*Mathematical Structures in Computer Science* (2022), **32**, pp. 1145–1175
doi:10.1017/S0960129522000275

CAMBRIDGE UNIVERSITY PRESS

PAPER

# Indexed and fibered structures for partial and total correctness assertions

U.E. Wolter[1*], A.R. Martini[2] and E.H. Häusler[3]

[1]Department of Informatics, University of Bergen, Bergen, Norway, [2]Av. Marechal Andrea 11/210, Porto Alegre, Brazil and [3]Departamento de Ciência da Computação, PUC-Rio, Rio de Janeiro, Brazil

*Corresponding author. Email: Uwe.Wolter@uib.no

(Received 6 August 2021; revised 31 July 2022; accepted 19 August 2022; first published online 19 September 2022)

**Abstract**

Hoare Logic has a long tradition in formal verification and has been continuously developed and used to verify a broad class of programs, including sequential, object-oriented, and concurrent programs. Here we focus on partial and total correctness assertions within the framework of Hoare logic and show that a comprehensive categorical analysis of its axiomatic semantics needs the languages of indexed and fibered category theory. We consider Hoare formulas with local, finite contexts, of program and logical variables. The structural features of Hoare assertions are presented in an indexed setting, while the logical features of deduction are modeled in the fibered one.

**Keywords:** Hoare logic; partial correctness assertions; total correctness assertions; indexed categories; fibrations

## 1. Introduction

Hoare Logic (Apt et al. 2009; Huth and Ryan 2004; Leino 2010; Loeckx and Sieber 1987) has a long tradition in formal verification and has been continuously developed and used to verify a broad class of programs, including sequential, object-oriented, and concurrent programs. This logic is comprised by a language in which one can formulate propositions about the partial and total correctness of while-programs and a deduction calculus with which one can prove that a certain proposition is true.

Partial correctness assertions are propositions of the form $\{P\}\, c\, \{Q\}$, where $P$, $Q$ are first-order formulas and $c$ is a while-program. The intuition behind such a specification is that if the program $c$ starts executing in a state where the assertion $P$ is true, then if $c$ terminates, it does so in a state where the assertion $Q$ holds. On the other hand, total correctness assertions are propositions of the form $[P]\, c\, [Q]$, and their intuitive meaning is that if the program $c$ starts executing in a state where the assertion $P$ is true, then $c$ terminates, and it does so in a state where the assertion $Q$ holds. Both partial and total correctness assertions are usually called Hoare triples.

We are interested to answer, at least partially, the fundamental question "What are the characteristic structural features of Hoare logic?" In contrast to the traditional exposition of Hoare logic, that relies on infinitary contexts of program variables, it is much more adequate to consider finite sets of program variables as contexts of programs and to work, in such a way, with finite "local" states and assertions about finite "local" states. This is a perfect match with the "categorical imperative" that morphisms in a category do have a unique source and target object. That is, any categorical analysis and presentation of logics should be based on local contexts for expressions

and formulas. Also, as shown in Examples 1, 2, and 3, this is the precise way a programmer writes and understands Hoare triples. Moreover, in current implementations of Hoare Logic as Bubel and Hähnle (2016), Martini (2020), Pierce et al. (2018*a*,b) and in programming languages which support specifications based on pre- and postconditions like Leino (2010), the programmer must explicitly declare the program and logical variables that appear in the Hoare specification. In other words: Our decision to use finite contexts of variables is a reasonable choice backed up by two essential rationals. First, the methodology enforced by a categorical treatment of logics and type theories leads naturally to the use of finite contexts. Second, we want to represent faithfully the essential components of Hoare specifications in programming practice. *Program code, logical formulas, and variable declarations are always finite objects in the mind of the working programmer*. Thus, our main goal here is to transform the global infinitary version of the Hoare logic for while-programs, presented in Section 2, into an equivalent local, finitary version. Our hope is that this finitary version of the Hoare logic for while-programs can serve as a blueprint for the design and study of Hoare style logics for other kinds of programs.

After developing a general and structured presentation of a finitary version of Hoare logic based on indexed categories, we have, at least, three reasons to move from the indexed setting to the fibered one. First, the fibered setting will allow us to put all the syntactic and semantic structures, developed so far, on a common conceptual ground and to relate and extend them. Second, it is technically quite uncomfortable to work with pseudo functors. To work instead with fibrations, the equivalent of pseudo functors is more reasonable and technically, less akward. Third, the essential reason in the light of logic is, however, that we need a "technological space" where logical deduction can take place.

The aim of this work is neither to replace traditional set-theoretical descriptions of logics by a corresponding categorical generalization nor to coin an axiomatization of just another abstract categorical framework for logics in the line of (partial) hyperdoctrines (Knijnenburg and Nordemann 1994), institutions (Goguen and Burstall 1992), and context institutions (Pawlowski 1995). Our aim is, in contrast, to demonstrate how indexed and fibered structures can be used, in a flexible and creative way, to model and reason about logical systems and to present how the syntactic and the semantic constituents of logical systems interplay with each other.

There are several categorical formalization's of Hoare logics on relatively high levels of abstraction and generality and our paper does not add much novelty to these papers. In Computer Science (as in many other branches of science), there is a "technological chain" which appears often as a "chain of abstractions and generalizations." Each step in this chain – in both directions from concrete to more abstract as well as from abstract to more concrete – is important and requires a substantial effort. To keep a certain branch of science alive, we have to maintain and to take care of the whole technological chain. Mathematics focuses traditionally at "most general results." It is, however, a social fact that the problem-solving strategy "look for the right most general result and instantiate it adequately to solve your concrete problem" is not feasible for the majority of us. In view of these remarks, one novelty of the paper *is a new description of a first step of abstraction from Hoare logic in programming practice to a categorical formalization of Hoare logic*.

The paper is organized as follows. To provide a unified and common ground for our categorical analysis, we recapitulate, first, in Section 2 basic concepts and constructions for our imperative language, and Hoare Logic. Section 3 analyzes the structural features of the Hoare logic with finite contexts of program and logical variables by means of indexed concepts. In Section 4, we transform, by means of the Grothendieck construction, the indexed functors into fibrations and we discuss Hoare triples and Hoare deduction calculus in the light of the corresponding fibered categories. Section 5 discusses how our work is related to other work in the "technological chain" and identifies, in more detail, novel contributions of the paper. We close the paper by a summary

https://doi.org/10.1017/S0960129522000275 Published online by Cambridge University Press

of the essential ideas treated in this discussion, that is to say, that the structural features of the language are presented in indexed categories while the features of deduction are in the fibered one. Moreover, we outline further work.

## 2. Background Material

In this section, we describe the syntax and semantics of our imperative language. We also present the fundamental concepts of Hoare Logic, that is, its semantics and proof theory, and the core ideas underlying indexed and fibered categories.

### The Syntax of IMP

This subsection describes the abstract syntax of our imperative language, called **IMP**. This is a small language equipped with array expressions, with which we can describe computations over the integers. In order to describe its abstract syntax, we need to fix some basic sets for values and variables. The set $\mathbb{B} = \{\mathbf{true}, \mathbf{false}\}$ of Boolean values, ranged over by metavariables $u, v, \ldots$; the set $\mathbb{Z} = \{\ldots, -2, 1, 0, 1, 2, \ldots\}$ of integer numbers, ranged over by metavariables $m, n, \ldots$; a countably infinite set **PVar** of program variables, ranged over by metavariables $x, y, \ldots$, and a countably infinite set of array variables **AVar** ranged over by metavariables $a, a_i, i \geq 0$. We assume the sets **PVar** and **AVar** to be *disjoint*.

The grammar for **IMP** comprises three syntactic categories: **AExp**, for arithmetic expressions, ranged over by $e, e', \ldots$; **BExp**, for Boolean expressions, ranged over by $b, b', \ldots$, and **Prg**, for programs, ranged over by $c, c', \ldots$ The following productions define the abstract syntax of **IMP** :

$$
\begin{aligned}
e \in \mathbf{AExp} &::= n \mid x \mid e_0 + e_1 \mid e_0 - e_1 \mid e_0 \times e_1 \mid a[e] \\
b \in \mathbf{BExp} &::= v \mid e_0 = e_1 \mid e_0 \leq e_1 \mid \neg b \mid b_0 \vee b_1 \mid b_0 \wedge b_1 \\
c \in \mathbf{Prg} &::= \underline{\mathbf{skip}} \mid x := e \mid c_0; c_1 \mid \underline{\mathbf{if}}\ b\ \underline{\mathbf{then}}\ c_0\ \underline{\mathbf{else}}\ c_1\ \underline{\mathbf{fi}} \mid \underline{\mathbf{while}}\ b\ \underline{\mathbf{do}}\ c\ \underline{\mathbf{od}} \mid a[e] := e'
\end{aligned}
$$

In order to evaluate an expression or to define the execution of a command, we need the notion of a *state*. This state has to define values for both program and array variables. A state for program variables is a function $\sigma_P : \mathbf{PVar} \to \mathbb{Z}$, while a state for array variables is a function $\sigma_A : \mathbf{AVar} \to (\mathbb{Z} \to \mathbb{Z})$, where $(\mathbb{Z} \to \mathbb{Z})$ is the set of all functions from integers to integers. We use the integers both as indexes and as values. It is up to the programmer to guarantee that index values are always greater or equal to zero. Thus, a state $\sigma$ is the disjoint union $\sigma \triangleq \sigma_P \uplus \sigma_A : \mathbf{PVar} \uplus \mathbf{AVar} \longrightarrow \mathbb{Z} \uplus (\mathbb{Z} \to \mathbb{Z})$, such that $\sigma(var) = \sigma_P(var)$ if $x \in \mathbf{PVar}$ and $\sigma(var) = \sigma_A(var)$, if $var \in \mathbf{AVar}$. The collection of all such states is named $\Sigma$. Given a state $\sigma_P$ and a program variable $x \in \mathbf{PVar}$, we denote by $\sigma_P[x \mapsto n]$ a new state that is everywhere like $\sigma_P$, except on $x$, where it is updated to the value $n$. The signature or type of the state update operator is $\_[\_ \mapsto \_] : (\mathbf{PVar} \to \mathbb{Z}) \to \mathbf{PVar} \to \mathbb{Z} \to (\mathbf{PVar} \to \mathbb{Z})$. Note that the type of $\sigma_P[x \mapsto n]$ asserts that it is also a state for program variables. Likewise, given an array $a$ and an array variable $a \in \mathbf{AVar}$, we denote by $a[i \mapsto n]$ a new array, that is everywhere like $a$ but on index $i$, where it is updated to the value $n$. The signature or type of the array update operator is $\_[\_ \mapsto \_] : (\mathbb{Z} \to \mathbb{Z}) \to \mathbb{Z} \to \mathbb{Z} \to (\mathbb{Z} \to \mathbb{Z})$. Note that the type of $a[i \mapsto n]$ asserts that it is also an array, that is, a function from integers to integers.

### Semantics of IMP

In this subsection, we specify the formal semantics of **IMP**. The meaning of arithmetic expressions is defined by primitive recursion on the syntactic structure of the formulas, while the interpretation of programs is given by a transition operational semantics. The following equations define a total function that, given a state $\sigma = \sigma_P \uplus \sigma_A$, maps arithmetic expressions to integers, and Boolean expressions to Boolean values. We assume $aop \in \{+, -, \times\}$, $rop \in \{=, \leq\}$, and $bop \in \{\wedge, \vee\}$.

https://doi.org/10.1017/S0960129522000275 Published online by Cambridge University Press

**Table 1.** Transition semantics for IMP

$$\longrightarrow \subseteq (\mathbf{Prg} \times \Sigma) \times (\mathbf{Prg} \times \Sigma)$$

$$\langle x := e, \sigma\rangle \longrightarrow \langle \underline{\mathbf{skip}}, \sigma'\rangle, \quad \sigma'_A = \sigma_A, \sigma'_P = \sigma_P[x \mapsto [\![e]\!]\sigma] \quad \text{(ass1)}$$

$$\langle a[e] := e', \sigma\rangle \longrightarrow \langle \underline{\mathbf{skip}}, \sigma'\rangle, \quad \sigma'_P = \sigma_P, \sigma'_A = \sigma_A([\![e]\!]\sigma \mapsto [\![e']\!]\sigma) \quad \text{(ass2)}$$

$$\langle \underline{\mathbf{if}}\ b\ \underline{\mathbf{then}}\ c_0\ \underline{\mathbf{else}}\ c_1\ \underline{\mathbf{fi}}, \sigma\rangle \longrightarrow \langle c_0, \sigma\rangle, \quad [\![b]\!]_\sigma = \mathit{true} \ \text{(if1)}$$

$$\langle \underline{\mathbf{if}}\ b\ \underline{\mathbf{then}}\ c_0\ \underline{\mathbf{else}}\ c_1\ \underline{\mathbf{fi}}, \sigma\rangle \longrightarrow \langle c_1, \sigma\rangle, \quad [\![b]\!]_\sigma = \mathit{false} \ \text{(if2)}$$

$$\langle \underline{\mathbf{while}}\ b\ \underline{\mathbf{do}}\ c\ \underline{\mathbf{od}}, \sigma\rangle \longrightarrow \langle \underline{\mathbf{if}}\ b\ \underline{\mathbf{then}}\ (c; \underline{\mathbf{while}}\ b\ \underline{\mathbf{do}}\ c\ \underline{\mathbf{od}})\ \underline{\mathbf{else}}\ \underline{\mathbf{skip}}\ \underline{\mathbf{fi}}, \sigma\rangle \quad \text{(while)}$$

$$\langle \underline{\mathbf{skip}}; c, \sigma\rangle \longrightarrow \langle c, \sigma\rangle \quad \text{(comp1)}$$

$$\frac{\langle c_1, \sigma\rangle \longrightarrow \langle c'_1, \sigma'\rangle}{\langle c_1; c_2, \sigma\rangle \longrightarrow \langle c'_1; c_2, \sigma'\rangle} \ \text{(comp2)}$$

| $[\![\_]\!]\_ : \mathbf{AExp} \to \Sigma \to \mathbb{Z}$ | $[\![\_]\!]\_ : \mathbf{BExp} \to \Sigma \to \mathbb{B}$ |
|---|---|
| $[\![n]\!]\sigma = n$ | $[\![v]\!]\sigma = v$ |
| $[\![x]\!]\sigma = \sigma_P(x)$ | $[\![\neg b]\!]\sigma = \neg[\![b]\!]\sigma$ |
| $[\![a[e]]\!]\sigma = \sigma_A(a)([\![e]\!]\sigma)$ | $[\![e_0\ rop\ e_1]\!]\sigma = [\![e_0]\!]\sigma\ rop\ [\![e_1]\!]\sigma,$ |
| $[\![e_0\ aop\ e_1]\!]\sigma = [\![e_0]\!]\sigma\ aop\ [\![e_1]\!]\sigma$ | $[\![b_0\ bop\ b_1]\!]\sigma = [\![b_0]\!]\sigma\ bop\ [\![b_1]\!]\sigma$ |

In structural operational semantics, the emphasis is on the individual steps of the execution. The semantics relates pairs of configurations $\delta \longrightarrow \delta'$ of the form $\delta = \langle c, \sigma\rangle$, where $c \in \mathbf{Prg}$, $\sigma \in \Sigma$. *Terminal* configurations have the form $\langle \underline{\mathbf{skip}}, \sigma\rangle$. The transition relation $\langle c, \sigma\rangle \longrightarrow \langle c', \sigma'\rangle$ expresses the first step of the execution of $c$ from state $\sigma$. There are two possible outcomes.If $\delta'$ is of the form $\langle c', \sigma'\rangle$, $c' \neq \underline{\mathbf{skip}}$ then the execution of $c$ from $\sigma$ is not completed. Otherwise, if $\delta' = \langle \underline{\mathbf{skip}}, \sigma'\rangle$ then the execution of $c$ from $\sigma$ has terminated with final state $\sigma'$. The single steps of the structural operational semantics of **IMP** programs is defined by the rules presented in Table 1.

A *derivation sequence* or *execution* of a program $c$ starting in state $\sigma$ is either: a *finite sequence* of configurations $\delta_0, \ldots, \delta_k, k \geq 0$ satisfying $\delta_0 = \langle c, \sigma\rangle, \delta_i \longrightarrow \delta_{i+1}, 0 \leq i < k, k \geq 0$, and $\delta_k$ is a *terminal* configuration; or an *infinite* sequence $\delta_0, \delta_1, \delta_2, \ldots$ of configurations satisfying $\delta_i \longrightarrow \delta_{i+1}, i \geq 0$. The expression $\delta_0 \overset{*}{\longrightarrow} \delta_k$ indicates that the execution from $\delta_0$ and $\delta_k$ has a finite number of steps, where $\overset{*}{\longrightarrow}$ is the reflexive, transitive closure of the relation $\longrightarrow$.

**Definition 1** (Semantics of Programs)**.** *The transition relation* $\longrightarrow$ *between configurations defines the meaning of programs as a partial function from states to states:*

$$[\![\_]\!]\_ : \mathit{Prg} \to (\Sigma \nrightarrow \Sigma) \qquad \mathit{with} \qquad [\![c]\!]\sigma = \begin{cases} \sigma' & \mathit{if}\ \langle c, \sigma\rangle \overset{*}{\longrightarrow} \langle \underline{\mathit{skip}}, \sigma'\rangle \\ \mathit{undefined} & \mathit{otherwise} \end{cases} \tag{1}$$

## Hoare Logic

The central feature of Hoare logic are the *Hoare triples* or, as they are often called, *partial correctness assertions*. We use both expressions interchangeably. A Hoare triple describes how the execution of a piece of code changes the state of the computation, and it is of the form $\{P\}\, c\, \{Q\}$,

https://doi.org/10.1017/S0960129522000275 Published online by Cambridge University Press

where $P, Q$ are assertions in a specification language and $c \in \mathbf{Prg}$ is a **IMP** program. $P$ is called the precondition and $Q$ the postcondition of the triple. It means that *for any state satisfying P, if the execution of c terminates, then the resulting state is a state satisfying Q*. Apart from partial correctness assertions, we also have *total correctness assertions*, expressions of the form $[P]\ c\ [Q]$. It means that *for any state satisfying P, the execution of c terminates, and the resulting state is a state satisfying Q*. Thus, total correctness is guaranteed by construction. We use the expressions *total correctness assertions* and *total Hoare triples* interchangeably.

**Remark 1** (Language of assertions)**.** We use the language of first-order logic to write assertions about computations over the integers. These assertions are built on top of *program variables* (**PVar**), *array variables* (**AVar**), and *logical variables*. We assume a countably infinite set **LVar** of logical variables, and such that **PVar**, **AVar**, **LVar** are mutually disjoint. The logical variables are the standard variables of first-order logic. They do not appear in programs. Their use in assertions are limited to write quantified formulas and also to save values of program variables in initial states. In the concrete examples of Hoare triples bellow, program and array variables are written in lowercase, while logical variables are capitalized. The language of assertions is named **Assn**.

**Example 1** (Program Swap)**.** The Hoare triple

$$\{x = X0 \wedge y = Y0\}\ temp := x; x := y; y := temp\ \{x = Y0 \wedge y = X0\}$$

asserts the partial and total correctness of a program that swaps the values of two program variables.

**Example 2** (Program Find)**.** The Hoare triple

$$\{Pre\}init; \underline{\mathbf{while}}\ B; \underline{\mathbf{do}}\ body; \underline{\mathbf{od}}\{Pos\}$$

asserts the partial and total correctness of a program that performs a linear search in an array of integers, where $Pre \triangleq n = Length \wedge Length \geq 0 \wedge key = K$ and $Pos \triangleq (0 \leq index \implies index < Length \wedge a[index] = key) \wedge (index = -1 \implies \forall J.0 \leq J < n \implies \neg(a[J] = key))$. Moreover, we take $init \triangleq index := -1; i := 0$, $B \triangleq (i < n) \wedge (index = -1)$ and $body \triangleq \underline{\mathbf{if}}\ (a[i] = key)\ index = i\ \underline{\mathbf{else}}\ \underline{\mathbf{skip}}\ \underline{\mathbf{fi}}; i := i + 1$.

**Example 3** (Insertion Sort)**.** The Hoare triple

$$\{Pre\}i := 1; \underline{\mathbf{while}}\ B\ \underline{\mathbf{do}}\ j := i; \underline{\mathbf{while}}\ C\ \underline{\mathbf{do}}\ body_2\ \underline{\mathbf{od}}\ i := 1 + 1\ \underline{\mathbf{od}}\{Pos\}$$

asserts the partial and total correctness of a program that sorts an array of integers. The array a is assumed to have length denoted by the logical variable *Length*. For this example, we abstract the property that the output array must be a permutation of the input array. We also assume that $Pre \triangleq n = Length \wedge Length > 0$, $Pos \triangleq \forall J, K.\ 0 \leq J < K < n \implies a[J] \leq a[K]$, $B \triangleq (i < n)$, $C \triangleq (j > 0) \wedge (a[j-1] > a[j])$, $body_2 \triangleq temp := a[j-1]; a[j-1] := a[j]; a[j] := temp; j := j - 1$.

