, so the postcondition on the existential side can refer to the left side. Here, it is not convenient to align $\mathsf{hav}\ y$ later than $\mathsf{hav}\ x$ because we need to align c with itself to exploit the assumption. The idea is to use a variable r that can be seen as predicting the final value of x on the left side. For this exposition we will consider r to be a logical variable that does not occur in code. We introduce r in a judgment that says y can match the predicted value:

$$\mathsf{skip} \mid \mathsf{hav}\ y : \mathbb{A}w \overset{\exists}{\approx\!>} \mathbb{A}w \wedge r = (\!|y|\!) \tag{11.2}$$

This is proved using eSkipHav and eConseq with validity of $\mathbb{A}w \Rightarrow (\exists|y.\ \mathbb{A}w \wedge r = (\!|y|\!))$. Using a frame rule (eFrame below), the assumption that c does not write y yields

$$c \mid c : \mathbb{A}w \wedge r = (\!|y|\!) \overset{\exists}{\approx\!>} \mathbb{A}w \wedge r = (\!|y|\!) \tag{11.3}$$

Next, by eHavSkip and eConseq we get

$$\mathsf{hav}\ x \mid \mathsf{skip} : \mathbb{A}w \wedge r = (\!|y|\!) \overset{\exists}{\approx\!>} \mathbb{A}w \wedge (r = (\!|x|\!) \Rightarrow r = (\!|y|\!)) \tag{11.4}$$

using that this is valid: $\mathbb{A}w \wedge r = (\!|y|\!) \Rightarrow (\forall x|.\ \mathbb{A}w \wedge (r = (\!|x|\!) \Rightarrow r = (\!|y|\!)))$. From (11.2–11.4) by eSeq, and eRewrite to eliminate skips, we get this key judgment that embodies the prophecy variable pattern:

$$c; \mathsf{hav}\ x \mid \mathsf{hav}\ y; c : \mathbb{A}w \overset{\exists}{\approx\!>} \mathbb{A}w \wedge (r = (\!|x|\!) \Rightarrow r = (\!|y|\!))$$

Now r occurs in neither the code nor the precondition, so by a rule eForall below we get

$$c; \mathsf{hav}\ x \mid \mathsf{hav}\ y; c : \mathbb{A}w \overset{\exists}{\approx\!>} (\forall r.\ \mathbb{A}w \wedge (r = (\!|x|\!) \Rightarrow r = (\!|y|\!)))$$

This yields (11.1) by eConseq.

For both $\forall\forall$ and $\forall\exists$ judgments there is a sound frame rule like that for unary logic.[28] We used this one:

$$\text{eFrame}\quad \frac{c \mid c' : \mathcal{P} \overset{\exists}{\approx\!>} \mathcal{Q} \qquad \mathsf{indep}(\mathsf{mods}(c)|\mathsf{mods}(c'), \mathcal{R})}{c \mid c' : \mathcal{P} \wedge \mathcal{R} \overset{\exists}{\approx\!>} \mathcal{Q} \wedge \mathcal{R}}$$

where $\mathsf{mods}(c)$ are the variables assigned or havoc'd in c. Finally, just as there are disjunction and conjunction rules, for pre- and post-conditions respectively, logics often include a rule for introducing existential in preconditions and less commonly a rule like the following one that we used. Here v is a logical variable that does not occur in code.[29]

$$\text{eForall}\quad \frac{c \mid c' : \mathcal{P} \overset{\exists}{\approx\!>} \mathcal{Q} \qquad \mathsf{indep}(v|v, \mathcal{P})}{c \mid c' : \mathcal{P} \overset{\exists}{\approx\!>} \forall v.\ \mathcal{Q}}$$

[28] Historically, variants had names including Invariance and Constancy, with Frame used where independence for heap locations is expressed using separating conjunction.

[29] Strictly speaking, in accord with our shallow embedding of store relations, one should consider that $\mathcal{Q}$ is a family of store relations $\mathcal{Q}_v$ indexed over $v \in \mathbb{Z}$ and define $\forall v.\ \mathcal{Q}$ as $\{(s, s') \mid \forall v \in \mathbb{Z}.\ (s, s') \in \mathcal{Q}_v\}$.

12. Conclusion

In this paper we augment a collection of relational Hoare logic rules with a straightforward rule for elimination of ghost variables, and a rule for deriving one correctness judgment from another by rewriting the commands involved to equivalent ones. The chosen notion of equivalence is that of KAT, allowing for the use of hypotheses to axiomatize the meaning of primitive commands and expressions when reasoning with KAT. The rewrites needed to derive a number of frequently proposed RHL rules require no hypotheses at all. Using a small set of hypotheses, we prove that any command is equivalent to one in automaton normal form, once it is instrumented with assignments to an explicit program counter variable. On this basis, we show that any correctness judgment proved in IAM style using an alignment automaton can be turned into a deductive proof in RHL+ using essentially the same assertions. One practical consequence is that automata-based alignment is not better than deductive in terms of strength of assertions needed. If some decidable fragment like linear arithmetic suffices for an automaton-based proof then that same fragment suffices for a deductive proof of the same judgment.

We also introduce a new notion, filtered alignment automata, for $\forall\exists$ properties. We introduce a new logic ERHL+ for $\forall\exists$ properties and show its alignment completeness. For both kinds of automata we show semantic completeness with respect to the relevant properties. Together with alignment completeness, this entails that RHL+ and ERHL+ are Cook complete with respect to $\forall\forall$ and $\forall\exists$ properties respectively.

Some rules which embody natural reasoning principles, such as the rule of conjunction (for $\forall\forall$) and transitive composition rules like rTrans, are not included in our logics because they are not needed for alignment completeness. We conjecture these are needed for entailment completeness which we posed as an interesting open problem which may lead to discovery of additional rules and reasoning principles.

Acknowledgments. We are grateful to the anonymous LMCS reviewers for insightful feedback which in particular helped improve the positioning of alignment completeness with respect to other criteria for program logics.

References

[ABGL24] Flavio Ascari, Roberto Bruni, Roberta Gori, and Francesco Logozzo. Sufficient incorrectness logic: SIL and separation SIL, 2024. arXiv:2310.18156.

[AdBO09] Krzysztof R. Apt, Frank S. de Boer, and Ernst-Rüdiger Olderog. *Verification of Sequential and Concurrent Programs*. Texts in Computer Science. Springer, 3 edition, 2009. doi:10.1007/978-1-84882-745-5.

[AKL+23] Timos Antonopoulos, Eric Koskinen, Ton Chanh Le, Ramana Nagasamudram, David A. Naumann, and Minh Ngo. An algebra of alignment for relational verification. *Proc. ACM Program. Lang.*, 7(POPL):573–603, 2023. Full version at https://arxiv.org/abs/2202.04278. doi:10.1145/3571213.

[AO19] Krzysztof R. Apt and Ernst-Rüdiger Olderog. Fifty years of Hoare's logic. *Formal Asp. Comput.*, 31(6), 2019.

[AP86] K. R. Apt and G. D. Plotkin. Countable nondeterminism and random assignment. *Journal of the ACM*, 33(4):724–767, 1986. doi:10.1145/6490.6494.

[BCB+21] Jan Baumeister, Norine Coenen, Borzoo Bonakdarpour, Bernd Finkbeiner, and César Sánchez. A temporal logic for asynchronous hyperproperties. In *Computer Aided Verification*, volume 12759 of *LNCS*, pages 694–717, 2021. doi:10.1007/978-3-030-81685-8_33.

[BCK13] Gilles Barthe, Juan Manuel Crespo, and César Kunz. Beyond 2-safety: Asymmetric product programs for relational program verification. In *Logical Foundations of Computer Science, International Symposium*, volume 7734 of *LNCS*, pages 29–43, 2013. doi:10.1007/978-3-642-35722-0_3.

[BDR04] Gilles Barthe, Pedro R. D'Argenio, and Tamara Rezk. Secure information flow by self-composition. In *IEEE Computer Security Foundations Workshop (CSFW'04)*, pages 100–114, 2004. doi:10.1109/CSFW.2004.17.

[BDR11] Gilles Barthe, Pedro R. D'Argenio, and Tamara Rezk. Secure information flow by self-composition. *Math. Struct. Comput. Sci.*, 21(6):1207–1252, 2011. doi:10.1017/S0960129511000193.

[BEG+19] Gilles Barthe, Renate Eilers, Pamina Georgiou, Bernhard Gleiss, Laura Kovács, and Matteo Maffei. Verifying relational properties using Trace Logic. In *2019 Formal Methods in Computer Aided Design*, pages 170–178, 2019. doi:10.23919/FMCAD.2019.8894277.

[Ben04] N. Benton. Simple relational correctness proofs for static analyses and program transformations. In *ACM Symposium on Principles of Programming Languages*, pages 14–25. ACM, 2004. doi:10.1145/964001.964003.

[Ber11] Lennart Beringer. Relational decomposition. In *Interactive Theorem Proving*, volume 6898 of *LNCS*, pages 39–54, 2011. doi:10.1007/978-3-642-22863-6_6.

[Beu24] Raven Beutner. Automated software verification of hyperliveness. In Bernd Finkbeiner and Laura Kovács, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 14571 of *LNCS*, pages 196–216, 2024. doi:10.1007/978-3-031-57249-4_10.

[BF22a] Raven Beutner and Bernd Finkbeiner. Prophecy variables for hyperproperty verification. In *IEEE Computer Security Foundations*, pages 471–485, 2022. doi:10.1109/CSF54842.2022.9919658.

