Mathematical Structures in Computer (2019), 1–00
doi:10.1017/xxxxx

CAMBRIDGE
UNIVERSITY PRESS

ARTICLE

Index and Fibred Structures for Partial and Total Correctness Assertions

U. E. Wolter, A. R. Martini, and E. H. Häusler

Department of Informatics - University of Bergen - Bergen - Norway
Uwe.Wolter@uib.no
Av. Marechal Andrea 11/210 - Porto Alegre - Brazil
alfio.martini@gmail.com
Departamento de Ciência da Computação - PUC-Rio - Rio de Janeiro - Brazil
hermann@inf.puc-rio

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 fibred 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 fibred 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); Loeckx and Sieber (1987); Leino (2010)) 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 its 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 tradition 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 any categorical analysis and presentation of logics should be based on local contexts for expressions and formulas. So, 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.

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 fibred one. First, the fibred 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 adequate.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 fibred 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.

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 a fibrations and we discuss Hoare triples and Hoare deduction calculus in the light of the corresponding fibred categories. We close the paper by a summary 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 fibred one.

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, i.e, its semantics and proof theory, and the core ideas underlying indexed and fibred 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 *mutually 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 σ 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 Σ. Given a state σ_P and a program variable $x \in \mathbf{PVar}$, we denote by $\sigma_P[x \mapsto n]$ a new state that is everywhere like σ_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 $a[i \mapsto j]$ asserts that it is also an array, i.e., 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 $bop \in \{\wedge, \vee\}$.

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

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 σ. There are two possible outcomes.If δ' is of the form $\langle c', \sigma' \rangle$, $c' \neq \underline{\mathbf{skip}}$ then the execution of c from σ is not completed. Otherwise, if $\delta' = \langle \underline{\mathbf{skip}}, \sigma' \rangle$ then the execution of c from σ has terminated with final state σ'. 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 σ 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 δ_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 δ_0 and δ_k has a finite number of steps, where $\overset{*}{\longrightarrow}$ is the reflexive, transitive closure of the relation $\longrightarrow$.

Table 1. Transition Semantics for IMP

$\longrightarrow\ \subseteq (\mathbf{Prg} \times \Sigma) \times (\mathbf{Prg} \times \Sigma)$
$\langle x := e, \sigma\rangle \longrightarrow \langle \underline{\mathbf{skip}}, \sigma'\rangle, \quad \sigma'_A = \sigma_A, \sigma'_P = \sigma_P[x \mapsto [\![e]\!]\sigma_P] \quad \text{(ass1)}$
$\langle a[e], \sigma\rangle \longrightarrow \langle \underline{\mathbf{skip}}, \sigma'\rangle, \quad \sigma'_P = \sigma_P, \sigma'_A = \sigma_A([\![e]\!]\sigma_P \mapsto [\![e']\!]\sigma_p) \quad \text{(ass2)}$
$\langle \underline{\mathbf{if}}\, b\, \underline{\mathbf{then}}\, c_0\, \underline{\mathbf{else}}\, c_1\, \underline{\mathbf{fi}}, \sigma\rangle \longrightarrow \langle c_0, \sigma\rangle, \quad [\![b]\!]_\sigma = true \ \text{(if1)}$
$\langle \underline{\mathbf{if}}\, b\, \underline{\mathbf{then}}\, c_0\, \underline{\mathbf{else}}\, c_1\, \underline{\mathbf{fi}}, \sigma\rangle \longrightarrow \langle c_1, \sigma\rangle, \quad [\![b]\!]_\sigma = false \ \text{(if2)}$
$\langle \underline{\mathbf{while}}\, b\, \underline{\mathbf{do}}\, c\, \underline{\mathbf{od}}, \sigma\rangle \longrightarrow \langle \underline{\mathbf{if}}\, b\, \underline{\mathbf{then}}\, (c; \underline{\mathbf{while}}\, b\, \underline{\mathbf{do}}\, c\, \underline{\mathbf{od}})\, \underline{\mathbf{else}}\, \underline{\mathbf{skip}}\, \underline{\mathbf{fi}}, \sigma\rangle \quad \text{(while)}$

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

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

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

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

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 $\{P\}\ c\ \{Q\}$, 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 2 (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 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

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

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

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

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

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 $Pos \triangleq 0 \leq index \implies index < Length \wedge a[index] = key \implies index = -1 \wedge \forall J. 0 \leq J < n \implies \neg(a[J] = key)$. Moreover $init \triangleq index := -1; i := 0$, $B \triangleq (i < n) \wedge (index = -1)$ and $body \triangleq \underline{\textbf{if}}\, (a[i] = key)\, index = i\, \underline{\textbf{else}}\, \underline{\textbf{skip}}\, \underline{\textbf{fi}}; i := i + 1$.

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

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

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, i.e., 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 $\mathbf{Env} = (\mathbf{LVar} \to \mathbb{Z})$.

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

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

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

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

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

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

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

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 $\mathbf{Env} \times \mathbf{Assn}$. 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 σ and environment α. 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 $\exists \sigma' \in \Sigma. [\![c]\!]\sigma = \sigma'$ 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 3. *Let $\{P\}\, c\, \{Q\}$ be a partial correctness assertion. Then the Hoare calculus is sound, i.e., 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 $\models [P]\, c\, [Q]$.*

Categories

We assume the reader has a working knowledge of first-order logic, i.e, 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.

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?"

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. 8)	finite contexts $\gamma \subseteq \mathbf{Var}$ (finite sets of program and array variables)	inclusions functions $in_{\gamma,\gamma'} : \gamma \hookrightarrow \gamma'$
$\overline{\mathsf{Cont}}$ (p. 8)	extended contexts $\lambda = (\gamma, \delta)$ with δ a finite set of logical variables	pairs of inclusion functions $(in_{\gamma,\gamma'}, in_{\delta,\delta'}) : (\gamma, \delta) \to (\gamma', \delta')$
Ent (p. 18)	pairs $(\lambda.Q)$ with Q a local state assertion in extended context λ	pairs of a morphism in $\overline{\mathsf{Cont}}$ and a semantic entailment between local state assertions
Pred (p. 19)	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 $\overline{\mathsf{Cont}}$ and an inclusion between local state predicates
Prg (p. 20)	$\lvert\mathsf{Prg}\rvert = \lvert\overline{\mathsf{Cont}}\rvert$	pairs of a morphism $(in_{\gamma,\gamma'}, in_{\delta,\delta'})$ in $\overline{\mathsf{Cont}}$ and a program in extended context λ'
Wp (p. 21)	$\lvert\mathsf{Wp}\rvert = \lvert\mathsf{Pred}\rvert$	pairs of a morphism in Prg and an inclusion of a local state predicate in a semantic weakest precondition
TC (p. 23)	$\lvert\mathsf{TC}\rvert = \lvert\mathsf{Ent}\rvert$	pairs of a morphism in Prg and a semantic entailment between a local state assertion and a syntactic weakest precondition

One basic structural feature of a Hoare logic we observed already. 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 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. From a practical point of view 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 as clearly shown in examples 1, 2 and 3. This is a perfect match with the “categorical imperative” that any categorical analysis and presentation of logics should be based on local contexts for expressions and formulas.

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 fibred 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.[a] 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[b] Cont with $|\mathsf{Cont}| \triangleq \wp_{fin}(\mathbf{Var})$, i.e., the set of all finite contexts and morphisms all inclusions 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 $\Sigma_\gamma \triangleq (\gamma \to \mathbb{D})$, i.e., 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 pre-composition:

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

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 build upon expressions which, in turn, are build 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 finite many program variables, only finite 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 are based on finite contexts of program and logical variables (see Martini (2020); Leino (2010); Pierce et al. (2018); Bubel and Hähnle (2016)).