The examples of Hoare triples above show that the arithmetic expressions used in the language of assertions contain logical variables as well. These extended language of arithmetic expressions is called $\mathbf{AExp}^{+}$ and this language *do not appear in programs, only in the language of assertions*. The syntax and semantic of extended arithmetic expressions needs to get a little fix, that is, the syntax must include logical variables and the semantics needs an *environment* (Huth and Ryan 2004), also called *assignment* (Loeckx and Sieber 1987), for free logical variables. An environment for the (free) logical variables in an assertion is a function $\alpha : \mathbf{LVar} \to \mathbb{Z}$. The set of all such environments is $\mathbf{Env} = (\mathbf{LVar} \to \mathbb{Z})$.

https://doi.org/10.1017/S0960129522000275 Published online by Cambridge University Press

$$e \in \mathbf{AExp}^{+} ::= n \mid x \mid l \mid e_0 + e_1 \mid e_0 - e_1 \mid e_0 \times e_1 \mid a[e]$$

$$[\![\_]\!]\_ : \mathbf{AExp}^{+} \to (\mathbf{PVar} \to \mathbb{Z}) \to (\mathbf{LVar} \to \mathbb{Z}) \to \mathbb{Z}$$

$$[\![n]\!]\sigma\alpha = n$$

$$[\![x]\!]\sigma\alpha = \sigma_P(x)$$

$$[\![l]\!]\sigma\alpha = \alpha(l), l \in \mathbf{LVar}$$

$$[\![a[e]]\!]\sigma\alpha = \sigma_A([\![e]\!]\sigma\alpha)$$

$$[\![e_0\ aop\ e_1]\!]\sigma\alpha = [\![e_0]\!]\sigma\alpha\ aop\ [\![e_1]\!]\sigma\alpha, \quad aop \in \{+, -, \times\}$$

In what follows, we assume the reader is familiar with the satisfaction relation between structures and formulas in first-order logic. See for instance Loeckx and Sieber (1987), Huth and Ryan (2004). However, in traditional exposition of logic like these and others, the satisfaction relation $\models$ is a subset of the Cartesian product **Env** × **Assn**. We have to consider states for program and array variables as well. Thus, our satisfaction relation, needed to define the semantics of Hoare triples, is a subset $(\_, \_) \models \_ \subseteq (\Sigma \times \mathbf{Env}) \times \mathbf{Assn}$. The notation $(\sigma, \alpha) \models A$ states that the program assertion $A$ is true at a state $\sigma$ and environment $\alpha$. A program assertion $A$ is called (arithmetic) *valid*, written $\models A$, iff $\forall \sigma \in \Sigma, \forall\, \alpha : \mathbf{LVar} \to \mathbb{Z}.\ (\sigma, \alpha) \models A$.

We say that a *partial correctness assertion* $\{P\}\ c\ \{Q\}$ *is true* at a state $\sigma \in \Sigma$ and in an environment $\alpha : \mathbf{LVar} \to \mathbb{Z}$, written $(\sigma, \alpha) \models \{P\}\ c\ \{Q\}$ iff $(\sigma, \alpha) \models P$ and for all $\sigma' \in \Sigma$: $[\![c]\!]\sigma = \sigma'$ implies $(\sigma', \alpha) \models Q$. Finally, a partial correctness assertion is (arithmetic) *valid*, written $\models \{P\}\ c\ \{Q\}$, iff $\forall \sigma \in \Sigma, \alpha : \mathbf{LVar} \to \mathbb{Z}.\ (\sigma, \alpha) \models \{P\}\ c\ \{Q\}$.

Likewise, we say that a *total correctness assertion* $[P]\ c\ [Q]$ *is true* at a state $\sigma \in \Sigma$ and in an environment $\alpha : \mathbf{LVar} \to \mathbb{Z}$, written $(\sigma, \alpha) \models [P]\ c\ [Q]$ iff $(\sigma, \alpha) \models P$ implies $\exists \sigma' \in \Sigma. [\![c]\!]\sigma = \sigma'$ and $(\sigma', \alpha) \models Q$. Finally, a total correctness assertion is (arithmetic) *valid*, written $\models [P]\ c\ [Q]$, iff $\forall \sigma \in \Sigma, \forall \alpha : \mathbf{LVar} \to \mathbb{Z}.\ (\sigma, \alpha) \models [P]\ c\ [Q]$. Note that *total correctness* implies *partial correctness.*

The following rules of the *Hoare Proof Calculus* define inductively the theorems of the Hoare Logic for total correctness assertions over **IMP** programs. Removing the rule TWh, we have a calculus for partial correctness assertions. Note that every occurrence of a program or an array variable in an assertion is free. Only logical variables can be bound (by means of quantification). In the rule for Ass bellow, the expression $Q[x/e]$ means the simultaneous replacement of every (free) occurrence of the program variable $x$ in the assertion $Q$ by the arithmetic expression $e$. By the same token, in the rule AAss, the expression $Q[a/a[e \mapsto e']]$ means the simultaneous replacement of every (free) occurrence of the array variable $a$ by the new array $a[e \mapsto e']$. In the rule TWh, $M$ is a measure function (loop variant) on a set $D$ equipped with a well-founded order $(D, <)$ (usually the set of natural numbers).

$$\frac{}{\vdash \{P\}\ \underline{\mathbf{skip}}\ \{P\}}\ \mathsf{Skip} \qquad \frac{}{\vdash \{Q[x/e]\}\ x := e\ \{Q\}}\ \mathsf{Ass} \qquad \frac{}{\vdash \{Q[a/a[e \mapsto e']]\}a[e] := e'\{Q\}}\ \mathsf{AAss}$$

$$\frac{\vdash \{P\}\ c_1\ \{Q\} \quad \vdash \{Q\}\ c_2\ \{R\}}{\vdash \{P\}\ c_1; c_2\ \{R\}}\ \mathsf{Comp} \qquad \frac{\vdash \{P \wedge B\}\ c_1\ \{Q\} \quad \vdash \{P \wedge \neg B\}\ c_2\ \{Q\}}{\vdash \{P\}\ \underline{\mathbf{if}}\ b\ \underline{\mathbf{then}}\{c_1\}\ \underline{\mathbf{else}}\ \{c_2\}\ \underline{\mathbf{fi}}\ \{Q\}}\ \mathsf{IfE}$$

$$\frac{\vdash \{P \wedge B\}c\{P\}}{\vdash \{P\}\ \underline{\mathbf{while}}\ b\ \underline{\mathbf{do}}\ c\ \underline{\mathbf{od}}\ \{P \wedge \neg B\}}\ \mathsf{PWh} \qquad \frac{\vdash \{P \wedge B\}c\{P\} \quad P \wedge B \to M > 0 \quad \{P \wedge B \wedge M = m\}c\{M < m\}}{\vdash [P]\ \underline{\mathbf{while}}\ b\ \underline{\mathbf{do}}\ c\ \underline{\mathbf{od}}\ [P \wedge \neg B]}\ \mathsf{TWh}$$

$$\frac{\vdash P \to Q \quad \vdash \{Q\}\ c\ \{R\}}{\vdash \{P\}\ c\ \{R\}}\ \mathsf{Stren} \qquad \frac{\vdash \{P\}c\{Q\} \quad \vdash Q \to R}{\vdash \{P\}\ c\ \{R\}}\ \mathsf{Weakn}$$

**Proposition 1.** Let $\{P\}\ c\ \{Q\}$ be a partial correctness assertion. Then, the Hoare calculus is sound, that is, every theorem is a valid formula. More precisely, we have $\vdash \{P\}\ c\ \{Q\}$ only if $\models \{P\}\ c\ \{Q\}$ and $\vdash [P]\ c\ [Q]$ only if $\models [P]\ c\ [Q]$.

https://doi.org/10.1017/S0960129522000275 Published online by Cambridge University Press

**Table 2.** Categories used and introduced in the paper

| Category | | Objects | Morphisms |
|---|---|---|---|
| Standard categories | | | |
| Set | | Sets | Total functions |
| Par | | Sets | Partial functions |
| Cat | | Small categories | Functors |
| Pre | | Preorders (seen as categories) | Monotone functions (functors) |
| Po | | Partial orders (seen as categories) | Monotone functions (functors) |
| Mon | | Monoids (seen as categories) | Monoid morphisms (functors) |
| Categories for Hoare logic | | | |
| Cont | (p. 1153) | Finite contexts $\gamma \subseteq$ **Var** (finite sets of program and array variables) | Inclusions functions $in_{\gamma,\gamma'} : \gamma \hookrightarrow \gamma'$ |
| $\overline{\text{Cont}}$ | (p. 1153) | Extended contexts $\lambda = (\gamma, \delta)$ with $\delta$ a finite set of logical variables | Pairs of inclusion functions $(in_{\gamma,\gamma'}, in_{\delta,\delta'}) : (\gamma, \delta) \to (\gamma', \delta')$ |
| Ent | (p. 1164) | Pairs $(\lambda.Q)$ with $Q$ a local state assertion in extended context $\lambda$ | Pairs of a morphism in $\overline{\text{Cont}}$ and a semantic entailment between local state assertions |
| Pred | (p. 1165) | Pairs $(\lambda..\mathscr{Q})$ with $\mathscr{Q} \in \wp(\Lambda_\lambda) = \wp(\Sigma_\gamma \times \Gamma_\delta)$ a local state predicate | Pairs of a morphism in $\overline{\text{Cont}}$ and an inclusion between local state predicates |
| Prg | (p. 1166) | $\lvert\text{Prg}\rvert = \lvert\overline{\text{Cont}}\rvert$ | Pairs of a morphism $(in_{\gamma,\gamma'}, in_{\delta,\delta'})$ in $\overline{\text{Cont}}$ and a program in extended context $\lambda'$ |
| Wp | (p. 1167) | $\lvert\text{Wp}\rvert = \lvert\text{Pred}\rvert$ | Pairs of a morphism in Prg and an inclusion of a local state predicate in a semantic weakest precondition |
| TC | (p. 1169) | $\lvert\text{TC}\rvert = \lvert\text{Ent}\rvert$ | Pairs of a morphism in Prg and a semantic entailment between a local state assertion and a syntactic weakest precondition |

## Categories and Fibrations

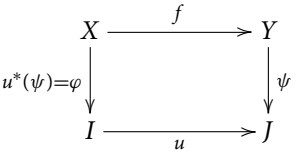
We assume the reader has a working knowledge of first-order logic, that is, its language, and basic model and proof theory. Likewise, the reader is required to have familiarity with the language of category theory, including basic limits and colimits constructions, as well as with the concepts of functors and natural transformations. However, in order to improve readability, we list in Table 2 all categories introduced and used in the paper and give a quick introduction to fibrations, which follows below.

The interplay between fibered and indexed constructions, we will rely on in this paper, is quite well-known. Indexed families of sets $(X_i)_{i\in I}$ and display maps $\varphi : X \to I$ be considered as the motivating set-theoretical concepts for their categorical counterparts, indexed and fibered categories, respectively. These concepts are actually equivalent. Given a family of sets $(X_i)_{i\in I}$, we take $X$ to be the disjoint union $\coprod_{i\in I} X_I = \{(x, i) \mid x \in I, i \in I\}$. This construction comes equipped with a projection function $\pi : \coprod_{i\in I} X_I \to I$, $(x, i) \mapsto i$. Conversely, given a function $\varphi : X \to I$, we take $X_I \triangleq \varphi^{-1}(i)$. The sets $\varphi^{-1}(i)$ are called the fibers of $X$. This defines a collection $(X_i)_{i\in I}$ together with an isomorphism $X \cong \coprod_{i\in I} X_I$.

**Definition 2** (Fibers)**.** *For any functor* $P : \mathsf{C} \to \mathsf{Ind}$*, the fiber* $\mathsf{C}_i$ *over an object* $i$ *of* $\mathsf{Ind}$ *is the subcategory of* $\mathsf{C}$ *given by the collection of objects* $a$ *such that* $P(a) = i$*, and the arrows* $u : c \to c'$ *of* $\mathsf{C}$ *for which* $P(u) = id_i$*.* □

https://doi.org/10.1017/S0960129522000275 Published online by Cambridge University Press

The idea of substitution can be seen as a motivation for the concept of *Cartesian arrow*. Consider a family $\psi : Y \to J$ over a set $J$. *Substitution* involves changing the index of the set $J$. Thus substitution along a function $u : I \to J$ involves creating a family of sets with the domain $I$ of $u$ as the new index set, and with fibers $Y_{u(i)}$ for $i \in I$. Thus, the family $(Y_j)_{j \in J}$ is mapped to a family $(X_i)_{i \in I}$ with $X_i = Y_{u(i)}$. This family can be obtained from the pullback of $\psi$ against $u$:

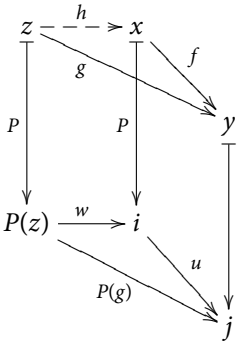



Thus, we have the set $X = \{(i, y) \in I \times Y \mid u(i) = \psi(y)\}$, with projections $\varphi : X \to I, f : X \to Y$. In this way, we obtain a new family $\varphi : X \to I$ over $I$, with fibers $X_i = \varphi^{-1}(i) \cong \{y \in Y \mid \psi(y) = u(i)\} = \psi^{-1}(u(i)) = Y_{u(i)}$. One normally writes $u^*(\psi)$ for the result $\varphi$ of substituting $\psi$ along $u$. The higher-order function $u^*$ is called the substitution function (functor).

**Definition 3** (Cartesian Arrow, Fibration)**.** *Let* $P : \mathsf{C} \to \mathsf{Ind}$ *be a functor.*

*(1) An arrow* $f : x \to y$ *in* $\mathsf{C}$ *is Cartesian over* $u : i \to j$ *in* $\mathsf{Ind}$*, if* $P(f) = u$ *and every* $g : z \to y$ *in* $\mathsf{C}$ *for which one has* $P(g) = w$*;* $u$ *for some* $w : P(z) \to i$*, uniquely determines an* $h : z \to x$ *in* $\mathsf{C}$ *above* $w$ *with* $g = h; f$*. We call* $f : x \to y$ *in the total category* $\mathsf{C}$ *Cartesian if it is Cartesian over its underlying* $u = P(f)$ *in* $\mathsf{Ind}$*.*

*(2) The functor* $P : \mathsf{C} \to \mathsf{Ind}$ *is a fibration if for every object* $y$ *in* $\mathsf{C}$ *and every* $u : i \to P(y)$ *in* $\mathsf{Ind}$*, there is a Cartesian arrow* $f : x \to y$ *in* $\mathsf{C}$ *above* $u$*. This Cartesian arrow is also called a Cartesian lifting of* $u$*.*




## 3. Hoare Logic and Indexed Categories

In this section, we analyze the Hoare logic of while-programs by means of indexed categories and indexed functors (natural transformations) between them. We are interested to answer, at least partially, the fundamental question "What are the characteristic structural features of Hoare logic?"

One basic structural feature of a Hoare logic we observed already, namely that a Hoare logic is defined in three steps: First, we define an appropriate concept of state and develop a corresponding suitable logic of states. Second, we define the syntax of programs as well as their semantics as state transforming entities. Third, we build a logic of programs based on the idea that a state transformation is reflected by a corresponding transformation of state assertions (predicates). We will proceed our analysis along this three step procedure.

In Section 2, the context of all programs is the same infinite set of program variables; thus, states are infinite "global" entities and assertions are defined, correspondingly, as statements about

https://doi.org/10.1017/S0960129522000275 Published online by Cambridge University Press

infinite "global entities." The fact that a program, as a finite entity build upon a finite set of program variables, changes only a finite fragment of an infinite global state remained implicit in the definitions.

However, in our categorical modeling of Hoare logic we consider Hoare triples as logical formulas in finite contexts of program and logical variables. As shown in Examples 1, 2, and 3, this is the precise way a programmer writes and understands Hoare triples. This understanding is aligned with modern approaches to the formalization of logics and type theories as in Crole (1993), Jacobs (2001), Pitts (2000), for example. Moreover, in current implementations of Hoare Logic as Bubel and Hähnle (2016), Martini (2020), Pierce et al. (2018*a*,b) and in programming languages which support specifications based on pre- and postconditions like Leino (2010), the programmer must explicitly declare, alongside the program, the program and logical variables that appear in the specification of the program. In other words: Our decision to use finite contexts of variables is a reasonable choice backed up by two essential rationals. First, the methodology enforced by a categorical treatment of logics and type theories leads naturally to the use of finite contexts.

Second, we want to represent faithfully the essential components of Hoare specifications in programming practice. *Program code, logical formulas and variable declarations are always finite objects in the mind of the working programmer*.

So, our second objective is to transform the global infinitary version of the Hoare logic for while-programs, presented in Section 2, into an equivalent local finitary version. Our analysis will be based on indexed and fibered concepts and structures, as presented, for example, in Martini et al. (2007), since these are the tools of choice to describe and reason about the structures arising by the transformation of monolithic infinite entities into infinite collections of inter-related finite entities.

### *3.1 Logic of local states*

For our analysis of the structural features of Hoare logics, the distinction between program variables **PVar** and array variables **AVar** is irrelevant; thus, we work, from now on, only with the set $\mathbf{Var} \triangleq \mathbf{PVar} \uplus \mathbf{AVar}$ where we refer to the elements of **Var** also simply as "program variables."

**Contexts and Local States:** Any program $c$ will at most change the values of the program variables in the finite set $pvr(c) \subseteq \mathbf{Var}$ of all program variables appearing in $c$.[1] Program $c$ may, however, be part of different bigger programs $c'$; thus, we should consider any finite set $\gamma \subseteq \mathbf{Var}$ with $pvr(c) \subseteq \gamma$ as a potential context of $c$. Therefore, we transform the infinite set **Var** of program variables into the partial order category[2] $\mathsf{Cont}$ with $|\mathsf{Cont}| \triangleq \wp_{fin}(\mathbf{Var})$, that is, the set of all finite contexts, and with morphisms all inclusion functions $in_{\gamma,\gamma'} : \gamma \hookrightarrow \gamma'$ corresponding to inclusions $\gamma \subseteq \gamma'$.

The semantics of a context $\gamma \in |\mathsf{Cont}| = \wp_{fin}(\mathbf{Var})$ is the corresponding set of local states $\Sigma_\gamma \triangleq (\gamma \to \mathbb{D})$, that is, of all type compatible functions $\sigma : \gamma \to \mathbb{D}$ where $\mathbb{D} \triangleq \mathbb{Z} \uplus (\mathbb{Z} \to \mathbb{Z})$. Any inclusion function $in_{\gamma,\gamma'} : \gamma \hookrightarrow \gamma'$ induces a reduction map $p_{\gamma',\gamma} : \Sigma_{\gamma'} \to \Sigma_\gamma$ given by precomposition:

$$p_{\gamma',\gamma}(\sigma') \triangleq in_{\gamma,\gamma'}; \sigma' \quad \text{for all local states } \sigma' \in \Sigma_{\gamma'} = (\gamma' \to \mathbb{D}). \tag{2}$$

The assignments $\gamma \mapsto \Sigma_\gamma$ and $in_{\gamma,\gamma'} \mapsto p_{\gamma',\gamma}$ define a functor $\mathtt{st} : \mathsf{Cont}^{op} \to \mathsf{Set}$.

**Extended Contexts and Local State Assertions:** In Hoare logic, assertions are used to describe properties of states. Assertions are built upon expressions which, in turn, are built upon program variables and logical variables. Only logical variables are quantified in assertions and the semantics of expressions and assertions depends only on program variables and free logical variables.

An assertion contains, besides finitely many program variables, only finitely many logical variables. Thus, we will work, besides contexts $\gamma \in \wp_{fin}(\mathbf{Var})$, also with finite sets $\delta \in \wp_{fin}(\mathbf{LVar})$ of (free) logical variables and use the term "(variable) declaration" for those finite sets of logical variables (see Examples 1, 2, 3). Indeed, modern implementations of tools built on top of Hoare logic

https://doi.org/10.1017/S0960129522000275 Published online by Cambridge University Press

are based on finite contexts of program and logical variables (see Bubel and Hähnle 2016; Leino 2010; Martini 2020; Pierce et al. 2018*a*).

In such a way, we can extend the category $\mathsf{Cont}$ to a category $\overline{\mathsf{Cont}}$ with objects $|\overline{\mathsf{Cont}}| \triangleq \wp_{fin}(\mathbf{Var}) \times \wp_{fin}(\mathbf{LVar})$ pairs of finite sets of local variables, also called "extended contexts," and morphisms given by pairs of inclusion functions $(in_{\gamma,\gamma'}, in_{\delta,\delta'}) : (\gamma, \delta) \to (\gamma', \delta')$.

The semantics of a declaration $\delta \in \wp_{fin}(\mathbf{LVar})$ is the set of local environments $\Gamma_\delta \triangleq (\delta \to \mathbb{Z})$, that is, of all functions $\alpha : \delta \to \mathbb{Z}$. Analogously to contexts, any inclusion function $in_{\delta,\delta'} : \delta \hookrightarrow \delta'$ between declarations induces a reduction map $p_{\delta',\delta} : \Gamma_{\delta'} \to \Gamma_\delta$ given by precomposition:

$$p_{\delta',\delta}(\alpha') \triangleq in_{\delta,\delta'}; \alpha' \quad \text{for all local environments } \alpha' \in \Gamma_{\delta'} = (\delta' \to \mathbb{Z}). \tag{3}$$

We can extend the functor $\mathtt{st} : \mathsf{Cont}^{op} \to \mathsf{Set}$ to a functor $\overline{\mathtt{st}} : \overline{\mathsf{Cont}}^{op} \to \mathsf{Set}$ that assigns to each "extended context" $\lambda = (\gamma, \delta)$ the corresponding set $\Lambda_\lambda \triangleq \Sigma_\gamma \times \Gamma_\delta$ of "extended local states" and to each pair $in_{\lambda,\lambda'} \triangleq (in_{\gamma,\gamma'}, in_{\delta,\delta'}) : \lambda \to \lambda'$ of inclusion functions the product $p_{\lambda',\lambda} \triangleq p_{\gamma',\gamma} \times p_{\delta',\delta} : \Lambda_{\lambda'} \to \Lambda_\lambda$ of reduction maps.

**Local State Assertions:** We consider for any extended context $\lambda = (\gamma, \delta) \in |\overline{\mathsf{Cont}}| = \wp_{fin}(\mathbf{Var}) \times \wp_{fin}(\mathbf{LVar})$ the corresponding set of local state assertions:

$$\mathtt{assn}(\lambda) = \mathtt{assn}(\gamma, \delta) \triangleq \{P \in \mathbf{Assn} \mid pvr(P) \subseteq \gamma, flv(P) \subseteq \delta\}, \tag{4}$$

where $flv(P)$ is the set of all free logical variables appearing in $P$. For any morphism $in_{\lambda,\lambda'} = (in_{\gamma,\gamma'}, in_{\delta,\delta'}) : \lambda \to \lambda'$ in $\overline{\mathsf{Cont}}$, we do have $\mathtt{assn}(\lambda) \subseteq \mathtt{assn}(\lambda')$ since $pvr(P) \subseteq \gamma$ entails $pvr(P) \subseteq \gamma'$ and $flv(P) \subseteq \delta$ entails $flv(P) \subseteq \delta'$, respectively. That is, we obtain an inclusion function $\mathtt{assn}(in_{\lambda,\lambda'}) : \mathtt{assn}(\lambda) \hookrightarrow \mathtt{assn}(\lambda')$. The assignments $\lambda \mapsto \mathtt{assn}(\lambda)$ and $in_{\lambda,\lambda'} \mapsto \mathtt{assn}(in_{\lambda,\lambda'})$ define obviously a functor $\mathtt{assn} : \overline{\mathsf{Cont}} \to \mathsf{Set}$.