[BF22b] Raven Beutner and Bernd Finkbeiner. Software verification of hyperproperties beyond k-safety. In Sharon Shoham and Yakir Vizel, editors, *Computer Aided Verification*, volume 13371 of *LNCS*, pages 341–362, 2022. doi:10.1007/978-3-031-13185-1_17.

[BGHS17] Gilles Barthe, Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub. Coupling proofs are probabilistic product programs. In *ACM Symposium on Principles of Programming Languages*, pages 161–174, 2017. doi:10.1145/3009837.3009896.

[BN99] Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge University Press, 1999.

[BNN16] Anindya Banerjee, David A. Naumann, and Mohammad Nikouei. Relational logic with framing and hypotheses. In *Foundations of Software Tech. and Theoretical Comp. Sci.*, pages 11:1–11:16, 2016. Technical report at http://arxiv.org/abs/1611.08992.

[BNN22] Anindya Banerjee, Ramana Nagasamudram, and David A. Naumann. Making relational Hoare logic alignment complete. *CoRR*, abs/2212.10338, 2022. doi:10.48550/ARXIV.2212.10338.

[BNNN22] Anindya Banerjee, Ramana Nagasamudram, David A. Naumann, and Mohammad Nikouei. A relational program logic with data abstraction and dynamic framing. *ACM Transactions on Programming Languages and Systems*, 44(4):25:1–25:136, 2022. doi:10.1145/3551497.

[CBG+18] Qinxiang Cao, Lennart Beringer, Samuel Gruetter, Josiah Dodds, and Andrew W Appel. VST-Floyd: A separation logic tool to verify correctness of C programs. *Journal of Automated Reasoning*, 61(1):367–422, 2018. doi:10.1007/s10817-018-9457-5.

[CKS96] Ernie Cohen, Dexter Kozen, and Frederick Smith. The complexity of Kleene algebra with tests. Technical Report TR96-1598, Cornell University, 1996.

[CMP20] Martin Clochard, Claude Marché, and Andrei Paskevich. Deductive verification with ghost monitors. *Proc. ACM Program. Lang.*, 4(POPL):2:1–2:26, 2020. doi:10.1145/3371070.

[Coo78] Stephen A. Cook. Soundness and completeness of an axiom system for program verification. *SIAM J. Comput.*, 7(1):70–90, 1978. doi:10.1137/0207005.

[CPSA19] Berkeley R. Churchill, Oded Padon, Rahul Sharma, and Alex Aiken. Semantic program alignment for equivalence checking. In *ACM Conf. on Program. Lang. Design and Implementation*, pages 1027–1040, 2019. doi:10.1145/3314221.3314596.

[dBdBZ80] Jacobus W. de Bakker, Arie de Bruin, and Jeffrey Zucker. *Mathematical Theory of Program Correctness*. Prentice-Hall international series in computer science. Prentice Hall, 1980.

[DFD22] Emanuele D'Osualdo, Azadeh Farzan, and Derek Dreyer. Proving hypersafety compositionally. *Proc. ACM Program. Lang.*, 6(OOPSLA2), 2022. doi:10.1145/3563298.

[dRE98] Willem-Paul de Roever and Kai Engelhardt. *Data Refinement: Model-Oriented Proof Methods and their Comparison*. Cambridge University Press, 1998.

[DS90] Edsger W. Dijkstra and Carel S. Scholten. *Predicate Calculus and Program Semantics*. Texts and Monographs in Computer Science. Springer, 1990. doi:10.1007/978-1-4612-3228-5.

[DYZD22] Robert Dickerson, Qianchuan Ye, Michael K. Zhang, and Benjamin Delaware. RHLE: modular deductive verification of relational $\forall \exists$ properties. In *Asian Symposium on Programming Languages and Systems*, volume 13658 of *LNCS*, pages 67–87, 2022. doi:10.1007/978-3-031-21037-2_4.

[EMH18] Marco Eilers, Peter Müller, and Samuel Hitz. Modular product programs. In *Programming Languages and Systems, European Symposium on Programming*, pages 502–529, 2018. doi:10.1007/978-3-319-89884-1_18.

[EMH20] Marco Eilers, Peter Müller, and Samuel Hitz. Modular product programs. *ACM Trans. Program. Lang. Syst.*, 42(1):3:1–3:37, 2020. doi:10.1145/3324783.

[FGP16] Jean-Christophe Filliâtre, Léon Gondelman, and Andrei Paskevich. The spirit of ghost code. *Formal Methods in System Design*, 48(3):152–174, 2016. doi:10.1007/s10703-016-0243-x.

[FKB18] Dan Frumin, Robbert Krebbers, and Lars Birkedal. Reloc: A mechanised relational logic for fine-grained concurrency. In *IEEE Symp. on Logic in Computer Science*, pages 442–451, 2018. doi:10.1145/3209108.3209174.

[Flo67] Robert Floyd. Assigning meaning to programs. In *Symp. on Applied Math. 19, Math. Aspects of Comp. Sci.*, pages 19–32. Amer. Math. Soc., 1967.

[Fra83] Nissim Francez. Product properties and their direct verification. *Acta Informatica*, 20:329–344, 1983. doi:10.1007/BF00264278.

[FSDF93] Cormac Flanagan, Amr Sabry, Bruce F. Duba, and Matthias Felleisen. The essence of compiling with continuations. In *ACM Conf. on Program. Lang. Design and Implementation*, pages 237–247, 1993. doi:10.1145/155090.155113.

[FV19] Azadeh Farzan and Anthony Vandikas. Automated hypersafety verification. In *Computer Aided Verification*, volume 11561 of *LNCS*, pages 200–218, 2019. doi:10.1007/978-3-030-25540-4_11.

[GKM14] Niels Bjørn Bugge Grathwohl, Dexter Kozen, and Konstantinos Mamouras. KAT + B! In *Joint Meeting of the EACSL Annual Conference on Computer Science Logic (CSL) and the ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 44:1–44:10, 2014. doi:10.1145/2603088.2603095.

[GS08] Benny Godlin and Ofer Strichman. Inference rules for proving the equivalence of recursive procedures. *Acta Inf.*, 45(6):403–439, 2008. doi:10.1007/s00236-008-0075-2.

[HHS93] C. A. R. Hoare, Jifeng He, and Augusto Sampaio. Normal form approach to compiler design. *Acta Informatica*, 30(8):701–739, 1993. doi:10.1007/BF01191809.

[HKLR13] Chris Hawblitzel, Ming Kawaguchi, Shuvendu K. Lahiri, and Henrique Rebêlo. Towards modularly comparing programs using automated theorem provers. In *CADE*, volume 7898 of *LNCS*, pages 282–299, 2013. doi:10.1007/978-3-642-38574-2_20.

[HKT00] David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic Logic*. MIT Press, Cambridge, MA, 2000.

[Hoa71] C. A. R. Hoare. Procedures and parameters: An axiomatic approach. In *Symposium on Semantics of Algorithmic Languages*, volume 188 of *Lecture Notes in Mathematics*, pages 102–116. 1971. doi:10.1007/BFb0059696.

[Hoa78] C. A. R. Hoare. Some properties of predicate transformers. *Journal of the ACM*, 25:461–480, 1978.

[ISV24] Shachar Itzhaky, Sharon Shoham, and Yakir Vizel. Hyperproperty verification as CHC satisfiability. In *Programming Languages and Systems, European Symposium on Programming*, volume 14577 of *LNCS*, pages 212–241, 2024. doi:10.1007/978-3-031-57267-8_9.

[JKJ+18] Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Ales Bizjak, Lars Birkedal, and Derek Dreyer. Iris from the ground up: A modular foundation for higher-order concurrent separation logic. *J. Funct. Program.*, 28:e20, 2018. doi:10.1017/S0956796818000151.

[Kle99] Thomas Kleymann. Hoare logic and auxiliary variables. *Formal Aspects Comput.*, 11(5):541–566, 1999. doi:10.1007/S001650050057.

[Koz97] Dexter Kozen. Kleene algebra with tests. *ACM Transactions on Programming Languages and Systems*, 19(3):427–443, 1997. doi:10.1145/256167.256195.

[Koz00] Dexter Kozen. On Hoare logic and Kleene algebra with tests. *ACM Trans. Comput. Log.*, 1(1):60–76, 2000. doi:10.1145/343369.343378.

[KS96] Dexter Kozen and Frederick Smith. Kleene algebra with tests: Completeness and decidability. In *International Workshop on Computer Science Logic*, volume 1258 of *LNCS*, pages 244–259, 1996. doi:10.1007/3-540-63172-0_43.

[KSF13] Máté Kovács, Helmut Seidl, and Bernd Finkbeiner. Relational abstract interpretation for the verification of 2-hypersafety properties. In *ACM Computer and Communications Security*, pages 211–222, 2013. doi:10.1145/2508859.2516721.

[KT01] Dexter Kozen and Jerzy Tiuryn. On the completeness of propositional Hoare logic. *Inf. Sci.*, 139(3-4):187–195, 2001. doi:10.1016/S0020-0255(01)00164-5.

[LS21] Leslie Lamport and Fred B. Schneider. Verifying hyperproperties with TLA. In *IEEE Computer Security Foundations*, pages 1–16, 2021. doi:10.1109/CSF51468.2021.00012.

[Nau06] David A. Naumann. From coupling relations to mated invariants for secure information flow. In *European Symposium on Research in Computer Security*, volume 4189 of *LNCS*, pages 279–296, 2006.

[NBG13] Aleksandar Nanevski, Anindya Banerjee, and Deepak Garg. Dependent type theory for verification of information flow and access control policies. *ACM Trans. Program. Lang. Syst.*, 35(2):6, 2013. doi:10.1145/2491522.2491523.