In such a way, we can extend the category 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 $\Gamma_\delta \triangleq (\delta \to \mathbb{Z})$, i.e., 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 pre-composition:

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

[a] More precisely only the values of those variables x may be changed that appear on the left hand side of an assignment $x := e$. We abstract here from this small subtlety.

[b] 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.

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 4 (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*

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

Proof. Due to our definitions we have $(pvr(P), flv(p)) \subseteq \lambda \subseteq \lambda'$ and that (σ', α') 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 5 (Logic of States is an Institution). A closer look at the development so far shows that we have actually defined an *Institution* (see Goguen and Burstall (1992), Diaconescu (2008)): 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 4, 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. □

Semantics of Local State Assertions: Any subset of Λ_λ 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, i.e., 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 λ 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 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 $\overline{\mathsf{Cont}}$, i.e., 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 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 Pre that not only preserves but also reflects order, i.e., is full seen as a functor.

Remark 6 (Cartesian Closed Category)**.** The preorder category $\mathtt{ent}(\lambda) = (\mathtt{assn}(\lambda), \Vdash_\lambda)$, for any object λ in $\overline{\mathsf{Cont}}$, is Cartesian closed with products $\wedge$, sums $\vee$ and exponentiation $\to$, i.e., we have

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

Proposition 4 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, i.e., for all assertions $P, Q \in \mathtt{assn}(\lambda)$ we have that $P \Vdash_\lambda Q$, i.e., $\mathtt{sem}_\lambda(P) \subseteq \mathtt{sem}_\lambda(Q)$, implies $P \Vdash_{\lambda'} Q$, i.e., $\mathtt{sem}_{\lambda'}(P) \subseteq \mathtt{sem}_{\lambda'}(Q)$. 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 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 $\mathtt{assn}$.

$$\begin{tikzcd}
\overline{\mathsf{Cont}} \arrow[d, bend right=50, "\mathtt{pred}"'] \arrow[d, bend left=50, "\mathtt{ent}"] \arrow[d, phantom, "\overset{\mathtt{sem}}{\Longleftarrow}"] \arrow[dr, bend left, "\mathtt{assn}"] \arrow[dr, phantom, "="] & \\
\mathsf{Pre} \arrow[r, "\mathtt{carr}"] & \mathsf{Set}
\end{tikzcd}$$

Finally, Proposition 4 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)):

Proposition 7. *The morphisms* $\mathtt{sem}_\lambda : \mathtt{ent}(\lambda) \to \mathtt{pred}(\lambda)$ *in* Pre *with* $\lambda \in |\overline{\mathsf{Cont}}|$ *constitute a natural transformation* $\mathtt{sem} : \mathtt{ent} \Rightarrow \mathtt{pred} : \overline{\mathsf{Cont}} \to \mathsf{Pre}$.

$$\begin{tikzcd}
\lambda = (\gamma, \delta) \arrow[d, "in_{\lambda,\lambda'} ="', "(in_{\gamma,\gamma'}, in_{\delta,\delta'})"] \\
\lambda' = (\gamma', \delta')
\end{tikzcd}
\qquad
\begin{tikzcd}
\Lambda_\lambda = \Sigma_\gamma \times \Gamma_\delta \\
\Lambda_{\lambda'} = \Sigma_{\gamma'} \times \Gamma_{\delta'} \arrow[u, "p_{\lambda',\lambda} ="', "p_{\gamma',\gamma} \times p_{\delta',\delta}"]
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathtt{pred}(\lambda) = (\wp(\Lambda_\lambda), \subseteq) \arrow[d, "p^{-1}_{\lambda',\lambda}"'] & \mathtt{ent}(\lambda) = (\mathtt{assn}(\lambda), \Vdash_\lambda) \arrow[l, "\mathtt{sem}_\lambda"'] \arrow[d, "\mathtt{ent}(in_{\lambda,\lambda'}) ="', "\mathtt{assn}(in_{\lambda,\lambda'})"] \\
\mathtt{pred}(\lambda') = (\wp(\Lambda_{\lambda'}), \subseteq) & \mathtt{ent}(\lambda') = (\mathtt{assn}(\lambda'), \Vdash_{\lambda'}) \arrow[l, "\mathtt{sem}_{\lambda'}"']
\end{tikzcd}$$

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, i.e., for each context γ we consider the corresponding set $\mathtt{prg}(\gamma) \triangleq \{c \in \mathbf{Prg} \mid pvr(c) \subseteq \gamma\}$ of programs in context γ. Our while-programs are sequential, i.e., 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" ε 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 γ 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 Σ_γ 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.[c] For any context γ we consider the monoid $\mathtt{pf}(\gamma)$ of local state transition maps with object Σ_γ and the whole hom-set $(\Sigma_\gamma \mathrel{\circ\!\!\!\rightarrow} \Sigma_\gamma) = \mathsf{Par}(\Sigma_\gamma, \Sigma_\gamma)$ as morphisms. "$\mathtt{pf}$" stands for "partial function".

For any inclusion function $in_{\gamma,\gamma'} : \gamma \hookrightarrow \gamma'$ we can define a function $\mathtt{pf}_{\gamma,\gamma'} : (\Sigma_\gamma \mathrel{\circ\!\!\!\rightarrow} \Sigma_\gamma) \to (\Sigma_{\gamma'} \mathrel{\circ\!\!\!\rightarrow} \Sigma_{\gamma'})$ that lifts any local state transition map $\tau : \Sigma_\gamma \mathrel{\circ\!\!\!\rightarrow} \Sigma_\gamma$ to a local state transition map $\tau' = \mathtt{pf}_{\gamma,\gamma'}(\tau) : \Sigma_{\gamma'} \mathrel{\circ\!\!\!\rightarrow} \Sigma_{\gamma'}$. 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 $(\sigma' : \gamma' \to \mathbb{D}) \in \mathrm{DD}(\tau') \subseteq \Sigma_{\gamma'}$

$$\begin{tikzcd}
\gamma' & \Sigma_{\gamma'} \arrow[d, "p_{\gamma',\gamma}"'] \arrow[dr, phantom, "pb"] & \mathrm{DD}(\tau') \arrow[l, hook'] \arrow[r, "\tau'"] \arrow[d, "p_{\gamma',\gamma}"'] \arrow[dr, phantom, "="] & \Sigma_{\gamma'} \arrow[d, "p_{\gamma',\gamma}"] \\
\gamma \arrow[u, hook, "in_{\gamma,\gamma'}"] & \Sigma_\gamma & \mathrm{DD}(\tau) \arrow[l, hook'] \arrow[r, "\tau"] & \Sigma_\gamma
\end{tikzcd}$$

$$\tau'(\sigma')(x) \triangleq \begin{cases} \tau(p_{\gamma',\gamma}(\sigma'))(x), & \text{if } x \in \gamma \\ \sigma'(x) & \text{, if } x \in \gamma' \setminus \gamma \end{cases} \tag{8}$$

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 $\tau_1, \tau_2 : \Sigma_\gamma \mathrel{\circ\!\!\!\rightarrow} \Sigma_\gamma$. Moreover, the definition entails $\mathtt{pf}_{\gamma,\gamma'}(id_{\Sigma_\gamma}) = id_{\Sigma_{\gamma'}}$ thus the function $\mathtt{pf}_{\gamma,\gamma'} : (\Sigma_\gamma \mathrel{\circ\!\!\!\rightarrow} \Sigma_\gamma) \to (\Sigma_{\gamma'} \mathrel{\circ\!\!\!\rightarrow} \Sigma_{\gamma'})$ 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 $\mathtt{pf} : \mathsf{Cont} \to \mathsf{Mon}$.