**Satisfaction Relation:** The usual inductive definition of the infinite version of satisfaction of assertions can be easily modified for local state assertions. We get, in such a way, a $|\overline{\mathsf{Cont}}|$-indexed family of satisfaction relations between extended local states, on one side, and local state assertions, on the other side:

$$\models_\lambda \subseteq \Lambda_\lambda \times \mathtt{assn}(\lambda) \quad \text{with } \lambda \in |\overline{\mathsf{Cont}}| = \wp_{fin}(\mathbf{Var}) \times \wp_{fin}(\mathbf{LVar}).$$

Moreover, satisfaction is compatible w.r.t. morphisms in $\overline{\mathsf{Cont}}$:

**Proposition 2** (Satisfaction Condition)**.** For any morphism $in_{\lambda,\lambda'} = (in_{\gamma,\gamma'}, in_{\delta,\delta'}) : \lambda \to \lambda'$ in $\overline{\mathsf{Cont}}$, any extended local state $(\sigma', \alpha') \in \Lambda_{\lambda'}$ and any local state assertion $P \in \mathtt{assn}(\lambda)$ we have

$$p_{\lambda',\lambda}(\sigma', \alpha') = (p_{\gamma',\gamma}(\sigma'), p_{\delta',\delta}(\alpha')) \models_\lambda P \quad \text{iff} \quad (\sigma', \alpha') \models_{\lambda'} \mathtt{assn}(in_{\lambda,\lambda'})(P) = P.$$

*Proof.* Due to our definitions, we have $(pvr(P), flv(P)) \subseteq \lambda \subseteq \lambda'$ and that $(\sigma', \alpha')$ coincides with $p_{\lambda',\lambda}(\sigma', \alpha')$ on $(pvr(P), flv(P))$; thus, the satisfaction condition states that the validity of an assertion $P$ only depends on the values assigned to the variables in $(pvr(P), flv(P))$. □

**Remark 2** (Logic of States is an Institution)**.** A closer look at the development so far shows that we have actually defined an *Institution* (see Diaconescu 2008; Goguen and Burstall 1992): The category of abstract signatures is the category $\overline{\mathsf{Cont}}$. The sentence functor is $\mathtt{assn} : \overline{\mathsf{Cont}} \to \mathsf{Set}$ while $\overline{\mathtt{st}} : \overline{\mathsf{Cont}}^{op} \to \mathsf{Set}$ is the model functor. Due to Proposition 2, the $|\overline{\mathsf{Cont}}|$-indexed family of satisfaction relations $\models_\lambda$ meets the necessary satisfaction condition.

As shown in Wolter et al. (2012), this allows us to define the semantics of assertions based on the contravariant power set construction. □

https://doi.org/10.1017/S0960129522000275 Published online by Cambridge University Press

**Semantics of Local State Assertions:** Any subset of $\Lambda_\lambda = \Sigma_\gamma \times \Gamma_\delta$ describes a certain property of extended local states and therefore we consider the elements of $\wp(\Lambda_\lambda)$ also as "state predicates."

A local state assertion $P \in \mathtt{assn}(\lambda)$ can be seen as the syntactic representation of a certain state predicate, namely of its semantics, that is, the set of all extended local states satisfying $P$:

$$\mathtt{sem}_\lambda(P) \triangleq \{(\sigma,\alpha) \in \Lambda_\lambda \mid (\sigma,\alpha) \models_\lambda P\} \in \wp(\Lambda_\lambda) \tag{5}$$

For all objects $\lambda$ in $\overline{\mathsf{Cont}}$, this defines a function $\mathtt{sem}_\lambda : \mathtt{assn}(\lambda) \to \wp(\Lambda_\lambda)$. To answer the question if this family of functions constitutes a relevant natural transformation from local state assertions to semantics, we construct first the target of this natural transformation: Composing the functor $\overline{\mathtt{st}}^{op} : \overline{\mathsf{Cont}} \to \mathsf{Set}^{op}$ with the contravariant power set functor $\mathtt{P} : \mathsf{Set}^{op} \to \mathsf{Pre}$, where $\mathsf{Pre}$ is the category of preorders and monotone functions, we obtain a functor $\mathtt{pred} : \overline{\mathsf{Cont}} \to \mathsf{Pre}$ with $\mathtt{pred}(\lambda) \triangleq (\wp(\Lambda_\lambda), \subseteq)$ for all objects $\lambda = (\gamma,\delta) \in |\overline{\mathsf{Cont}}|$ and with

$$\mathtt{pred}(in_{\lambda,\lambda'}) \triangleq p^{-1}_{\lambda',\lambda} : (\wp(\Lambda_\lambda), \subseteq) \longrightarrow (\wp(\Lambda_{\lambda'}, \subseteq))$$

for all morphisms $in_{\lambda,\lambda'} : \lambda \to \lambda'$ in $\overline{\mathsf{Cont}}$, that is, for all state predicates $\mathscr{P} \subseteq \Lambda_\lambda$ we have

$$p^{-1}_{\lambda',\lambda}(\mathscr{P}) = \{(\sigma',\alpha') \in \Lambda_{\lambda'} \mid p_{\lambda',\lambda}(\sigma',\alpha') = (p_{\gamma',\gamma}(\sigma'), p_{\delta',\delta}(\alpha')) \in \mathscr{P}\}. \tag{6}$$

Since the formation of inverse images is monotone w.r.t. set inclusions, we obtain indeed a functor from $\overline{\mathsf{Cont}}$ into $\mathsf{Pre}$. Note that the preorders $\mathtt{pred}(\lambda) = (\wp(\Lambda_\lambda), \subseteq)$ are even partial orders.

Second, we can borrow the order relation in $(\wp(\Lambda_\lambda), \subseteq)$, to define semantic entailment.

**Semantic Entailment:** For any local state assertions $P, Q \in \mathtt{assn}(\lambda)$ we define

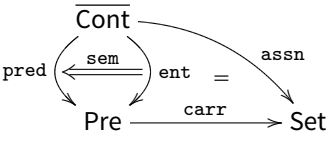
$$P \Vdash_\lambda Q \quad \text{iff} \quad \mathtt{sem}_\lambda(P) \subseteq \mathtt{sem}_\lambda(Q). \tag{7}$$

In categorical terms, we extend the set $\mathtt{assn}(\lambda)$ to a preorder $\mathtt{ent}(\lambda) \triangleq (\mathtt{assn}(\lambda), \Vdash_\lambda)$ in such a way that the function $\mathtt{sem}_\lambda : \mathtt{assn}(\lambda) \to \wp(\Lambda_\lambda)$ turns into a morphism $\mathtt{sem}_\lambda : \mathtt{ent}(\lambda) \to \mathtt{pred}(\lambda)$ in the category $\mathsf{Pre}$ that not only preserves but also reflects order, that is, $\mathtt{sem}_\lambda$ is a full functor.

**Remark 3** (Cartesian Closed Category)**.** The preorder category $\mathtt{ent}(\lambda) = (\mathtt{assn}(\lambda), \Vdash_\lambda)$, for any object $\lambda$ in $\overline{\mathsf{Cont}}$, is Cartesian closed with products $\wedge$, sums $\vee$ and exponentiation $\to$, that is, we have

$$P \wedge Q \Vdash_\lambda R \quad \text{iff} \quad P \Vdash_\lambda (Q \to R). \qquad \square$$

Proposition 2 entails, that for any morphism $in_{\lambda,\lambda'} : \lambda \to \lambda'$ in $\overline{\mathsf{Cont}}$ the corresponding inclusion function $\mathtt{assn}(in_{\lambda,\lambda'}) : \mathtt{assn}(\lambda) \hookrightarrow \mathtt{assn}(\lambda')$ is monotone w.r.t. semantic entailment, that is, for all assertions $P, Q \in \mathtt{assn}(\lambda)$ we have that $P \Vdash_\lambda Q$, that is, $\mathtt{sem}_\lambda(P) \subseteq \mathtt{sem}_\lambda(Q)$, implies $P \Vdash_{\lambda'} Q$, that is, $\mathtt{sem}_{\lambda'}(P) \subseteq \mathtt{sem}_{\lambda'}(Q)$. This means that the inclusion function $\mathtt{assn}(in_{\lambda,\lambda'}) : \mathtt{assn}(\lambda) \hookrightarrow \mathtt{assn}(\lambda')$ establishes, actually, a morphism $\mathtt{ent}(in_{\lambda,\lambda'}) \triangleq \mathtt{assn}(in_{\lambda,\lambda'}) : \mathtt{ent}(\lambda) \to \mathtt{ent}(\lambda')$ in the category $\mathsf{Pre}$. In such a way, the functor $\mathtt{assn} : \overline{\mathsf{Cont}} \to \mathsf{Set}$ lifts up to a functor $\mathtt{ent} : \overline{\mathsf{Cont}} \to \mathsf{Pre}$ such that the composition of $\mathtt{ent}$ with the forgetful functor $\mathtt{carr} : \mathsf{Pre} \to \mathsf{Set}$, assigning to each preorder its carrier set, equals $\mathtt{assn}$.




Finally, Proposition 2 ensures also that the morphisms $\mathtt{sem}_\lambda : \mathtt{ent}(\lambda) \to (\wp(\Lambda_\lambda), \subseteq)$ constitute a natural transformation $\mathtt{sem} : \mathtt{ent} \Rightarrow \mathtt{pred} : \overline{\mathsf{Cont}} \to \mathsf{Pre}$ as stated in the following proposition (compare Lemma 3.7 in Wolter et al. 2012):

https://doi.org/10.1017/S0960129522000275 Published online by Cambridge University Press

**Proposition 3.** The morphisms $\mathtt{sem}_\lambda : \mathtt{ent}(\lambda) \to \mathtt{pred}(\lambda)$ in Pre with $\lambda \in |\overline{\mathsf{Cont}}|$ constitute a natural transformation $\mathtt{sem} : \mathtt{ent} \Rightarrow \mathtt{pred} : \overline{\mathsf{Cont}} \to \mathsf{Pre}$.

$$\begin{array}{ccccc}
\lambda = (\gamma, \delta) & & \Lambda_\lambda = \Sigma_\gamma \times \Gamma_\delta & & \mathtt{pred}(\lambda) = (\wp(\Lambda_\lambda), \subseteq) \xleftarrow{\mathtt{sem}_\lambda} \mathtt{ent}(\lambda) = (\mathtt{assn}(\lambda), \Vdash_\lambda) \\
{\scriptstyle in_{\lambda,\lambda'} =} \downarrow {\scriptstyle (in_{\gamma,\gamma'}, in_{\delta,\delta'})} & & {\scriptstyle p_{\lambda',\lambda} =} \uparrow {\scriptstyle p_{\gamma',\gamma} \times p_{\delta',\delta}} & & p^{-1}_{\lambda',\lambda} \downarrow \qquad\qquad {\scriptstyle \mathtt{ent}(in_{\lambda,\lambda'}) =} \downarrow {\scriptstyle \mathtt{assn}(in_{\lambda,\lambda'})} \\
\lambda' = (\gamma', \delta') & & \Lambda_{\lambda'} = \Sigma_{\gamma'} \times \Gamma_{\delta'} & & \mathtt{pred}(\lambda') = (\wp(\Lambda_{\lambda'}), \subseteq) \xleftarrow{\mathtt{sem}_{\lambda'}} \mathtt{ent}(\lambda') = (\mathtt{assn}(\lambda'), \Vdash_{\lambda'})
\end{array}$$

### *3.2 Local programs and state transition semantics*

Programs are defined prior to and independent of logical variables and the semantics of programs are partial state transition maps between corresponding sets of states (see Definition 1). In this subsection, we develop a local finitary version of the state transition semantics of programs.

**Local Programs – Syntax:** "Local programs" are programs in a context; that is, for each context $\gamma$ we consider the corresponding set $\mathtt{prg}(\gamma) \triangleq \{c \in \mathbf{Prg} \mid pvr(c) \subseteq \gamma\}$ of programs in context $\gamma$. Our while-programs are sequential, that is, for any two local programs $c_1, c_2 \in \mathtt{prg}(\gamma)$ there is a unique local program $c_1; c_2 \in \mathtt{prg}(\gamma)$ and the concatenation operator $\_;\_$ is, in addition, assumed to be associative. Adding to $\mathtt{prg}(\gamma)$ an "empty program" $\varepsilon$ such that $c; \varepsilon = \varepsilon; c = c$ for all $c \in \mathtt{prg}(\gamma)$, we upgrade $\mathtt{prg}(\gamma)$ to a monoid. We consider monoids as categories with exactly one object. In abuse of notation, we denote the syntactic category with the only object $\gamma$ and the set of morphisms $\mathtt{prg}(\gamma)$, where composition is sequential concatenation of programs, also by $\mathtt{prg}(\gamma)$. For any inclusion function $in_{\gamma,\gamma'} : \gamma \hookrightarrow \gamma'$, we get obviously an inclusion functor $\mathtt{prg}_{\gamma,\gamma'} : \mathtt{prg}(\gamma) \hookrightarrow \mathtt{prg}(\gamma')$ thus the assignments $\gamma \mapsto \mathtt{prg}(\gamma)$ and $in_{\gamma,\gamma'} \mapsto \mathtt{prg}_{\gamma,\gamma'}$ define a functor $\mathtt{prg} : \mathsf{Cont} \to \mathsf{Mon}$.

**Transitions of Local States:** The semantics of a local program $c \in \mathtt{prg}(\gamma)$ is a partial function from the corresponding set $\Sigma_\gamma$ of local states into itself. To define this semantics precisely and in a well-structured way, we present, first, a brief account of those partial state transition maps.

We denote by Par the category of all sets and partial functions between sets.[3] For any context $\gamma$, we consider the monoid $\mathtt{pf}(\gamma)$ of local state transition maps with object $\Sigma_\gamma$ and the whole hom-set $(\Sigma_\gamma \nrightarrow \Sigma_\gamma) = \mathsf{Par}(\Sigma_\gamma, \Sigma_\gamma)$ as morphisms. "pf" stands for "partial function."

For any inclusion function $in_{\gamma,\gamma'} : \gamma \hookrightarrow \gamma'$, we can define a function $\mathtt{pf}_{\gamma,\gamma'} : (\Sigma_\gamma \nrightarrow \Sigma_\gamma) \to (\Sigma_{\gamma'} \nrightarrow \Sigma_{\gamma'})$ that lifts any local state transition map $\tau : \Sigma_\gamma \nrightarrow \Sigma_\gamma$ to a local state transition map $\tau' = \mathtt{pf}_{\gamma,\gamma'}(\tau) : \Sigma_{\gamma'} \nrightarrow \Sigma_{\gamma'}$. The construction goes like this: we define the domain of definition $\mathrm{DD}(\tau') \triangleq p^{-1}_{\gamma',\gamma}(\mathrm{DD}(\tau))$ using the corresponding reduction map $p_{\gamma',\gamma} : \Sigma_{\gamma'} \to \Sigma_\gamma$, defined in (2). Now we set for all $(\sigma' : \gamma' \to \mathbb{D}) \in \mathrm{DD}(\tau') \subseteq \Sigma_{\gamma'}$

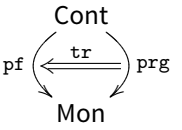
$$\begin{array}{ccccc}
\gamma' & & \Sigma_{\gamma'} & \hookleftarrow \mathrm{DD}(\tau') \xrightarrow{\tau'} & \Sigma_{\gamma'} \\
\uparrow {\scriptstyle in_{\gamma,\gamma'}} & & \downarrow {\scriptstyle p_{\gamma',\gamma}} & \downarrow {\scriptstyle p_{\gamma',\gamma}} & \downarrow {\scriptstyle p_{\gamma',\gamma}} \\
\gamma & & \Sigma_\gamma & \hookleftarrow \mathrm{DD}(\tau) \xrightarrow{\tau} & \Sigma_\gamma
\end{array} \qquad pb \qquad = \qquad \tau'(\sigma')(x) \triangleq \begin{cases} \tau(p_{\gamma',\gamma}(\sigma'))(x), & \text{if } x \in \gamma \\ \sigma'(x), & \text{if } x \in \gamma' \setminus \gamma \end{cases} \tag{8}$$

Thus, we obtain, especially, $\tau'; p_{\gamma',\gamma} = p_{\gamma',\gamma}; \tau$ in Par. This ensures $\mathtt{pf}_{\gamma,\gamma'}(\tau_1; \tau_2) = \mathtt{pf}_{\gamma,\gamma'}(\tau_1); \mathtt{pf}_{\gamma,\gamma'}(\tau_2)$ for all $\tau_1, \tau_2 : \Sigma_\gamma \nrightarrow \Sigma_\gamma$. Moreover, the definition entails $\mathtt{pf}_{\gamma,\gamma'}(id_{\Sigma_\gamma}) = id_{\Sigma_{\gamma'}}$ thus the function $\mathtt{pf}_{\gamma,\gamma'} : (\Sigma_\gamma \nrightarrow \Sigma_\gamma) \to (\Sigma_{\gamma'} \nrightarrow \Sigma_{\gamma'})$ establishes a functor $\mathtt{pf}_{\gamma,\gamma'} : \mathtt{pf}(\gamma) \to \mathtt{pf}(\gamma')$ between the monoids $\mathtt{pf}(\gamma)$ and $\mathtt{pf}(\gamma')$.

We do have $\mathtt{pf}_{\gamma,\gamma} = id_{\mathtt{pf}(\gamma)}$, and for any inclusions $\gamma \subseteq \gamma' \subseteq \gamma''$, we obtain $\mathtt{pf}_{\gamma,\gamma'}; \mathtt{pf}_{\gamma',\gamma''} = \mathtt{pf}_{\gamma,\gamma''}$ since the formation of inverse images is compositional and since $\gamma'' \setminus \gamma = (\gamma'' \setminus \gamma') \cup (\gamma' \setminus \gamma)$. In such a way, the assignments $\gamma \mapsto \mathtt{pf}(\gamma)$ and $in_{\gamma,\gamma'} \mapsto \mathtt{pf}_{\gamma,\gamma'}$ define a functor $\mathtt{pf} : \mathsf{Cont} \to \mathsf{Mon}$.

https://doi.org/10.1017/S0960129522000275 Published online by Cambridge University Press

**Local Programs – State Transition Semantics:** The state transition semantics of local programs is simply defined by restricting the global state transition semantics $[\![\_]\!] : \mathbf{Prg} \to (\Sigma \nrightarrow \Sigma)$ from Definition 1 to local programs and local states, respectively. We show that such a restriction of the




global semantics can be constructed in a way that we obtain a family $\mathtt{tr}_\gamma : \mathtt{prg}(\gamma) \to \mathtt{pf}(\gamma)$, $\gamma \in |\mathsf{Cont}|$ of functors between monoids establishing a natural transformation $\mathtt{tr} : \mathtt{prg} \Rightarrow \mathtt{pf}$. This natural transformation represents the state transition semantics of local programs.

First, we show that for any context $\gamma$ the global state transition semantics $[\![\_]\!] : \mathbf{Prg} \to (\Sigma \nrightarrow \Sigma)$ of programs restricts to a functorial state transition semantics $\mathtt{tr}_\gamma : \mathtt{prg}(\gamma) \to (\Sigma_\gamma \nrightarrow \Sigma_\gamma)$ for the corresponding local states: Analogously to (2), the inclusion function $in_\gamma : \gamma \hookrightarrow \mathbf{Var}$ induces a reduction map $p_\gamma : \Sigma \to \Sigma_\gamma$ with $p_\gamma(\varrho) \triangleq in_\gamma ; \varrho$ for all global states $\varrho \in \Sigma = (\mathbf{Var} \to \mathbb{D})$. $p_\gamma : \Sigma \to \Sigma_\gamma$ is surjective since $\mathbb{D}$ is not empty. By means of $p_\gamma$, we can restrict now for any local program $c \in \mathtt{prg}(\gamma)$ the partial function $[\![c]\!] : \Sigma \nrightarrow \Sigma$ to a partial function $\mathtt{tr}_\gamma(c) : \Sigma_\gamma \nrightarrow \Sigma_\gamma$: We define $\mathrm{DD}(\mathtt{tr}_\gamma(c)) \triangleq p_\gamma(\mathrm{DD}([\![c]\!]))$; thus, there exists for any local state $\sigma \in \mathrm{DD}(\mathtt{tr}_\gamma(c))$ a global state $\varrho \in \mathrm{DD}([\![c]\!])$ with $p_\gamma(\varrho) = \sigma$ and we can set $\mathtt{tr}_\gamma(c)(\sigma) \triangleq p_\gamma([\![c]\!](\varrho))$. Why does this work?

$$\begin{array}{ccccc}
\Sigma & \hookleftarrow & \mathrm{DD}([\![c]\!]) & \xrightarrow{[\![c]\!]} & \Sigma \\
\downarrow{\scriptstyle p_\gamma} & (1) & \downarrow{\scriptstyle p_\gamma} & = & \downarrow{\scriptstyle p_\gamma} \\
\Sigma_\gamma & \hookleftarrow & \mathrm{DD}(\mathtt{tr}_\gamma(c)) & \xrightarrow{\mathtt{tr}_\gamma(c)} & \Sigma_\gamma
\end{array}$$

Any program $c$ changes at most the values for the program variables in $pvr(c)$, that is, for any global state $\varrho \in \mathrm{DD}([\![c]\!])$ and any program variable $x \notin pvr(c)$ we have $[\![c]\!](\varrho)(x) = \varrho(x)$. We defined $c \in \mathtt{prg}(\gamma)$ iff $prv(c) \subseteq \gamma$, thus for any global states $\varrho, \varrho' \in \Sigma$ it holds that $\varrho \in \mathrm{DD}([\![c]\!])$ and $p_\gamma(\varrho) = p_\gamma(\varrho')$ implies $\varrho' \in \mathrm{DD}([\![c]\!])$ and $p_\gamma([\![c]\!](\varrho)) = p_\gamma([\![c]\!](\varrho'))$. This ensures that the definition of $\mathtt{tr}_\gamma(c)$ is independent of representatives as well as that the square (1) in the diagram above is a pullback in $\mathsf{Set}$, that is, we have $[\![c]\!]; p_\gamma = p_\gamma ; \mathtt{tr}_\gamma(c)$ in $\mathsf{Par}$.