[NBN23] Ramana Nagasamudram, Anindya Banerjee, and David A. Naumann. Alignment complete relational Hoare logics for some and all. *CoRR*, abs/2307.10045v5, 2023. Version v5 has details elided in later versions, and discussion of entailment completeness. doi:10.48550/arXiv.2307.10045.

[Nip02] Tobias Nipkow. Hoare logics for recursive procedures and unbounded nondeterminism. In *Computer Science Logic*, volume 2471 of *LNCS*, pages 103–119, 2002. doi:10.1007/3-540-45793-3_8.

[NN21] Ramana Nagasamudram and David A. Naumann. Alignment completeness for relational Hoare logics. In *IEEE Symp. on Logic in Computer Science*, 2021. Extended version at https://arxiv.org/abs/2101.11730.

[O'H20] Peter W O'Hearn. Incorrectness logic. *Proc. ACM Program. Lang.*, 4(POPL):10:1–10:32, 2020. doi:10.1145/3371078.

[PdAC+22] Benjamin C. Pierce, Arthur Azevedo de Amorim, Chris Casinghino, Marco Gaboardi, Michael Greenberg, Cătălin Hriţcu, Vilhelm Sjöberg, Andrew Tolmach, and Brent Yorgey. *Programming Language Foundations*, volume 2 of *Software Foundations*. Electronic textbook, 2022. Version 6.2, http://softwarefoundations.cis.upenn.edu.

[PFG18] Lauren Pick, Grigory Fedyukovich, and Aarti Gupta. Exploiting synchrony and symmetry in relational verification. In *Computer Aided Verification*, volume 10981 of *LNCS*, pages 164–182, 2018. doi:10.1007/978-3-319-96145-3_9.

[Rin99] Martin Rinard. Credible compilation. Technical Report MIT-LCS-TR-776, MIT, March 1999. URL: https://people.csail.mit.edu/rinard/paper/credibleCompilation.html.

[RM99] Martin Rinard and Darko Marinov. Credible compilation with pointers. In *Proceedings of the FLoC Workshop on Run-Time Result Verification*, 1999. URL: https://people.csail.mit.edu/rinard/paper/credibleCompilation.html.

[SD16] Marcelo Sousa and Isil Dillig. Cartesian Hoare logic for verifying k-safety properties. In *ACM Conf. on Program. Lang. Design and Implementation*, pages 57–69, 2016. doi:10.1145/2908080.2908092.

[SGSV19] Ron Shemer, Arie Gurfinkel, Sharon Shoham, and Yakir Vizel. Property directed self composition. In *Computer Aided Verification*, volume 11561 of *LNCS*, pages 161–179, 2019. doi:10.1007/978-3-030-25540-4_9.

[SM03] Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE J. Selected Areas in Communications*, 21(1):5–19, January 2003.

[SV16] Ofer Strichman and Maor Veitsman. Regression verification for unbalanced recursive functions. In *FM 2016: Formal Methods*, pages 645–658, 2016.

[TA05] Tachio Terauchi and Alex Aiken. Secure information flow as a safety problem. In *International Static Analysis Symposium*, volume 3672 of *LNCS*, pages 352–367, 2005. doi:10.1007/11547662_24.

[UTK21] Hiroshi Unno, Tachio Terauchi, and Eric Koskinen. Constraint-based relational verification. In *Computer Aided Verification*, volume 12759 of *LNCS*, pages 742–766, 2021. doi:10.1007/978-3-030-81685-8_35.

[WDLC18] Yuepeng Wang, Isil Dillig, Shuvendu K. Lahiri, and William R. Cook. Verifying equivalence of database-driven applications. *Proc. ACM Program. Lang.*, 2(POPL):56:1–56:29, 2018. doi:10.1145/3158144.

[Win93] Glynn Winskel. *The Formal Semantics of Programming Languages - an Introduction*. Foundation of computing series. MIT Press, 1993.

[Yan07] Hongseok Yang. Relational separation logic. *Theoretical Computer Science*, 375(1-3):308–334, 2007. doi:10.1016/j.tcs.2006.12.036.

Appendix A. Additional definitions

Figure 22 defines the label of a command, recursing only in the sequence case. Figure 23 presents the pre-post relation denoted by a command, defined inductively in big-step style.

$$\mathsf{lab}(\mathsf{skip}^n) = \mathsf{lab}(x :=^n e) = \mathsf{lab}(\mathsf{hav}^n\ x) = \mathsf{lab}(\mathsf{if}^n\ gcs\ \mathsf{fi}) = \mathsf{lab}(\mathsf{do}^n\ gcs\ \mathsf{od}) = n$$
$$\mathsf{lab}(c; d) = \mathsf{lab}(c)$$

Figure 22: Definition of $\mathsf{lab}(c)$.

$$[\![\,\mathsf{skip}^n\,]\!]\,s\,s \qquad \frac{t = s[x \mapsto [\![\,e\,]\!](s)]}{[\![\,x := e\,]\!]\,s\,t} \qquad \frac{v \in \mathbb{Z}}{[\![\,\mathsf{hav}\ x\,]\!]\,s\,s[x \mapsto v]} \qquad \frac{[\![\,c\,]\!]\,s\,u \qquad [\![\,d\,]\!]\,u\,t}{[\![\,c; d\,]\!]\,s\,t}$$

$$\frac{e \to b \text{ is in } gcs \qquad s \in [\![\,e\,]\!] \qquad [\![\,b\,]\!]\,s\,t}{[\![\,\mathsf{if}\ gcs\ \mathsf{fi}\,]\!]\,s\,t}$$

$$\frac{e \to b \text{ is in } gcs \qquad s \in [\![\,e\,]\!] \qquad [\![\,b\,]\!]\,s\,u \qquad [\![\,\mathsf{do}\ gcs\ \mathsf{od}\,]\!]\,u\,t}{[\![\,\mathsf{do}\ gcs\ \mathsf{od}\,]\!]\,s\,t} \qquad \frac{s \in [\![\,\neg\mathsf{enab}(gcs)\,]\!]}{[\![\,\mathsf{do}\ gcs\ \mathsf{od}\,]\!]\,s\,s}$$

Figure 23: Denotational semantics $[\![\,c\,]\!]$ (omitting labels which are unconstrained).

Appendix B. Axioms for normal form equivalence

For the sake of straightforward presentation, command equivalence has been formulated in terms of a single fixed set of hypotheses that axiomatize simple assignments and boolean expressions (Defs. 5.3 and 5.4). However, many useful equivalences require no hypotheses. Only a few specific hypotheses are needed to prove the normal form theorem, and in this section we spell those out.

Observe that if $c \,/\, f \hookrightarrow gcs$ then there is a finite set of instances of this relation that supports the fact, namely the normal forms of subprograms of c. This enables us to define a set of axioms that are useful for reasoning about a given program c and its normal form. The definition is parameterized on c and on the choice of a pc variable and final label.

Definition B.1. The ***normal form axioms***, $\mathsf{nfax}(pc, c, f)$, comprises the following set of equations.

(diffTest): $(\!|\,?i\,|\!)\,;(\!|\,?j\,|\!) = 0$ for i and j in $\mathsf{labs}(c) \cup \{f\}$ such that $i \neq j$
(setTest): $(\!|\,!i\,|\!)\,;(\!|\,?i\,|\!) = (\!|\,!i\,|\!)$ for i in $\mathsf{labs}(c) \cup \{f\}$
(totIf): $\neg(\!|\,\mathsf{enab}(gcs)\,|\!) = 0$ for every $\mathsf{if}\ gcs\ \mathsf{fi}$ that occurs in c
(testCommuteAsgn): $(\!|\,?i\,|\!);(\!|\,x := e\,|\!) = (\!|\,x := e\,|\!);(\!|\,?i\,|\!)$ for every $x := e$ in c such that $x \not\equiv pc$, and every i in $\mathsf{labs}(c) \cup \{f\}$
(testCommuteHav): $(\!|\,?i\,|\!);(\!|\,\mathsf{hav}\ x\,|\!) = (\!|\,\mathsf{hav}\ x\,|\!);(\!|\,?i\,|\!)$ for every $\mathsf{hav}\ x$ in c such that $x \not\equiv pc$, and every i in $\mathsf{labs}(c) \cup \{f\}$

Lemma B.2. For any pc, c, f, the equations $\mathsf{nfax}(pc, c, f)$ follow by KAT reasoning from the equations of Hyp.

Proof. Observe that, in light of Lemma 5.2, the set Hyp can characterized in terms of $\mathfrak{L}$ as follows:

- $(\!|\, e \,|\!) = 0$ for every boolean expression e such that $(\!|\, e \,|\!)^{\mathfrak{L}} = \emptyset$
- $(\!|\, e_0 \,|\!); (\!|\, x := e \,|\!); \neg(\!|\, e_1 \,|\!) = 0$ for every x, e, e_0, e_1 such that $(\!|\, e_0 \,|\!)^{\mathfrak{L}}; (\!|\, x := e \,|\!)^{\mathfrak{L}}; \neg(\!|\, e_1 \,|\!)^{\mathfrak{L}} = \emptyset$.
- $(\!|\, e_0 \,|\!); (\!|\, \mathsf{hav}\ x \,|\!); \neg(\!|\, e_1 \,|\!) = 0$ for every x, e, e_0, e_1 such that $(\!|\, e_0 \,|\!)^{\mathfrak{L}}; (\!|\, \mathsf{hav}\ x \,|\!)^{\mathfrak{L}}; \neg(\!|\, e_1 \,|\!)^{\mathfrak{L}} = \emptyset$.

Now we proceed.