Local Programs - State Transition Semantics: The state transition semantics of local programs is simply defined by restricting the global state transition semantics $[\![\_]\!] : \mathbf{Prg} \to (\Sigma \mathrel{\circ\!\!\!\rightarrow} \Sigma)$ 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.

$$\begin{tikzcd}
\mathsf{Cont} \arrow[d, bend right=60, "\mathtt{pf}"', ""{name=L}] \arrow[d, bend left=60, "\mathtt{prg}", ""'{name=R}] \\
\mathsf{Mon}
\arrow[Rightarrow, from=R, to=L, "\mathtt{tr}"']
\end{tikzcd}$$

First, we show that for any context γ the global state transition semantics $[\![\_]\!] : \mathbf{Prg} \to (\Sigma \mathrel{\circ\!\!\!\rightarrow} \Sigma)$ of programs restricts to a functorial state transition semantics $\mathtt{tr}_\gamma : \mathtt{prg}(\gamma) \to (\Sigma_\gamma \mathrel{\circ\!\!\!\rightarrow} \Sigma_\gamma)$ 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 $p_\gamma(\rho) \triangleq in_\gamma; \rho$ for all global states $\rho \in \Sigma = (\mathbf{Var} \to \mathbb{D})$.

[c] A partial function $f : A \mathrel{\circ\!\!\!\rightarrow} B$ 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 $f; g : A \mathrel{\circ\!\!\!\rightarrow} C$ of two partial functions $f : A \mathrel{\circ\!\!\!\rightarrow} B$, $g : B \mathrel{\circ\!\!\!\rightarrow} C$ 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 $a \in \mathrm{DD}(f; g)$.

$p_\gamma : \Sigma \to \Sigma_\gamma$ is surjective since $\mathbb{D}$ is not empty. By means of p_γ we can restrict now for any local program $c \in \mathtt{prg}(\gamma)$ the partial function $\llbracket c \rrbracket : \Sigma \nrightarrow \Sigma$ to a partial function $\mathtt{tr}_\gamma(c) : \Sigma_\gamma \nrightarrow \Sigma_\gamma$: We define $\mathrm{DD}(\mathtt{tr}_\gamma(c)) \triangleq p_\gamma(\mathrm{DD}(\llbracket c \rrbracket))$ thus there exists for any local state $\sigma \in \mathrm{DD}(\mathtt{tr}_\gamma(c))$ a global state $\rho \in \mathrm{DD}(\llbracket c \rrbracket)$ with $p_\gamma(\rho) = \sigma$ and we can set $\mathtt{tr}_\gamma(c)(\sigma) \triangleq p_\gamma(\llbracket c \rrbracket(\rho))$. Why does this work?

$$\begin{tikzcd}
\Sigma \arrow[d, "p_\gamma"] \arrow[dr, phantom, "(1)"] & \mathrm{DD}(\llbracket c \rrbracket) \arrow[l, hook'] \arrow[r, "\llbracket c \rrbracket"] \arrow[d, "p_\gamma"] \arrow[dr, phantom, "="] & \Sigma \arrow[d, "p_\gamma"'] \\
\Sigma_\gamma & \mathrm{DD}(\mathtt{tr}_\gamma(c)) \arrow[l, hook'] \arrow[r, "\mathtt{tr}_\gamma(c)"] & \Sigma_\gamma
\end{tikzcd}$$

Any program c changes at most the values for the program variables in $pvr(c)$, i.e., for any global state $\rho \in \mathrm{DD}(\llbracket c \rrbracket)$ and any program variable $x \notin pvr(c)$ we have $\llbracket c \rrbracket(\rho)(x) = \rho(x)$. We defined $c \in \mathtt{prg}(\gamma)$ iff $prv(c) \subseteq \gamma$, thus for any global states $\rho, \rho' \in \Sigma$ it holds that $\rho \in \mathrm{DD}(\llbracket c \rrbracket)$ and $p_\gamma(\rho) = p_\gamma(\rho')$ implies $\rho' \in \mathrm{DD}(\llbracket c \rrbracket)$ and $p_\gamma(\llbracket c \rrbracket(\rho)) = p_\gamma(\llbracket c \rrbracket(\rho'))$ 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 Set, i.e., we have $\llbracket c \rrbracket; p_\gamma = p_\gamma; \mathtt{tr}_\gamma(c)$ in Par.

For any local programs $c_1, c_2 \in \mathtt{prg}(\gamma)$ the compositionality $\llbracket c_1; c_2 \rrbracket = \llbracket c_1 \rrbracket; \llbracket c_2 \rrbracket$ 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 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

$$\begin{tikzcd}
\Sigma_{\gamma'} \arrow[d, "p_{\gamma',\gamma}"'] \arrow[dr, phantom, "pb"] & \mathrm{DD}(\mathtt{pf}_{\gamma,\gamma'}(\mathtt{tr}_\gamma(c))) \arrow[l, hook'] \arrow[r, "\mathtt{pf}_{\gamma,\gamma'}(\mathtt{tr}_\gamma(c))"] \arrow[d, "p_{\gamma',\gamma}"'] \arrow[dr, phantom, "="] & \Sigma_{\gamma'} \arrow[d, "p_{\gamma',\gamma}"'] \\
\Sigma_\gamma & \mathrm{DD}(\mathtt{tr}_\gamma(c)) \arrow[l, hook'] \arrow[r, "\mathtt{tr}_\gamma(c)"] & \Sigma_\gamma
\end{tikzcd}$$

$$\mathtt{tr}_{\gamma'}(c) = \mathtt{pf}_{\gamma,\gamma'}(\mathtt{tr}_\gamma(c)) : \Sigma_{\gamma'} \nrightarrow \Sigma_{\gamma'} \qquad (9)$$

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

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, i.e., 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 $\llbracket c \rrbracket$, 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 pre-condition P describing properties of the "input states" $\sigma \in \Sigma_\gamma$ and a post-condition Q describing properties of the corresponding "output states" $\mathtt{tr}_\gamma(c)(\sigma) \in \Sigma_\gamma$.

We 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 8 (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, refer to the same local environment α. 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 [d].

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 8, the definition of $\overline{\tau}$ in (10) and the definition of inverse images for partial functions in footnote (d). □

In Theorem 1 the state transition map τ 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 kind 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

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

in Subsection 3.2 for plain contexts, to extended contexts:

$$\begin{tikzcd}
\overline{\mathsf{Cont}} \arrow[d, bend right=60, "\overline{\mathtt{pf}}"'] \arrow[d, bend left=60, "\overline{\mathtt{prg}}"] \arrow[d, phantom, "\overset{\overline{\mathtt{tr}}}{\Longleftarrow}"] \\
\mathsf{Mon}
\end{tikzcd}$$

