Available online at www.sciencedirect.com

ScienceDirect

ELSEVIER

Electronic Notes in Theoretical Computer Science 348 (2020) 125–145

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Indexed and Fibred Structures for Hoare Logic

U. E. Wolter[1]

Department of Informatics - University of Bergen - Bergen - Norway

A. R. Martini [2]

Av. Marechal Andrea 11/210 - Porto Alegre - Brazil

E. H. Häusler[3]

Departamento de Ciência da Computação - PUC-Rio - Rio de Janeiro - Brazil

Abstract

Indexed and fibred categorical concepts are widely used in computer science as models of logical systems and type theories. Here we focus on Hoare logic and show that a comprehensive categorical analysis of its axiomatic semantics needs the languages of indexed category and fibred category theory. The structural features of the language are presented in an indexed setting, while the logical features of deduction are modeled in the fibred one. Especially, Hoare triples arise naturally as special arrows in a fibred category over a syntactic category of programs, while deduction in the Hoare calculus can be characterized categorically by the heuristic *deduction = generation of cartesian arrows + composition of arrows.*

Keywords: Hoare logic, weakest liberal precondition, indexed categories, institution, fibrations, Grothendieck construction

1 Introduction

A logic for reasoning about programs has to rely on a logic for reasoning about states and other specific concepts, such as environments and memory locations. Besides that, everything has to fit compositionally in a way that deduction on properties on states can be directly used on deduction on statements that must be combined in order to derive properties on programs. Each constituent at the syntactic level

[1] Email: Uwe.Wolter@uib.no

[2] Email: alfio.martini@gmail.com

[3] Email: hermann@inf.puc-rio

https://doi.org/10.1016/j.entcs.2020.02.008

1571-0661/© 2020 The Author(s). Published by Elsevier B.V.

of the language should correspond to a semantic concept and this correspondence has to preserve structure and logic.

It is known that functors naturally preserve the structure, but they do not have to preserve logic properties. Usually, if one wants to preserve logical properties, a certain notion of universality has to be used. A widely used concept for providing this notion of universality is the concept of *cartesian arrow*. Although cartesian arrows are embedded in the definition of fibred categories, any indexed category also provides these cartesian arrows as well, albeit with more structural information [1,4].

The interplay between fibred and indexed constructions, we will rely on in this paper, is quite well-known for the Category and the Logic community. Indexed families of sets ($\{A_i\}_{i \in I}$) and display mappings ($d : \biguplus_{i \in I} A_i \to I$) can be considered as the motivating set-theoretical concepts for their categorical counterparts, indexed and fibred categories (fibrations), respectively. In set theory, these counterparts are logically equivalent, but fibred and indexed categories are formally equivalent only when they are taken from a 2-category point of view (see, for instance, [13]).

The aim of the paper 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 [8], institutions [6], and context institutions [11]. 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.

Hoare logic [7] is presented in the literature in quite diverse ways. The weakest liberal precondition [3], for example, can be defined in a semantic way by predicate transformers or in a syntactic way. The semantics of programs can be defined by partial state transition functions or by predicate transformers. As the *main result* of the paper we show how all these diverse and divergent features of Hoare logic can be described in a smooth, systematic and uniform way using the language of indexed categories and the language of fibred categories. Our analysis shows that a highly appropriate way to deal with axiomatic semantics, at least in the format of Hoare logic, is to use both categorical tools in a way that the structural features of the language are presented in an indexed setting and the features of deduction in the fibred one. This underlines, especially, that a good understanding of the correlations between indexed and fibred concepts can be very helpful in order to present, analyze and use these logics in an appropriate way.

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 from domain theory, the **IMP** language, and Hoare Logic. We also provide a list with the essential categories used throughout the text. Section 3 analyzes the structural features of the Hoare logic by means of indexed concepts. Especially, we are concerned about a categorical account of the weakest liberal precondition. In Section 4 we transform, by means of the Grothendieck construction, the indexed account into a fibred one 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 Preliminary Concepts

The material in this section is, for the most part, well-known and it is included here in order to fix notation and to improve readability of the paper.

2.1 A Small Imperative Language

The **IMP** language is a simple imperative language with only three data types. 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 $m, n, \ldots$ The countably infinite set **Loc** of memory locations, ranged over by metavariables $x, y, \ldots$ We use the terms locations, variables and identifier interchangeably. Also, we consider programs that use a finite number of locations and we assume there are enough locations available for any program. The grammar for **IMP** comprises three syntactic categories. **AExp**, arithmetic expressions, ranged over by $a, a', \ldots$ **BExp**, boolean expressions, ranged over by $b, b', \ldots$ **Prg**, programs, ranged over by $c, c', \ldots$ The following productions define the abstract syntax of **IMP** :

$$\begin{aligned}
a \in \mathbf{AExp} &::= n \mid x \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \times a_1 \\
b \in \mathbf{BExp} &::= v \mid a_0 = a_1 \mid a_0 \leq a_1 \mid \neg b \mid b_0 \vee b_1 \mid b_0 \wedge b_1 \\
c \in \mathbf{Prg} &::= \underline{\mathbf{skip}} \mid x := a \mid c_0; c_1 \mid \underline{\mathbf{if}}\ b\ \underline{\mathbf{then}}\ c_0\ \underline{\mathbf{else}}\ c_1\ \underline{\mathbf{fi}} \mid \underline{\mathbf{while}}\ b\ \underline{\mathbf{do}}\ c\ \underline{\mathbf{od}}
\end{aligned}$$

In order to evaluate an expression or to define the execution of a command, we need the notion of a *memory state*. A memory state σ is an element of the set Σ, which contains all functions from locations to integers: $\sigma \in \Sigma = (\mathbf{Loc} \to \mathbb{Z})$. Given a state σ, we denote by $\sigma[x \mapsto n]$ the memory where the value of x is updated to n, i.e.,

$$\sigma[x \mapsto n](y) = \begin{cases} n & \text{if } y = x \\ \sigma(y) & \text{if } y \neq x \end{cases}$$