(diffTest) If $i \neq j$ then $(\!|\, ?i \wedge ?j \,|\!)^{\mathfrak{L}} = \emptyset$, so Hyp contains $(\!|\, ?i \wedge ?j \,|\!) = 0$. So we have $(\!|\, ?i \,|\!)\ ;\ (\!|\, ?j \,|\!) = (\!|\, ?i \wedge ?j \,|\!) = 0$ using the definition of $(\!|\, - \,|\!)$.

(setTest) Hyp contains $(\!|\, \mathsf{skip} \,|\!); (\!|\, !i \,|\!)\ ;\ \neg(\!|\, ?i \,|\!) = 0$, and $(\!|\, \mathsf{skip} \,|\!)$ is 1. Now observe that $(\!|\, !i \,|\!) = (\!|\, !i \,|\!)\ ;\ ((\!|\, ?i \,|\!) + \neg(\!|\, ?i \,|\!)) = (\!|\, !i \,|\!)\ ;\ (\!|\, ?i \,|\!) + 1; (\!|\, !i \,|\!)\ ;\ \neg(\!|\, ?i \,|\!) = (\!|\, !i \,|\!)\ ;\ (\!|\, ?i \,|\!)$ using KAT laws and $1; (\!|\, !i \,|\!)\ ;\ \neg(\!|\, ?i \,|\!) = 0$.

(totIf) If $\mathsf{if}\ gcs\ \mathsf{fi}$ that occurs in c then we have $(\neg(\!|\, \mathsf{enab}(gcs) \,|\!))^{\mathfrak{L}} = \emptyset$ as a consequence of the $\mathsf{total\text{-}if}$ condition (Def. 3.1). So the equation $\neg(\!|\, \mathsf{enab}(gcs) \,|\!) = 0$ is in Hyp.

(testCommuteAsgn) The equation $(\!|\, ?i \,|\!); (\!|\, x := e \,|\!) = (\!|\, x := e \,|\!); (\!|\, ?i \,|\!)$, is equivalent to the conjunction of $(\!|\, ?i \,|\!); (\!|\, x := e \,|\!); \neg(\!|\, ?i \,|\!) = 0$ and $\neg(\!|\, ?i \,|\!); (\!|\, x := e \,|\!); (\!|\, ?i \,|\!) = 0$ using KAT laws. By definition of $(\!|\, - \,|\!)$ and boolean algebra, $\neg(\!|\, ?i \,|\!); (\!|\, x := e \,|\!); (\!|\, ?i \,|\!) = 0$ is equivalent to $(\!|\, \neg ?i \,|\!); (\!|\, x := e \,|\!); \neg(\!|\, \neg ?i \,|\!) = 0$. Both $?i \Rightarrow (?i)^x_e$ and $\neg ?i \Rightarrow (\neg ?i)^x_e$ are valid because $pc \not\equiv x$, so Hyp contains both equations $(\!|\, ?i \,|\!); (\!|\, x := e \,|\!); \neg(\!|\, ?i \,|\!) = 0$ and $(\!|\, \neg ?i \,|\!); (\!|\, x := e \,|\!); \neg(\!|\, \neg ?i \,|\!) = 0$.

(testCommuteHav) The equation $(\!|\, ?i \,|\!); (\!|\, \mathsf{hav}\ x \,|\!) = (\!|\, \mathsf{hav}\ x \,|\!); (\!|\, ?i \,|\!)$, is equivalent to the conjunction of $(\!|\, ?i \,|\!); (\!|\, \mathsf{hav}\ x \,|\!); \neg(\!|\, ?i \,|\!) = 0$ and $\neg(\!|\, ?i \,|\!); (\!|\, \mathsf{hav}\ x \,|\!); (\!|\, ?i \,|\!) = 0$ using KAT laws. By definition of $(\!|\, - \,|\!)$ and boolean algebra, $\neg(\!|\, ?i \,|\!); (\!|\, \mathsf{hav}\ x \,|\!); (\!|\, ?i \,|\!) = 0$ is equivalent to $(\!|\, \neg ?i \,|\!); (\!|\, \mathsf{hav}\ x \,|\!); \neg(\!|\, \neg ?i \,|\!) = 0$. Both $(\!|\, ?i \,|\!); (\!|\, \mathsf{hav}\ x \,|\!); \neg(\!|\, ?i \,|\!) = 0$ and $(\!|\, \neg ?i \,|\!); (\!|\, \mathsf{hav}\ x \,|\!); \neg(\!|\, \neg ?i \,|\!) = 0$ are instances of the form $(\!|\, e \,|\!); (\!|\, \mathsf{hav}\ x \,|\!); \neg(\!|\, e \,|\!) = 0$, and we are assuming $x \not\equiv pc$ so they are both in Hyp and we are done. □

Remark B.3. $\mathsf{nfax}(pc, c, f)$ is finite, and every equation is equivalent to one of the form $KE = 0$. So entailments of the form $\mathsf{nfax}(pc, c, f) \vdash KE_0 = KE_1$ are decidable in PSPACE [CKS96]. □

Lemma B.4. All equations in $\mathsf{nfax}(pc, c, f)$ are true in $\mathfrak{L}$, for any pc, c, f.

This holds because the equations follow from Hyp (Lemma B.2) and the equations in Hyp are true in $\mathfrak{L}$ (Lemma 5.6). Note that c need not be ok for this result. However, truth of (totIf) does depend on the $\mathsf{total\text{-}if}$ condition (Def. 3.1).

Lemma B.5. The following hold for any c, pc, f such that pc does not occur in c.

(diffTestNeg) $\mathsf{nfax}(pc, c, f) \vdash (\!|\, ?i \,|\!) = (\!|\, ?i \,|\!); \neg(\!|\, ?j \,|\!)$ and $(\!|\, ?i \,|\!) \leq \neg(\!|\, ?j \,|\!)$
for $i \neq j$ with i, j in $\mathsf{labs}(c) \cup \{f\}$.

(nf-enab-labs) $\mathsf{nfax}(pc, c, f) \vdash (\!|\, \mathsf{enab}(gcs) \,|\!) = (\!|\, (\vee i\ :\ i \in \mathsf{labs}(c)\ :\ ?i) \,|\!)$
if $\mathsf{okf}(c, f)$ and $c\ /\ f \hookrightarrow gcs$.

(nf-lab-enab) $\mathsf{nfax}(pc, c, f) \vdash (\!|\, ?\mathsf{lab}(c) \,|\!) \leq (\!|\, \mathsf{enab}(gcs) \,|\!)$
if $\mathsf{okf}(c, f)$ and $c\ /\ f \hookrightarrow gcs$.

(nf-enab-disj) $\mathsf{nfax}(pc, c, f) \vdash (\!|\, \mathsf{enab}(gcs_0) \,|\!); (\!|\, \mathsf{enab}(gcs_1) \,|\!) = 0$
if $\mathsf{okf}(c, f)$ and gcs_0 and gcs_1 are the normal form bodies of disjoint subprograms of c.

(nf-enab-corr) $\mathsf{nfax}(pc, c, f) \vdash$
$(\!|\, \mathsf{enab}(gcs) \,|\!); (\!|\, gcs \,|\!) = (\!|\, \mathsf{enab}(gcs) \,|\!); (\!|\, gcs \,|\!); ((\!|\, \mathsf{enab}(gcs) \,|\!) + (\!|\, ?f \,|\!))$
if $\mathsf{okf}(c, f)$ and $c\ /\ f \hookrightarrow gcs$ and $pc \notin vars(c)$.

(nf-enab-corr) can be abbreviated $\mathsf{nfax}(pc, c, f) \vdash (\!|\, gcs \,|\!) : (\!|\, \mathsf{enab}(gcs) \,|\!) \rightsquigarrow (\!|\, \mathsf{enab}(gcs) \,|\!) + (\!|\, ?f \,|\!)$.

Proof. (diffTestNeg) follows from (diffTest) using boolean algebra.

$$\frac{c \,/\, \mathsf{lab}(d) \hookrightarrow gcs_0 \qquad d \,/\, f \hookrightarrow gcs_1}{c; d \,/\, f \hookrightarrow gcs_0 \,[\!]\, gcs_1}$$

$$\frac{\mathsf{bodies}(gcs) \,/\, f \hookrightarrow nfs}{\mathsf{if}^n\ gcs\ \mathsf{fi} \,/\, f \hookrightarrow map((\lambda(e \to c)\,.\,?n \wedge e \to \,!\mathsf{lab}(c)),\, gcs) \,[\!]\, \mathsf{concat}(nfs)}$$

$$\frac{\mathsf{bodies}(gcs) \,/\, f \hookrightarrow nfs}{\mathsf{do}^n\ gcs\ \mathsf{od} \,/\, f \hookrightarrow map((\lambda(e \to c)\,.\,?n \wedge e \to \,!\mathsf{lab}(c)),\, gcs) \,[\!]\, ?n \wedge \neg\mathsf{enab}(gcs) \to !f \,[\!]\, \mathsf{concat}(nfs)}$$

$$\frac{c \,/\, f \hookrightarrow gcs}{[c] \,/\, f \hookrightarrow [gcs]} \qquad\qquad \frac{c \,/\, f \hookrightarrow gcs \qquad cs \,/\, f \hookrightarrow nfs}{c :: cs \,/\, f \hookrightarrow gcs :: nfs}$$

Figure 24: Normal form bodies for if and do.