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 $\tau, \tau' : \Sigma_\gamma \mathrel{\circ\!\!\!\rightarrow} \Sigma_\gamma$ thus we can chose $\overline{\mathtt{pf}}(\lambda)$ to be the submonoid of $(\Lambda_\lambda \mathrel{\circ\!\!\!\rightarrow} \Lambda_\lambda) = \mathsf{Par}(\Lambda_\lambda, \Lambda_\lambda)$ given by all extended state transition maps $\overline{\tau}_\delta \triangleq \tau \times id_{\Gamma_\delta} : \Sigma_\gamma \times \Gamma_\delta \mathrel{-\!\!\circ\!\!\!\rightarrow} \Sigma_\gamma \times \Gamma_\delta$ with $\tau \in \mathtt{pf}(\gamma) = (\Sigma_\gamma \mathrel{\circ\!\!\!\rightarrow} \Sigma_\gamma)$. Note, that $\mathtt{pf}(\gamma)$ and $\overline{\mathtt{pf}}(\lambda)$ are isomorphic, i.e., for each declaration δ 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 $\overline{\tau}_\delta = \tau \times id_{\Gamma_\delta} : \Lambda_\lambda \mathrel{\circ\!\!\!\rightarrow} \Lambda_\lambda$ in $\overline{\mathtt{pf}}(\lambda)$ the extended state transition map $\overline{\mathtt{pf}}_{\lambda,\lambda'}(\overline{\tau}_\delta) \triangleq \overline{\mathtt{pf}_{\gamma,\gamma'}(\tau)}_{\delta'} = \mathtt{pf}_{\gamma,\gamma'}(\tau) \times id_{\Gamma_{\delta'}} : \Lambda_{\lambda'} \mathrel{\circ\!\!\!\rightarrow} \Lambda_{\lambda'}$ in $\overline{\mathtt{pf}}(\lambda')$. Our construction transforms equation (8)

$$\begin{tikzcd}
\Sigma_{\gamma'} \times \Gamma_{\delta'} \arrow[r, -, "\mathtt{pf}_{\gamma,\gamma'}(\tau)\times id_{\Gamma_{\delta'}}"] & \circ \arrow[r] & \Sigma_{\gamma'} \times \Gamma_{\delta'} \arrow[d, mapsto, "p_{\gamma',\gamma}\times p_{\delta',\delta}"] \\
\Sigma_{\gamma'} \times \Gamma_{\delta'} \arrow[u, phantom] & & \\
\Sigma_\gamma \times \Gamma_\delta \arrow[r, -, "\tau\times id_{\Gamma_\delta}"'] & \circ \arrow[r] \arrow[uu, mapsto, "\overline{\mathtt{pf}}_{\lambda,\lambda'}"] & \Sigma_\gamma \times \Gamma_\delta
\end{tikzcd}$$

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

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

in 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 $\overline{\mathtt{pf}}_{\lambda,\lambda} = id_{\overline{\mathtt{pf}}(\lambda)}$ 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 $\mathtt{pf}(\gamma) = (\Sigma_\gamma \mathrel{\circ\!\!\!\rightarrow} \Sigma_\gamma)$ to a functor $\overline{\mathtt{tr}}_\lambda : \overline{\mathtt{prg}}(\lambda) \to \overline{\mathtt{pf}}(\lambda)$ between the monoids $\overline{\mathtt{prg}}(\lambda)$ and $\overline{\mathtt{pf}}(\lambda) \subseteq (\Lambda_\lambda \mathrel{\circ\!\!\!\rightarrow} \Lambda_\lambda)$. 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 $\tau, \tau' : \Sigma_\gamma \mathrel{\circ\!\!\!\rightarrow} \Sigma_\gamma$. 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 Set as a subcategory of 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 Par into 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 $f : A \mathrel{\circ\!\!\!\rightarrow} B$ 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 $f : A \mathrel{\circ\!\!\!\rightarrow} B$ 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, i.e., injective on objects and on morphisms.

For each extended context λ 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 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 Mon establish a natural isomorphism $\mathtt{tc} : \overline{\mathtt{pf}} \Rightarrow \mathtt{if}$.

$$\begin{tikzcd}[row sep=large]
\overline{\mathsf{Cont}} \arrow[d, bend right=60, "\mathtt{if}"'{name=L}] \arrow[d, bend left=60, "\overline{\mathtt{pf}}"{name=R}] \arrow[from=R, to=L, Rightarrow, phantom, ""] \\
\mathsf{Mon}
\arrow[from=R, to=L, Rightarrow, "\mathtt{tc}"']
\end{tikzcd}$$

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, i.e., with τ 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 $\mathtt{if}_{\lambda,\lambda'}(\overline{\tau}_\delta^{-1}) = \mathtt{tc}_{\lambda'}(\overline{\mathtt{pf}}_{\lambda,\lambda'}(\overline{\tau}_\delta)) = (\overline{\mathtt{pf}}_{\lambda,\lambda'}(\overline{\tau}_\delta))^{-1} : (\wp(\Lambda_{\lambda'}), \subseteq) \to (\wp(\Lambda_{\lambda'}), \subseteq)$

$$\begin{tikzcd}
(\wp(\Lambda_{\lambda'}), \subseteq) & {} & (\wp(\Lambda_{\lambda'}), \subseteq) \arrow[ll, "\mathtt{tc}_{\lambda'}(\overline{\mathtt{pf}}_{\lambda,\lambda'}(\overline{\tau}_\delta))"'] \\
(\wp(\Lambda_\lambda), \subseteq) \arrow[u, "p^{-1}_{\lambda',\lambda}"] & {} \arrow[u, mapsto, "\mathtt{if}_{\lambda,\lambda'}"] & (\wp(\Lambda_\lambda), \subseteq) \arrow[ll, "\mathtt{tc}_\lambda(\overline{\tau}_\delta)"] \arrow[u, "p^{-1}_{\lambda',\lambda}"']
\end{tikzcd}$$

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

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

Completely analogously, we can use the embedding $\mathtt{P_{DD}}^{op} : \mathsf{Par} \to \mathsf{Pre}^{op}$ to construct for each extended context λ in $\overline{\mathsf{Cont}}$ the image $\mathtt{if}^{\mathrm{DD}}(\lambda) \triangleq \mathtt{P_{DD}}^{op}(\overline{\mathtt{pf}}(\lambda))$ 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 $\overline{\mathtt{pf}}(\lambda)$. "$\mathtt{pc}$" stands for "partial correctness". For any morphism $in_{\lambda,\lambda'} : \lambda \to \lambda'$ in $\overline{\mathsf{Cont}}$, we can define a functor $\mathtt{if^{DD}}_{\lambda,\lambda'} \triangleq \mathtt{pc}_\lambda^{-1}; \overline{\mathtt{pf}}_{\lambda,\lambda'}; \mathtt{pc}_{\lambda'} : \mathtt{if}(\lambda) \to \mathtt{if}(\lambda')$. $\overline{\mathtt{pf}} : \overline{\mathsf{Cont}} \to \mathsf{Mon}$ 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 $\mathtt{pc} : \overline{\mathtt{pf}} \Rightarrow \mathtt{if}^{\mathrm{DD}}$.

$$\begin{tikzcd}[row sep=large]
\overline{\mathsf{Cont}} \arrow[d, bend right=60, "\mathtt{if}^{\mathrm{DD}}"'{name=L}] \arrow[d, bend left=60, "\overline{\mathtt{pf}}"{name=R}] \\
\mathsf{Mon}
\arrow[from=R, to=L, Rightarrow, "\mathtt{pc}"']
\end{tikzcd}$$

We fix the above discussion in the following theorem.