In order to define the denotational semantics of expressions and programs, we use the metalanguage of the lambda calculus enriched with boolean and arithmetic constants (operations). Besides lambda abstraction and function application, we have the conditional operator $t \to t_0, t_1$, which returns t_0 if $t = true$ and t_1, if $t = false$. The interpretation functions are defined by structural recursion over the abstract syntax for expressions and programs. For any set D, we have the flat CPO $D_\perp \triangleq D \cup \{\perp\}$ (where $\perp \notin D$) such that $d \sqsubseteq_{D_\perp} d'$ iff $d = d'$ or $d = \perp$. The element $\perp_D$ represents undefinedness or a divergence in a computation. Given a function $f : D \to E_\perp$, we define its lifting $f^* : D_\perp \to E_\perp$, such that $f^*(d) = f(d)$ for all $d \in D$ and $f^*(\perp_D) = \perp_E$. Every function in $(D_\perp \to E_\perp)$ and $(D \to E_\perp)$ is continuous. We take for granted the standard denotation functions $\mathsf{E}[\![\cdot]\!] : \mathbf{AExp} \to \Sigma \to \mathbb{Z}$ and $\mathsf{B}[\![\cdot]\!] : \mathbf{BExp} \to \Sigma \to \mathbb{B}$ and assume they are defined elsewhere.

$$\mathsf{P}[\![\cdot]\!] : \mathbf{Prg} \to \Sigma \to \Sigma_\perp$$

$$\mathsf{P}[\![x := a]\!] = \lambda\sigma.\ \sigma[x \mapsto \mathsf{E}[\![a]\!]\sigma]$$

$$\mathsf{P}[\![\ \underline{\mathbf{skip}}\ ]\!] = \lambda\sigma.\ \sigma$$

$$\mathsf{P}[\![c_0; c_1]\!] = \lambda\sigma.\ (\mathsf{P}[\![c_1]\!]^*(\mathsf{P}[\![c_0]\!]\sigma)$$

$$\mathsf{P}[\![\underline{\mathbf{if}}\ b\ \underline{\mathbf{then}}\ c_0\ \underline{\mathbf{else}}\ c_1\ \underline{\mathbf{fi}}]\!] = \lambda\sigma.\ \mathsf{B}[\![b]\!]\sigma \to \mathsf{P}[\![c_0]\!]\sigma, \mathsf{P}[\![c_1]\!]\sigma$$

$$\mathsf{P}[\![\underline{\mathbf{while}}\ b\ \underline{\mathbf{do}}\ c\ \underline{\mathbf{od}}]\!] = \mathit{fix}(\Gamma_{b,c}) = \textstyle\bigsqcup_{n \geq 0} \Gamma^n_{b,c}(\perp_{\Sigma \to \Sigma_\perp}), \quad \text{where}$$

$$\Gamma_{b,c} : (\Sigma \to \Sigma_\perp) \to (\Sigma \to \Sigma_\perp) = \lambda\varphi.\ \lambda\sigma.\ \mathsf{B}[\![b]\!]\sigma \to \varphi^*(\mathsf{P}[\![c]\!]\sigma), \sigma$$

2.2 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 triple describes how the execution of a piece of code changes the state of the computation. A Hoare triple 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. An informal understanding about the meaning of a Hoare triple can be given as follows: *any terminating execution of c from a state satisfying P ends up in a state satisfying Q.*

Our specification logic is built of boolean expressions which are enriched with quantifiers and a countably infinite set of integer (logical) variables **IVar**. The extended language of arithmetic expressions **AExpV** and the specification language of program assertions **Assn** are defined by the following abstract syntax, where $i, j, \ldots$ range over integer variables, $P, Q, R, \ldots$ over program assertions, $aop \in \{+, -, \times\}$, $bop \in \{\wedge, \vee, \to\}$.

$$a \in \mathbf{AExpV} ::= n \mid x \mid i \mid a_0\ aop\ a_1$$

$$P \in \mathbf{Assn} \quad ::= a_0 \leq a_1 \mid a_0 = a_1 \mid \neg P \mid P_0\ bop\ P_1 \mid \forall i.\ P \mid \exists i.\ P$$

The set of free variables in arithmetic expressions is comprised by the integer variables that occur in it. Free variables in program assertions are comprised by all integer variables which are not in the scope of existential and universal quantifiers, as presented in any introduction to first order logic. Both sets of free variables (FV) are easily defined by structural induction. For instance, $FV(x = i \wedge y = j) = \{i, j\}$

and $FV(\forall i.\ \forall j.\ (i \geq 0 \wedge j > 0 \to i = xb + y \wedge 0 \leq y < j)) = \emptyset$, assuming x, y are locations.

An environment for the (free) integer variables in an expression is a function $env : \mathbf{IVar} \to \mathbb{Z}$. The set of all such enviroments is $Env = (\mathbf{IVar} \to \mathbb{Z})$. The extended interpretation function $\mathsf{E}[\![\cdot]\!] : \mathbf{AExpV} \to Env \to \Sigma \to \mathbb{Z}$ receives now an additional argument. The definition is analogous to the case of arithmetic expressions $\mathbf{AExp}$. The only interesting cases are $\mathsf{E}[\![i]\!]\ env\ \sigma = env(i)$ and $\mathsf{E}[\![x]\!]\ env\ \sigma = \sigma(x)$.

In order to define satisfaction and validity of Hoare triples, we need an analogous definition for program assertions, i.e., we have to define $\sigma \models_{env} P$ for every program assertion $P \in \mathbf{Assn}$, arbitrary environments $env : \mathbf{IVar} \to \mathbb{Z}$ and states $\sigma \in \Sigma_\perp$. The definition is standard and is done by structural induction on the structure of P as is usually done in logic. We assume that $\perp \models_{env} P$.

A program assertion P is valid in an environment $env : \mathbf{IVar} \to \mathbb{Z}$, written $\models_{env} P$, iff $\forall \sigma \in \Sigma_\perp.\ \sigma \models_{env} P$. A program assertion P is called (arithmetic) *valid*, written $\models P$, iff $\forall\ env : \mathbf{IVar} \to \mathbb{Z}.\ \models_{env} P$. We say that Q is a logical consequence of P in an environment $env : \mathbf{IVar} \to \mathbb{Z}$, written $P \models_{env} Q$, iff $\forall \sigma \in \Sigma_\perp.\ \sigma \models_{env} P \to \sigma \models_{env} Q$. Moreover, we say that Q is a logical consequence of P, written $P \models Q$ iff $\forall \sigma \in \Sigma_\perp.\ \forall\ env : \mathbf{IVar} \to \mathbb{Z}.\ \sigma \models_{env} P \to \sigma \models_{env} Q$.