(nf-enab-labs) By rule induction on the normal form judgment.
For $\mathsf{do}^n\ gcs_0\ \mathsf{od}$, the normal form body is explicitly defined to include, not only a command with guard $?n \wedge e$ for each $e \to d$ in gcs_0 but also a command with guard $?n \wedge \neg\mathsf{enab}(\ldots)$ for their complement. The disjunction of their enabling conditions simplifies to $?n$. By the normal form rule and induction hypothesis, the enabling condition for the collected normal form bodies is the disjunction over their label tests, hence the result.
For $\mathsf{if}^n\ gcs_0\ \mathsf{fi}$, the normal form body includes disjuncts $?n \wedge e$ for each of the guard expressions e in gcs_0. Consider the case of two branches, i.e., gcs_0 is $e_0 \to d_0 \,[\!]\, e_1 \to d_1$, the disjunction is $(?n \wedge e_0) \vee (?n \wedge e_1)$. Observe

$$\begin{array}{lll} & (\!|\, (?n \wedge e_0) \vee (?n \wedge e_1) \,|\!) & \\ = & (\!|\, ?n \,|\!); (\!|\, e_0 \,|\!) + (\!|\, ?n \,|\!); (\!|\, e_1 \,|\!) & \text{def } (\!|-|\!) \\ = & (\!|\, ?n \,|\!); ((\!|\, e_0 \,|\!) + (\!|\, e_1 \,|\!)) & \text{KAT law} \\ = & (\!|\, ?n \,|\!); (\!|\, e_0 \vee e_1 \,|\!) & \text{def } (\!|-|\!) \\ = & (\!|\, ?n \,|\!) & \text{axiom (totIf), unit law} \end{array}$$

The rest of the argument is like for do. The general case (multiple branches) is similar.

(nf-lab-enab) follows from (nf-enab-labs), using boolean algebra and definition of $(\!|-|\!)$ (distributing over $\vee$).

(nf-enab-disj) is proved by induction on normal form, using that by ok subprograms have distinct labels, and (nf-enab-labs) and (diffTest).

(nf-enab-corr) The proof is by induction on the normal form relation and by calculation using KAT laws and the axioms. It uses (testCommuteAsgn) so pc must not occur in c. □

Lemma 5.14. If $\mathsf{okf}(c, f)$ and $c \,/\, f \hookrightarrow gcs$ then $[\![\, \mathsf{enab}(gcs) \,]\!] = [\![\, (\vee i \,:\, i \in \mathsf{labs}(c) \,:\, ?i) \,]\!]$.

Proof. Follows by definitions from provability lemma (nf-enab-labs) of Lemma B.5, using Lemmas B.4 and 5.2. □

Appendix C. Normal form bodies

Figure 24 gives the general cases of normal forms for if- and do-commands, which subsume the special cases given in Figure 13. The definition of normal form bodies for commands is mutually inductive with the definition of normal form body list for nonempty command

lists, which is given by the last two rules in Figure 24. The rules use square brackets for singleton list and :: for list cons. They use $\mathsf{bodies}(gcs)$ and $\mathsf{concat}(nfs)$ defined by

$$\begin{array}{lcl} \mathsf{bodies}(e \to c) & = & [c] \\ \mathsf{bodies}(e \to c \,[\!]\, gcs) & = & c :: \mathsf{bodies}(gcs) \\ \mathsf{concat}([gcs]) & = & gcs \\ \mathsf{concat}(gcs : nfs) & = & gcs \,[\!]\, \mathsf{concat}(nfs) \end{array}$$

Appendix D. Additional proofs

Theorem 9.7 (semantic soundness and completeness of filtered automata)**.** We have $A, A' \models \mathcal{P} \overset{\exists}{\approx\!>} \mathcal{Q}$ iff there are L, R, J, K and a valid annotation an of $\prod(A, A', L, R, J, K)$ for $\mathcal{P} \rightsquigarrow \mathcal{Q}$ such that K is an adequate filtering for an. Moreover, if A, A' act on variable stores, and $A, A', \mathcal{P}, \mathcal{Q}$ are finitely supported, then there are finitely supported such L, R, J, K, an (and witness V).

Proof. We prove the main statement by mutual implication, and afterwards consider the issue of finite support.

(right implies left) If an is a valid annotation of $\prod(A, A', L, R, J, K)$ for $\mathcal{P} \overset{\exists}{\approx\!>} \mathcal{Q}$ and K is an adequate filtering for an, then by Lemma 9.6 the product is $\mathcal{P}$-adequate. Since an is valid for $\mathcal{P} \rightsquigarrow \mathcal{Q}$ we have $\prod(A, A', L, R, J, K) \models \mathcal{P} \rightsquigarrow \mathcal{Q}$ by Lemma 4.6. So by Lemma 9.4 we have $A, A' \models \mathcal{P} \overset{\exists}{\approx\!>} \mathcal{Q}$.

(left implies right) Suppose $A, A' \models \mathcal{P} \overset{\exists}{\approx\!>} \mathcal{Q}$. The idea is to use a left-first sequential alignment automaton such that, once the left-only execution has finished, with final store t, the filter condition only keeps states that are in some terminating run of A' that ends in a store t' such that $(t, t') \in \mathcal{Q}$.

To make this precise we need a few definitions. Let $\mathsf{Traces}(A)$ be the set of all nonempty and finite sequences of states of automata A that are consecutive under A's transition relation. In what follows, we use α, β to range over traces, and write $start(\alpha)$ and $end(\alpha)$ for the first and last states of α. For $s \in Sto$, $n' \in Ctrl'$, and $s' \in Sto'$, define $termQtr(s, n', s')$ to be the set of terminated traces of A' that begin at (n', s') and end in a state t' for which $(s, t') \in \mathcal{Q}$. That is,

$$termQtr(s, n', s') \;\hat{=}\; \{\alpha \in \mathsf{Traces}(A') \mid start(\alpha) = (n', s') \wedge \exists t'. end(\alpha) = (fin', t') \wedge (s, t') \in \mathcal{Q}\}.$$

Define the restriction to traces of minimum length, as follows.[30]

$$shortQtr(s, n', s') \;\hat{=}\; \{\alpha \in termQtr(s, n', s') \mid \mathsf{len}(\alpha) = \min(\mathsf{map}(\mathsf{len}, termQtr(s, n', s')))\}$$

We need the following facts:

(i) Suppose α is in $termQtr(t, m', s')$ and (n', t') is in α. Then $termQtr(t, n', t')$ is nonempty.
(ii) $termQtr(t, n', s')$ is nonempty iff $shortQtr(t, n', s')$ is nonempty.
(iii) If (s, s') is in $\mathcal{P}$ and there is an A-trace from $(init, s)$ to (fin, t), then $termQtr(t, init', s') \neq \emptyset$.

[30] One may wonder how min is defined, in case $termQtr(s, n', s')$ is empty (and hence so is $\mathsf{map}(\mathsf{len}, termQtr(s, n', s'))$), but that does not matter because in this case $shortQtr(s, n', s')$ is empty.

Both (i) and (ii) follow directly by definitions. Fact (iii) is a consequence of the theorem's assumption that $A, A' \models \mathcal{P} \overset{\exists}{\approx\!>} \mathcal{Q}$. If $(s, s') \in \mathcal{P}$ and there is an A-trace from (init, s) to (fin, t) then by $A, A' \models \mathcal{P} \overset{\exists}{\approx\!>} \mathcal{Q}$ there is some t' and A'-trace α from (init', s') to (fin', t') with $(t, t') \in \mathcal{Q}$. By definition, α is in $\mathit{termQtr}(t, \mathit{init}', s')$.

Next we define L, R, J as needed for left-first sequential alignment product. So we let $J \mathrel{\hat{=}} \emptyset$. In case the transition relations $\mapsto, \mapsto'$ of A, A' are total (as in the case of program automata) it suffices to define $L \mathrel{\hat{=}} [*|\mathit{init}'] \wedge \neg[\mathit{fin}|*]$ and $R \mathrel{\hat{=}} [\mathit{fin}|*] \wedge \neg[\mathit{fin}|\mathit{fin}']$. In general, to ensure liveness in the sense required by Def. 4.2, we need

$$L \mathrel{\hat{=}} [*|\mathit{init}'] \wedge \neg[\mathit{fin}|*] \wedge (\mathsf{dom}(\mapsto) \times \mathit{states}')$$
$$R \mathrel{\hat{=}} [\mathit{fin}|*] \wedge \neg[\mathit{fin}|\mathit{fin}'] \wedge (\mathit{states} \times \mathsf{dom}(\mapsto'))$$

Define K by[31]

$$((n, n'), (s, s')) \in K \quad \mathrel{\hat{=}} \quad n' = \mathit{init}' \vee (n = \mathit{fin} \wedge \mathit{termQtr}(s, n', s') \neq \emptyset)$$

Let an be the canonical semantic annotation for $\mathcal{P}$, i.e., $\mathit{an}(n, n')$ is the strongest invariant of $\prod(A, A', L, R, J, K)$ with respect to $\mathcal{P}$. To be precise, for any n, n':

$$\mathit{an}(n, n') \mathrel{\hat{=}} \{(t, t') \mid \exists s, s'.\, (s, s') \in \mathcal{P} \wedge ((\mathit{init}, \mathit{init}'), (s, s')) \mapsto_K^* ((n, n'), (t, t'))\}$$

By construction, this annotation satisfies $\mathcal{P} \Rightarrow \mathit{an}(\mathit{init}, \mathit{init}')$ and moreover it satisfies the verification conditions, i.e., it is valid. It remains to show that $\mathit{an}(\mathit{fin}, \mathit{fin}') \Rightarrow \mathcal{Q}$ (i.e., it is an annotation for $\mathcal{P} \rightsquigarrow \mathcal{Q}$). By definition of K, any stores t, t' for which $((\mathit{fin}, \mathit{fin}'), (t, t'))$ is reachable via $\mapsto_K$ satisfy $\mathcal{Q}$. Because $\mathit{an}(\mathit{fin}, \mathit{fin}')$ is exactly these stores, we conclude that $\mathit{an}(\mathit{fin}, \mathit{fin}') \Rightarrow \mathcal{Q}$.