For any local programs $c_1, c_2 \in \mathtt{prg}(\gamma)$ the compositionality $[\![c_1 ; c_2]\!] = [\![c_1]\!]; [\![c_2]\!]$ of global state transition semantics gives us compositionality $\mathtt{tr}_\gamma(c_1 ; c_2) = \mathtt{tr}_\gamma(c_1) ; \mathtt{tr}_\gamma(c_2)$ of the corresponding local state transition semantics at hand. Moreover, we set $\mathtt{tr}_\gamma(\varepsilon) \triangleq id_{\Sigma_\gamma}$ for the empty program $\varepsilon \in \mathtt{prg}(\gamma)$. This ensures that the function $\mathtt{tr}_\gamma : \mathtt{prg}(\gamma) \to (\Sigma_\gamma \nrightarrow \Sigma_\gamma)$ establishes indeed a functor $\mathtt{tr}_\gamma : \mathtt{prg}(\gamma) \to \mathtt{pf}(\gamma)$ from the monoid $\mathtt{prg}(\gamma)$ into the monoid $\mathtt{pf}(\gamma) = (\Sigma_\gamma \nrightarrow \Sigma_\gamma)$.

Second, we validate that the family $\mathtt{tr}_\gamma : \mathtt{prg}(\gamma) \to \mathtt{pf}(\gamma)$, $\gamma \in |\mathsf{Cont}|$ of functors provides a natural transformation $\mathtt{tr} : \mathtt{prg} \Rightarrow \mathtt{pf}$: For any morphism $in_{\gamma,\gamma'} : \gamma \to \gamma'$ in $\mathsf{Cont}$ we have the inclusion functor $\mathtt{prg}(in_{\gamma,\gamma'}) = \mathtt{prg}_{\gamma,\gamma'} : \mathtt{prg}(\gamma) \hookrightarrow \mathtt{prg}(\gamma')$ and the functor $\mathtt{pf}(in_{\gamma,\gamma'}) = \mathtt{pf}_{\gamma,\gamma'} : \mathtt{pf}(\gamma) \to \mathtt{pf}(\gamma')$. To validate the naturality condition we show that

$$\begin{array}{ccccc}
\Sigma_{\gamma'} & \hookleftarrow & \mathrm{DD}(\mathtt{pf}_{\gamma,\gamma'}(\mathtt{tr}_\gamma(c))) & \xrightarrow{\mathtt{pf}_{\gamma,\gamma'}(\mathtt{tr}_\gamma(c))} & \Sigma_{\gamma'} \\
\downarrow{\scriptstyle p_{\gamma',\gamma}} & pb & \downarrow{\scriptstyle p_{\gamma',\gamma}} & = & \downarrow{\scriptstyle p_{\gamma',\gamma}} \\
\Sigma_\gamma & \hookleftarrow & \mathrm{DD}(\mathtt{tr}_\gamma(c)) & \xrightarrow{\mathtt{tr}_\gamma(c)} & \Sigma_\gamma
\end{array} \qquad \mathtt{tr}_{\gamma'}(c) = \mathtt{pf}_{\gamma,\gamma'}(\mathtt{tr}_\gamma(c)) : \Sigma_{\gamma'} \multimap \Sigma_{\gamma'} \tag{9}$$

for each local program $c \in \mathtt{prg}(\gamma) \subseteq \mathtt{prg}(\gamma')$.

https://doi.org/10.1017/S0960129522000275 Published online by Cambridge University Press

Due to (8), we have $\mathtt{pf}_{\gamma,\gamma'}(\mathtt{tr}_\gamma(c)); p_{\gamma',\gamma} = p_{\gamma',\gamma}; \mathtt{tr}_\gamma(c)$ in Par. Since $p_\gamma = p_{\gamma'}; p_{\gamma',\gamma}$, the definition of the functors $\mathtt{tr}\_$ entails $p_{\gamma'}; \mathtt{tr}_{\gamma'}(c); p_{\gamma',\gamma} = p_{\gamma'}; p_{\gamma',\gamma}; \mathtt{tr}_\gamma(c) = p_{\gamma'}; \mathtt{pf}_{\gamma,\gamma'}(\mathtt{tr}_\gamma(c)); p_{\gamma',\gamma}$ and thus $\mathtt{tr}_{\gamma'}(c); p_{\gamma',\gamma} = p_{\gamma',\gamma}; \mathtt{tr}_\gamma(c) = \mathtt{pf}_{\gamma,\gamma'}(\mathtt{tr}_\gamma(c)); p_{\gamma',\gamma}$ in Par since $p_{\gamma'}$ is surjective, that is, epic, in Par. From the last equation we can conclude $\mathrm{DD}(\mathtt{tr}_{\gamma'}(c)) = \mathrm{DD}(\mathtt{pf}_{\gamma,\gamma'}(\mathtt{tr}_\gamma(c)))$ and that $\mathtt{tr}_{\gamma'}(c)(\sigma')(x) = \mathtt{tr}_\gamma(c)(p_{\gamma',\gamma}(\sigma'))(x) = \mathtt{pf}_{\gamma,\gamma'}(\mathtt{tr}_\gamma(c))(\sigma')(x)$ for all $\sigma' \in \mathrm{DD}(\mathtt{tr}_{\gamma'}(c))$ and all $x \in \gamma$. For all $x \in \gamma' \setminus \gamma$ and thus also $x \notin pvr(c)$, we have $\mathtt{tr}_{\gamma'}(c)(\sigma')(x) = \sigma'(x) = \mathtt{pf}_{\gamma,\gamma'}(\mathtt{tr}_\gamma(c))(\sigma')(x)$ due to the properties of $[\![c]\!]$, mentioned above, the definition of $\mathtt{tr}_{\gamma'}$ and the definition of $\mathtt{pf}_{\gamma,\gamma'}(\mathtt{tr}_\gamma(c))$.

### *3.3 State transition maps as predicate transformers*

Hoare triples are a logical means to describe and reason about the semantics of programs thereby relying on a corresponding logic of states. We consider here "local partial correctness assertions" $\lambda : \{P\}\, c\, \{Q\}$ and "local total correctness assertions" $\lambda : [P]\, c\, [Q]$ for extended contexts $\lambda = (\gamma, \delta)$ such that $c \in \mathtt{prg}(\gamma)$ and $P, Q \in \mathtt{assn}(\lambda)$.

A correctness assertion is an assertion about the state transition semantics $\mathtt{tr}_\gamma(c) : \Sigma_\gamma \nrightarrow \Sigma_\gamma$ of the local program $c \in \mathtt{prg}(\gamma)$ and is represented by a pair of local state assertions – a precondition $P$ describing properties of the "input states" $\sigma \in \Sigma_\gamma$ and a postcondition $Q$ describing properties of the corresponding "output states" $\mathtt{tr}_\gamma(c)(\sigma) \in \Sigma_\gamma$.

One can observe, however, that correctness assertions can be defined and investigated independent of programs namely as assertions about arbitrary state transition maps $\tau : \Sigma_\gamma \nrightarrow \Sigma_\gamma$. Following this observation, we develop in this subsection a local version of the predicate transformer semantics, as introduced in Dijkstra (1975), not only for programs but for arbitrary state transition maps. We consider as well total as partial correctness semantics and show that any of these semantics is equivalent to the state transition semantics.

**Correctness Assertions:** To underline the implicational "nature" of correctness assertions, we adapt an arrow notation for correctness assertions about arbitrary state transition maps.

**Definition 4** (General Correctness Assertions)**.** *Let be given an extended context* $\lambda = (\gamma, \delta)$ *and two assertions* $P, Q \in \mathtt{assn}(\lambda)$.

(1) *We say that a state transition map* $\tau : \Sigma_\gamma \nrightarrow \Sigma_\gamma$ *satisfies the implication* $P \Rightarrow Q$ *in the sense of "total correctness," written* $\tau \models_\lambda (TC, P \Rightarrow Q)$*, iff for all* $(\sigma, \alpha) \in \Lambda_\lambda = \Sigma_\gamma \times \Gamma_\delta$ *we have that* $(\sigma, \alpha) \models_\lambda P$ *implies* $\sigma \in \mathrm{DD}(\tau)$ *and* $(\tau(\sigma), \alpha) \models_\lambda Q$.

(2) *Correspondingly, we say that a state transition map* $\tau : \Sigma_\gamma \nrightarrow \Sigma_\gamma$ *satisfies the implication* $P \Rightarrow Q$ *in the sense of "partial correctness," written* $\tau \models_\lambda (PC, P \Rightarrow Q)$*, iff for all* $(\sigma, \alpha) \in \Lambda_\lambda$ *we have that* $(\sigma, \alpha) \models_\lambda P$ *implies* $(\tau(\sigma), \alpha) \models_\lambda Q$ *or* $\sigma \notin \mathrm{DD}(\tau)$.

An essential observation is that, in both cases, the satisfaction statement for the precondition and the postcondition, respectively, refers to the same local environment $\alpha$. This means that a *correctness assertion* can be seen as an *implication with implicitly universally quantified free logical variables*. On the other side, this gives us a hint how to extend state transition maps, in a reasonable way, to local environments: For any state transition map $\tau : \Sigma_\gamma \nrightarrow \Sigma_\gamma$ and any extended context $\lambda = (\gamma, \delta)$, we obtain an extended state transition map $\overline{\tau}_\delta = \tau \times id_{\Gamma_\delta} : \Sigma_\gamma \times \Gamma_\delta \nrightarrow \Sigma_\gamma \times \Gamma_\delta$ with

$$\mathrm{DD}(\overline{\tau}_\delta) \triangleq \mathrm{DD}(\tau) \times \Gamma_\delta \quad \text{and} \quad \overline{\tau}_\delta(\sigma, \alpha) \triangleq (\tau(\sigma), \alpha) \text{ for all } (\sigma, \alpha) \in \mathrm{DD}(\overline{\tau}_\delta). \tag{10}$$

Following Djikstra's idea of "programs as predicate transformers," we can give now an equivalent formulation of correctness assertions based on the semantics of assertions and the formation of inverse images.[4]

https://doi.org/10.1017/S0960129522000275 Published online by Cambridge University Press

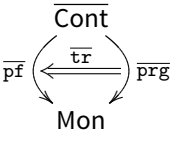
**Theorem 1** (Correctness and Inverse Images)**.** For any state transition map $\tau : \Sigma_\gamma \nrightarrow \Sigma_\gamma$, any extended context $\lambda = (\gamma, \delta)$ and any assertions $P, Q \in \mathtt{assn}(\lambda)$ the following equivalences hold:

(1) $\tau \models_\lambda (TC, P \Rightarrow Q)$ iff $\mathtt{sem}_\lambda(P) \subseteq (\tau \times id_{\Gamma_\delta})^{-1}(\mathtt{sem}_\lambda(Q))$.
(2) $\tau \models_\lambda (PC, P \Rightarrow Q)$ iff $\mathtt{sem}_\lambda(P) \subseteq (\tau \times id_{\Gamma_\delta})^{-1}(\mathtt{sem}_\lambda(Q)) \cup (\Lambda_\lambda \setminus (\mathrm{DD}(\tau) \times \Gamma_\delta))$.

*Proof.* This follows immediately from the definition of the semantics of assertions in (5), Definition 4, the definition of $\overline{\tau}$ in (10), and the definition of inverse images for partial functions in footnote (4). □

In Theorem 1, the state transition map $\tau$ serves as a predicate transformer in the sense that the formation of inverse images transforms the state predicate $\mathtt{sem}_\lambda(Q)$ into the state predicate $\overline{\tau}_\delta^{-1}(\mathtt{sem}_\lambda(Q))$ or $\overline{\tau}_\delta^{-1}(\mathtt{sem}_\lambda(Q)) \cup \Lambda_\lambda \setminus \mathrm{DD}(\overline{\tau})$, respectively. In this subsection, we develop a full categorical account of these two kinds of predicate transformer semantics of state transition maps.

**Extended State Transition Maps:** To be able to relate and combine the logic of local states with the semantics of local programs, it is necessary to lift up the state transition semantics, developed in Subsection 3.2 for plain contexts, to extended contexts:




The functor $\overline{\mathtt{prg}} : \overline{\mathsf{Cont}} \to \mathsf{Mon}$ is simply defined by $\overline{\mathtt{prg}}(\lambda) \triangleq \mathtt{prg}(\gamma) = \{c \in \mathbf{Prg} \mid pvr(c) \subseteq \gamma\}$ for all extended contexts $\lambda = (\gamma, \delta) \in |\overline{\mathsf{Cont}}|$ and by assigning to each morphism $in_{\lambda,\lambda'} = (in_{\gamma,\gamma'}, in_{\delta,\delta'}) : \lambda \to \lambda'$ in $\overline{\mathsf{Cont}}$ the inclusion functor $\overline{\mathtt{prg}}_{\lambda,\lambda'} = \mathtt{prg}_{\gamma,\gamma'} : \overline{\mathtt{prg}}(\lambda) \hookrightarrow \overline{\mathtt{prg}}(\lambda')$.

The definition of extended state transition maps in (10) is the key ingredient to lift up the functor $\mathtt{pf} : \mathsf{Cont} \to \mathsf{Mon}$ to a functor $\overline{\mathtt{pf}} : \overline{\mathsf{Cont}} \to \mathsf{Mon}$: Let be given an extended context $\lambda = (\gamma, \delta)$. It is easy to verify that we have $(\overline{\tau; \tau'})_\delta = \overline{\tau}_\delta; \overline{\tau'}_\delta$ for arbitrary state transition maps $\tau, \tau' : \Sigma_\gamma \nrightarrow \Sigma_\gamma$ thus we can choose $\overline{\mathtt{pf}}(\lambda)$ to be the submonoid of $(\Lambda_\lambda \nrightarrow \Lambda_\lambda) = \mathsf{Par}(\Lambda_\lambda, \Lambda_\lambda)$ given by all extended state transition maps $\overline{\tau}_\delta \triangleq \tau \times id_{\Gamma_\delta} : \Sigma_\gamma \times \Gamma_\delta \rightharpoonup \Sigma_\gamma \times \Gamma_\delta$ with $\tau \in \mathtt{pf}(\gamma) = (\Sigma_\gamma \nrightarrow \Sigma_\gamma)$. Note, that $\mathtt{pf}(\gamma)$ and $\overline{\mathtt{pf}}(\lambda)$ are isomorphic, that is, for each declaration $\delta$ we produce a "copy" of $\mathtt{pf}(\gamma)$! For any morphism $in_{\lambda,\lambda'} = (in_{\gamma,\gamma'}, in_{\delta,\delta'}) : \lambda \to \lambda'$ in $\overline{\mathsf{Cont}}$, we can extend the functor $\mathtt{pf}_{\gamma,\gamma'} : \mathtt{pf}(\gamma) \to \mathtt{pf}(\gamma')$ to a functor $\overline{\mathtt{pf}}_{\lambda,\lambda'} : \overline{\mathtt{pf}}(\lambda) \to \overline{\mathtt{pf}}(\lambda')$ assigning to any extended state transition map $\overline{\tau}_\delta = \tau \times id_{\Gamma_\delta} : \Lambda_\lambda \nrightarrow \Lambda_\lambda$ in $\overline{\mathtt{pf}}(\lambda)$ the extended state transition map $\overline{\mathtt{pf}}_{\lambda,\lambda'}(\overline{\tau}_\delta) \triangleq \overline{\mathtt{pf}_{\gamma,\gamma'}(\tau)}_{\delta'} = \mathtt{pf}_{\gamma,\gamma'}(\tau) \times id_{\Gamma_{\delta'}} : \Lambda_{\lambda'} \nrightarrow \Lambda_{\lambda'}$ in $\overline{\mathtt{pf}}(\lambda')$. Our construction transforms equation (8)

$$\begin{array}{ccc} \Sigma_{\gamma'} \times \Gamma_{\delta'} & \xrightarrow{\mathtt{pf}_{\gamma,\gamma'}(\tau) \times id_{\Gamma_{\delta'}}} & \Sigma_{\gamma'} \times \Gamma_{\delta'} \\ {\scriptstyle p_{\gamma',\gamma} \times p_{\delta',\delta}} \downarrow & \overline{\mathtt{pf}}_{\lambda,\lambda'} \uparrow & \downarrow {\scriptstyle p_{\gamma',\gamma} \times p_{\delta',\delta}} \\ \Sigma_\gamma \times \Gamma_\delta & \xrightarrow[\tau \times id_{\Gamma_\delta}]{} & \Sigma_\gamma \times \Gamma_\delta \end{array}$$

$\mathtt{pf}_{\gamma,\gamma'}(\tau); p_{\gamma',\gamma} = p_{\gamma',\gamma}; \tau$ in $\mathsf{Par}$ into the equation

$$\overline{\mathtt{pf}}_{\lambda,\lambda'}(\overline{\tau}_\delta); p_{\lambda',\lambda} = p_{\lambda',\lambda}; \overline{\tau}_\delta \tag{11}$$

in $\mathsf{Par}$ for the product $p_{\lambda',\lambda} = p_{\gamma',\gamma} \times p_{\delta',\delta} : \Lambda_{\lambda'} \to \Lambda_\lambda$ of reduction maps.

This ensures that we have indeed defined a functor $\overline{\mathtt{pf}}_{\lambda,\lambda'} : \overline{\mathtt{pf}}(\lambda) \to \overline{\mathtt{pf}}(\lambda')$ between the monoids $\overline{\mathtt{pf}}(\lambda)$ and $\overline{\mathtt{pf}}(\lambda')$. Moreover, it can be shown, analogously to Subsection 3.2, that

https://doi.org/10.1017/S0960129522000275 Published online by Cambridge University Press

$\overline{\mathtt{pf}}_{\lambda,\lambda} = id_{\overline{\mathtt{pf}}(\lambda)}$ and that $\overline{\mathtt{pf}}_{\lambda,\lambda'};\overline{\mathtt{pf}}_{\lambda',\lambda''} = \overline{\mathtt{pf}}_{\lambda,\lambda''}$ for any inclusions $\lambda \subseteq \lambda' \subseteq \lambda''$; thus, the assignments $\lambda \mapsto \overline{\mathtt{pf}}(\lambda)$ and $in_{\lambda,\lambda'} \mapsto \overline{\mathtt{pf}}_{\lambda,\lambda'}$ define indeed a functor $\overline{\mathtt{pf}} : \overline{\mathsf{Cont}} \to \mathsf{Mon}$.

Finally, we can extend the functor $\mathtt{tr}_\gamma : \mathtt{prg}(\gamma) \to \mathtt{pf}(\gamma)$ between the monoids $\mathtt{prg}(\gamma)$ and $\mathtt{pf}(\gamma) = (\Sigma_\gamma \nrightarrow \Sigma_\gamma)$ to a functor $\overline{\mathtt{tr}}_\lambda : \overline{\mathtt{prg}}(\lambda) \to \overline{\mathtt{pf}}(\lambda)$ between the monoids $\overline{\mathtt{prg}}(\lambda)$ and $\overline{\mathtt{pf}}(\lambda) \subseteq (\Lambda_\lambda \nrightarrow \Lambda_\lambda)$. We simply set $\overline{\mathtt{tr}}_\lambda(c) \triangleq \overline{\mathtt{tr}_\gamma(c)}_\delta = \mathtt{tr}_\gamma(c) \times id_{\Gamma_\delta}$ for each $c \in \overline{\mathtt{prg}}(\lambda) = \mathtt{prg}(\gamma)$. Functoriality of $\overline{\mathtt{tr}}_\lambda$ is ensured by the functoriality of $\mathtt{tr}_\gamma$ and the equation $\overline{\tau;\tau'}_\delta = \overline{\tau}_\delta;\overline{\tau'}_\delta$ for arbitrary state transition maps $\tau, \tau' : \Sigma_\gamma \nrightarrow \Sigma_\gamma$. Moreover, equation (9) guarantees that the family $\overline{\mathtt{tr}}_\lambda : \overline{\mathtt{prg}}(\lambda) \to \overline{\mathtt{pf}}(\lambda)$, $\lambda \in |\overline{\mathsf{Cont}}|$ of functors provides a natural transformation $\overline{\mathtt{tr}} : \overline{\mathtt{prg}} \Rightarrow \overline{\mathtt{pf}}$.

**Two contravariant Power Set Functors:** To find an adequate formalization of the two kinds of predicate transformations, appearing in Theorem 1, we take a closer look at the inverse image construction for partial functions. We consider $\mathsf{Set}$ as a subcategory of $\mathsf{Par}$!

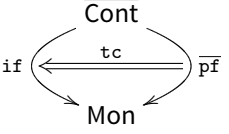
The contravariant power set functor $\mathtt{P} : \mathsf{Set}^{op} \to \mathsf{Pre}$, assigning to each set $A$ the partial order $(\wp(A), \subseteq)$ and to each function $f : A \to B$ the inverse image functor $f^{-1} : (\wp(B), \subseteq) \to (\wp(A), \subseteq)$ with $f^{-1}(B') \triangleq \{a \in A \mid f(a) \in B'\}$ for all subsets $B' \subseteq B$, can be extended in two different ways to a contravariant power set functor from $\mathsf{Par}$ into $\mathsf{Pre}$.

The "standard" functor $\mathtt{P} : \mathsf{Par}^{op} \to \mathsf{Pre}$ is related to total correctness and assigns to each set $A$ the partial order $(\wp(A), \subseteq)$ and to each partial function $f : A \nrightarrow B$ the inverse image functor $f^{-1} : (\wp(B), \subseteq) \to (\wp(A), \subseteq)$ with $f^{-1}(B') \triangleq \{a \in A \mid a \in \mathrm{DD}(f), f(a) \in B'\}$ for all subsets $B' \subseteq B$.

The "non-standard" functor $\mathtt{P}_{\mathrm{DD}} : \mathsf{Par}^{op} \to \mathsf{Pre}$ is, in turn, related to partial correctness and assigns to each set $A$ the partial order $(\wp(A), \subseteq)$ and to each partial function $f : A \nrightarrow B$ the modified inverse image functor $f^{-1}_{\mathrm{DD}} : (\wp(B), \subseteq) \to (\wp(A), \subseteq)$ with $f^{-1}_{\mathrm{DD}}(B') \triangleq f^{-1}(B') \cup (A \setminus \mathrm{DD}(f))$ for all subsets $B' \subseteq B$.