Theorem 2 (Natural isomorphisms)**.** For each extended context $\lambda = (\gamma, \delta)$, the assignments $\mathtt{tc}_\lambda(\overline{\tau}_\delta) \triangleq \overline{\tau}_\delta^{-1} : (\wp(\Lambda_\lambda), \subseteq) \to (\wp(\Lambda_\lambda), \subseteq)$, $\quad \mathtt{pc}_\lambda(\overline{\tau}_\delta) \triangleq (\overline{\tau}_\delta)^{-1}_{\mathrm{DD}} : (\wp(\Lambda_\lambda), \subseteq) \to (\wp(\Lambda_\lambda), \subseteq)$, 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 $\mathtt{P} : \mathsf{Par}^{op} \to \mathsf{Pre}$ and $\mathtt{P_{DD}} : \mathsf{Par}^{op} \to \mathsf{Pre}$ of the contravariant power set functor $\mathtt{P} : \mathsf{Set}^{op} \to \mathsf{Pre}$ 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 $\mathtt{tc}$ and $\mathtt{pc}$ 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

$$\begin{tikzcd}[row sep=huge, column sep=huge]
\overline{\mathsf{Cont}} \arrow[d, bend right=70, "\mathtt{if}"'{name=I}] \arrow[d, "\overline{\mathtt{pf}}"{name=P}] \arrow[d, bend left=70, "\overline{\mathtt{prg}}"{name=R}] \\
\mathsf{Mon}
\arrow[from=P, to=I, Rightarrow, "\mathtt{tc}"', shorten=3pt]
\arrow[from=R, to=P, Rightarrow, "\overline{\mathtt{tr}}"', shorten=3pt]
\end{tikzcd}$$

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

$$\begin{tikzcd}
\lambda' & (\lambda' \xrightarrow{c} \lambda') \arrow[r, mapsto, "\overline{\mathtt{tr}}_{\lambda'}"] & (\Lambda_{\lambda'} \arrow[r, "\overline{\mathtt{tr}}_{\lambda'}(c)"{name=U}] \arrow[d, "p_{\lambda',\lambda}"] & \Lambda_{\lambda'}) \arrow[r, mapsto, "\mathtt{tc}_{\lambda'}"] \arrow[d, "p_{\lambda',\lambda}"] & ((\wp(\Lambda_{\lambda'}), \subseteq) & (\wp(\Lambda_{\lambda'}), \subseteq)) \arrow[l, "\mathtt{wp}_{\lambda'}(c)"'{name=W}] \\
\lambda \arrow[u, "in_{\lambda,\lambda'}"] & (\lambda \xrightarrow{c} \lambda) \arrow[u, mapsto, "\overline{\mathtt{prg}}_{\lambda,\lambda'}"] \arrow[r, mapsto, "\overline{\mathtt{tr}}_{\lambda}"] & (\Lambda_{\lambda} \arrow[r, "\overline{\mathtt{tr}}_{\lambda}(c)"'{name=D}] & \Lambda_{\lambda}) \arrow[r, mapsto, "\mathtt{tc}_{\lambda}"] & ((\wp(\Lambda_{\lambda}), \subseteq) \arrow[u, "p^{-1}_{\lambda',\lambda}"] & (\wp(\Lambda_{\gamma}), \subseteq)) \arrow[u, "p^{-1}_{\lambda',\lambda}"] \arrow[l, "\mathtt{wp}_{\lambda}(c)"{name=V}]
\arrow[from=D, to=U, mapsto, "\overline{\mathtt{pf}}_{\lambda,\lambda'}"]
\arrow[from=V, to=W, mapsto, "\mathtt{if}_{\lambda,\lambda'}"]
\end{tikzcd} \tag{13}$$

We defined the state transition semantics $\mathtt{tr}_{\gamma}(c) : \Sigma_{\gamma} \circ\!\!\!\rightarrow \Sigma_{\gamma}$, $\Sigma_{\gamma} = (\gamma \to \mathbb{D})$ 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 $[\![c]\!] : \Sigma \circ\!\!\!\rightarrow \Sigma$ for global states. For any inclusion function $in_{\gamma,\gamma'} : \gamma \to \gamma'$ we have $\mathtt{prg}(\gamma) \subseteq \mathtt{prg}(\gamma')$ and $\mathtt{pf}_{\gamma,\gamma'}(\mathtt{tr}_{\gamma}(c)) : \Sigma_{\gamma'} \circ\!\!\!\rightarrow \Sigma_{\gamma'}$ 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 $[\![c]\!] : \Sigma \circ\!\!\!\rightarrow \Sigma$ and since $[\![c]\!](\rho)(x) = \rho(x)$ for any global state $\rho \in \mathrm{DD}([\![c]\!])$ 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 $\overline{\mathtt{tr}}_{\lambda}(c) \triangleq \mathtt{tr}_{\gamma}(c) \times id_{\Gamma_{\delta}} : \Lambda_{\lambda} \circ\!\!\!\rightarrow \Lambda_{\lambda}$ and $\overline{\mathtt{pf}}_{\lambda,\lambda'}(\overline{\mathtt{tr}}_{\lambda}(c)) \triangleq \mathtt{pf}_{\gamma,\gamma'}(\mathtt{tr}_{\gamma}(c)) \times id_{\Gamma_{\delta'}} : \Lambda_{\lambda'} \circ\!\!\!\rightarrow \Lambda_{\lambda'}$ 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 $\mathtt{wp}$ 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.

$$\begin{tikzcd}[row sep=huge, column sep=huge]
\overline{\mathsf{Cont}} \arrow[d, bend right=70, "\mathtt{if}"'{name=I}] \arrow[d, "\overline{\mathtt{pf}}"{name=P}] \arrow[d, bend left=70, "\overline{\mathtt{prg}}"{name=R}] \\
\mathsf{Mon}
\arrow[from=P, to=I, Rightarrow, "\mathtt{pc}"', shorten=3pt]
\arrow[from=R, to=P, Rightarrow, "\overline{\mathtt{tr}}"', shorten=3pt]
\end{tikzcd}$$

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, $\mathtt{tc}$ and $\mathtt{pc}$ 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 $\mathtt{wlp}_\lambda(c)(\mathscr{Q}) = (\mathtt{tr}_\gamma(c) \times id_{\Gamma_\delta})^{-1}(\mathscr{Q}) \cup (\mathrm{DD}(\mathtt{tr}_\gamma(c)) \times \Gamma_\delta) \subseteq \Lambda_\lambda$.

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 $\lambda = (pvr(c) \cup pvr(Q), flv(Q))$.

Since our language of expressions is expressive enough, we can express weakest preconditions syntactically: 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 $\overline{\mathsf{Cont}}$.

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, i.e., 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 corresponding fibred acccount in the next section.

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 a finitary version of Hoare logic based on indexed categories. There are now, at least, three reasons to move from the indexed setting to the fibred one. First, the fibred 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 adequate.Third, the essential reason in the light of logic is, however, that we need a "technological space" where logical deduction can take place.

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 fibred 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 fibred 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 fibred category of state assertions and semantic entailments:

Definition 9 (Category Ent)**.** *The category* 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, i.e., since* $\mathtt{ent}$ *is a functor and since the* $\mathtt{ent}(\lambda) = (\mathtt{assn}(\lambda), \Vdash_\lambda)$ *are preorder categories.*

$$\begin{tikzcd}
\lambda'' & (\mathtt{assn}(\lambda''), \Vdash_{\lambda''}) & R \arrow[r, phantom, "\Vdash_{\lambda''}"] \arrow[dr, dashed, "(in_{\lambda',\lambda''}, \Vdash_{\lambda''})"'] & P \arrow[r, phantom, "\Vdash_{\lambda''}"] & Q \\
\lambda' \arrow[u, "in_{\lambda',\lambda''}"] & (\mathtt{assn}(\lambda'), \Vdash_{\lambda'}) \arrow[u, hook, "\mathtt{ent}(in_{\lambda',\lambda''})"] & & P \arrow[u, mapsto] \arrow[r, phantom, "\Vdash_{\lambda'}"] \arrow[dr, dashed, "(in_{\lambda,\lambda'}, \Vdash_{\lambda'})"'] & Q \arrow[u, mapsto] \\
\lambda \arrow[u, "in_{\lambda,\lambda'}"] & (\mathtt{assn}(\lambda), \Vdash_{\lambda}) \arrow[u, hook, "\mathtt{ent}(in_{\lambda,\lambda'})"] & & & Q \arrow[u, mapsto]
\end{tikzcd}$$

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 fibres* $\Pi^{-1}_{\mathsf{Ent}}(id_\lambda) \simeq \mathtt{ent}(\lambda)$.