It remains to show K is an adequate filtering for an.

For the enabled condition in the Def. 9.5, we need for any n, n', $\breve{an}(n, n') \Rightarrow L \vee R \vee J \vee [\mathit{fin}|\mathit{fin}']$. By construction, an invariant of the product is $[*|\mathit{init}'] \vee [\mathit{fin}|*]$. So by definition of an we get $\breve{an}(n, n') \Rightarrow L \vee R \vee J \vee [\mathit{fin}|\mathit{fin}']$ for any n, n'.

The left-permissiveness condition in Def. 9.5 holds because K allows all left steps. Joint-productivity holds because J is empty. To show right-productivity we need a measure V.

To define V, first observe that all elements of $\mathit{shortQtr}(t, n', s')$ have the same length, if there are any elements. So we can write $\mathsf{len}(\mathit{shortQtr}(t, n', s'))$ for that number, provided the set is not empty. Now define V, a function from product states to naturals, by

$$\begin{array}{lcll} V((n, n'), (s, s')) & = & 0 & \text{if } n \neq \mathit{fin} \\ V((\mathit{fin}, n'), (t, s')) & = & \mathsf{len}(\mathit{shortQtr}(t, n', s')) & \text{if } \mathit{shortQtr}(t, n', s') \neq \emptyset \\ V((\mathit{fin}, n'), (t, s')) & = & 0 & \text{otherwise} \end{array}$$

For right productivity, consider any product state $((n, n'), (t, t'))$ and assume $(t, t') \in \mathit{an}(n, n')$ and $((n, n'), (t, t')) \in R$. We must show there is some m' and u', and a transition from $(n', t') \mapsto' (m', u')$ such that

- $((n, m'), (t, u')) \in K(n, m')$, and
- $V((n, m'), (t, u')) < V((n, n'), (t, t'))$

Observe that in general, from $(t, t') \in \mathit{an}(n, n')$ we have that $((n, n'), (t, t'))$ is reachable from some $((\mathit{init}, \mathit{init}'), (s, s'))$ with $(s, s') \in \mathcal{P}$.

From $((n, n'), (t, t')) \in R$ we have that $n = \mathit{fin}$ and $n' \neq \mathit{fin}'$. Now, in general $K \vee [\mathit{init}|\mathit{init}']$ is an invariant of an LRJK-automaton, so because $\mathit{fin} \neq \mathit{init}$ we have that

[31] It also works to define K using shortest traces only.

$a\check{n}(\mathit{fin}, n')$ implies $K(\mathit{fin}, n')$. In particular we have $((\mathit{fin}, n'), (t, t'))$ is in $K(\mathit{fin}, n')$. Thus by definition of K we have that $\mathit{termQtr}(t, n', t') \neq \emptyset$, unless $n' = \mathit{init}'$. In case $n' = \mathit{init}'$ we get $\mathit{termQtr}(t, \mathit{init}', t') \neq \emptyset$ as follows. By the general observation above, from $(t, t') \in \mathit{an}(\mathit{fin}, \mathit{init}')$ we have that $((\mathit{fin}, \mathit{init}'), (t, t'))$ is reachable from some $((\mathit{init}, \mathit{init}'), (s, s'))$ with $(s, s') \in \mathcal{P}$. This gives a terminated A-trace. Now we can appeal to fact (iii) above to get $\mathit{termQtr}(t, \mathit{init}', t') \neq \emptyset$.

Now by fact (ii) we have $\mathit{shortQtr}(t, n', t')$ is nonempty. Choose any α in $\mathit{shortQtr}(t, n', t')$. The first state of α is (n', t'), by definitions, and $\mathsf{len}(\alpha) > 1$ because $n' \neq \mathit{fin}'$. Let (m', u') be the second state of α, so we have $(n', t') \mapsto' (m', u')$. Using facts (i) and (ii), the set $\mathit{termQtr}(t, m', u')$ is nonempty, so by definition of K we have $((\mathit{fin}, m'), (t, u')) \in K(\mathit{fin}, m')$, proving the first bullet above.

Since $\mathit{shortQtr}(t, n', t')$ is nonempty (and traces have nonzero length) we have $V((\mathit{fin}, n'), (t, t')) \neq 0$. Since (m', u') is the second state in α which is in $\mathit{shortQtr}(t, n', t')$, we have $V((\mathit{fin}, m'), (t, u')) = V((\mathit{fin}, n'), (t, t')) - 1$ whence $V((\mathit{fin}, m'), (t, u')) < V((\mathit{fin}, n'), (t, t'))$, proving the second bullet above and completing the proof of the main statement of the theorem.

Concerning finite support, suppose the stores of A and A' are variable stores, and suppose $A, A', \mathcal{P}, \mathcal{Q}$ are finitely supported. We must show that $L, R, J, K, \mathit{an}, V$ are too. Let X be all the variables on which $\mathcal{P}$ or $\mathcal{Q}$ depend (on the left or right), together with all the variables in the support of the transition relation of A or A'. Note that in any trace α of either automaton, any variable $y \notin X$ is unchanged through the trace. Moreover if β is obtained from α by setting y to another fixed value, then β is a trace of the automaton. Because X supports $\mathcal{P}$, we have that strongest invariants, are supported by X; in brief, an is supported by X. Because X supports $\mathcal{Q}$, each set $\mathit{termQtr}(s, n', s')$ is determined by the projections of s and s' on X, and the same for $\mathit{shortQtr}(s, n', s')$. Thus the condition $\mathit{termQtr}(s, n', s') \neq \emptyset$ in the definition of K is supported by X, so K is as well. We have L, R, J supported by X because $\mathsf{dom}(\mapsto)$ and $\mathsf{dom}(\mapsto')$ are by assumption. Finally, V is supported by X because the condition $\mathit{shortQtr}(s, n', s') \neq \emptyset$ in its definition is. □

Remark D.1. A natural question is whether the L, R, J, K conditions, annotation an, and measure V used in the proof of Theorem 9.7 are expressible in a first order assertion language. Prior works on Hoare logic showed expressivity of weakest conditions in Peano arithmetic, based on Gödel encodings of stores, execution traces, etc. [AdBO09, dBdBZ80] as needed for an, K, V. Thus Peano arithmetic suffices for our result. □

Theorem 9.10. Suppose we have the following.
(a) $\mathsf{okf}(c, f)$ and $\mathsf{okfd}(c', f')$.
(b) an is a valid annotation of $\prod(\mathsf{aut}(c, f), \mathsf{aut}(c', f'), L, R, J, K)$ for $\mathcal{S} \rightsquigarrow \mathcal{T}$.
(c) K is an adequate filtering for an and $\prod(\mathsf{aut}(c, f), \mathsf{aut}(c', f'), L, R, J, K)$.
(d) an, $\mathcal{S}$, $\mathcal{T}$, L, R, J, and the witness V for adequate filtering, all have finite support.
Then the judgment $c \mid c' : \mathcal{S} \overset{\exists}{\approx\!>} \mathcal{T}$ has a proof in ERHL+.

Proof. The proof of this theorem proceeds in the same way as the proof of Theorem 7.1. As in Theorem 7.1, the proof uses only relational assertions derived from $\mathit{an}(i, j)$, L, R, and K.

We start by choosing a variable pc that is fresh with respect to $\mathcal{S}$, $\mathcal{T}$, c, c', an, L, R, J, K, and the V that witnesses adequacy assumption (c). Existence of such a variable is ensured by the assumption (d) of finite support. By Lemma 5.11 there are gcs and gcs' such that $c \,/\, f \hookrightarrow \mathit{gcs}$ and $c' \,/\, f' \hookrightarrow \mathit{gcs}'$. Let $n = \mathsf{lab}(c)$ and $n' = \mathsf{lab}(c')$.

By Theorem 5.13 we have

$$!n; \mathsf{do}\ gcs\ \mathsf{od} \simeq \mathsf{add}^{pc}(c); !f \qquad \text{and} \qquad !n'; \mathsf{do}\ gcs'\ \mathsf{od} \simeq \mathsf{add}^{pc}(c'); !f'$$

Define store relation $\mathcal{Q}$ to be $\mathcal{Q}_{an} \wedge \mathcal{Q}_{pc}$ where

$$\begin{array}{ll} \mathcal{Q}_{an}: & (\wedge i, j \ :\ i \in \mathsf{labs}(c) \cup \{f\} \wedge j \in \mathsf{labs}(c') \cup \{f'\} \ :\ \langle\!\langle ?i \mid ?j \rangle\!\rangle \ \Rightarrow an(i,j)) \\ \mathcal{Q}_{pc}: & (\vee i, j \ :\ i \in \mathsf{labs}(c) \cup \{f\} \wedge j \in \mathsf{labs}(c') \cup \{f'\} \ :\ \langle\!\langle ?i \mid ?j \rangle\!\rangle\ ) \end{array}$$

We will derive

$$\mathsf{do}\ gcs\ \mathsf{od} \mid \mathsf{do}\ gcs'\ \mathsf{od} : \mathcal{Q} \overset{\exists}{\approx\!>} \mathcal{Q} \wedge \neg\langle\!\langle \mathsf{enab}(gcs) \rangle\!\rangle \wedge \neg\langle\!\langle \mathsf{enab}(gcs') \rangle\!\rangle \tag{D.1}$$

To do so, we start by noting K being an adequate filtering implies there is a variant function V from $((\mathsf{labs}(c) \cup \{f\}) \times (\mathsf{labs}(c') \cup \{f'\})) \times ((\mathsf{Var} \to \mathbb{Z}) \times (\mathsf{Var} \to \mathbb{Z}))$ to D, where $(D, \prec)$ is some well-ordered set. Define the ***pc-encoded variant*** $\tilde{V}$ as $\tilde{V}(s, s') \mathrel{\hat{=}} V((s(pc), s'(pc)), (s, s'))$. Note that $\tilde{V}$ is a function on variable stores. In what follows, we will also use the curried form $V^{\downarrow}(n, n')$ defined for any n, n', s, s' by $V^{\downarrow}(n, n')(s, s') \mathrel{\hat{=}} V((n, n'), (s, s'))$.