A categorical treatment of Hoare Logic needs more precise definitions of Hoare Logic concepts. We say that a triple $\{P\}\ c\ \{Q\}$ is true at a state $\sigma \in \Sigma_\perp$ and in an $env : \mathbf{IVar} \to \mathbb{Z}$, written $\sigma \models_{env} \{P\}\ c\ \{Q\}$ iff $\sigma \models_{env} P$ implies $\mathsf{P}[\![c]\!]^*\sigma \models Q$. The triple is valid in an environment $env : \mathbf{IVar} \to \mathbb{Z}$, written $\models_{env} \{P\}\ c\ \{Q\}$ iff $\forall \sigma \in \Sigma_\perp.\ \sigma \models_{env} \{P\}\ c\ \{Q\}$. Finally, a partial correctness assertion is (arithmetic) *valid*, written $\models \{P\}\ c\ \{Q\}$, iff $\forall\ env : \mathbf{IVar} \to \mathbb{Z}.\ \models_{env} \{P\}\ c\ \{Q\}$.

Let $env : \mathbf{IVar} \to \mathbb{Z}$ be an environment and P an assertion. Then the *extension of the assertion* P w.r.t. env, written as P^{env} is defined as $P^{env} \triangleq \{\sigma \in \Sigma_\perp \mid \sigma \models_{env} P\}$. Then it follows that $P \models Q$ iff $\forall\ env : \mathbf{IVar} \to \mathbb{Z}.\ P^{env} \subseteq Q^{env}$. By the same token, we have $\models \{P\}\ c\ \{Q\}$ iff $\forall\ env : \mathbf{IVar} \to \mathbb{Z}.\ \mathsf{P}[\![c]\!]^*(P^{env}) \subseteq Q^{env}$.

Let $c \in \mathbf{Prg}, Q \in \mathbf{Assn}$ and $env : \mathbf{IVar} \to \mathbb{Z}$. Then the *semantic weakest liberal precondition*, written $\mathsf{wlp}^{env}[\![c, Q]\!]$, of Q with respect to c in env is defined by $\mathsf{wlp}^{env}[\![c, Q]\!] \triangleq \{\sigma \in \Sigma_\perp \mid \mathsf{P}[\![c]\!]^*\sigma \models_{env} Q\} = (\mathsf{P}[\![c]\!]^*)^{-1}(Q^{env})$. Thus we have $\models \{P\}\ c\ \{Q\}$ iff $\forall\ env : \mathbf{IVar} \to \mathbb{Z}.\ P^{env} \subseteq \mathsf{wlp}^{env}[\![c, Q]\!]$.

The following rules of the *Hoare Proof Calculus* define inductively the partial correctness relation $\vdash \subseteq \mathbf{Assn} \times \mathbf{Prg} \times \mathbf{Assn}$, where the expression $Q[x/a]$ means the simultaneous replacement of every ocurrence of the location x in the assertion Q by the arithmetic expression a.

$$\frac{}{\vdash \{P\}\underline{\mathbf{skip}}\{P\}}\ \mathsf{Skip} \qquad \frac{}{\vdash \{Q[x/a]\}x := a\{Q\}}\ \mathsf{Assn} \qquad \frac{\vdash \{P\}c_1\{Q\} \quad \vdash \{Q\}c_2\{R\}}{\vdash \{P\}c_1; c_2\{R\}}\ \mathsf{Comp}$$

$$\frac{\vdash \{P \wedge b\}c_1\{Q\} \quad \vdash \{P \wedge \neg b\}c_2\{Q\}}{\vdash \{P\}\ \underline{\mathbf{if}}\ b\ \underline{\mathbf{then}}\{c_1\}\ \underline{\mathbf{else}}\ \{c_2\}\ \underline{\mathbf{fi}}\ \{Q\}}\ \mathsf{IfElse} \qquad \frac{\vdash \{P \wedge b\}c\{P\}}{\vdash \{P\}\ \underline{\mathbf{while}}\ b\ \underline{\mathbf{do}}\ c\ \underline{\mathbf{od}}\ \{P \wedge \neg b\}}\ \mathsf{PWhile}$$

$$\frac{\vdash P \to Q \quad \vdash \{Q\}c\{R\}}{\vdash \{P\}c\{R\}}\ \mathsf{PreStr} \qquad \frac{\vdash \{P\}c\{Q\} \quad \vdash Q \to R}{\vdash \{P\}c\{R\}}\ \mathsf{PosWk}$$

Let $\{P\}\ c\ \{Q\}$ be a partial correctness assertion. Then the Hoare calculus is sound, i.e., every theorem is a valid formula.

$$\vdash \{P\}\ c\ \{Q\} \quad \text{only if} \quad \models \{P\}\ c\ \{Q\}$$

For each program c and assertion Q we assume there is an assertion $wlp(c, Q) \in$ **Assn**. Intuitively, it is the weakest precondition that ensures that if the execution of c terminates, then Q holds in the final state, i.e., with the property that $\models \{wlp(c, Q)\}\ c\ \{Q\}$. We also assume that for any $env : \mathbf{IVar} \to \mathbb{Z}$ that the program assertion language is expressive enough, i.e.,

$$wlp(c, Q)^{env} = \mathsf{wlp}^{env}[\![c, Q]\!] \tag{1}$$

2.3 Categories

The reader is assumed to be familiar with the basic notions of category theory, such as adjunctions, fibrations and indexed categories [1]. To improve readability, Table 1 lists all categories introduced and used in the paper.

Table 1
Categories used and introduced in the paper

Category	**Objects**	**Morphisms**
Standard categories		
Cpo	complete partial orders	continuous functions
Set	sets	total functions
Po	partial orders	monotone functions
Cat	small categories	functors
Pre	preorders	monotone functions
Categories for Hoare logic		
Str	components	sub-component relationships
Prg	components	sub-component relationships and programs
Cont	contexts γ (finite subsets of logical variables)	context inclusions
Assn	pairs $(\gamma.P)$ with P an assertion about states in context γ	pairs of a context inclusion and a semantic entailment between state assertions
Pca	pairs $(\gamma.P)$ with P an assertion about states in context γ	partial correctness assertions (triples of a program, a context inclusion and a semantic entailment between state assertions)
St	pairs $(\gamma.\mathcal{E})$ with $\mathcal{E}$ a predicate, i.e., a set $\mathcal{E} \subseteq \Sigma_\perp \times env(\gamma)$ of states	predicate inclusions (pairs of a context inclusion and an inclusion between a predicate and an extended predicate)
Sem	pairs $(\gamma.\mathcal{E})$ with $\mathcal{E}$ a predicate, i.e., a set $\mathcal{E} \subseteq \Sigma_\perp \times env(\gamma)$ of states	partial correctness relationships (triples of a program, a context inclusion and an inclusion between a predicate and a transformed predicate)

3 Hoare Logic and Indexed Categories

As exemplified in the last section, Hoare Logic addresses state based systems and is concerned with the behavior of programs. In our example of a sequential imperative programming language, we are dealing with systems with a single component system, i.e., a single program or algorithm. In general, however we may consider also systems constituted by different components.