**From State Transition Maps to Predicate Transformers:** Both functors $\mathtt{P} : \mathsf{Par}^{op} \to \mathsf{Pre}$ and $\mathtt{P}_{\mathrm{DD}} : \mathsf{Par}^{op} \to \mathsf{Pre}$ are embeddings, that is, injective on objects and on morphisms.

For each extended context $\lambda$ in $\overline{\mathsf{Cont}}$, the image $\mathtt{if}(\lambda) \triangleq \mathtt{P}^{op}(\overline{\mathtt{pf}}(\lambda))$ of the submonoid $\overline{\mathtt{pf}}(\lambda)$ of $(\Lambda_\lambda \nrightarrow \Lambda_\lambda) = \mathsf{Par}(\Lambda_\lambda, \Lambda_\lambda)$ w.r.t. $\mathtt{P}^{op} : \mathsf{Par} \to \mathsf{Pre}^{op}$ becomes therefore a submonoid of $\mathsf{Pre}((\wp(\Lambda_\lambda), \subseteq), (\wp(\Lambda_\lambda), \subseteq))^{op}$ and we get an isomorphism $\mathtt{tc}_\lambda : \overline{\mathtt{pf}}(\lambda) \to \mathtt{if}(\lambda)$ in $\mathsf{Mon}$ with $\mathtt{tc}_\lambda(\overline{\tau}_\delta) \triangleq \overline{\tau}_\delta^{-1} : (\wp(\Lambda_\lambda), \subseteq) \to (\wp(\Lambda_\lambda), \subseteq)$ for each morphism $\overline{\tau}_\delta : \Lambda_\lambda \nrightarrow \Lambda_\lambda$ in $\overline{\mathtt{pf}}(\lambda)$. "$\mathtt{if}$" and "$\mathtt{tc}$" stand for "inverse image function" and "total correctness," respectively. For any morphism




$in_{\lambda,\lambda'} : \lambda \to \lambda'$ in $\overline{\mathsf{Cont}}$, we can define a functor $\mathtt{if}_{\lambda,\lambda'} \triangleq \mathtt{tc}_\lambda^{-1};\overline{\mathtt{pf}}_{\lambda,\lambda'};\mathtt{tc}_{\lambda'} : \mathtt{if}(\lambda) \to \mathtt{if}(\lambda')$. $\overline{\mathtt{pf}} : \overline{\mathsf{Cont}} \to \mathsf{Mon}$ is a functor; thus, this definition ensures that the assignments $\lambda \mapsto \mathtt{if}(\lambda)$ and $in_{\lambda,\lambda'} \mapsto \mathtt{if}_{\lambda,\lambda'}$ constitute a functor $\mathtt{if} : \overline{\mathsf{Cont}} \to \mathsf{Mon}$ and that, in addition, the isomorphisms $\mathtt{tc}_\lambda : \overline{\mathtt{pf}}(\lambda) \to \mathtt{if}(\lambda)$ in $\mathsf{Mon}$ establish a natural isomorphism $\mathtt{tc} : \overline{\mathtt{pf}} \Rightarrow \mathtt{if}$.

For each extended context $\lambda = (\gamma, \delta)$, the monoid $\mathtt{if}(\lambda)$ is constituted by all inverse image functors of the form $\overline{\tau}_\delta^{-1} = (\tau \times id_{\Gamma_\delta})^{-1} : (\wp(\Lambda_\lambda), \subseteq) \to (\wp(\Lambda_\lambda), \subseteq)$, $\Lambda_\lambda = \Sigma_\gamma \times \Gamma_\delta$ (interpreted as morphisms in the opposite direction) with $\tau : \Sigma_\gamma \nrightarrow \Sigma_\gamma$ a partial function, that is, with $\tau$ ranging over all morphisms in $\mathtt{pf}(\gamma) = \mathsf{Par}(\Sigma_\gamma, \Sigma_\gamma)$ and thus $\overline{\tau}_\delta : \Lambda_\lambda \nrightarrow \Lambda_\lambda$ ranging over all morphisms in $\overline{\mathtt{pf}}(\lambda) \subseteq \mathsf{Par}(\Lambda_\lambda, \Lambda_\lambda)$. The functor $\mathtt{if}_{\lambda,\lambda'} : \mathtt{if}(\lambda) \to \mathtt{if}(\lambda')$ assigns to $\mathtt{tc}_\lambda(\overline{\tau}_\delta) = \overline{\tau}_\delta^{-1}$ the inverse image functor $\mathtt{if}_{\lambda,\lambda'}(\overline{\tau}_\delta^{-1}) = \mathtt{tc}_{\lambda'}(\overline{\mathtt{pf}}_{\lambda,\lambda'}(\overline{\tau}_\delta)) = (\overline{\mathtt{pf}}_{\lambda,\lambda'}(\overline{\tau}_\delta))^{-1} : (\wp(\Lambda_{\lambda'}), \subseteq) \to (\wp(\Lambda_{\lambda'}), \subseteq)$

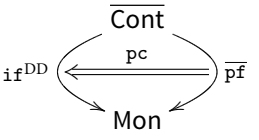
https://doi.org/10.1017/S0960129522000275 Published online by Cambridge University Press

$$\begin{array}{ccc} (\wp(\Lambda_{\lambda'}), \subseteq) & \xleftarrow{\mathtt{tc}_{\lambda'}(\overline{\mathtt{pf}}_{\lambda,\lambda'}(\overline{\tau}_\delta))} & (\wp(\Lambda_{\lambda'}), \subseteq) \\ \uparrow{\scriptstyle p^{-1}_{\lambda',\lambda}} & \uparrow{\scriptstyle \mathtt{if}_{\lambda,\lambda'}} & \uparrow{\scriptstyle p^{-1}_{\lambda',\lambda}} \\ (\wp(\Lambda_\lambda), \subseteq) & \xleftarrow[\mathtt{tc}_\lambda(\overline{\tau}_\delta)]{} & (\wp(\Lambda_\lambda), \subseteq) \end{array}$$

while the equation (11) in Par is transformed into an equation in Pre:

$$p^{-1}_{\lambda',\lambda}; \mathtt{if}_{\lambda,\lambda'}(\overline{\tau}_\delta) = \overline{\tau}^{-1}_\delta; p^{-1}_{\lambda',\lambda} \tag{12}$$

Completely analogously, we can use the embedding $\mathrm{P}_{\mathrm{DD}}{}^{op} : \mathsf{Par} \to \mathsf{Pre}^{op}$ to construct for each extended context $\lambda$ in $\overline{\mathsf{Cont}}$ the image $\mathtt{if}^{\mathrm{DD}}(\lambda) \triangleq \mathrm{P}_{\mathrm{DD}}{}^{op}(\overline{\mathtt{pf}}(\lambda))$ of the submonoid $\overline{\mathtt{pf}}(\lambda)$ of $\mathsf{Par}(\Lambda_\lambda, \Lambda_\lambda)$ and obtain a submonoid of $\mathsf{Pre}((\wp(\Lambda_\lambda), \subseteq), (\wp(\Lambda_\lambda), \subseteq))^{op}$. We get an isomorphism $\mathtt{pc}_\lambda : \overline{\mathtt{pf}}(\lambda) \to \mathtt{if}^{\mathrm{DD}}(\lambda)$ in Mon with $\mathtt{pc}_\lambda(\overline{\tau}_\delta) \triangleq (\overline{\tau}_\delta)^{-1}_{\mathrm{DD}} : (\wp(\Lambda_\lambda), \subseteq) \to (\wp(\Lambda_\lambda), \subseteq)$ for each morphism $\overline{\tau}_\delta : \Lambda_\lambda \nrightarrow \Lambda_\lambda$ in $\overline{\mathtt{pf}}(\lambda)$. "pc" stands for "partial correctness." For any morphism $in_{\lambda,\lambda'} : \lambda \to \lambda'$ in $\overline{\mathsf{Cont}}$, we can define a functor $\mathtt{if}^{\mathrm{DD}}{}_{\lambda,\lambda'} \triangleq \mathtt{pc}^{-1}_\lambda; \overline{\mathtt{pf}}_{\lambda,\lambda'}; \mathtt{pc}_{\lambda'} : \mathtt{if}(\lambda) \to \mathtt{if}(\lambda')$.




$\overline{\mathtt{pf}} : \overline{\mathsf{Cont}} \to \mathsf{Mon}$ is a functor thus the assignments $\lambda \mapsto \mathtt{if}(\lambda)$ and $in_{\lambda,\lambda'} \mapsto \mathtt{if}_{\lambda,\lambda'}$ constitute a functor $\mathtt{if}^{\mathrm{DD}} : \overline{\mathsf{Cont}} \to \mathsf{Mon}$ and, in addition, the isomorphisms $\mathtt{pc}_\lambda : \overline{\mathtt{pf}}(\lambda) \to \mathtt{if}^{\mathrm{DD}}(\lambda)$ in Mon establish a natural isomorphism $\mathtt{pc} : \overline{\mathtt{pf}} \Rightarrow \mathtt{if}^{\mathrm{DD}}$.

We fix the above discussion in the following theorem.

**Theorem 2** (Natural isomorphisms)**.** For each extended context $\lambda = (\gamma, \delta)$, the assignments $\mathtt{tc}_\lambda(\overline{\tau}_\delta) \triangleq \overline{\tau}^{-1}_\delta : (\wp(\Lambda_\lambda), \subseteq) \to (\wp(\Lambda_\lambda), \subseteq)$, $\mathtt{pc}_\lambda(\overline{\tau}_\delta) \triangleq (\overline{\tau}_\delta)^{-1}_{\mathrm{DD}} : (\wp(\Lambda_\lambda), \subseteq) \to (\wp(\Lambda_\lambda), \subseteq)$, define, respectively, the natural isomorphisms for total and partial correctness $tc : \overline{\mathtt{pf}} \Rightarrow \mathtt{if} : \overline{\mathsf{Cont}} \to \mathsf{Mon}$ and $pc : \overline{\mathtt{pf}} \Rightarrow \mathtt{if}^{\mathrm{DD}} : \overline{\mathsf{Cont}} \to \mathsf{Mon}$.

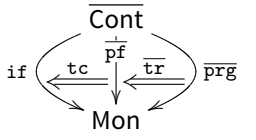
**Semantic Equivalences:** Based on two different extensions $\mathrm{P} : \mathsf{Par}^{op} \to \mathsf{Pre}$ and $\mathrm{P}_{\mathrm{DD}} : \mathsf{Par}^{op} \to \mathsf{Pre}$ of the contravariant power set functor $\mathrm{P} : \mathsf{Set}^{op} \to \mathsf{Pre}$ to the category Par, we presented two distinct predicate transformer semantics for partial functions – the total correctness semantics $\mathtt{tc} : \overline{\mathtt{pf}} \Rightarrow \mathtt{if}$, converting partial functions into inverse image functors, and the partial correctness semantics $\mathtt{pc} : \overline{\mathtt{pf}} \Rightarrow \mathtt{if}^{\mathrm{DD}}$, converting partial functions into modified inverse image functors.

Since both natural transformations tc and pc are natural isomorphisms, we have, especially, shown in such a way that the total correctness semantics and the partial correctness semantics are equivalent from a structural point of view.

Therefore, it will be sufficient to concentrate our further investigations of the structural features of Hoare logics on one of these semantics. We will focus on total correctness since partial correctness has been discussed in Wolter et al. (2020).

### *3.4 Weakest precondition semantics of local programs*

We are well prepared now to come back to the set-theoretic characterizations of correctness




https://doi.org/10.1017/S0960129522000275 Published online by Cambridge University Press

assertions in Theorem 1. We can define a predicate transformer semantics $\mathtt{wp} \triangleq \overline{\mathtt{tr}}; \mathtt{tc} : \overline{\mathtt{prg}} \Rightarrow \mathtt{if}$ of programs by composing the state transition semantics $\overline{\mathtt{tr}} : \overline{\mathtt{prg}} \Rightarrow \overline{\mathtt{pf}}$ of programs with the total correctness semantics $\mathtt{tc} : \overline{\mathtt{pf}} \Rightarrow \mathtt{if}$ of partial functions. "wp" stands for "weakest preconditions" a term we discuss later in this subsection.

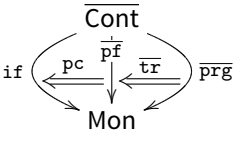
Diagram (13) visualizes the definition of the "weakest precondition semantics" $\mathtt{wp} = \overline{\mathtt{tr}}; \mathtt{tc}$ of programs and summarizes our efforts to develop indexed semantics of local programs:

$$\begin{array}{ccccccccc}
\lambda' & (\lambda' \xrightarrow{c} \lambda') & \overset{\overline{\mathtt{tr}}_{\lambda'}}{\longmapsto} & (\Lambda_{\lambda'} \overset{\overline{\mathtt{tr}}_{\lambda'}(c)}{\multimap} \Lambda_{\lambda'}) & \overset{\mathtt{tc}_{\lambda'}}{\longmapsto} & ((\wp(\Lambda_{\lambda'}), \subseteq) & \overset{\mathtt{wp}_{\lambda'}(c)}{\longleftarrow} & (\wp(\Lambda_{\lambda'}), \subseteq)) \\
\uparrow{\scriptstyle in_{\lambda,\lambda'}} & \uparrow{\scriptstyle \overline{\mathtt{prg}}_{\lambda,\lambda'}} & & \downarrow{\scriptstyle p_{\lambda',\lambda}} \quad \uparrow{\scriptstyle \overline{\mathtt{pf}}_{\lambda,\lambda'}} \quad \downarrow{\scriptstyle p_{\lambda',\lambda}} & & \uparrow{\scriptstyle p^{-1}_{\lambda',\lambda}} & & \uparrow{\scriptstyle \mathtt{if}_{\lambda,\lambda'}} \quad \uparrow{\scriptstyle p^{-1}_{\lambda',\lambda}} & \\
\lambda & (\lambda \xrightarrow{c} \lambda) & \overset{\overline{\mathtt{tr}}_{\lambda}}{\longmapsto} & (\Lambda_{\lambda} \overset{\overline{\mathtt{tr}}_{\lambda}(c)}{\multimap} \Lambda_{\lambda}) & \overset{\mathtt{tc}_{\lambda}}{\longmapsto} & ((\wp(\Lambda_{\lambda}), \subseteq) & \overset{\mathtt{wp}_{\lambda}(c)}{\longleftarrow} & (\wp(\Lambda_{\lambda}), \subseteq))
\end{array} \tag{13}$$

We defined the state transition semantics $\mathtt{tr}_\gamma(c) : \Sigma_\gamma \nrightarrow \Sigma_\gamma$, $\Sigma_\gamma = (\gamma \to \mathbb{D})$ of a local program $c \in \mathtt{prg}(\gamma) = \{c \in \mathbf{Prg} \mid pvr(c) \subseteq \gamma\}$ as a restriction of the corresponding state transition map $[\![c]\!] : \Sigma \nrightarrow \Sigma$ for global states. For any inclusion function $in_{\gamma,\gamma'} : \gamma \to \gamma'$ we have $\mathtt{prg}(\gamma) \subseteq \mathtt{prg}(\gamma')$ and $\mathtt{pf}_{\gamma,\gamma'}(\mathtt{tr}_\gamma(c)) : \Sigma_{\gamma'} \nrightarrow \Sigma_{\gamma'}$ simply extends $\mathtt{tr}_\gamma(c)$ by the identity on $\Sigma_{\gamma' \setminus \gamma}$. We get $\mathtt{pf}_{\gamma,\gamma'}(\mathtt{tr}_\gamma(c)) = \mathtt{tr}_{\gamma'}(c)$ since $\mathtt{tr}_\gamma(c)$ and $\mathtt{tr}_{\gamma'}(c)$ are both restriction of the same partial map $[\![c]\!] : \Sigma \nrightarrow \Sigma$ and since $[\![c]\!](\varrho)(x) = \varrho(x)$ for any global state $\varrho \in \mathrm{DD}([\![c]\!])$ and any program variable $x \notin pvr(c)$. For the same reason, we obtained also the equation $\mathtt{tr}_{\gamma'}(c); p_{\gamma',\gamma} = p_{\gamma',\gamma}; \mathtt{tr}_\gamma(c)$ in the category Par for the reduction map $p_{\gamma',\gamma} : \Sigma_{\gamma'} \to \Sigma_\gamma$ induced by precomposition with $in_{\gamma,\gamma'} : \gamma \to \gamma'$.

To be able to reason about the semantics of local programs, we had to lift up the state transition semantics to extended contexts $\lambda = (\gamma, \delta)$, corresponding sets $\Lambda_\lambda = \Sigma_\gamma \times \Gamma_\delta$ of "extended local states" and thus to pairs $in_{\lambda,\lambda'} = (in_{\gamma,\gamma'}, in_{\delta,\delta'}) : \lambda \to \lambda'$ of inclusion functions and products $p_{\lambda',\lambda} = p_{\gamma',\gamma} \times p_{\delta',\delta} : \Lambda_{\lambda'} \to \Lambda_\lambda$ of reduction maps. Guided by Theorem 1, this extension was done by simply adjoining identity maps. For each local program $c \in \overline{\mathtt{prg}}(\lambda) \triangleq \mathtt{prg}(\gamma)$ we set $\overline{\mathtt{tr}}_\lambda(c) \triangleq \mathtt{tr}_\gamma(c) \times id_{\Gamma_\delta} : \Lambda_\lambda \nrightarrow \Lambda_\lambda$ and $\overline{\mathtt{pf}}_{\lambda,\lambda'}(\overline{\mathtt{tr}}_\lambda(c)) \triangleq \mathtt{pf}_{\gamma,\gamma'}(\mathtt{tr}_\gamma(c)) \times id_{\Gamma_{\delta'}} : \Lambda_{\lambda'} \nrightarrow \Lambda_{\lambda'}$ thus $\overline{\mathtt{pf}}_{\lambda,\lambda'}(\overline{\mathtt{tr}}_\lambda(c)) = \mathtt{tr}_{\gamma'}(c) \times id_{\Gamma_{\delta'}} = \overline{\mathtt{tr}}_{\lambda'}(c)$ and, moreover, $\overline{\mathtt{tr}}_{\lambda'}(c); p_{\lambda',\lambda} = p_{\lambda',\lambda}; \overline{\mathtt{tr}}_\lambda(c)$ in Par.

Finally, we transformed the extended state transition semantics into the predicate transformer semantics wp by means of the "standard" contravariant power set functor $\mathtt{P} : \mathsf{Par}^{op} \to \mathsf{Pre}$. We set $\mathtt{wp}_\lambda(c) \triangleq \mathtt{tc}_\lambda(\overline{\mathtt{tr}}_\lambda(c)) = \overline{\mathtt{tr}}_\lambda(c)^{-1}$, and get $\mathtt{if}_{\lambda,\lambda'}(\mathtt{wp}_\lambda(c)) = \mathtt{if}_{\lambda,\lambda'}(\mathtt{tc}_\lambda(\overline{\mathtt{tr}}_\lambda(c))) = \mathtt{tc}_{\lambda'}(\overline{\mathtt{pf}}_{\lambda,\lambda'}(\overline{\mathtt{tr}}_\lambda(c))) = \mathtt{tc}_{\lambda'}(\overline{\mathtt{tr}}_{\lambda'}(c))$ and the equation $p^{-1}_{\lambda',\lambda}; \mathtt{wp}_{\lambda'}(c) = \mathtt{wp}_\lambda(c); p^{-1}_{\lambda',\lambda}$ in Pre.




Besides the predicate transformer semantics $\mathtt{wp} = \overline{\mathtt{tr}}; \mathtt{tc} : \overline{\mathtt{prg}} \Rightarrow \mathtt{if}$, we can also define a "weakest liberal precondition semantics" $\mathtt{wlp} \triangleq \overline{\mathtt{tr}}; \mathtt{pc} : \overline{\mathtt{prg}} \Rightarrow \mathtt{if}$ of programs by composing the state transition semantics $\overline{\mathtt{tr}} : \overline{\mathtt{prg}} \Rightarrow \overline{\mathtt{pf}}$ of programs with the partial correctness semantics $\mathtt{pc} : \overline{\mathtt{pf}} \Rightarrow \mathtt{if}$ of partial functions instead.

As discussed at the end of Subsection 3.3, tc and pc are natural isomorphisms; thus, the weakest precondition semantics and the weakest liberal precondition semantics of local programs are structural equivalent and we will focus on the weakest precondition semantics.

**Weakest Preconditions:** The notion of "weakest preconditions" has been introduced in Dijkstra (1975) and reflects the equivalences in Theorem 1. The weakest precondition of a local program $c \in \overline{\mathtt{prg}}(\lambda)$ with respect to a state predicate $\mathscr{Q} \subseteq \Lambda_\lambda$ is the state predicate $\mathtt{wp}_\lambda(c)(\mathscr{Q}) = (\mathtt{tr}_\gamma(c) \times id_{\Gamma_\delta})^{-1}(\mathscr{Q}) \subseteq \Lambda_\lambda$. Correspondingly, the weakest liberal precondition is the state predicate $\mathtt{wlp}_\lambda(c)(\mathscr{Q}) = (\mathtt{tr}_\gamma(c) \times id_{\Gamma_\delta})^{-1}(\mathscr{Q}) \cup (\mathrm{DD}(\mathtt{tr}_\gamma(c)) \times \Gamma_\delta) \subseteq \Lambda_\lambda$.

https://doi.org/10.1017/S0960129522000275 Published online by Cambridge University Press

For a program $c$ and an assertion $Q$, we consider the weakest precondition $\mathtt{wp}_\lambda(c)(\mathtt{sem}_\lambda(Q))$ and the weakest liberal precondition $\mathtt{wlp}_\lambda(c)(\mathtt{sem}_\lambda(Q))$ where $\lambda = (pvr(c) \cup pvr(Q), flv(Q))$.

An important and non-trivial result concerning Hoare logic is that the *Hoare Proof Calculus*, presented at the end of Section 2, is complete (compare Cook 1978 and Apt et al. 2009). This completeness result is based on another non-trivial result stating that our language of expressions is expressive enough, in the sense, that we can represent weakest preconditions syntactically (see Theorem 3.4 in Apt et al. 2009): There exist assertions $wp(c, Q), wlp(c, Q) \in \mathtt{assn}(\lambda)$ such that $\mathtt{sem}_\lambda(wp(c, Q)) = \mathtt{wp}_\lambda(c)(\mathtt{sem}_\lambda(Q))$ and $\mathtt{sem}_\lambda(wlp(c, Q)) = \mathtt{wlp}_\lambda(c)(\mathtt{sem}_\lambda(Q))$, respectively.