Now to show (D.1), we instantiate rule eDo with $\mathcal{Q} := \mathcal{Q}$, $\mathcal{L} := \tilde{L}$, $\mathcal{R} := \tilde{R}$ and $V := \tilde{V}$. The side condition of eDo is the same as the side condition of rDo. Further, the instantiation of $\mathcal{Q}$, $\mathcal{L}$, and $\mathcal{R}$ is the same as in the proof of Theorem 7.1. Thus, the proof that the side condition is valid is identical. We now turn to proofs of the premises of eDo. For each, we list a few representative cases.

There are three sets of premises of eDo for the two loops, with these forms:

(left-only) $b \mid \mathsf{skip} : \mathcal{Q} \wedge \langle\!\langle e \rangle\!\rangle \wedge \tilde{L} \overset{\exists}{\approx\!>} \mathcal{Q}$ for all $e \to b$ in gcs

(right-only) $\mathsf{skip} \mid b' : \mathcal{Q} \wedge \rangle\!\langle e' \rangle\!\rangle \wedge \tilde{R} \wedge (\tilde{V} = k) \overset{\exists}{\approx\!>} \mathcal{Q} \wedge (\tilde{V} \prec k)$ for all $e' \to b'$ in gcs', all $k \in D$

(joint) $b \mid b' : \mathcal{Q} \wedge \langle\!\langle e \mid e' \rangle\!\rangle\ \wedge \neg\tilde{L} \wedge \neg\tilde{R} \overset{\exists}{\approx\!>} \mathcal{Q}$ for all $e \to b$ in gcs and $e' \to b'$ in gcs'

Joint cases. We start by making note of the following. For any m and m' we can prove

$$!m \mid !m' : an(m, m') \overset{\exists}{\approx\!>} \mathcal{Q}. \tag{D.2}$$

We arrive at this as follows. Since $an(m, m')$ is independent of pc on both sides, the judgment $!m \mid !m' : an(m, m') \overset{\exists}{\approx\!>} an(m, m') \wedge \langle\!\langle ?m \mid ?m' \rangle\!\rangle$ can be proved using eAsgnAsgn with eConseq to simplify the precondition. Then, we use eConseq with the fact that $an(m, m') \wedge \langle\!\langle ?m \mid ?m' \rangle\!\rangle \Rightarrow \mathcal{Q}$ to obtain (D.2). Additionally, the following implication is valid.

$$\mathcal{Q} \wedge \langle\!\langle ?n \mid ?n' \rangle\!\rangle\ \wedge \neg\tilde{L} \wedge \neg\tilde{R} \Rightarrow an(n, n') \wedge \langle\!\langle ?n \mid ?n' \rangle\!\rangle\ \wedge \tilde{J} \tag{D.3}$$

We arrive at this using adequacy condition (c) of the theorem (in particular, the enabled condition in Def. 9.5) and the fact that $Q \wedge \langle\!\langle ?n \mid ?n' \rangle\!\rangle \Rightarrow an(n, n')$. See (7.6).

Now for the joint premises. We spell out just two cases.

- $\mathsf{hav}\ x;\ !m \mid \mathsf{skip};\ !m' : \mathcal{Q} \wedge \langle\!\langle ?n \mid ?n' \rangle\!\rangle\ \wedge \neg\tilde{L} \wedge \neg\tilde{R} \overset{\exists}{\approx\!>} \mathcal{Q}$, where $\mathsf{sub}(n, c) = \mathsf{hav}^n\ x$, $m = \mathsf{fsuc}(n, c, f)$, $\mathsf{sub}(n', c') = \mathsf{skip}^{n'}$, and $m' = \mathsf{fsuc}(n', c', f')$.

  We construct a deductive proof using eSeq with judgments (D.2) and

  $$\mathsf{hav}^n\ x \mid \mathsf{skip}^{n'} : \mathcal{Q} \wedge \langle\!\langle ?n \mid ?n' \rangle\!\rangle\ \wedge \neg\tilde{L} \wedge \neg\tilde{R} \overset{\exists}{\approx\!>} an(m, m')$$

  To prove the latter, first by eHavSkip we have $\mathsf{hav}^n\ x \mid \mathsf{skip}^{n'} : (\dot{\forall} x_{|}.\ an(m, m')) \overset{\exists}{\approx\!>} an(m, m')$. Now strengthen the precondition using eConseq for which we need to show

$\mathcal{Q} \wedge \{\!|?n \,|\, ?n'|\!\} \wedge \neg\tilde{L} \wedge \neg\tilde{R} \Rightarrow (\dot{\forall} x_{|}.\, an(m,m'))$. In doing so, we'll appeal to the corresponding *pc*-encoded VC (Lemma 9.8) which is

$$\forall v \in \mathbb{Z}.\, an(n,n') \wedge \{\!|?n \,|\, ?n'|\!\} \wedge \tilde{J} \wedge K^{\downarrow}(m,m')^{x|}_{v|} \Rightarrow an(m,m')^{x|}_{v|}$$

The eConseq step is justified as follows.

	$\mathcal{Q} \wedge \{\!|?n \,|\, ?n'|\!\} \wedge \neg\tilde{L} \wedge \neg\tilde{R}$	
$\Rightarrow$	$an(n,n') \wedge \{\!|?n \,|\, ?n'|\!\} \wedge \tilde{J}$	by (D.3)
$\Rightarrow$	$an(n,n') \wedge \{\!|?n \,|\, ?n'|\!\} \wedge \tilde{J} \wedge (\forall v \in \mathbb{Z}.\, K^{\downarrow}(m,m')^{x|}_{v|})$	semantics, adequacy, see below (†)
$\Leftrightarrow$	$\forall v.\, an(n,n') \wedge \{\!|?n \,|\, ?n'|\!\} \wedge \tilde{J} \wedge K^{\downarrow}(m,m')^{x|}_{v|}$	pred calc
$\Rightarrow$	$\forall v.\, an(m,m')^{x|}_{v|}$	apply VC (Lemma 9.8)
$\Leftrightarrow$	$\dot{\forall} x_{|}.\, an(m,m')$	by def

To justify the step marked (†) above, we rely on the theorem's assumption of adequate filtering. Specifically, we use use joint-productivity to prove that $an(n,n') \wedge \{\!|?n \,|\, ?n'|\!\} \wedge \tilde{J}$ implies $(\forall v \in \mathbb{Z}.\, K^{\downarrow}(m,m')^{x|}_{v|})$. Because Def. 9.5 is in terms of automata states, we need to reason pointwise and take care with our notational abuses. Observe for any s, s' that

$$(s,s') \in (an(n,n') \wedge \{\!|?n \,|\, ?n'|\!\} \wedge \tilde{J})$$

equivales, by definitions and freshness of *pc*,

$$(s,s') \in an(n,n') \wedge s(pc) = n \wedge s'(pc) = n' \wedge ((n,n'),(s,s')) \in J$$

By the assumptions $\mathsf{sub}(n,c) = \mathsf{hav}^n\ x$ and $m = \mathsf{fsuc}(n,c,f)$, using Definition 4.10 of program automata which in this case is based on semantics of $\mathsf{hav}^n\ x$,

we have $(n,s) \mapsto (m, s[x \mapsto v])$ for all $v \in \mathbb{Z}$. By joint-productivity, for any v there is some m'', t' such that $(n',s') \mapsto' (m'',t')$ and $((m,m''),(s[x \mapsto v])) \in K$. But by semantics and assumptions $\mathsf{sub}(n',c') = \mathsf{skip}^{n'}$ and $m' = \mathsf{fsuc}(n',c',f')$ we must have $m'' = m'$ and $t' = s'$, so we have $((m,m'),(s[x \mapsto v],s')) \in K$. This is equivalent to $(s[x \mapsto v],s') \in K^{\downarrow}(m,m')$ and to $(s,s') \in K^{\downarrow}(m,m')^{x|}_{v|}$. So we have $(\forall v \in \mathbb{Z}.\, K^{\downarrow}(m,m')^{x|}_{v|})$, so (†) is proved.

- $!m \mid !m' : \mathcal{Q} \wedge \{\!|?n \,|\, ?n' \wedge e'|\!\} \wedge \neg\tilde{L} \wedge \neg\tilde{R} \overset{\exists}{\approx\!>} \mathcal{Q}$ where $\mathsf{sub}(n,c) = \mathsf{skip}^n$, $m = \mathsf{fsuc}(n,c,f)$, $\mathsf{sub}(n',c') = \mathsf{if}^{n'}\ gcs'_0\ \mathsf{fi}$ with $e' \to d'$ in gcs'_0, $m' = \mathsf{lab}(d')$.

  The corresponding VC is similar to the assign/if VC in Figure 19, specifically:

$$J \wedge \breve{an}(n,n') \wedge \{\!|e'|\!\} \wedge K(m,m') \Rightarrow \hat{an}(m,m')$$

Of course we use the pc-encoded version.