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, i.e., 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 10. 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 context) and propositional reasoning in the fibres $\mathtt{ent}(\lambda) = (\mathtt{assn}(\lambda), \Vdash_\lambda) \simeq \Pi^{-1}_{\mathsf{Ent}}(id_\lambda)$ (compare Remark 6). 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 fibred category of state predicates and inclusions of state predicates:

Definition 11 (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 $\mathtt{pred}$ is a functor, i.e., 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.*

$$\begin{tikzcd}
\lambda'' & (\wp(\Lambda_{\lambda''}),\subseteq) \\
\lambda' \arrow[u, "in_{\lambda',\lambda''}"] & (\wp(\Lambda_{\lambda'}),\subseteq) \arrow[u, "\mathtt{pred}(in_{\lambda',\lambda''})", "=p^{-1}_{\lambda'',\lambda'}"'] \\
\lambda \arrow[u, "in_{\lambda,\lambda'}"] & (\wp(\Lambda_\lambda),\subseteq) \arrow[u, "\mathtt{pred}(in_{\lambda,\lambda'})", "=p^{-1}_{\lambda',\lambda}"']
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathscr{R} \arrow[dr, dashed, "(in_{\lambda',\lambda''},\subseteq)"'] & \subseteq p^{-1}_{\lambda'',\lambda'}(\mathscr{P}) & \subseteq & p^{-1}_{\lambda'',\lambda}(\mathscr{Q}) \\
& \mathscr{P} \arrow[u, mapsto, "p^{-1}_{\lambda'',\lambda'}"'] \arrow[drr, dashed, "(in_{\lambda,\lambda'},\subseteq)"'] & \subseteq & p^{-1}_{\lambda',\lambda}(\mathscr{Q}) \arrow[u, mapsto, "p^{-1}_{\lambda'',\lambda'}"'] \\
& & & \mathscr{Q} \arrow[u, mapsto, "p^{-1}_{\lambda',\lambda}"']
\end{tikzcd}$$

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 fibres $\Pi^{-1}_{\mathsf{Pred}}(id_\lambda)\simeq(\wp(\Lambda_\lambda),\subseteq)$.

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.

$$\begin{tikzcd}
& \mathsf{Pred} \arrow[dl, "\Pi_{\mathsf{Pred}}"] \\
\mathsf{Ent} \arrow[ur, "\mathtt{Sem}"] \arrow[d, "\Pi_{\mathsf{Ent}}"'] & \\
\overline{\mathsf{Cont}}^{op} &
\end{tikzcd}$$

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 $\mathtt{Sem};\Pi_{\mathsf{Pred}}=\Pi_{\mathsf{Ent}}$. $\mathtt{Sem}$ assigns to each local state assertion $(\lambda.Q)$ its local semantics $(\lambda.\mathtt{sem}_\lambda(Q))$ and to each entailment $(in_{\lambda,\lambda'},\Vdash_{\lambda'}):(\lambda'.P)\to(\lambda.Q)$, i.e., $P\Vdash_{\lambda'}Q$ the corresponding semantic inclusion $(in_{\lambda,\lambda'},\subseteq):(\lambda'.\mathtt{sem}_{\lambda'}(P))\to(\lambda.\mathtt{sem}_\lambda(Q))$, i.e., $\mathtt{sem}_{\lambda'}(P)\subseteq p^{-1}_{\lambda',\lambda}(\mathtt{sem}_\lambda(Q))$.

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 12 (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 11, e.g., we obtain the commutative diagram on the left 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 fibre $\Pi^{-1}_{\mathsf{Pred}}(id_{\lambda'})\simeq\mathtt{pred}(\lambda')$ and a Cartesian arrow.

$$\begin{tikzcd}
\mathscr{R} \arrow[r, "(id_{\lambda''},\subseteq)"] \arrow[dr, "(in_{\lambda',\lambda''},\subseteq)"'] & p^{-1}_{\lambda'',\lambda'}(\mathscr{P}) \arrow[r, "(id_{\lambda''},\subseteq)"] \arrow[d, "(in_{\lambda',\lambda''},=)"] & p^{-1}_{\lambda'',\lambda}(\mathscr{Q}) \arrow[d, "(in_{\lambda',\lambda''},=)"] \\
 & \mathscr{P} \arrow[r, "(id_{\lambda'},\subseteq)"] \arrow[dr, "(in_{\lambda,\lambda'},\subseteq)"'] & p^{-1}_{\lambda',\lambda}(\mathscr{Q}) \arrow[d, "(in_{\lambda,\lambda'},=)"] \\
 & & \mathscr{Q}
\end{tikzcd}$$

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 13 (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* λ *to* λ' *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* λ *is* $(id_\lambda,\varepsilon)=(in_{\lambda,\lambda},\varepsilon)$ *where* ε *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* $in_{\lambda,\lambda''}=in_{\lambda,\lambda'};in_{\lambda',\lambda''}$.

$$\begin{tikzcd}
\lambda'' & & \overline{\mathtt{prg}}(\lambda'') & & \lambda'' \arrow[r, "c_1"] & \lambda'' \arrow[r, "c_2"] & \lambda'' \\
\lambda' \arrow[u, "in_{\lambda',\lambda''}"] & & \overline{\mathtt{prg}}(\lambda') \arrow[u, hook] & & & \lambda' \arrow[u, mapsto, "in_{\lambda',\lambda''}"'] \arrow[ul, dashed, "(in_{\lambda',\lambda''},c_1)"] \arrow[r, "c_2"] & \lambda' \arrow[u, mapsto, "in_{\lambda',\lambda''}"'] \\
\lambda \arrow[u, "in_{\lambda,\lambda'}"] & & \overline{\mathtt{prg}}(\lambda) \arrow[u, hook] & & & & \lambda \arrow[u, mapsto, "in_{\lambda,\lambda'}"'] \arrow[ul, dashed, "(in_{\lambda,\lambda'},c_2)"]
\end{tikzcd}$$

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 fibres* $\Pi^{-1}_{\mathsf{Prg}}(id_\lambda)\simeq\mathtt{prg}(\lambda)^{op}$. □

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 to 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 λ in Prg the preorder category $\mathtt{Wp}(\lambda)\triangleq(\wp(\Lambda_\lambda),\subseteq)$ and to each morphism $(in_{\lambda,\lambda'},c):\lambda\to\lambda'$ in 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 $\mathsf{Pre}((\wp(\Lambda_\lambda),\subseteq),(\wp(\Lambda_\lambda),\subseteq))^{op}$, i.e., $\mathtt{wp}_\lambda(c_1;c_2)=\mathtt{wp}_\lambda(c_2);\mathtt{wp}_\lambda(c_1)$, and due to the results in Subsection 3.4 (see diagram (13)).