In the following, we analyze Hoare Logic with both indexed and fibred categorical structures as developed, for example, in [10]. We intend to give an answer to the fundamental question "What are the characteristic structural features of Hoare Logic?" and use these structures to give a categorical presentation for it.

In the present section we will lift this insight onto a more abstract and structured level. We will develop, step by step, an abstract categorical presentation of the Hoare logic of **IMP** by means of indexed categorical concepts and constructions. Hoare Logic arises in two steps: First, a "logic of states" is defined and, based on this, a "logic of programs" is developed.

3.1 Logic of States

In this subsection we analyze the concept of states in **IMP** as well as the language and semantics of assertions about states. Parallel to this analysis we develop a general structure of a "logic of states".

System structure and States: In the simple language **IMP**, we can describe programs on a single "machine". In our categorical presentation we will use the identifier C to denote this unique component. The semantics of a system component is given by its states. In case of **IMP**, the semantics of the unique component C is the cpo $\Sigma_\bot = (\mathbf{Loc} \to \mathbb{Z})_\bot$. If we consider now the very simple category Str consisting only of the object C and its identity we can state that the definition of the concept of a state for **IMP** can be captured by the definition of a functor $st : \mathsf{Str}^{op} \to \mathsf{Cpo}$ with $st(C) \triangleq \Sigma_\bot = (\mathbf{Loc} \to \mathbb{Z})_\bot$, where Cpo is the category with complete partial orders as objects and continuous functions as morphisms.

Note that Str has only one object since we consider, following the traditional exposition, programs only in the context of all program variables (locations). In case we would consider programs in the context of finite subsets of **Loc**, the category Str^{op} would turn into a finite product category ([12]).

Contexts and Environments: Any categorical analysis and presentation of logics emphasizes the importance of local contexts for expressions and formulas. In our case, this "categorical imperative" suggests that it is not fully appropriate to take the countably infinite set **IVar** as the only (default) context for all assertions about states. Instead, it is more appropriate to work with finite subsets of **IVar** as "local contexts". Following this insight, contexts in our categorical presentation of **IMP** are given by the (syntactic) partial order category $\mathsf{Cont} \triangleq (\wp_{fin}(\mathbf{IVar}), \subseteq)$.

The semantics of contexts is given by environments, i.e., by a functor env : $\mathsf{Cont}^{op} \to \mathsf{Set}$ with $env(\gamma) \triangleq (\gamma \to \mathbb{Z})$ for all contexts $\gamma \in \wp_{fin}(\mathbf{IVar})$ and with

$env(\gamma \subseteq \gamma') = p_{\gamma',\gamma} : (\gamma' \to \mathbb{Z}) \to (\gamma \to \mathbb{Z})$ a *projection map* given by pre-composition $p_{\gamma',\gamma}(e') \triangleq in_{\gamma,\gamma'}; e'$ for all environments $e' \in (\gamma' \to \mathbb{Z})$ where $in_{\gamma,\gamma'} : \gamma \to \gamma'$ is the inclusion map corresponding to the inclusion $\gamma \subseteq \gamma'$.

Assertions: In case of **IMP**, all assertions are statements about the states of the unique system component, and their definition is based on locations for this single component. In our categorical analysis, we abstract from locations and define

$assn(\gamma) \triangleq \{P \in \mathbf{Assn} \mid free(P) \subseteq \gamma\}, \quad$ for all $\gamma \in \wp_{fin}(\mathbf{IVar})$

where $free(P) \subseteq \mathbf{IVar}$ is the finite set of all free integer variables in P. This defines a functor $assn : \mathsf{Cont} \to \mathsf{Set}$. For any inclusion $\gamma \subseteq \gamma'$ the corresponding inclusion map $assn(\gamma \subseteq \gamma') : assn(\gamma) \to assn(\gamma')$ just states that for any assertion P the inclusion $free(P) \subseteq \gamma$ entails the inclusion $free(P) \subseteq \gamma'$.

Generalization Assertions: In case of a system with more components we should have assertions for each single component. For example, different components should have different locations (addresses). Moreover, we should be able to translate statements about a subsystem into statements about the corresponding compound system (but not the other way around). Therefore we propose to assume for the general case a functor $assn : \mathsf{Str} \times \mathsf{Cont} \to \mathsf{Set}$.

Satisfaction: The inductive definition of satisfaction in Section 2.2 can be easily modified to take into account finite contexts. We will get, in such a way, satisfaction for **IMP** as a family of ternary relations between states, environments, and assertions indexed by contexts $\gamma \in |\mathsf{Cont}|$

$$\models_\gamma \subseteq \Sigma_\perp \times env(\gamma) \times assn(\gamma).$$

Moreover, the definition of satisfaction for **IMP** ensures compatibility w.r.t. context extensions:

Proposition 3.1 (Satisfaction is compatible w.r.t. context extension) *For any inclusion $\gamma \subseteq \gamma'$, any state $\sigma \in \Sigma_\perp$, any environment $e' \in env(\gamma')$ and any assertion $P \in assn(\gamma)$ we have*

$$(\sigma, env(\gamma \subseteq \gamma')(e')) = (\sigma, p_{\gamma',\gamma}(e')) \models_\gamma P \quad \textit{iff} \quad (\sigma, e') \models_{\gamma'} P.$$

Generalization Satisfaction: In the general setting we have the functors $st : \mathsf{Str}^{op} \to \mathsf{Cat}$, $env : \mathsf{Cont}^{op} \to \mathsf{Cat}$, and $assn : \mathsf{Str} \times \mathsf{Cont} \to \mathsf{Set}$. Cat is the category of all small categories and all functors between them where a category C is small iff its collection of morphisms (and thus also its collection of objects) is a set. Satisfaction is defined as a family of ternary relations between states, environments, and assertions indexed by objects $C \in |\mathsf{Str}|$ (component identifiers) and contexts $\gamma \in |\mathsf{Cont}|$

$$\models_{C,\gamma} \subseteq |st(C)| \times |env(\gamma)| \times |assn(C, \gamma)|.$$

Moreover, the following compatibility condition should be satisfied: For any context morphism $i : \gamma \to \gamma'$, any morphism $r : C \to C'$ in Str, any state $\sigma' \in$

$|st(C')|$, any environment $e' \in |env(\gamma')|$ and any assertion $P \in assn(\gamma)$ we have $(st(r)(\sigma'), env(i)(e')) \models_{C,\gamma} P \quad \text{iff} \quad (\sigma', e') \models_{C',\gamma'} assn(r,i)(P)$.