$\mathtt{sem}$ is a natural transformation; thus, the equations $p^{-1}_{\lambda',\lambda}; \mathtt{wp}_{\lambda'}(c) = \mathtt{wp}_\lambda(c); p^{-1}_{\lambda',\lambda}$ and $p^{-1}_{\lambda',\lambda}; \mathtt{wlp}_{\lambda'}(c) = \mathtt{wlp}_\lambda(c); p^{-1}_{\lambda',\lambda}$ ensure that syntactic weakest preconditions are context independent: It holds that $\mathtt{sem}_{\lambda'}(wp(c, Q)) = \mathtt{wp}_{\lambda'}(c)(\mathtt{sem}_{\lambda'}(Q))$ and $\mathtt{sem}_{\lambda'}(wlp(c, Q)) = \mathtt{wlp}_{\lambda'}(c)(\mathtt{sem}_{\lambda'}(Q))$, respectively, for any morphism $in_{\lambda,\lambda'} : \lambda \to \lambda'$ in $\overline{\mathsf{Cont}}$.

The context independence of syntactic weakest preconditions ensures also that they are monoton w.r.t. semantic entailment, that is, $P \Vdash_\lambda Q$ implies $wp(c, P) \Vdash_{\lambda'} wp(c, Q)$ for all inclusion functions $in_{\lambda,\lambda'} : \lambda \to \lambda'$ and all programs $c : \lambda' \to \lambda'$: $P \Vdash_\lambda Q$ is defined in (7) by the inclusion $\mathtt{sem}_\lambda(P) \subseteq \mathtt{sem}_\lambda(Q)$. Proposition 2 ensures that this inclusion entails the inclusion $\mathtt{sem}_{\lambda'}(P) \subseteq \mathtt{sem}_{\lambda'}(Q)$; thus, we obtain also the inclusion $\mathtt{wp}_{\lambda'}(c)(\mathtt{sem}_{\lambda'}(P)) \subseteq \mathtt{wp}_{\lambda'}(c)(\mathtt{sem}_{\lambda'}(Q))$ and thus $\mathtt{sem}_{\lambda'}(wlp(c, P)) \subseteq \mathtt{sem}_{\lambda'}(wlp(c, Q))$ due to context independence. The last inclusion, however, means nothing but $wp(c, P) \Vdash_{\lambda'} wp(c, Q)$ according to (7).

To denote the correctness of local programs, we go back to the traditional Hoare triples: A local total correctness assertion $\lambda : [P]\, c\, [Q]$ is valid, written $\models_\lambda [P]\, c\, [Q]$, if, and only if, $\mathtt{tr}_\gamma(c) \models_\lambda (TC, P \Rightarrow Q)$ and, analogously, a local partial correctness assertion $\lambda : \{P\}\, c\, \{Q\}$ is valid, written $\models_\lambda \{P\}\, c\, \{Q\}$, if, and only if, $\mathtt{tr}_\gamma(c) \models_\lambda (PC, P \Rightarrow Q)$.

Instantiating Theorem 1 by the state transition semantics $\mathtt{tr}_\gamma(c)$ of programs, we can summarize that correctness assertions can be equivalently expressed by means of semantic weakest preconditions while the existence of corresponding syntactic weakest preconditions gives us, finally, an equivalent formulation of correctness by means of semantic entailment at hand.

**Corollary 1** (Correctness Assertions)**.** *For any extended context* $\lambda = (\gamma, \delta)$*, any program* $c \in \overline{\mathtt{prg}}(\lambda)$*, and any assertions* $P, Q \in \mathtt{assn}(\lambda)$ *the following equivalences hold:*

(1) $\models_\lambda [P]\, c\, [Q]$ *iff* $\mathtt{sem}_\lambda(P) \subseteq \mathtt{wp}_\lambda(c)(\mathtt{sem}_\lambda(Q))$ *iff* $P \Vdash_\lambda wp(c, Q)$.
(2) $\models_\lambda \{P\}\, c\, \{Q\}$ *iff* $\mathtt{sem}_\lambda(P) \subseteq \mathtt{wlp}_\lambda(c)(\mathtt{sem}_\lambda(Q))$ *iff* $P \Vdash_\lambda wlp(c, Q)$.

Syntactic weakest preconditions are assertions and thus only uniquely determined up to logical equivalence, that is, up to isomorphisms in $\mathtt{ent}(\lambda) = (\mathtt{assn}(\lambda), \Vdash_\lambda)$. An indexed account of the structural features of syntactic weakest preconditions and of a deduction calculus for Hoare triples would have to relay therefore on pseudo functors. We consider this as not quite adequate and prefer to develop directly a fibered account in the next section.

**Remark 4** (Notation)**.** We use the same notation "wp," with typographic variations, to denote different concepts related to "weak preconditions." Thereby, we apply the general notational conventions used in the paper: *wp* (mathit-font) denotes a function, $\mathtt{wp}$ (mathtt-font) denotes a natural transformation, $\mathtt{Wp}$ (mathtt-font) will denote a functor and $\mathsf{Wp}$ (mathsf-font) a category.

## 4. Hoare Logic and Fibrations

The presentation of the structural features of the traditional infinitary version of Hoare logic, as outlined in Section 2, is essentially an indexed one. In the last section, we have elucidated this observation by developing a general and structured presentation of the semantic features of a finitary version of Hoare logic based on indexed categories.

https://doi.org/10.1017/S0960129522000275 Published online by Cambridge University Press

Guided by the three reasons, discussed in the introductory section, we will move now from the indexed setting to the fibered one and present a fully fledged fibered account of Hoare logic.

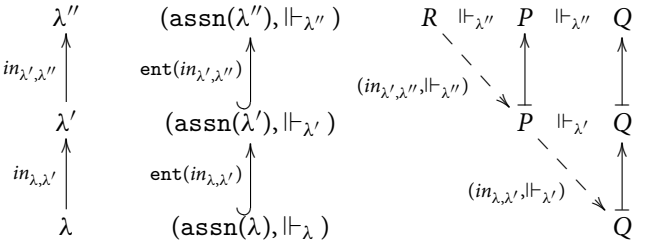
### *4.1 Fibrations for the logic of local states*

The Grothendieck construction (see Barr and Wells 1990) is the main technique to transform an indexed category into a fibered category (fibration). There are different variants of the Grothendieck construction, and we do not include a general definition of the different variants needed here. We describe, however, in detail all the fibered structures obtained by transforming the indexed structures in Section 3.

The indexed version of the logic of local states is manifested by the natural transformation $\mathtt{sem} : \mathtt{ent} \Rightarrow \mathtt{pred} : \overline{\mathsf{Cont}} \to \mathsf{Pre}$. Transforming, first, the functor (indexed category) $\mathtt{ent} : \overline{\mathsf{Cont}} \to \mathsf{Pre}$, we get a fibered category of state assertions and semantic entailments:

**Definition 5** (Category $\mathsf{Ent}$). *The category* $\mathsf{Ent}$ *of "local state assertions" and "semantic entailment" is defined as follows:*

- *objects: all pairs* $(\lambda.Q)$ *of an extended context* $\lambda \in |\overline{\mathsf{Cont}}|$ *and an assertion* $Q \in \mathtt{assn}(\lambda)$.
- *morphisms: from* $(\lambda'.P)$ *to* $(\lambda.Q)$ *are all pairs* $(in_{\lambda,\lambda'}, \Vdash_{\lambda'})$ *with* $in_{\lambda,\lambda'} : \lambda \to \lambda'$ *a morphism in* $\overline{\mathsf{Cont}}$ *and* $P \Vdash_{\lambda'} Q = \mathtt{ent}(in_{\lambda,\lambda'})(Q)$ *a morphism in* $\mathtt{ent}(\lambda') = (\mathtt{assn}(\lambda'), \Vdash_{\lambda'})$.
- *identities: the identity on* $(\lambda.Q)$ *is* $(id_\lambda, =)$ *where* $id_\lambda = in_{\lambda,\lambda}$.
- *composition: the composition of two morphisms* $(in_{\lambda',\lambda''}, \Vdash_{\lambda''}) : (\lambda''.R) \to (\lambda'.P)$ *and* $(in_{\lambda,\lambda'}, \Vdash_{\lambda'}) : (\lambda'.P) \to (\lambda.Q)$ *is the morphism* $(in_{\lambda,\lambda''}, \Vdash_{\lambda''}) : (\lambda''.R) \to (\lambda.Q)$ *where* $in_{\lambda,\lambda''} = in_{\lambda,\lambda'}; in_{\lambda',\lambda''}$. *Composition is well-defined due to the monotonicity of context extensions w.r.t. semantic entailment and the associativity of semantic entailment, that is, since* $\mathtt{ent}$ *is a functor and since the* $\mathtt{ent}(\lambda) = (\mathtt{assn}(\lambda), \Vdash_\lambda)$ *are preorder categories.*




*We obtain a projection functor* $\Pi_{\mathsf{Ent}} : \mathsf{Ent} \to \overline{\mathsf{Cont}}^{op}$ *with* $\Pi_{\mathsf{Ent}}(\lambda.Q) \triangleq \lambda$ *and* $\Pi_{\mathsf{Ent}}((in_{\lambda,\lambda'}, \Vdash_{\lambda'}) : (\lambda'.P) \to (\lambda.Q)) \triangleq (in_{\lambda,\lambda'} : \lambda \to \lambda')$ *with fibers* $\Pi^{-1}_{\mathsf{Ent}}(id_\lambda) \simeq \mathtt{ent}(\lambda)$.

The diagram in Definition 5 (and the diagrams in the following Definitions 6, 7, 8 and 9) visualizes the corresponding Grothendieck construction of morphisms (dashed arrows) and should help the reader to validate the well-definedness of the composition of those morphisms.

The general properties of Grothendieck constructions provide:

**Theorem 3.** The functor $\Pi_{\mathsf{Ent}} : \mathsf{Ent} \to \overline{\mathsf{Cont}}^{op}$ is a split fibration where $(in_{\lambda,\lambda'}, \Vdash_{\lambda'}) : (\lambda'.P) \to (\lambda.Q)$ is a Cartesian arrow if, and only if, $P$ and $Q = \mathtt{ent}(in_{\lambda,\lambda'})(Q)$ are equivalent w.r.t. semantic entailment, that is, isomorphic in $\mathtt{ent}(\lambda') = (\mathtt{assn}(\lambda'), \Vdash_{\lambda'})$.

Given $in^{op}_{\lambda,\lambda'} : \lambda' \to \lambda$ in $\overline{\mathsf{Cont}}^{op}$ and $Q \in \mathtt{assn}(\lambda)$ the standard choice for a corresponding Cartesian arrow is $(in_{\lambda,\lambda'}, \Vdash_{\lambda'}) : (\lambda', Q) \to (\lambda, Q)$.

**Remark 5.** The presentation of assertions about states as a fibration makes evident that the deductive apparatus on those assertions is essentially based on substitution (changing of

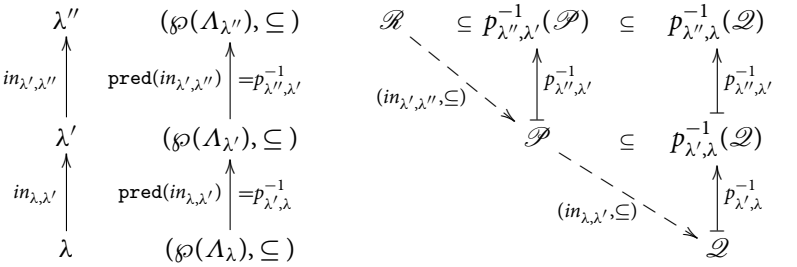
https://doi.org/10.1017/S0960129522000275 Published online by Cambridge University Press

context) and propositional reasoning in the fibers $\mathtt{ent}(\lambda) = (\mathtt{assn}(\lambda), \Vdash_\lambda) \simeq \Pi^{-1}_{\mathsf{Ent}}(id_\lambda)$ (compare Remark 3). That we have a fibration ensures that every first-order variable is universally quantified and that the reasoning on them is sound. On the other hand, existentially quantified assertions have their sound semantics provided by the Cartesian structure of the fibration. □

Transforming, second, the functor (indexed category) $\mathtt{pred} : \overline{\mathsf{Cont}} \to \mathsf{Pre}$, we get a fibered category of state predicates and inclusions of state predicates:

**Definition 6** (Category Pred)**.** *The category* Pred *of "local state predicates" and inclusions is defined as follows:*

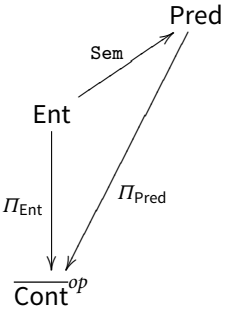
- *objects: all pairs* $(\lambda.\mathscr{Q})$ *of an extended context* $\lambda \in |\overline{\mathsf{Cont}}|$ *and a state predicate* $\mathscr{Q} \in \wp(\Lambda_\lambda)$.
- *morphisms: from* $(\lambda'.\mathscr{P})$ *to* $(\lambda.\mathscr{Q})$ *are all pairs* $(in_{\lambda,\lambda'}, \subseteq)$ *with* $in_{\lambda,\lambda'} : \lambda \to \lambda'$ *a morphism in* $\overline{\mathsf{Cont}}$ *and* $\mathscr{P} \subseteq p^{-1}_{\lambda',\lambda}(\mathscr{Q})$ *a morphism in* $\mathtt{pred}(\lambda') = (\wp(\Lambda_{\lambda'}), \subseteq)$.
- *identities: the identity on* $(\lambda.\mathscr{P})$ *is* $(id_\lambda, =)$.
- *composition: the composition of two morphisms* $(in_{\lambda',\lambda''}, \subseteq) : (\lambda''.\mathscr{R}) \to (\lambda'.\mathscr{P})$ *and* $(in_{\lambda,\lambda'}, \subseteq) : (\lambda'.\mathscr{P}) \to (\lambda.\mathscr{Q})$ *is the morphism* $(in_{\lambda,\lambda''}, \subseteq) : (\lambda''.\mathscr{R}) \to (\lambda.\mathscr{Q})$ *where* $in_{\lambda,\lambda''} = in_{\lambda,\lambda'}; in_{\lambda',\lambda''}$. *Composition is well-defined since* pred *is a functor, that is, we have* $p^{-1}_{\lambda',\lambda}; p^{-1}_{\lambda'',\lambda'} = p^{-1}_{\lambda'',\lambda}$, *and since the* $\mathtt{pred}(\lambda) = (\wp(\Lambda_\lambda), \subseteq)$ *are partial order categories.*




*We obtain a projection functor* $\Pi_{\mathsf{Pred}} : \mathsf{Pred} \to \overline{\mathsf{Cont}}^{op}$ *with* $\Pi_{\mathsf{Pred}}(\lambda.\mathscr{Q}) \triangleq \lambda$ *and* $\Pi_{\mathsf{Pred}}((in_{\lambda,\lambda'}, \subseteq) : (\lambda'.\mathscr{P}) \to (\lambda.\mathscr{Q})) \triangleq (in_{\lambda,\lambda'} : \lambda \to \lambda')$ *with fibers* $\Pi^{-1}_{\mathsf{Pred}}(id_\lambda) \simeq (\wp(\Lambda_\lambda), \subseteq)$.

The general properties of Grothendieck constructions provide:

**Theorem 4.** The functor $\Pi_{\mathsf{Pred}} : \mathsf{Pred} \to \overline{\mathsf{Cont}}^{op}$ is a split fibration where the Cartesian arrows are exactly the morphisms $(in_{\lambda,\lambda'}, =) : (\lambda'.p^{-1}_{\lambda',\lambda}(\mathscr{Q})) \to (\lambda.\mathscr{Q})$ for all $in^{op}_{\lambda,\lambda'} : \lambda' \to \lambda$ in $\overline{\mathsf{Cont}}^{op}$ and all state predicates $\mathscr{Q} \in \wp(\Lambda_\gamma)$. These are the only Cartesian arrows since inclusion $\subseteq$ is anti-symmetric.

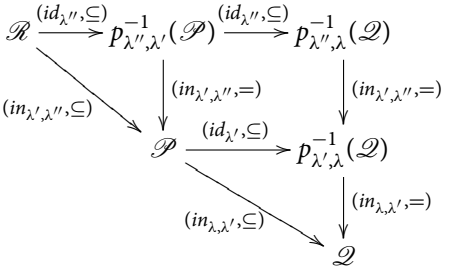



https://doi.org/10.1017/S0960129522000275 Published online by Cambridge University Press

Finally, the Grothendieck construction transforms the natural transformation (indexed functor) $\mathtt{sem} : \mathtt{ent} \Rightarrow \mathtt{pred}$ into a functor $\mathtt{Sem} : \mathsf{Ent} \to \mathsf{Pred}$ such that $\mathtt{Sem}; \Pi_{\mathsf{Pred}} = \Pi_{\mathsf{Ent}}$. Sem assigns to each local state assertion $(\lambda.Q)$ its local semantics $(\lambda.\mathtt{sem}_\lambda(Q))$ and to each entailment $(in_{\lambda,\lambda'}, \Vdash_{\lambda'}) : (\lambda'.P) \to (\lambda.Q)$, that is, $P \Vdash_{\lambda'} Q$ the corresponding semantic inclusion $(in_{\lambda,\lambda'}, \subseteq) : (\lambda'.\mathtt{sem}_{\lambda'}(P)) \to (\lambda.\mathtt{sem}_\lambda(Q))$, that is, $\mathtt{sem}_{\lambda'}(P) \subseteq p^{-1}_{\lambda',\lambda}(\mathtt{sem}_\lambda(Q))$.

The way "semantic" entailment is defined in (7) exactly by inclusions of state predicates becomes manifested by the fact that the functor $\mathtt{Sem} : \mathsf{Ent} \to \mathsf{Pred}$ is full.

**Remark 6** (Commutative Diagrams)**.** The reader should be aware that the diagrams we use to visualize the construction of a fibration and to validate its well-definedness turn into



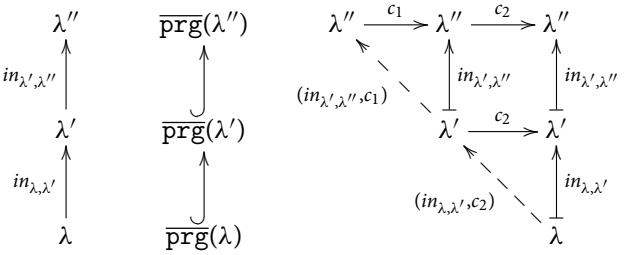

commutative diagrams in the resulting fibration. In case of the construction of Pred in Definition 6, for example, we obtain the commutative diagram above in Pred. The vertical arrows are Cartesian arrows, and the diagram shows also that each morphism $(in_{\lambda,\lambda'}, \subseteq) : (\lambda'.\mathscr{P}) \to (\lambda.Q)$ can be factorized into the composition $(in_{\lambda,\lambda'}, \subseteq) = (id_{\lambda'}, \subseteq); (in_{\lambda,\lambda'}, =)$ of a morphism in the fiber $\Pi^{-1}_{\mathsf{Pred}}(id_{\lambda'}) \simeq \mathtt{pred}(\lambda')$ and a Cartesian arrow.

### *4.2 Fibrations for local programs and weakest precondition semantics*

The functor $\overline{\mathtt{prg}} : \overline{\mathsf{Cont}} \to \mathsf{Mon}$ can be transformed into a category Prg of local programs.

**Definition 7** (Category Prg)**.** *The category* Prg *of "local programs":*

- *objects:* $|\mathsf{Prg}| \triangleq |\overline{\mathsf{Cont}}|$ *is the set of all extended contexts* $\lambda = (\gamma, \delta)$.
- *morphisms: from* $\lambda$ *to* $\lambda'$ *are all pairs* $(in_{\lambda,\lambda'}, c) : \lambda \to \lambda'$ *with* $in_{\lambda,\lambda'} : \lambda \to \lambda'$ *a morphism in* $\overline{\mathsf{Cont}}$ *and* $c \in \overline{\mathtt{prg}}(\lambda') = \mathtt{prg}(\gamma')$.
- *identities: the identity on* $\lambda$ *is* $(id_\lambda, \varepsilon) = (in_{\lambda,\lambda}, \varepsilon)$ *where* $\varepsilon$ *is the empty program.*
- *composition: the composition of two morphisms* $(in_{\lambda,\lambda'}, c_2) : \lambda \to \lambda'$ *and* $(in_{\lambda',\lambda''}, c_1) : \lambda' \to \lambda''$ *is the morphism* $(in_{\lambda,\lambda''}, c_1; c_2) : \lambda \to \lambda''$ *where* $in_{\lambda,\lambda''} = in_{\lambda,\lambda'}; in_{\lambda',\lambda''}$.




https://doi.org/10.1017/S0960129522000275 Published online by Cambridge University Press

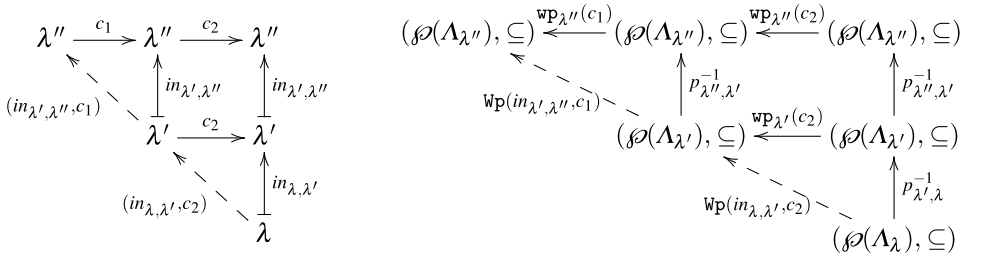
*Moreover, we obtain a projection functor* $\Pi_{\mathsf{Prg}} : \mathsf{Prg} \to \overline{\mathsf{Cont}}$ *with* $\Pi_{\mathsf{Prg}}(\lambda) \triangleq \lambda$ *and* $\Pi_{\mathsf{Prg}}((in_{\lambda,\lambda'}, c) : \lambda \to \lambda') \triangleq (in_{\lambda,\lambda'} : \lambda \to \lambda')$ *with fibers* $\Pi^{-1}_{\mathsf{Prg}}(id_\lambda) \simeq \mathtt{prg}(\lambda)^{op}$. □

The functor $\Pi_{\mathsf{Prg}} : \mathsf{Prg} \to \overline{\mathsf{Cont}}$ is an opfibration thus the opposite functor $\Pi^{op}_{\mathsf{Prg}} : \mathsf{Prg}^{op} \to \overline{\mathsf{Cont}}^{op}$ becomes a fibration. On the other side, the identity on $|\mathsf{Prg}| \triangleq |\overline{\mathsf{Cont}}|$ extends to an embedding $\mathtt{E}_{cont} : \overline{\mathsf{Cont}} \to \mathsf{Prg}$, such that $\mathtt{E}_{cont}; \Pi_{\mathsf{Prg}} = id_{\overline{\mathsf{Cont}}}$, mapping $in_{\lambda,\lambda'}$ to $(in_{\lambda,\lambda'}, \varepsilon)$.