$$\begin{tikzcd}
\lambda'' \arrow[r, "c_1"] & \lambda'' \arrow[r, "c_2"] & \lambda'' \\
 & \lambda' \arrow[u, "|->", "in_{\lambda',\lambda''}"'] \arrow[u, phantom] \arrow[ul, dashed, "{(in_{\lambda',\lambda''},c_1)}"] \arrow[r, "c_2"] & \lambda' \arrow[u, "|->", "in_{\lambda',\lambda''}"'] \\
 & & \lambda \arrow[u, "|->", "in_{\lambda,\lambda'}"'] \arrow[ul, dashed, "{(in_{\lambda,\lambda'},c_2)}"]
\end{tikzcd}
\qquad
\begin{tikzcd}
(\wp(\Lambda_{\lambda''}),\subseteq) & (\wp(\Lambda_{\lambda''}),\subseteq) \arrow[l, "\mathtt{wp}_{\lambda''}(c_1)"'] & (\wp(\Lambda_{\lambda''}),\subseteq) \arrow[l, "\mathtt{wp}_{\lambda''}(c_2)"'] \\
 & (\wp(\Lambda_{\lambda'}),\subseteq) \arrow[u, "p^{-1}_{\lambda'',\lambda'}"'] \arrow[ul, dashed, "{\mathtt{Wp}(in_{\lambda',\lambda''},c_1)}"] & (\wp(\Lambda_{\lambda'}),\subseteq) \arrow[u, "p^{-1}_{\lambda'',\lambda'}"'] \arrow[l, "\mathtt{wp}_{\lambda'}(c_2)"'] \\
 & & (\wp(\Lambda_\lambda),\subseteq) \arrow[u, "p^{-1}_{\lambda',\lambda}"'] \arrow[ul, dashed, "{\mathtt{Wp}(in_{\lambda,\lambda'},c_2)}"]
\end{tikzcd}$$

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

Definition 14 (Category Wp)**.** *The category* 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* 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, i.e., 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.*

$$\begin{tikzcd}
\lambda'' \\
\lambda' \arrow[u, "|->", "{(in_{\lambda',\lambda''},c_1)}"] \\
\lambda \arrow[u, "|->", "{(in_{\lambda,\lambda'},c_2)}"]
\end{tikzcd}
\qquad
\begin{tikzcd}
(\wp(\Lambda_{\lambda''}),\subseteq) \\
(\wp(\Lambda_{\lambda'}),\subseteq) \arrow[u, "|->", "{\mathtt{Wp}(in_{\lambda',\lambda''},c_1)}"] \\
(\wp(\Lambda_\lambda),\subseteq) \arrow[u, "|->", "{\mathtt{Wp}(in_{\lambda,\lambda'},c_2)}"]
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathscr{R} \arrow[drr, dashed, "{((in_{\lambda',\lambda''},c_1),\subseteq)}"'] & \subseteq & \mathtt{wp}_{\lambda''}(c_1)(p^{-1}_{\lambda'',\lambda'}(\mathscr{P})) & \subseteq & \mathtt{wp}_{\lambda''}(c_1;c_2)(p^{-1}_{\lambda'',\lambda}(\mathscr{Q})) \\
 & & \mathscr{P} \arrow[u, "|->", "{\mathtt{Wp}(in_{\lambda',\lambda''},c_1)}"'] \arrow[drr, dashed, "{((in_{\lambda,\lambda'},c_2),\subseteq)}"'] & \subseteq & \mathtt{wp}_{\lambda'}(c_2)(p^{-1}_{\lambda',\lambda}(\mathscr{Q})) \arrow[u, "|->", "{\mathtt{Wp}(in_{\lambda',\lambda''},c_1)}"'] \\
 & & & & \mathscr{Q} \arrow[u, "|->", "{\mathtt{Wp}(in_{\lambda,\lambda'},c_2)}"']
\end{tikzcd}$$

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

The general properties of Grothendieck constructions provide:

Theorem 5. The functor $\mathit{\Pi}_{\mathsf{Wp}}:\mathsf{Wp}\to\mathsf{Prg}^{op}$ 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 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, i.e., 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 fibres $\Pi^{-1}_{\mathsf{Pred}}(id_{\lambda}) \simeq (\wp(\Lambda_{\lambda}), \subseteq)$ and $\Pi^{-1}_{\mathsf{Wp}}(\mathtt{E}^{op}_{cont}(id_{\lambda})) = \Pi^{-1}_{\mathsf{Wp}}((id_{\lambda}, \varepsilon))$.

$$\begin{tikzcd}
\mathsf{Pred} \arrow[r, "\mathtt{E}_{pred}"] \arrow[d, "\Pi_{\mathsf{Pred}}"'] & \mathsf{Wp} \arrow[d, "\Pi_{\mathsf{Wp}}"] \\
\overline{\mathsf{Cont}}^{op} \arrow[r, "\mathtt{E}^{op}_{cont}"] & \mathsf{Prg}^{op}
\end{tikzcd}$$

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 fibred setting:

$$\begin{tikzcd}
 & \mathsf{Pred} \arrow[rr, "\mathtt{E}_{pred}"] \arrow[ddl, "\Pi_{\mathsf{Pred}}"] & & \mathsf{Wp} \arrow[ddl, "\Pi_{\mathsf{Wp}}"] \\
\mathsf{Ent} \arrow[ur, "\mathtt{Sem}"] \arrow[rr, dashed, "\mathtt{E}_{ent}"] \arrow[d, "\Pi_{\mathsf{Ent}}"'] & & \mathsf{TC} \arrow[ur, dashed, "\mathtt{TSem}"] \arrow[d, dashed, "\Pi_{\mathsf{TC}}"'] & \\
\overline{\mathsf{Cont}}^{op} \arrow[rr, "\mathtt{E}^{op}_{cont}"] & & \mathsf{Prg}^{op} &
\end{tikzcd}$$

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 TC of total correctness assertions about local programs by extending, step by step, the category Ent of local state assertions. In parallel three new functors should be constructed connecting the new category 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 $\Pi_{\mathsf{TC}} = \mathtt{TSem}; \Pi_{\mathsf{Pred}}$, means that the calculus is sound. If $\mathtt{TSem}$ is, in addition, full, the calculus is also complete.
(2) **Conservative extension:** The functor $\mathtt{E}_{ent} : \mathsf{Ent} \to \mathsf{TC}$ should be an embedding such that the top and the front face commute, i.e., $\mathtt{E}_{ent}; \mathtt{TSem} = \mathtt{Sem}; \mathtt{E}_{pred}$ and $\mathtt{E}_{ent}; \Pi_{\mathsf{TC}} = \Pi_{\mathsf{Ent}}; \mathtt{E}^{o}_{cont}p$. 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 TC comprises all deducible correctness assertions.