Extension and Semantic Entailment: We can combine the two functors $st : \mathsf{Str}^{op} \to \mathsf{Cpo}$ and $env : \mathsf{Cont}^{op} \to \mathsf{Set}$ into a functor $st \otimes env : (\mathsf{Str} \times \mathsf{Cont})^{op} \to \mathsf{Set}$ with $(st \otimes env)(C, \gamma) \triangleq st(C) \times env(\gamma) = \Sigma_\perp \times env(\gamma)$ for all pairs of objects $(C,\gamma) \in |\mathsf{Str}\times\mathsf{Cont}|$ and with $(st\otimes env)(id_C, \gamma \subseteq \gamma') \triangleq id_{\Sigma_\perp} \times p_{\gamma',\gamma} : \Sigma_\perp \times env(\gamma') \to \Sigma_\perp \times env(\gamma)$ for all pairs of morphisms $(id_C, \gamma \subseteq \gamma')$ in $\mathsf{Str} \times \mathsf{Cont}$.

By applying the contra-variant power set construction we obtain then a functor $(st \otimes env)^{-1} : \mathsf{Str} \times \mathsf{Cont} \to \mathsf{Po}$, where Po is the category of partial orders and monotone functions, with $(st \otimes env)^{-1}(C, \gamma) \triangleq (\wp(\Sigma_\perp \times env(\gamma)), \subseteq)$ for all pairs of objects $(C, \gamma) \in |\mathsf{Str} \times \mathsf{Cont}|$ and with

$$(st \otimes env)^{-1}(id_C, \gamma \subseteq \gamma') \triangleq (id_{\Sigma_\perp} \times p_{\gamma',\gamma})^{-1} : \wp(\Sigma_\perp \times env(\gamma)) \to \wp(\Sigma_\perp \times env(\gamma'))$$

for all pairs of morphisms $(id_C, \gamma \subseteq \gamma')$ in $\mathsf{Str} \times \mathsf{Cont}$, i.e., for all $\mathcal{E} \subseteq \Sigma_\perp \times env(\gamma)$ we have

$$(id_{\Sigma_\perp} \times p_{\gamma',\gamma})^{-1}(\mathcal{E}) = \{(\sigma, e') \in \Sigma_\perp \times env(\gamma') \mid (\sigma, p_{\gamma',\gamma}(e')) \in \mathcal{E}\}.$$

Remark 3.2 [Logic of States as an Institution] A closer look at the development so far shows that we have described actually an institution **ST** [6]. The category of abstract signatures is the product category $\mathsf{Str} \times \mathsf{Cont}$. Our sentence functor is the functor $assn : \mathsf{Str} \times \mathsf{Cont} \to \mathsf{Set}$ and the two functors $st : \mathsf{Str}^{op} \to \mathsf{Cat}$ and $env : \mathsf{Cont}^{op} \to \mathsf{Cat}$ can be combined into a functor $st \otimes env : (\mathsf{Str} \times \mathsf{Cont})^{op} \to \mathsf{Cat}$ with $(st \otimes env)(C, \gamma) \triangleq st(C) \times env(\gamma)$ for all pairs of objects $(C, \gamma) \in |\mathsf{Str} \times \mathsf{Cont}|$ and with $(st \otimes env)(r, i) \triangleq st(r) \times env(i) : st(C') \times env(\gamma') \to st(C) \times env(\gamma)$ for all pairs of morphisms $(r : C \to C', i : \gamma \to \gamma')$ in $\mathsf{Str} \times \mathsf{Cont}$.

The family of ternary satisfaction relations can be considered as a family of binary satisfaction relations in **ST** and the satisfaction condition above becomes the satisfaction relation in **ST**.

Based on the development so far, we can define for any context $\gamma \in |\mathsf{Cont}|$ a map $ext_\gamma : assn(\gamma) \to \wp(\Sigma_\perp \times env(\gamma))$ providing for each assertion $P \in assn(\gamma)$ its extension

$$ext_\gamma(P) \triangleq \{(\sigma, e) \in \Sigma_\perp \times env(\gamma) \mid (\sigma, e) \models_\gamma P\}.$$

We will denote here semantic entailment by $P \Vdash Q$ instead of $P \models Q$ as in Section 2.2. Semantic entailment for single assertions $P, Q \in assn(\gamma)$ with finite context γ is defined then by

$$P \Vdash_\gamma Q \quad \text{iff} \quad ext_\gamma(P) \subseteq ext_\gamma(Q) \tag{2}$$

In categorical terms, this means that we extend $assn(\gamma)$ to a preorder $assn_{\Vdash}(\gamma) \triangleq (assn(\gamma), \Vdash_\gamma)$ such that ext_γ turns into a monotone function $ext_\gamma : (assn(\gamma), \Vdash_\gamma) \to (\wp(\Sigma_\perp \times env(\gamma)), \subseteq)$.