Based on the results in Subsection 3.4, we can transform the weakest precondition natural transformation $\mathtt{wp} = \overline{\mathtt{tr}}; \mathtt{tc} : \overline{\mathtt{prg}} \Rightarrow \mathtt{if}$ into a functor $\mathtt{Wp} : \mathsf{Prg} \to \mathsf{Pre}$. Note that this is not one of the traditional Grothendieck constructions. $\mathtt{Wp}$ assigns to each object $\lambda$ in $\mathsf{Prg}$ the preorder category $\mathtt{Wp}(\lambda) \triangleq (\wp(\Lambda_\lambda), \subseteq)$ and to each morphism $(in_{\lambda,\lambda'}, c) : \lambda \to \lambda'$ in $\mathsf{Prg}$ the functor

$$\mathtt{Wp}(in_{\lambda,\lambda'}, c) \triangleq (p^{-1}_{\lambda',\lambda}; \mathtt{wp}_{\lambda'}(c) : (\wp(\Lambda_\lambda), \subseteq) \longrightarrow (\wp(\Lambda_{\lambda'}), \subseteq)).$$

$\mathtt{Wp}$ preserves identities $\mathtt{Wp}(id_\lambda, \varepsilon) = p^{-1}_{\lambda,\lambda}; \mathtt{wp}_\lambda(\varepsilon) = id_{(\wp(\Lambda_\lambda),\subseteq)}; id_{(\wp(\Lambda_\lambda),\subseteq)} = id_{(\wp(\Lambda_\lambda),\subseteq)}$ and is also compatible with composition $\mathtt{Wp}((in_{\lambda,\lambda'}, c_2); (in_{\lambda',\lambda''}, c_1)) = \mathtt{Wp}(in_{\lambda,\lambda''}, c_1; c_2) = p^{-1}_{\lambda'',\lambda}; \mathtt{wp}_\lambda(c_1; c_2) = (p^{-1}_{\lambda',\lambda}; \mathtt{wp}_\lambda(c_2)); (p^{-1}_{\lambda'',\lambda'}; \mathtt{wp}_\lambda(c1)) = \mathtt{Wp}(in_{\lambda,\lambda''}, c_2); \mathtt{Wp}(in_{\lambda,\lambda''}, c_1)$ since $\mathtt{wp}_\lambda$ is a monoid morphism from $\overline{\mathtt{prg}}(\lambda)$ into a submonoid of $\mathsf{Pre}((\wp(\Lambda_\lambda), \subseteq), (\wp(\Lambda_\lambda), \subseteq))^{op}$, that is, $\mathtt{wp}_\lambda(c_1; c_2) = \mathtt{wp}_\lambda(c_2); \mathtt{wp}_\lambda(c_1)$, and due to the results in Subsection 3.4 (see diagram (13)).



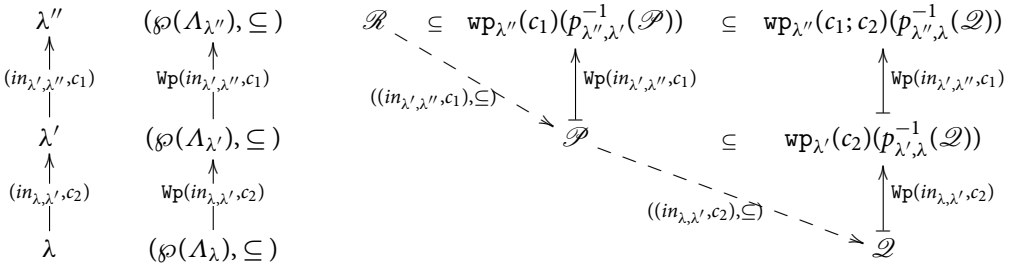

Note that the functor property of $\mathtt{Wp}$ entails that the syntactic weakest preconditions $wp(c_1, wp(c_2, Q))$ and $wp(c_1; c_2, Q)$ are logical equivalent. This equivalence is important for the discussion in Section 4.3.

Applying now to $\mathtt{Wp} : \mathsf{Prg} \to \mathsf{Pre}$ the appropriate variant of the traditional Grothendieck construction, we get the category $\mathsf{Wp}$ of semantic weakest preconditions.

**Definition 8** (Category $\mathsf{Wp}$)**.** *The category* $\mathsf{Wp}$ *of "local state predicates" and semantic weakest preconditions is defined as follows:*

- *objects:* $|\mathsf{Wp}| \triangleq |\mathsf{Pred}|$ *is the set of all pairs* $(\lambda.\mathscr{Q})$ *of an extended context* $\lambda \in |\overline{\mathsf{Cont}}|$ *and a state predicate* $\mathscr{Q} \in \wp(\Lambda_\lambda)$.
- *morphisms: from* $(\lambda'.\mathscr{P})$ *to* $(\lambda.\mathscr{Q})$ *are all pairs* $((in_{\lambda,\lambda'}, c), \subseteq)$ *with* $(in_{\lambda,\lambda'}, c) : \lambda \to \lambda'$ *a morphism in* $\mathsf{Prg}$ *and* $\mathscr{P} \subseteq \mathtt{Wp}(in_{\lambda,\lambda'}, c)(\mathscr{Q}) = \mathtt{wp}_{\lambda'}(c)(p^{-1}_{\lambda',\lambda}(\mathscr{Q}))$ *a morphism in* $\mathtt{Wp}(\lambda') = (\wp(\Lambda_{\lambda'}), \subseteq)$.
- *identities: the identity on* $(\lambda.\mathscr{P})$ *is* $((id_\lambda, \varepsilon), =)$.
- *composition: the composition of two morphisms* $((in_{\lambda',\lambda''}, c_1), \subseteq) : (\lambda''.\mathscr{R}) \to (\lambda'.\mathscr{P})$ *and* $((in_{\lambda,\lambda'}, c_2), \subseteq) : (\lambda'.\mathscr{P}) \to (\lambda.\mathscr{Q})$ *is the morphism* $((in_{\lambda,\lambda''}, c_1; c_2), \subseteq) : (\lambda''.\mathscr{R}) \to (\lambda.\mathscr{Q})$. *Composition is well-defined since* $\mathtt{Wp}$ *is a functor, that is, we have* $\mathtt{Wp}(in_{\lambda,\lambda'}, c_2); \mathtt{Wp}(in_{\lambda',\lambda''}, c_1) = \mathtt{Wp}(in_{\lambda,\lambda''}, c_1; c_2)$, *and since the* $\mathtt{Wp}(\lambda) = (\wp(\Lambda_\lambda), \subseteq)$ *are partial order categories.*

https://doi.org/10.1017/S0960129522000275 Published online by Cambridge University Press

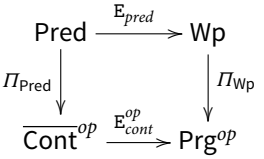



*The assignments* $\Pi_{\mathsf{Wp}}(\lambda.\mathscr{Q}) \triangleq \lambda$ *and* $\Pi_{\mathsf{Wp}}(((in_{\lambda,\lambda'}, c), \subseteq) : (\lambda'.\mathscr{P}) \to (\lambda.\mathscr{Q})) \triangleq ((in_{\lambda,\lambda'}, c) : \lambda \to \lambda')$ *define a projection functor* $\Pi_{\mathsf{Wp}} : \mathsf{Wp} \to \mathsf{Prg}^{op}$ *with fibers* $\Pi^{-1}_{\mathsf{Wp}}((id_\lambda, \varepsilon)) \simeq (\wp(\Lambda_\lambda), \subseteq)$.

The general properties of Grothendieck constructions provide:

**Theorem 5.** The functor $\Pi_{\mathsf{Wp}} : \mathsf{Wp} \to \mathsf{Prg}^{op}$ is a split fibration where the Cartesian arrows are exactly the morphisms $((in_{\lambda,\lambda'}, c), =) : (\lambda'.\mathtt{wp}_\lambda(p^{-1}_{\lambda',\lambda}(\mathscr{Q}))) \to (\lambda.\mathscr{Q})$ for all $(in_{\lambda,\lambda'}, c)^{op} : \lambda' \to \lambda$ in $\mathsf{Prg}^{op}$ and all state predicates $\mathscr{Q} \in \wp(\Lambda_\gamma)$. These are the only Cartesian arrows since inclusion $\subseteq$ is anti-symmetric.

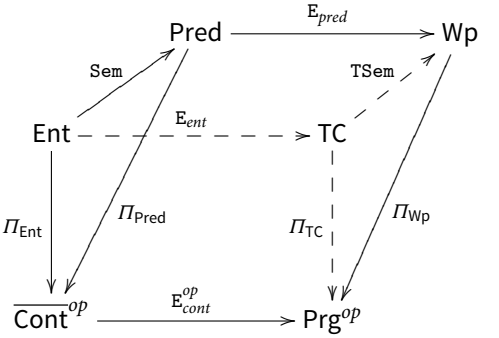
**Theorem 6** (Conservative Extension)**.** The identity on $|\mathsf{Wp}| \triangleq |\mathsf{Pred}|$ extends to an embedding




$\mathtt{E}_{pred} : \mathsf{Pred} \to \mathsf{Wp}$ mapping $(in_{\lambda,\lambda'}, \subseteq) : (\lambda'.\mathscr{P}) \to (\lambda.\mathscr{Q})$ to $((in_{\lambda,\lambda'}, \varepsilon), \subseteq) : (\lambda'.\mathscr{P}) \to (\lambda.\mathscr{Q})$. The resulting square commutes, that is, we have $\mathtt{E}_{pred}; \Pi_{\mathsf{Wp}} = \Pi_{\mathsf{Pred}}; \mathtt{E}^{op}_{cont}$. Moreover, it is a pullback square since $\mathtt{E}_{pred}$ establishes isomorphisms between the fibers $\Pi^{-1}_{\mathsf{Pred}}(id_\lambda) \simeq (\wp(\Lambda_\lambda), \subseteq)$ and $\Pi^{-1}_{\mathsf{Wp}}(\mathtt{E}^{op}_{cont}(id_\lambda)) = \Pi^{-1}_{\mathsf{Wp}}((id_\lambda, \varepsilon))$. Note that the pullback property means that the semantic fibration $\Pi_{\mathsf{Wp}} : \mathsf{Wp} \to \mathsf{Prg}^{op}$ is a "conservative extension" of the semantic fibration $\Pi_{\mathsf{Pred}} : \mathsf{Pred} \to \overline{\mathsf{Cont}}^{op}$, in the sense, that no new relations between local state predicates are introduced. The semantics of states is unchanged.

### *4.3 Fibration for Hoare logic*

The continuous lines in the diagram below show what we have gained so far in the fibered setting:




https://doi.org/10.1017/S0960129522000275 Published online by Cambridge University Press

The right face represents the logic of local states where entailment between local state assertions is defined semantically by inclusions between corresponding local state predicates. The logic of local states comprises as well all general first-order logic assertions as the theory of the data types of our language of expressions. The back face shows the extension of the category of local state predicates by semantic weakest preconditions for local programs.

The task of a Hoare proof calculus is nothing but to generate the missing category $\mathsf{TC}$ of total correctness assertions about local programs by extending, step by step, the category $\mathsf{Ent}$ of local state assertions. In parallel, three new functors should be constructed connecting the new category $\mathsf{TC}$ to the framework, developed so far. The natural requirements for a Hoare proof calculus can be reflected, in terms of fibrations, by the following objectives:

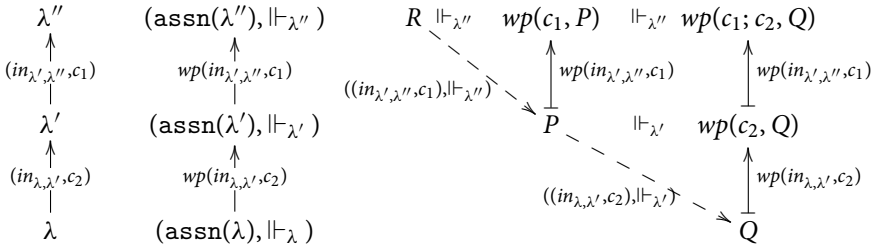
(1) **Soundness:** The existence of a functor $\mathtt{TSem} : \mathsf{TC} \to \mathsf{Wp}$ such that $\Pi_{\mathsf{TC}} = \mathtt{TSem}; \Pi_{\mathsf{Wp}}$, means that the calculus is sound. If $\mathtt{TSem}$ is, in addition, full, the calculus is also complete.
(2) **Conservative extension:** The functor $\mathrm{E}_{ent} : \mathsf{Ent} \to \mathsf{TC}$ should be an embedding such that the top and the front face commute, that is, $\mathrm{E}_{ent}; \mathtt{TSem} = \mathtt{Sem}; \mathrm{E}_{pred}$ and $\mathrm{E}_{ent}; \Pi_{\mathsf{TC}} = \Pi_{\mathsf{Ent}}; \mathrm{E}^{o}_{cont} p$. In addition, the front square should be a pullback square. Note that due to pullback decomposition, this implies that also the top square becomes a pullback.
(3) **Fibration:** Requiring that the resulting functor $\Pi_{\mathsf{TC}} : \mathsf{TC} \to \mathsf{Prg}^{op}$ is a fibration, we enforce the application of deduction rules until $\mathsf{TC}$ comprises all deducible correctness assertions.

The existence of syntactic weakest preconditions allows us, due to Corollary 1, to describe the category $\mathsf{TC}$ of total correctness assertions independent of a concrete deduction calculus. Analogously to the construction of $\mathsf{Wp}$, we need, however, some preparations. For any morphism $(in_{\lambda,\lambda'}, c) : \lambda \to \lambda'$ in $\mathsf{Prg}$, we consider the function $wp(in_{\lambda,\lambda'}, c) : \mathtt{assn}(\lambda) \to \mathtt{assn}(\lambda')$ with $wp(in_{\lambda,\lambda'}, c)(P) \triangleq wp(c, P)$ for all $P \in \mathtt{assn}(\lambda)$. $wp(\varepsilon, Q)$ and $Q$ are logically equivalent thus we can assume w.l.o.g. that $wp(\varepsilon, Q) = Q$ thus $wp(in_{\lambda,\lambda'}, \varepsilon)$ becomes an inclusion function. As discussed in Subsection 3.4, syntactic weakest preconditions are context independent, and, in addition, they are monotone, that is, $P \Vdash_{\lambda} Q$ implies $wp(c, P) \Vdash_{\lambda'} wp(c, Q)$ for all inclusion functions $in_{\lambda,\lambda'} : \lambda \to \lambda'$ and all programs $c : \lambda' \to \lambda'$. This means that the function $wp(in_{\lambda,\lambda'}, c) : \mathtt{assn}(\lambda) \to \mathtt{assn}(\lambda')$ defines, actually, a functor from $(\mathtt{assn}(\lambda), \Vdash_{\lambda})$ into $(\mathtt{assn}(\lambda'), \Vdash_{\lambda'})$. Moreover, we have that $wp(c_1, wp(c_2, Q))$ and $wp(c_1; c_2, Q)$ are logical equivalent, that is, isomorphic in $(\mathtt{assn}(\lambda), \Vdash_{\lambda})$, for arbitrary programs $c_1, c_2 : \lambda \to \lambda$ and arbitrary local state assertions $Q \in \mathtt{assn}(\lambda)$.

**Definition 9** (Category $\mathsf{TC}$). *The category* $\mathsf{TC}$ *of "local state assertions" and "total correctness assertions" is defined as follows:*

- *objects:* $|\mathsf{TC}| \triangleq |\mathsf{Ent}|$ *is the set of all pairs* $(\lambda.Q)$ *of an extended context* $\lambda \in |\overline{\mathsf{Cont}}|$ *and a local state assertion* $Q \in \mathtt{assn}(\lambda)$.
- *morphisms: a morphism* $((in_{\lambda,\lambda'}, c), \Vdash_{\lambda'}) : (\lambda'.P) \to (\lambda.Q)$ *is given by a morphism* $(in_{\lambda,\lambda'}, c) : \lambda \to \lambda'$ *in* $\mathsf{Prg}$ *such that the condition* $P \Vdash_{\lambda'} wp(c, Q)$ *is satisfied.*
- *identities: the identity on* $(\lambda.P)$ *is* $((id_{\lambda}, \varepsilon), \Vdash_{\lambda})$ *with the logical equivalence* $P \Vdash_{\lambda} wp(\varepsilon, P)$.
- *composition: the composition of two morphisms* $((in_{\lambda',\lambda''}, c_1), \Vdash_{\lambda''}) : (\lambda''.R) \to (\lambda'.P)$ *and* $((in_{\lambda,\lambda'}, c_2), \Vdash_{\lambda'}) : (\lambda'.P) \to (\lambda.Q)$ *is the morphism* $((in_{\lambda,\lambda''}, c_1; c_2), \Vdash_{\lambda''}) : (\lambda''.R) \to (\lambda.Q)$. *Composition is well-defined since both functors* $wp(in_{\lambda,\lambda'}, c_2); wp(in_{\lambda',\lambda''}, c_1)$ *and* $wp(in_{\lambda,\lambda''}, c_1; c_2)$ *are natural isomorphic, and since the* $\mathtt{ent}(\lambda) = (\mathtt{assn}(\lambda), \Vdash_{\lambda})$ *are preorder categories.*

https://doi.org/10.1017/S0960129522000275 Published online by Cambridge University Press




*The assignments* $\Pi_{\mathsf{TC}}(\lambda.Q) \triangleq \lambda$ *and* $\Pi_{\mathsf{TC}}(((in_{\lambda,\lambda'}, c), \Vdash_{\lambda'}) : (\lambda'.P) \to (\lambda.Q)) \triangleq ((in_{\lambda,\lambda'}, c) : \lambda \to \lambda')$ *define a projection functor* $\Pi_{\mathsf{TC}} : \mathsf{TC} \to \mathsf{Prg}^{op}$ *with fibers* $\Pi^{-1}_{\mathsf{TC}}((id_\lambda, \varepsilon)) \simeq \mathtt{ent}(\lambda)$.

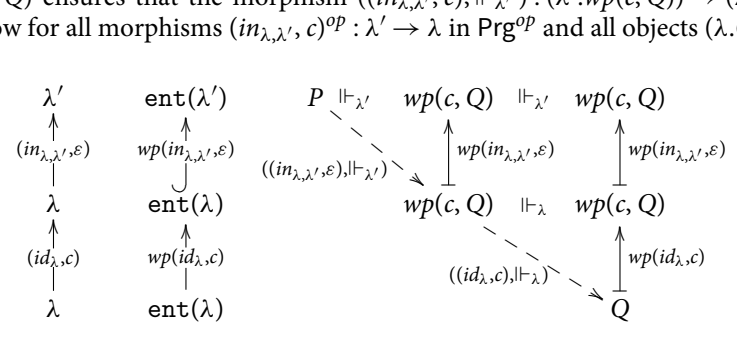
We discuss now if our three objectives for a Hoare proof calculus are indeed satisfied:

**Soundness:** We can define a functor $\mathtt{TSem} : \mathsf{TC} \to \mathsf{Wp}$ assigning to any object $(\lambda.Q)$ in TC the object $(\lambda.\mathtt{sem}_\lambda(Q))$ in Wp and to any morphism $((in_{\lambda,\lambda'}, c), \Vdash_{\lambda'}) : (\lambda'.P) \to (\lambda.Q)$ in TC with $P \Vdash_{\lambda'} wp(c, Q)$ the morphism $((in_{\lambda,\lambda'}, c), \subseteq) : (\lambda'.\mathtt{sem}_{\lambda'}(P)) \to (\lambda.\mathtt{sem}_\lambda(Q))$ in Wp with $\mathtt{sem}_{\lambda'}(P) \subseteq \mathtt{wp}_{\lambda'}(c)(p^{-1}_{\lambda',\lambda}(\mathtt{sem}_\lambda(Q)))$. Due to Proposition 3 as well as the definition and context independence of syntactic weakest preconditions, we have $\mathtt{wp}_{\lambda'}(c)(p^{-1}_{\lambda',\lambda}(\mathtt{sem}_\lambda(Q))) = \mathtt{wp}_{\lambda'}(c)(\mathtt{sem}_{\lambda'}(Q)) = \mathtt{sem}_{\lambda'}(wp(c, Q))$; thus, the assignments are well-defined. The functor property can be shown straightforwardly and the required commutativity $\Pi_{\mathsf{TC}} = \mathtt{TSem}; \Pi_{\mathsf{Wp}}$ is simply ensured by definition.

**Conservative extension:** Analogously to Theorem 6, the identity on $|\mathsf{TC}| \triangleq |\mathsf{Ent}|$ extends to an embedding $\mathrm{E}_{ent} : \mathsf{Ent} \to \mathsf{TC}$ mapping $(in_{\lambda,\lambda'}, \Vdash_{\lambda'}) : (\lambda'.P) \to (\lambda.Q)$ to $((in_{\lambda,\lambda'}, \varepsilon), \Vdash_{\lambda'}) : (\lambda'.P) \to (\lambda.Q)$. $P \Vdash_{\lambda'} Q$ implies $P \Vdash_{\lambda'} wp(\varepsilon, Q) = Q$, thus the embedding is well-defined. The resulting front square commutes, that is, we have $\mathrm{E}_{ent}; \Pi_{\mathsf{TC}} = \Pi_{\mathsf{Ent}}; \mathrm{E}^{op}_{cont}$. Moreover, it is a pullback square since $\mathrm{E}_{ent}$ establishes isomorphisms between the fibers $\Pi^{-1}_{\mathsf{Ent}}(id_\lambda) \simeq \mathtt{ent}_\lambda$ and $\Pi^{-1}_{\mathsf{TC}}(\mathrm{E}^{op}_{cont}(id_\lambda)) = \Pi^{-1}_{\mathsf{TC}}((id_\lambda, \varepsilon))$.