The existence of syntactic weakest preconditions allows us, due to Corollary 1, to describe the category TC of total correctness assertions independent of a concrete deduction calculus. Analogously to the construction of Wp, we need, however, some preparations. For any morphism $(in_{\lambda,\lambda'}, c) : \lambda \to \lambda'$ a in 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 logical 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 monoton w.r.t. semantic entailment, i.e., $P \Vdash_{\lambda} Q$ 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, i.e., 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 15 (Category TC)**.** *The category* 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* 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.*

$$\begin{tikzcd}
\lambda'' & (\mathtt{assn}(\lambda''), \Vdash_{\lambda''}) & R \arrow[rrd, dashed, "{((in_{\lambda',\lambda''},c_1),\Vdash_{\lambda''})}"'] & \Vdash_{\lambda''} & wp(c_1, P) & \Vdash_{\lambda''} & wp(c_1; c_2, Q) \\
\lambda' \arrow[u, "{(in_{\lambda',\lambda''},c_1)}" description] & (\mathtt{assn}(\lambda'), \Vdash_{\lambda'}) \arrow[u, "{wp(in_{\lambda',\lambda''},c_1)}" description] & & & P \arrow[u, mapsto, "{wp(in_{\lambda',\lambda''},c_1)}"'] \arrow[rrd, dashed, "{((in_{\lambda,\lambda'},c_2),\Vdash_{\lambda'})}"'] & \Vdash_{\lambda'} & wp(c_2, Q) \arrow[u, mapsto, "{wp(in_{\lambda',\lambda''},c_1)}"'] \\
\lambda \arrow[u, "{(in_{\lambda,\lambda'},c_2)}" description] & (\mathtt{assn}(\lambda), \Vdash_{\lambda}) \arrow[u, "{wp(in_{\lambda,\lambda'},c_2)}" description] & & & & & Q \arrow[u, mapsto, "{wp(in_{\lambda,\lambda'},c_2)}"']
\end{tikzcd}$$

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 fibres* $\Pi_{\mathsf{TC}}^{-1}((id_{\lambda}, \varepsilon)) \simeq \mathtt{ent}(\lambda)$.

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 $\mathtt{sem}_{\lambda'}(P) \subseteq \mathtt{wp}_{\lambda'}(c)(p_{\lambda',\lambda}^{-1}(\mathtt{sem}_{\lambda}(Q)))$. Due to Proposition 7 as well as the definition and context independence of syntactic weakest preconditions, we have $\mathtt{wp}_{\lambda'}(c)(p_{\lambda',\lambda}^{-1}(\mathtt{sem}_{\lambda}(Q))) = \mathtt{wp}_{\lambda'}(c)(\mathtt{sem}_{\lambda'}(Q)) = \mathtt{sem}_{\lambda'}(wp(c, Q))$ 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 $\mathtt{E}_{ent} : \mathsf{Ent} \to \mathsf{TC}$ 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, i.e., we have $\mathtt{E}_{ent}; \Pi_{\mathsf{TC}} = \Pi_{\mathsf{Ent}}; \mathtt{E}_{cont}^{op}$. Moreover, it is a pullback square since $\mathtt{E}_{ent}$ establishes isomorphisms between the fibres $\Pi_{\mathsf{Ent}}^{-1}(id_{\lambda}) \simeq \mathtt{ent}_{\lambda}$ and $\Pi_{\mathsf{TC}}^{-1}(\mathtt{E}_{cont}^{op}(id_{\lambda})) = \Pi_{\mathsf{TC}}^{-1}((id_{\lambda}, \varepsilon))$.

We know that $\mathtt{wp}_{\lambda'}(\varepsilon)$ is the identity on $(\wp(\Lambda_{\lambda'}), \subseteq)$ thus the commutativity $\mathtt{Sem}; \mathtt{E}_{pred} = \mathtt{E}_{ent}; \mathtt{TSem}$ 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 Prg^{op} and all objects $(\lambda.Q)$ in TC.

$$\begin{tikzcd}
\lambda' & \mathtt{ent}(\lambda') & P & \Vdash_{\lambda'} \arrow[d, phantom] & wp(c,Q) & \Vdash_{\lambda''} & wp(c,Q) \\
\lambda \arrow[u, "(in_{\lambda,\lambda'}{,}\varepsilon)" description] & \mathtt{ent}(\lambda) \arrow[u, hook, "wp(in_{\lambda,\lambda'}{,}\varepsilon)" description] & & & wp(c,Q) \arrow[u, mapsto, "wp(in_{\lambda,\lambda'}{,}\varepsilon)"'] \arrow[from=1-3, dashed, "((in_{\lambda,\lambda'}{,}\varepsilon){,}\Vdash_{\lambda'})"'] \arrow[drr, dashed, "((id_{\lambda}{,}c){,}\Vdash_{\lambda})"'] & \Vdash_{\lambda} & wp(c,Q) \arrow[u, mapsto, "wp(in_{\lambda,\lambda'}{,}\varepsilon)"'] \\
\lambda \arrow[u, "(id_{\lambda}{,}c)" description] & \mathtt{ent}(\lambda) \arrow[u, "wp(id_{\lambda}{,}c)" description] & & & & & Q \arrow[u, mapsto, "wp(id_{\lambda}{,}c)"']
\end{tikzcd}$$

Instantiating Remark 12, 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).

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. 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 fibred 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 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 fibred presentation, we have been able to formalize precisely the intuition that Hoare triples are a kind of fibred entity, i.e., Hoare triples arise naturally as special arrows in a fibred 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, 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.

Competing interests: The authors declare none.

References

Apt, K. R., **de Boer, F. S.**, **and Olderog, E.** 2009. *Verification of Sequential and Concurrent Programs*. Texts in Computer Science. Springer.

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

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

Diaconescu, R. 2008. *Institution-Independent Model Theory*. Birkhäuser Basel, 1st edition.

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

Goguen, J. A. and Burstall, R. M. 1992. Institutions: Abstract Model Theory for Specification and Programming. *Journal of the ACM*, 39(1):95–146.

Huth, M. and Ryan, M. D. 2004. *Logic in computer science - modelling and reasoning about systems (2. ed.)*. Cambridge University Press.

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*, Volume 4, Issue 02:117–146.

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

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

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.**, editors, *COMPASS/ADT*, volume 1130 of *Lecture Notes in Computer Science*, pp. 436–457. Springer.

Pierce, B. C., **de Amorim, A. A.**, **Casinghino, C.**, **Gaboardi, M.**, **Greenberg, M.**, **Hriţcu, C.**, **Sjöberg, V.**, **Tolmach, A.**, **and Yorgey, B.** 2018. *Programming Language Foundations*. Software Foundations series, volume 2. Electronic textbook.

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.

Extensions w.r.t. the Workshop Submission

This submission is not only an extension of the workshop publication, but also a substantial rewriting and revision.

Background Material

- The imperative language was augmented with array variables so that we included examples of Hoare triples for programs that perform computations over an array of integers, e.g, linear search and insertion sort.
- The traditional global semantics was defined this time with structural operational semantics instead of denotational semantics. This change provides a more straightforward definition of the semantics of a program as a partial function instead of continuous functions between CPO's from states to states. This also improved the readability of the paper, since with partial functions, we have a simple mathematical tool to express non-termination.
- In this work, we also cover not only partial correctness, but also total correctness. The use of partial functions support straightforward and direct definitions of both concepts.

Categorical Development - Indexed Categories

- The material of Hoare logic from the point of view of Indexed Categories was rewritten from scratch. In the workshop submission, we dealt only with partial correctness assertions with finite contexts of logical variables. Now we deal with finite contexts of both program and logical variables. An entire new set of tools and techniques had to be developed to deal with these extended contexts.
- On top of these constructions, we developed a logic of local state assertions, which is based 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.
- Based on a abstract theory of partial transition maps, we have developed a new reformulation of Dijkstra's theory of predicate transformers.
- As a new result we showed that partial correctness semantics and total correctness semantics are structurally equivalent.

Categorical Development - Fibred Categories

- The methodology and style is analogous to the workshop submission.
- We have again used systematically Grothendieck constructions in order to construct a collection of fibrations.
- These fibrations are all new, since we are dealing now with extended contexts of both program and logical variables.
- To deal with extended contexts of both program and logical variables we had to utilize, however, constructions that go beyond the traditional Grothendieck constructions.