Remark 3.3 [Cartesian Closed Category] The preorder category $(assn(\gamma), \Vdash_\gamma)$, for any context γ, is Cartesian closed with products $\wedge$, sums $\vee$ and exponentiation $\to$, i.e., we have

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

Proposition 3.1 ensures now, that for any context inclusion $\gamma \subseteq \gamma'$ the corresponding mapping $assn(\gamma \subseteq \gamma')$ is monotonic w.r.t. semantic entailment, i.e., we obtain a functor

$$assn(\gamma \subseteq \gamma') : (assn(\gamma), \Vdash_\gamma) \to (assn(\gamma'), \Vdash_{\gamma'}).$$

Globally, we get, in such a way, a functor $assn : \mathsf{Str} \times \mathsf{Cont} \to \mathsf{Pre}$ where Pre is the category of preorders and monotone functions. Moreover, compatibility of satisfaction ensures that extensions behave in a "natural way":

Proposition 3.4 *The family of functors* $ext_\gamma : (assn(\gamma), \Vdash_\gamma) \to (\wp(\Sigma_\perp \times env(\gamma)), \subseteq)$ *with* $\gamma \in |\mathsf{Cont}|$ *constitutes a natural transformation* $ext : assn \Rightarrow (st \otimes env)^{-1} : \mathsf{Str} \times \mathsf{Cont} \to \mathsf{Pre}$*, where we recall that* Po *is a subcategory of* Pre*.*

$$\begin{tikzcd}
C \arrow[d, "id_C"'] & \gamma \arrow[d, "\subseteq"'] & \Sigma_\perp & env(\gamma) & (\wp(\Sigma_\perp \times env(\gamma)), \subseteq) \arrow[d, "(id_{\Sigma_\perp} \times p_{\gamma',\gamma})^{-1}"'] & (assn(\gamma), \Vdash_\gamma) \arrow[l, "ext_\gamma"'] \arrow[d, "assn(\gamma \subseteq \gamma')"] \\
C & \gamma' & \Sigma_\perp \arrow[u, "id_{\Sigma_\perp}"] & env(\gamma') \arrow[u, "p_{\gamma',\gamma}"] & (\wp(\Sigma_\perp \times env(\gamma')), \subseteq) & (assn(\gamma'), \Vdash_{\gamma'}) \arrow[l, "ext_{\gamma'}"']
\end{tikzcd}$$

Generalization Extension and Semantic Entailment: The general case works completely analogously: The two functors $st : \mathsf{Str}^{op} \to \mathsf{Cat}$ and $env : \mathsf{Cont}^{op} \to \mathsf{Cat}$ can be combined into a functor $st \otimes env : (\mathsf{Str} \times \mathsf{Cont})^{op} \to \mathsf{Cat}$ with

$$(st \otimes env)(C, \gamma) \triangleq st(C) \times env(\gamma)$$

a product of categories for all pairs of objects $(C, \gamma) \in |\mathsf{Str} \times \mathsf{Cont}|$ and with

$$(st \otimes env)(r, i) \triangleq st(r) \times env(i) : st(C') \times env(\gamma') \to st(C) \times env(\gamma)$$

for all pairs of morphisms $(r : C \to C', i : \gamma \to \gamma')$ in $\mathsf{Str} \times \mathsf{Cont}$.

The functor $(st \otimes env)^{-1} : \mathsf{Str} \times \mathsf{Cont} \to \mathsf{Po}$ is obtained by applying the contravariant power construction for categories. That is, we consider for all pairs of objects $(C, \gamma) \in |\mathsf{Str} \times \mathsf{Cont}|$ the partial order of all subcategories of $st(C) \times env(\gamma)$:

$$(st \otimes env)^{-1}(C, \gamma) \triangleq (\wp(st(C) \times env(\gamma)), \subseteq)$$

and for all pairs of morphisms $(r, i) : (C, \gamma) \to (C', \gamma')$ in $\mathsf{Str} \times \mathsf{Cont}$ we consider the pre-image functor

$$(st \otimes env)^{-1}(r, i) \triangleq (st(r) \times env(i))^{-1}.$$

For any component identifier $C \in |\mathsf{Str}|$ and any context $\gamma \in |\mathsf{Cont}|$ we get a map $ext_{C,\gamma} : assn(C, \gamma) \to \wp(st(C) \times env(\gamma))$ with $ext_{C,\gamma}(P)$ the full subcategory of $st(C) \times env(\gamma)$ given by the set of objects

$$|ext_{C,\gamma}(P)| \triangleq \{(\sigma, e) \in |st(C) \times env(\gamma)| \mid (\sigma, e) \models_{C,\gamma} P\}.$$

Semantic entailment for single assertions is defined by $\quad P \Vdash_{C,\gamma} Q \quad$ iff $\quad ext_{C,\gamma}(P) \subseteq ext_{C,\gamma}(Q)$.

This means that we extend $assn(C,\gamma)$ to a preorder category $assn_{\Vdash}(C,\gamma) \triangleq (assn(C,\gamma), \Vdash_{C,\gamma})$ such that $ext_{C,\gamma}$ turns into a full functor

$$ext_{C,\gamma} : (assn(C,\gamma), \Vdash_{C,\gamma}) \to (\wp(st(C) \times env(\gamma)), \subseteq).$$

Compatibility of satisfaction ensures, that for any context morphism $i : \gamma \to \gamma'$ and any morphism $r : C \to C'$ in Str the corresponding mapping $assn(r,i)$ is monotonic w.r.t. semantic entailment, i.e., we have a functor

$$assn(r,i) : (assn(C,\gamma), \Vdash_{C,\gamma}) \to (assn(C',\gamma'), \Vdash_{C',\gamma'}).$$

Globally, we get, in such a way a functor $assn : \mathsf{Str} \times \mathsf{Cont} \to \mathsf{Pre}$. Moreover, compatibility of satisfaction ensures that the functors $ext_{C,\gamma}$ constitute a natural transformation $ext : assn \Rightarrow (st \otimes env)^{-1}$. Note, that the pre-image of a full subcategory w.r.t. a functor is a full subcategory as well.

$$\begin{tikzcd}
C \arrow[d, "r"'] & \gamma \arrow[d, "i"'] & st(C) & env(\gamma) & (\wp(st(C) \times env(\gamma)), \subseteq) \arrow[d, "(st(r) \times env(i))^{-1}"] & assn_{\Vdash}(C,\gamma) \arrow[l, "ext_{C,\gamma}"'] \arrow[d, "assn(r,i)"] \\
C' & \gamma' & st(C') \arrow[u, "st(r)"] & env(\gamma') \arrow[u, "env(i)"] & (\wp(st(C') \times env(\gamma')), \subseteq) & assn_{\Vdash}(C',\gamma') \arrow[l, "ext_{C',\gamma'}"']
\end{tikzcd}$$

3.2 Logic of Programs

Here we extend our analysis to programs in **IMP** and their semantics. Based on this we give an indexed categorical account of partial correctness assertions based on the concept of the weakest liberal precondition.