We prove the judgment using (D.2) and eConseq with the following implication.

	$\mathcal{Q} \wedge \{\!|?n \,|\, ?n' \wedge e'|\!\} \wedge \neg\tilde{L} \wedge \neg\tilde{R}$	
$\Rightarrow$	$an(n,n') \wedge \{\!|?n \,|\, ?n' \wedge e'|\!\} \wedge \tilde{J}$	using (D.3)
$\Rightarrow$	$an(n,n') \wedge \{\!|?n \,|\, ?n' \wedge e'|\!\} \wedge \tilde{J} \wedge K^{\downarrow}(m,m')$	adequacy, semantics
$\Rightarrow$	$an(m,m')$	apply VC

The adequacy step uses in particular joint productivity, which says for any n, n', s, s', m, t that if $(s,s') \in an(n,n')$ and $((n,n'),(s,s')) \in J$ and $(n,s) \mapsto (m,t)$, then there are m'', t' with $(n',s') \mapsto' (m'',t')$ and $((m,m''),(t,t')) \in K$. Instantiating this with $m := \mathsf{fsuc}(n,c,f)$ and $t := s$ (since the command at n is skip), we get that there exist m'', t' with $(n',s') \mapsto' (m'',t')$ and $((m,m''),(t,t')) \in K$. By semantics and control determinacy

(assumption (a) of the theorem, and Lemma 9.9) and the fact that e' is true in s', we have that $m'' = m'$ since m' is the unique successor, and by semantics $t' = s'$. So we have $(n', s') \mapsto' (m', s')$ and $((m, m'), (s, s')) \in K$, and equivalently $(s, s') \in K^{\downarrow}(m, m')$.

Right-only cases. These cases are proved using the same rules as the joint cases, plus one additional rule: eDisj, in the form eDisjN derived from it. This is needed for the same reason rDisjN is needed in the proof of alignment completeness for RHL+ (Theorem 7.1). The joint cases determine a unique starting and ending pair of control points, which determines the VC to appeal to. The right-only cases do not determine a control point on the left. So we consider an arbitrary left control point, and then combine all cases using eDisjN. In passing, we note the following:

$$(s, s') \in \langle\!\langle ?m \mid ?m' \rangle\!\rangle \;\Rightarrow V^{\downarrow}(m, m')(s, s') = \tilde{V}(s, s') \quad \text{for any } s, s', m, m' \tag{D.4}$$

$$\mathcal{Q} \wedge [\![?n']\!] \Leftrightarrow (\vee\, n \;:\; n \in \mathsf{labs}(c) \cup \{f\} \;:\; \mathcal{Q} \wedge \langle\!\langle ?n \mid ?n' \rangle\!\rangle\ ) \tag{D.5}$$

Additionally, to streamline the proofs below, we note the following can be proved.

$$\mathsf{skip} \mid !m' : an(n, m') \wedge \langle\!\langle ?n]\!] \wedge V^{\downarrow}(n, m') \prec k \overset{\exists}{\approx\!>} \mathcal{Q} \wedge \tilde{V} \prec k \tag{D.6}$$

We obtain this judgment as follows. By eSkipAsgn and the fact that all the terms in the precondition are independent of pc on the right, we have $\mathsf{skip} \mid !m' : an(n, m') \wedge \langle\!\langle ?n]\!] \wedge V^{\downarrow}(n, m') \prec k \overset{\exists}{\approx\!>} an(n, m') \wedge \langle\!\langle ?n \mid ?m' \rangle\!\rangle \wedge V^{\downarrow}(n, m') \prec k$. We then weaken the postcondition using eConseq: suppose $an(n, m') \wedge \langle\!\langle ?n \mid ?m' \rangle\!\rangle \wedge V^{\downarrow}(n, m') \prec k$. By (7.5), this implies $\mathcal{Q} \wedge \langle\!\langle ?n \mid ?m' \rangle\!\rangle \wedge V^{\downarrow}(n, m') \prec k$. We then have $\mathcal{Q} \wedge \tilde{V} \prec k$ by (D.4).

We now detail the proofs for a couple of representative right-only cases. Towards that end, we pick an arbitrary control point $n \in \mathsf{labs}(c) \cup \{f\}$ on the left. The judgments below conjoin $\langle\!\langle ?n]\!]$ to preconditions, and hence, don't exactly match the premises of eDo. However, since n is an arbitrary label, by (D.5) and application of eDisjN, each proof below gives rise to a corresponding right-only premise of rule eDo (such use of a disjunction rule is spelled out in more detail in the proof of Theorem 7.1). Note there's a premiss for each literal value k in D. So let $k \in D$ be arbitrary, in the following.

- $\mathsf{skip} \mid !m' : \mathcal{Q} \wedge \langle\!\langle ?n \mid ?n' \rangle\!\rangle \wedge \tilde{R} \wedge \tilde{V} = k \overset{\exists}{\approx\!>} \mathcal{Q} \wedge \tilde{V} \prec k$, where $\mathsf{sub}(n', c') = \mathsf{skip}^{n'}$ and $m' = \mathsf{fsuc}(n', c', f')$.

  We obtain this from (D.6) by eConseq, strengthening the precondition as follows.

	$\mathcal{Q} \wedge \langle\!\langle ?n \mid ?n' \rangle\!\rangle \wedge \tilde{R} \wedge \tilde{V} = k$	
$\Rightarrow$	$an(n, n') \wedge \langle\!\langle ?n \mid ?n' \rangle\!\rangle \wedge \tilde{R} \wedge V^{\downarrow}(n, n') = k$	(7.5) and (D.4)
$\Rightarrow$	$an(n, n') \wedge \langle\!\langle ?n \mid ?n' \rangle\!\rangle \wedge \tilde{R} \wedge K^{\downarrow}(n, m') \wedge V^{\downarrow}(n, m') \prec k$	adequacy, semantics (†)
$\Rightarrow$	$an(n, m') \wedge \langle\!\langle ?n]\!] \wedge V^{\downarrow}(n, m') \prec k$	apply VC (Lemma 9.8)

  In the step marked (†), we use the semantics of skip and apply the right-productivity condition, noting that the only transitions from n' are to m', to obtain $K^{\downarrow}(n, m') \wedge V^{\downarrow}(n, m') \prec V^{\downarrow}(n, n')$. Using $V^{\downarrow}(n, n') = k$, this yields $K^{\downarrow}(n, m') \wedge V^{\downarrow}(n, m') \prec k$. As in the proofs of the joint-only premises of eDo above, we do not spell this step out in detail.
- $\mathsf{skip} \mid !m' : \mathcal{Q} \wedge \langle\!\langle ?n \mid ?n' \wedge e' \rangle\!\rangle \wedge \tilde{R} \wedge \tilde{V} = k \overset{\exists}{\approx\!>} \mathcal{Q} \wedge \tilde{V} \prec k$, where $\mathsf{sub}(n', c') = \mathsf{if}^{n'}\ gcs'\ \mathsf{fi}$ and $e' \to d'$ is in gcs' and $m' = \mathsf{lab}(d')$.

As in the preceding case we obtain this from (D.6) by eConseq, strengthening the precondition as follows.

$$\begin{array}{ll} & \mathcal{Q} \wedge \{\!|?n \mid ?n' \wedge e'|\!\} \ \wedge \tilde{R} \wedge \tilde{V} = k \\ \Rightarrow & \quad \text{(7.5) and (D.4)} \\ & an(n,n') \wedge \{\!|?n \mid ?n' \wedge e'|\!\} \ \wedge \tilde{R} \wedge V^{\downarrow}(n,n') = k \\ \Rightarrow & \quad (\dagger) \text{ adequacy, semantics, see below} \\ & an(n,n') \wedge \{\!|?n \mid ?n' \wedge e'|\!\} \ \wedge \tilde{R} \wedge K^{\downarrow}(n,m') \wedge V^{\downarrow}(n,m') \prec k \\ \Rightarrow & \quad \text{apply VC from Figure 18, see Lemma 9.8} \\ & an(n,m') \wedge \{\!|?n|\!\} \wedge V^{\downarrow}(n,m') \prec k \end{array}$$

To prove the step marked (†), consider any (s,s') that satisfies the antecedent. By right productivity there are m'', t' such that $(n',s') \mapsto' (m'',t')$ and $((n,m''),(s,t')) \in K$ and $V((n,m''),(s,t')) \prec V((n,n'),(s,t'))$. By definition of automata and semantics of if, the store is unchanged so t' is s'. That is, we have m'' such that $(n',s') \mapsto' (m'',s')$ and $((n,m''),(s,s')) \in K$ and $V((n,m''),(s,s')) \prec V((n,n'),(s,s'))$. Because (s,s') satisfies $\{\!|?n \mid ?n' \wedge e'|\!\}$ and c' is control deterministic, we must have $m'' = m'$, hence $((n,m'),(s,s')) \in K$ and $V((n,m'),(s,s')) \prec V((n,n'),(s,s'))$. So (s,s') satisfies $K^{\downarrow}(n,m')$ and $V^{\downarrow}(n,m') \prec k$ as needed for the consequent of (†).

The other right-only cases are similarly proved.

Left-only cases. These are very similar to the right-only cases and are omitted. Justifications of eConseq are simpler since we don't have to reason about $V^{\downarrow}$ decreasing. Rather than using right-productivity of K, these cases use its left-permissivity.

Finishing the proof. Now that we've established all the premises of eDo, the rest of the proof proceeds exactly like the proof of Theorem 7.1 but using eSeq, eConseq, eRewrite, eGhost. □