We know that $\mathtt{wp}_{\lambda'}(\varepsilon)$ is the identity on $(\wp(\Lambda_{\lambda'}), \subseteq)$; thus, the commutativity $\mathtt{Sem}; \mathrm{E}_{pred} = \mathrm{E}_{ent}; \mathtt{TSem}$ of the top square, and thus also its pullback property, can be easily shown.

**Fibration:** The functor $\Pi_{\mathsf{TC}} : \mathsf{TC} \to \mathsf{Prg}^{op}$ is a fibration since the equivalence of $wp(c_1, wp(c_2, Q))$ and $wp(c_1; c_2, Q)$ ensures that the morphism $((in_{\lambda,\lambda'}, c), \Vdash_{\lambda'}) : (\lambda'.wp(c, Q)) \to (\lambda.Q)$ in TC is a Cartesian arrow for all morphisms $(in_{\lambda,\lambda'}, c)^{op} : \lambda' \to \lambda$ in $\mathsf{Prg}^{op}$ and all objects $(\lambda.Q)$ in TC.




Instantiating Remark 6, we realize, first, that any morphism $(in_{\lambda,\lambda'}, c) : \lambda \to \lambda'$ in Prg can be factorized into the composition $(in_{\lambda,\lambda'}, c) = (id_\lambda, c); (in_{\lambda,\lambda'}, \varepsilon)$ of a morphism $(in_{\lambda,\lambda'}, \varepsilon) : \lambda \to \lambda'$, originating from $\overline{\mathsf{Cont}}$, and a new kind of morphism $(id_\lambda, c) : \lambda \to \lambda$ introducing programs. Second, we see that any morphism $((in_{\lambda,\lambda'}, c), \Vdash_{\lambda'}) : (\lambda'.P) \to (\lambda.Q)$ in TC can, correspondingly, be factorized $((in_{\lambda,\lambda'}, c), \Vdash_{\lambda'}) = ((in_{\lambda,\lambda'}, \varepsilon), \Vdash_{\lambda'}); ((id_\lambda, c), \Vdash_\lambda)$ into a morphism $((in_{\lambda,\lambda'}, \varepsilon), \Vdash_{\lambda'}) : (\lambda', P) \to (\lambda, wp(c, Q))$ originating from Ent and a Cartesian arrow $((id_\lambda, c), \Vdash_\lambda) : (\lambda.wp(c, Q)) \to (\lambda.Q)$ characterizing the program $c$ (see the diagram above).

https://doi.org/10.1017/S0960129522000275 Published online by Cambridge University Press

That is, to describe the extension of the category Ent to the category TC we need only the new arrows $((id_\lambda, c), \Vdash_\lambda) : (\lambda.wp(c, Q)) \to (\lambda.Q)$. Generating these special kind of Cartesian arrows is the essential task of a Hoare proof calculus. More precisely, a Hoare proof calculus does nothing but to extend the category Ent by new morphisms utilizing two procedures:

(1) **Construction of Cartesian arrows:** Generate the Cartesian arrow $((id_\lambda, c), \Vdash_\lambda) : (\lambda.wp(c, Q)) \to (\lambda.Q)$ for any object $(\lambda.Q)$ in $|\mathsf{Ent}| = |\mathsf{TC}|$ and any "program" morphism $(id_\lambda, c) : \lambda \to \lambda$ in Prg. These are the rules Skip, Assn, AAssn, IfE, PWh and TWh.
(2) **Composition:** Close everything w.r.t. composition
   a. by composing new morphisms in TC with new morphisms in TC (rule Comp) and
   b. by pre- and post-composing new morphisms in TC with given morphisms from Ent (rules Stren and Weakn).

In summary: Our discussion shows that we reached indeed all three objectives for the Hoare logic of total correctness assertions. As shown in Subsection 3.3, total correctness semantics and partial correctness semantics are structurally equivalent; thus, a corresponding variant of a categorical account of Hoare logic for partial correctness assertions and weakest liberal preconditions can be developed straightforwardly in a completely analogous way.

## 5. Related Works

Cook (1978) seems to be the seminal article for the mathematical study of Hoare logic (HL). Cook was the first deeply examining syntactical and semantic components related to HL and proving its soundness. A very interesting discussion in this work concerns the role of *data type specifications*. That is, assertions intended to formalize the relevant aspects of data types that we should use in connection with the rule of consequence to have correctness of programs supporting the data types. It was the first article considering Hoare logic as a logic parametrized by a data types specification in a modular way. The completeness theorem, on the contrary, was approached in Cook (1978) with less emphasis on modularity. Since Cook's work, many articles discussed how the data type specification integrates into HL more or less naturally. Indexing and Fibering are among the most worked out approaches to describe this integration. We briefly discuss some of the most relevant or recent articles on this below.

In the indexed/fibered approach to logical systems formalization, we consider a cartesian closed category or a preorder category to define truth values. Sometimes, other more sophisticated categories, such as topoi or higher-order categories, have their internal logic used to provide truth values. For example, the truth values may arise from a fibration construction when formalizing predicates in a categorical semantics for FOL (First-Order Logic), Typed-FOL or HOL (Higher Order Logic), respectively. At the same time, we use the internal logic to provide semantics to a set-like language using topoi. Of course, Hoare logic belongs to the first case since there is no mandatory need for a set-like language in an imperative program semantics. On the other hand, indexed categories are more related to the algebraic system specifications (compare Goguen and Burstall 1992). As we illustrate in this article, indexed categories and fibrations are two faces of the same coin, not only mathematically speaking but also in programming language semantics. We discuss, in the following, more papers related to the fibration approach. Indexed categories are more frequently used for defining categorical semantics for general logic (compare Diaconescu 2008; Wolter et al. 2012).

As an article, following the indexed approach to HL, it is worth to mention Goncharov and Schröder (2013). It defines order-enriched monads to induce a CPO structure on the monad itself rather than on the base category. The goal is to use this CPO structure as truth values by observing that any order-enriched monad induces a (weak) truth-value object. The enrichment has to do with the side effects, and finally, it presents a generic Hoare calculus for monadic side

https://doi.org/10.1017/S0960129522000275 Published online by Cambridge University Press

effecting programs. It proves the relative completeness of this Hoare calculus using the weakest preconditions system. Monads are the tool of choice to encapsulate side effects; thus, a monadic construction involves side effecting encapsulation naturally. It is, however, unusual that the enrichment of the base category with truth values happens on top of the monad itself. We see it as one difference in the treatment of truth values provided by our article. We discuss potential enrichments on the base category in the next paragraph. As in our case, Kleisli category is the semantic counterpart of the weakest precondition predicate transformers for computation. The Kleisli category relates to the partiality monad. We think that we have the advantage of providing a more detailed explanation of many fibrations provided by the Grothendick construction. In Goncharov and Schröder (2013), fibrations are not considered and the overall presentation follows indexed constructions. It is important to mention that the mechanism of obtaining the truth values on top of the monad has the drawback of having an assertion logic that is, in general, at least the full intuitionistic logic (see pages 4 and 5 in Goncharov and Schröder 2013). The fibrations we provide obtain always at least full intuitionistic FOL, and we consider this as a more adequate contribution.

Gaboardi et al. (2021) discusses and formalizes what is nowadays called Graded Hoare Logic (GHL). GHL is a family of Hoare logic extensions aiming to provide new deductive mechanisms to cope with some additional information to reason about side effects relative to programs. Such side effects can take cost analyses, probabilistic computations or security features into account when reasoning about program correctness. Even quantum effects can be included in this list of side effects, although this case is not considered in Gaboardi et al. (2021). A graded program language semantics is obtained by considering (new) type systems with fine-grained information added on top of the original semantics. Monads can be graded to consider embedding side effects into a pure language, as discussed above in connection with Goncharov and Schröder (2013). Almost all GHL proposed semantics use some side effects encapsulation in a monad, sometimes comonads, providing graded (co)monads. Historically, graded monads appeared first in functional semantics to deal with side effects in $\lambda$-calculus-based languages. Gaboardi et al. (2021) seem to be the first article that considers imperative languages in a uniform graded treatment. It takes graded categories to generalize the graded (co)monadic framework. There is some advantage of taking grading as a denotational approach instead of having it due to some imposition provided by (incremental) grading. It shows that graded categories abstract monadic and co-monadic semantics for grading. Afterwards, it considers an extension to the novel structure of graded Freyd categories. A Freyd category is a way of obtaining a set-like model on top of any (locally small) category with a terminal object. The construction of a Freyd category starts with the global sections of the terminal object and employs (coherent) fibrations to add more and more structure and logic incrementally. Thus, the semantic framework for GHL, on top of (graded) Freyd categories, uses a fibrational setting. It is similar to our fibrational approach, with the main difference that we do this for the standard reasoning on imperative program correctness. We only comment on the possibility of augmenting the data type side effects in our article, mainly to a (mixed) quantum-based programming language. The fact that we conduct our fibrational approach for Hoare logic semantics on a fibration semantics is consistent with what is discussed and reported in Gaboardi et al. (2021). Our contribution goes deeper since we have a detailed explanation of the relation to indexed (algebraic) Hoare semantics.

Martin et al. (2006) provide an elegant approach addressing the question what kind of categorical semantics one needs to read off from it an instance of a complete and sound set of Hoare logic rules. It is an entirely theoretical work pointing at that a particular kind of traced symmetric monoidal categories can be such mathematical structure. The traced symmetric monoidal categories for while-programs read off Hoare's original set of rules. The article further shows how to utilize the approach to cope with extensions of while-programs, including pointers and other features. A functor from a traced symmetric category to a preordered category that plays the truth values category is called a verification functor. The verification functor, a monoidal functor, is

https://doi.org/10.1017/S0960129522000275 Published online by Cambridge University Press

interpreted as a Hoare triple. The logical rules arise naturally from this abstract view. Adding new features to while-programs, such as pointers, is made by lifting the monoidal verification functors from a (new) preordered category that includes the semantic domains abstraction for the Heap and the Store. The monoidal requirement ensures, in a certain way, that this lifting is cartesian. The lifting is not described in terms of fibrations, but they are implicitly there. Compared with other articles devoted to Hoare logic semantics, we can say that Martin et al. (2006) uses fewer higher-categorical constructions than the other articles mentioned in this section. Our approach provides a complete and more detailed categorical explanation of this technique in the language of fibrations and indexed categories. We consider this as another novel contribution of our paper.

Hasuo (2015) and Aguirre and Katsumata (2020) discuss some monadic models of computational effects that can be used to provide semantics to weakest (liberal) preconditions predicate transformers taking into account a variety of side effects. Section 2 in Hasuo (2015) describes many contravariant monadic functors that obtain enriched monads, as in Goncharov and Schröder (2013). Some examples appear in both papers, such as the powerset and lifting cases which can also be found in category theory textbooks (see Crole 1993 as one possible reference). The goal of Hasuo (2015) is to provide semantics for the logic of predicates in forced games. However, the framework can be also applied to the Hoare logic of imperative while-programs.

Aguirre and Katsumata (2020) are more abstract than our approach and geared to the treatment of monadic effects. Given a fibration $P : \mathsf{E} \to \mathsf{C}$, for every $\mathsf{C}$-arrow $f : x \to y$ (morally a program) the fibered structure gives a functor from the fiber over $y$ (i.e. predicates over $y$) to the fiber over $x$ (i.e. predicates over $x$), which can be seen as the "weakest precondition" of $f$. From this starting point, Aguirre and Katsumata (2020) study how a monad $T$ modeling an effect on $\mathsf{C}$ can be lifted to a monad on $\mathsf{E}$ such that there is a fibration between the corresponding Kleisli categories that gives a weakest precondition transformer for effectful computations. In particular, they study the case where $\mathsf{E}$ is a slice category for some object $o$ of $\mathsf{C}$ representing truth values and investigate how to construct monadic liftings from $o$-carried Eilenberg-Moore $T$-algebras. They provide one example in which they instantiate their framework for a concrete imperative language, but the remaining examples are kept abstract. Our approach and intended application are different. Aguirre and Katsumata (2020) assume they have some abstract categories of programs and predicates, and a fibration between them. Our construction is more explicit and starts from the ground up, with a concrete imperative while language and a concrete assertion language. Then, we obtain the fibrational structure via the Grothendieck construction. This is a more appropriate setting in which to discuss issues like soundness and completeness of the Hoare calculus as properties of the functors relating the different categories. The fact that $\mathsf{TC}$ is a fibration also allows us to reason about syntactic weakest preconditions, which Aguirre and Katsumata (2020) do not cover.

In summary, this is in our view the novelty we provide with respect to Aguirre and Katsumata (2020) (which also applies to other related work, e.g. Hasuo 2015): (1) Application to a concrete programming language and assertion language. (2) Construction of the categorical structures "from the ground up." (3) Explicit separation of syntax and semantics that allows for an easier and more direct discussion of soundness and completeness.

## 6. Conclusion

The traditional presentation of the structural features of Hoare logic is based on a global context of program and logical variables. However, a categorical reformulation of these constructions must be based on local contexts for expressions and formulas. We recast the conceptual framework of Hoare logic from the perspective of both indexed and fibered categories.

With indexed categories, we developed a logic of local states, with finite contexts of program and array variables. On top of this logic, we develop a logic of local state assertions, which is based

https://doi.org/10.1017/S0960129522000275 Published online by Cambridge University Press

on extended contexts of both program and logical variables. After that, we presented the transition semantics of programs with finitary contexts by developing suitable categorical constructions for restricting the traditional, global transition semantics. This local transition semantics of programs is based on a more general theory of partial state transition maps. Theorem 1 is a reformulation of the idea of "programs as predicate transformers" using general partial state transition maps. Corollary 1 is an application of this result for partial transition maps generated by programs.

On the other hand, there are some important reasons to present Hoare logic also with fibrations. The most essential one is that fibrations provide a mathematical workspace, where logical deduction can take place. By translating the indexed categorical presentation into a fibered presentation, we have been able to formalize precisely the intuition that Hoare triples are a kind of fibered entity, that is, Hoare triples arise naturally as special arrows in a fibered category over a syntactic category of programs. Moreover, deduction in Hoare calculi can be characterized categorically by the heuristic *deduction = generation of cartesian arrows + composition of arrows.*

As a further work, using the techniques and tools developed in this paper as a blueprint, we are currently in the early stages of developing a Hoare logic for a quantum programming language (QPL). For QPL, the logic of states is twofold. We have the logic of quantum states and the logic of classical states. To have both of them together, in a well-integrated way, we use Indexed Categories and Fibrations. The logic of programs develops on top of this twofold category of classical-quantum states.

Many imperative programming languages, like C, allow us to declare and allocate local program variables in the middle of a program. Such programs can no longer be modeled by endomorphisms on the set of environments. We developed already some ideas how to extend and vary our approach to deal also with "allocations of local variables." It will be interesting to see if we can utilize Hasuo (2015), Aguirre and Katsumata (2020) to work out such an extension in detail.

Summing up, we think that one of the most important contributions of our article is to show in detail both sides of the two most used mechanisms to provide categorical and modular semantics for Hoare style logics. More research is needed to have what we described in this paper as a parametric framework to derive the detailed correctness proof and its associated set of Hoare rules. This is an additional step to the goals stated in Cook (1978).

**Acknowledgements.** We thank the anonymous reviewers for their careful reading of our manuscript and their many insightful comments and suggestions.

**Conflicts of interest.** The authors declare none.

## Notes

**1** More precisely, only variables $x$ that appear on the left hand side of an assignment $x := e$ may be changed. We abstract here from this small subtlety.

**2** A preorder can be seen as a small category with at most one morphism between any two objects thus we consider the category Pre of preorders and its subcategory Po of partial orders as subcategories of the category Cat of all small categories.

**3** A partial function $f : A \nrightarrow B$ is given by a set $\mathrm{DD}(f) \subseteq A$, called the domain of definition of $f$, and a span of a total inclusion function $in_{\mathrm{DD}(f),A} : \mathrm{DD}(f) \hookrightarrow A$ and a total function $f : \mathrm{DD}(f) \to B$. In case $\mathrm{DD}(f) = A$ and thus $in_{\mathrm{DD}(f),A} = id_A$, $f$ is a usual total function. The composition $f; g : A \nrightarrow C$ of two partial functions $f : A \nrightarrow B$, $g : B \nrightarrow C$ is defined by means of an inverse image (pullback) construction in Set: $\mathrm{DD}(f; g) \triangleq f^{-1}(\mathrm{DD}(g))$ and $f; g(a) \triangleq g(f(a))$ for all $a \in \mathrm{DD}(f; g)$.

**4** For a partial function $f : A \nrightarrow B$ the inverse image of a subset $B' \subseteq B$ is given by $f^{-1}(B') \triangleq \{a \in A \mid a \in \mathrm{DD}(f), f(a) \in B'\}$.

## References

Aguirre, A. and Katsumata, S. (2020). Weakest preconditions in fibrations. *Electronic Notes in Theoretical Computer Science* **352** 5–27. The 36th Mathematical Foundations of Programming Semantics Conference, 2020.

Apt, K. R., de Boer, F. S. and Olderog, E. (2009). *Verification of Sequential and Concurrent Programs*, Texts in Computer Science, Springer Dordrecht Heidelberg London New York.

https://doi.org/10.1017/S0960129522000275 Published online by Cambridge University Press

Barr, M. and Wells, C. (1990). *Category Theory for Computing Science*, Series in Computer Science, London, Prentice Hall International.

Bubel, R. and Hähnle, R. (2016). Key-hoare. In: *Deductive Software Verification - The KeY Book - From Theory to Practice*, 571–589.

Cook, S. A. (1978). Soundness and completeness of an axiom system for program verification. *SIAM Journal on Computing* **7** (1) 70–90.

Crole, R. J. (1993). *Categories for Types*, Cambridge, Cambridge University Press.

Diaconescu, R. (2008). *Institution-Independent Model Theory*, 1st ed., Basel, Birkhäuser.

Dijkstra, E. W. (1975). Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM* **18** (8) 453–457.

Gaboardi, M., Katsumata, S., Orchard, D. and Sato, T. (2021). Graded Hoare logic and its categorical semantics. In: Yoshida, N. (ed.) *Programming Languages and Systems - 30th European Symposium on Programming, ESOP 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27–April 1, 2021, Proceedings*, Lecture Notes in Computer Science, vol. 12648, Springer, 234–263.

Goguen, J. A. and Burstall, R. M. (1992). Institutions: Abstract model theory for specification and programming. *Journal of the ACM* **39** (1) 95–146.

Goncharov, S. and Schröder, L. (2013). A relatively complete generic hoare logic for order-enriched effects. In: *2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*, 273–282.

Hasuo, I. (2015). Generic weakest precondition semantics from monads enriched with order. *Theoretical Computer Science* **604** 2–29. Coalgebraic Methods in Computer Science.

Huth, M. and Ryan, M. D. (2004). *Logic in Computer Science - Modelling and Reasoning about Systems*, 2nd edition, Cambridge, Cambridge University Press.

Jacobs, B. (2001). *Categorical Logic and Type Theory*, Studies in Logic and the Foundations of Mathematics, vol. 141, Amsterdam, Elsevier.

Knijnenburg, P. and Nordemann, F. (1994). Partial hyperdoctrines: Categorical models for partial function logic and hoare logic. *Mathematical Structures in Computer Science, Cambridge University Press* **4** (02) 117–146.

Leino, R. (2010). Dafny: An automatic program verifier for functional correctness. In: *16th International Conference, LPAR-16, Dakar, Senegal*, Springer Berlin Heidelberg, 348–370.

Loeckx, J. and Sieber, K. (1987). *The Foundations of Program Verification*, 2nd ed., Stuttgart, Wiley-Teubner.

Martin, U., Mathiesen, E. A. and Oliva, P. (2006). Hoare logic in the abstract. In: Ésik, Z. (ed.) *Computer Science Logic*, Lecture Notes in Computer Science, vol. 4207, Berlin, Heidelberg, Springer, 501–515.

Martini, A. (2020). Reasoning about partial correctness assertions in Isabelle/HOL. *RITA* **27** (3) 84–101.

Martini, A., Wolter, U. and Haeusler, E. H. (2007). Fibred and indexed categories for abstract model theory. *Journal of the IGPL* **15** (5–6) 707–739.

Pawlowski, W. (1995). Context institutions. In: Haveraaen, M., Owe, O. and Dahl, O.-J. (eds.) *COMPASS/ADT*, Lecture Notes in Computer Science, vol. 1130, Springer, 436–457.

Pierce, B. C., de Amorim, A. A., Casinghino, C., Gaboardi, M., Greenberg, M., Hriţcu, C., Sjöberg, V., Tolmach, A. and Yorgey, B. (2018a). *Programming Language Foundations*, Software Foundations Series, vol. 2, Electronic Textbook.

Pierce, B. C., de Amorim, A. A., Casinghino, C., Gaboardi, M., Greenberg, M., Hriţcu, C., Sjöberg, V. and Yorgey, B. (2018b). *Logical Foundations*, Software Foundations Series, vol. 1, Electronic Textbook.

Pitts, A. M. (2000). Categorical logic. In: Abramsky, S., Gabbay, D. M. and Maibaum, T. S. E. (eds.) *Handbook of Logic in Computer Science, Volume 5. Algebraic and Logical Structures*, Oxford University Press, 39–128.

Wolter, U., Martini, A. and Haeusler, E. H. (2012). Towards a uniform presentation of logical systems by indexed categories and adjoint situations. *Journal of Logic and Computation, Oxford University Press* **25** (1) 57–93. Advance Access article.

Wolter, U., Martini, A. R. and Haeusler, E. H. (2020). Indexed and fibred structures for Hoare logic. *Electronic Notes in Theoretical Computer Science* **348** 125–145.

**Cite this article:** Wolter UE, Martini AR and Häusler EH (2022). Indexed and fibered structures for partial and total correctness assertions. *Mathematical Structures in Computer Science* **32**, 1145–1175. https://doi.org/10.1017/S0960129522000275