Programs: Programs in **IMP** are defined in a way that we can naturally consider them as arrows in a syntactic category Prg with C the only object. The identity on C can be considered as the program $\underline{\textbf{skip}}$ and composition in Prg is given by sequential composition of programs.

The categories Str and Prg do have the same object and the definition of the denotational semantics of programs based on the equations

$$\llbracket\, \underline{\textbf{skip}} \,\rrbracket\sigma = \sigma \quad \text{and} \quad \llbracket c_1 ; c_2 \rrbracket^* \sigma = \llbracket c_2 \rrbracket^* (\llbracket c_1 \rrbracket \sigma) \tag{3}$$

means that we extend the functor $st : \mathsf{Str}^{op} \to \mathsf{Cpo}$ to a functor $sem : \mathsf{Prg}^{op} \to \mathsf{Cpo}$ with $sem(c) \triangleq \llbracket c \rrbracket^* : \Sigma_\bot \to \Sigma_\bot$ for all the "program arrows" c in Prg. Extension means that we have $st(C) = sem(C) = \Sigma_\bot = (\mathbf{Loc} \to \mathbb{Z})_\bot$.

Generalization Programs: Analogously, we extend Str to a category Prg with $|\mathsf{Str}| = |\mathsf{Prg}|$ by introducing new arrows which represent programs/methods and we extend the functor $st : \mathsf{Str}^{op} \to \mathsf{Cat}$ to a functor $sem : \mathsf{Prg}^{op} \to \mathsf{Cat}$.

Weakest Liberal Precondition: In the same way as before we can combine the two functors $sem : \mathsf{Prg}^{op} \to \mathsf{Cpo}$ and $env : \mathsf{Cont}^{op} \to \mathsf{Set}$ into a functor $sem \otimes env :$

$(\mathsf{Prg} \times \mathsf{Cont})^{op} \to \mathsf{Set}$ with

$$(sem \otimes env)(C, \gamma) \triangleq sem(C) \times env(\gamma) = \Sigma_\perp \times env(\gamma)$$

for all pairs of objects $(C, \gamma) \in |\mathsf{Prg} \times \mathsf{Cont}| = |\mathsf{Str} \times \mathsf{Cont}|$ and

$$(sem \otimes env)(c, \gamma \subseteq \gamma') \triangleq [\![c]\!]^* \times p_{\gamma',\gamma} : \Sigma_\perp \times env(\gamma') \to \Sigma_\perp \times env(\gamma)$$

for all pairs of morphisms $(c, \gamma \subseteq \gamma')$ in $\mathsf{Prg} \times \mathsf{Cont}$. The elements in $\Sigma_\perp \times env(\gamma)$ are our "states" while the maps $[\![c]\!]^* \times p_{\gamma',\gamma}$ are "state transition functions". By applying the contra-variant power construction we obtain a functor $(sem \otimes env)^{-1} : \mathsf{Prg} \times \mathsf{Cont} \to \mathsf{Po}$ which extends the functor $(st \otimes env)^{-1}$. That is, we have

$$(sem \otimes env)^{-1}(C, \gamma) \triangleq (\wp(\Sigma_\perp \times env(\gamma)), \subseteq)$$

for all pairs of objects $(C, \gamma) \in |\mathsf{Prg} \times \mathsf{Cont}| = |\mathsf{Str} \times \mathsf{Cont}|$ and

$$(sem \otimes env)^{-1}(c, \gamma \subseteq \gamma') \triangleq ([\![c]\!]^* \times p_{\gamma',\gamma})^{-1}$$

for all pairs of morphisms $(c, \gamma \subseteq \gamma')$ in $\mathsf{Prg} \times \mathsf{Cont}$. The elements in $\wp(\Sigma_\perp \times env(\gamma))$ are our "predicates" while the maps $([\![c]\!]^* \times p_{\gamma',\gamma})^{-1}$ are "predicate transformers".

The "predicate transformer" functor $(sem \otimes env)^{-1}$ provides just another way to present the semantic weakest liberal precondition from Section 2.2.

Lemma 3.5 *Given a program morphism c in Prg, a context $\gamma \in |\mathsf{Cont}|$, and an assertion $Q \in assn(\gamma)$, we have for any state $\sigma \in \Sigma_\perp$ and any environment $e \in env(\gamma)$*

$$\sigma \in \mathsf{wlp}^e[\![c, Q]\!] \quad \textit{iff} \quad (\sigma, e) \in ([\![c]\!]^* \times id_\gamma)^{-1}(ext_\gamma(Q))$$

The quest for a weakest liberal precondition and, especially, the interplay between the semantic and the syntactic version of weakest liberal precondition can be presented now in our indexed framework as follows:

Indexed categorical quest for weakest liberal precondition: Extend the functor $assn : \mathsf{Str} \times \mathsf{Cont} \to \mathsf{Pre}$ to a functor $wlp : \mathsf{Prg} \times \mathsf{Cont} \to \mathsf{Pre}$ such that the family of functors

$$ext_\gamma : (assn(\gamma), \Vdash_\gamma) \to (\wp(\Sigma_\perp \times env(\gamma)), \subseteq)$$

with $\gamma \in |\mathsf{Cont}|$ constitutes also a natural transformation $ext : wlp \Rightarrow (sem \otimes env)^{-1}$.

$$\begin{tikzcd}
C \arrow[d, "c"'] & \gamma \arrow[d, "\subseteq"'] & \Sigma_\perp & env(\gamma) & (\wp(\Sigma_\perp \times env(\gamma)), \subseteq) \arrow[d, maps to, "([\![c]\!]^* \times p_{\gamma',\gamma})^{-1}"] & (assn(\gamma), \Vdash_\gamma) \arrow[l, "ext_\gamma"'] \arrow[d, maps to, "wlp(c,\gamma\subseteq\gamma')?"] \\
C & \gamma' & \Sigma_\perp \arrow[u, maps to, "{[\![c]\!]^*}"] & env(\gamma') \arrow[u, maps to, "p_{\gamma',\gamma}"] & (\wp(\Sigma_\perp \times env(\gamma')), \subseteq) & (assn(\gamma'), \Vdash_{\gamma'}) \arrow[l, "ext_{\gamma'}"']
\end{tikzcd}$$

Analysis of the Indexed categorical quest for weakest liberal precondition: Any arrow $(c, \gamma \subseteq \gamma')$ in $\mathsf{Prg} \times \mathsf{Cont}$ can be decomposed into $(c, id_\gamma); (id_C, \gamma \subseteq$

$\gamma')$ with $(id_C, \gamma \subseteq \gamma')$ an arrow in $\mathsf{Str} \times \mathsf{Cont}$. Since we require that wlp is an extension of $assn$ we have $wlp(id_C, \gamma \subseteq \gamma') = assn(\gamma \subseteq \gamma')$. So the new requirement is actually that we have functors $wlp(c, id_\gamma) : (assn(\gamma), \Vdash_\gamma) \to (assn(\gamma), \Vdash_\gamma)$. For notational convenience we will use for these ”new functors” the notation $wlp_\gamma(c) \triangleq wlp(c, id_\gamma)$.

The requirement that $wlp_\gamma(c)$ is a map with $wlp_\gamma(c); ext_\gamma = ext_\gamma; (\llbracket c \rrbracket^* \times id_{env(\gamma)})^{-1}$ is equivalent to condition (1) and comprises the requirement that our language of assertions is *expressive*. The additional requirement that $wlp_\gamma(c)$ becomes a functor means that the syntactic weakest liberal precondition is monotonic w.r.t. semantic entailment.

To validate that the naturality requirements $wlp_\gamma(c); ext_\gamma = ext_\gamma; (\llbracket c \rrbracket^* \times id_{env(\gamma)})^{-1}$ for the new arrows (c, id_γ) in $\mathsf{Prg} \times \mathsf{Cont}$ are sufficient to ensure naturality for all arrows in $\mathsf{Prg} \times \mathsf{Cont}$, we have to check if the following cube commutes:

$$\begin{tikzcd}
(\wp(\Sigma_\perp \times env(\gamma')), \subseteq) & & & (assn(\gamma'), \Vdash_{\gamma'}) \arrow[lll, "ext_{\gamma'}"'] \\
& (\wp(\Sigma_\perp \times env(\gamma)), \subseteq) \arrow[ul, "(id \times p_{\gamma',\gamma})^{-1}"] & (assn(\gamma), \Vdash_\gamma) \arrow[l, "ext_\gamma"'] \arrow[ur, "assn(\gamma \subseteq \gamma')"'] & \\
& (\wp(\Sigma_\perp \times env(\gamma)), \subseteq) \arrow[u, "(\llbracket c \rrbracket^* \times id_{env(\gamma)})^{-1}"] \arrow[dl, "(id \times p_{\gamma',\gamma})^{-1}"] & (assn(\gamma), \Vdash_\gamma) \arrow[l, "ext_\gamma"'] \arrow[u, "wlp_\gamma(c)"'] \arrow[dr, "assn(\gamma \subseteq \gamma')"'] & \\
(\wp(\Sigma_\perp \times env(\gamma')), \subseteq) \arrow[uuu, "(\llbracket c \rrbracket^* \times id_{env(\gamma')})^{-1}"] & & & (assn(\gamma'), \Vdash_{\gamma'}) \arrow[lll, "ext_{\gamma'}"'] \arrow[uuu, "wlp_{\gamma'}(c)"'] \arrow[from=1-1, to=4-2, phantom, "(1)"] \arrow[from=1-2, to=2-3, phantom, "(2)"] \arrow[from=3-2, to=4-3, phantom, "(3)"] \arrow[from=1-4, to=4-3, phantom, "(4)"]
\end{tikzcd}$$

That the small inner and the big outer square commute is the naturality requirement for the new functors wlp_γ. Left square (1) commutes due to the functoriality of the contra-variant power set construction and the equalities $(\llbracket c \rrbracket^* \times id_{env(\gamma)}); (id \times p_{\gamma',\gamma}) = (\llbracket c \rrbracket^* \times p_{\gamma',\gamma}) = (id \times p_{\gamma',\gamma}); (\llbracket c \rrbracket^* \times id_{env(\gamma)})$. Top square (2) and bottom square (3) commute since the functors env_γ constitute a natural transformation $ext : assn \Rightarrow (st \otimes env)^{-1}$ (see Subsection 3.1). That the right square (4) commutes means that the weakest liberal precondition of an assertion P should only depend on the free variables in P. This requirement is satisfied for any traditional Hoare logic since those logics usually rely on the finite set of free variables in P as the implicit finite context of P.

Compositionality of Weakest Liberal Precondition: Traditional logical deduction systems are defined based on (and can only provide) logical equivalence. In our indexed setting this is reflected by the observation that it is nearly impossible to define/deduce syntactic weakest liberal preconditions in such a way that we have compositionality up to syntactic identity. That is, for an assertion $P \in assn(\gamma)$ and programs c_1, c_2 we will, in general, not have $wlp_\gamma(c_1; c_2)(P) = wlp_\gamma(c_2)(wlp_\gamma(c_1)(P))$. We will only have that $wlp_\gamma(c_1; c_2)(P)$ and $wlp_\gamma(c_2)(wlp_\gamma(c_1)(P))$ are logical equivalent, i.e., isomorphic in $(assn(\gamma), \Vdash_\gamma)$. This means that our “indexed categorical quest” has to be relaxed to become reasonable.

Relaxed Indexed categorical quest for weakest liberal precondition: Extend the functor $assn : \mathsf{Str} \times \mathsf{Cont} \to \mathsf{Pre}$ to a *pseudo functor* $wlp : \mathsf{Prg} \times \mathsf{Cont} \to \mathsf{Pre}$ such that the family of functors

$$ext_\gamma : (assn(\gamma), \Vdash_\gamma) \to (\wp(\Sigma_\perp \times env(\gamma)), \subseteq)$$

with $\gamma \in |\mathsf{Cont}|$ constitutes also a natural transformation $ext : wlp \Rightarrow (sem \otimes env)^{-1}$.

4 Hoare Logic and Fibrations

The traditional presentation of the structural features of Hoare logic, as outlined in Section 2.2, is essentially an indexed one. In the last section we have elucidated this observation by lifting the traditional presentation to a more general and structured presentation based on indexed categories. There are now, at least, two reasons to move from the indexed setting to a fibred one:

(i) A more structural reason is that it is technically quite uncomfortable to work with pseudo functors. To work, in those cases, with the fibred equivalent of pseudo functors is more reasonable.

(ii) The essential reason in light of logic is, however, that we need a "technological space" where logical deduction can take place.

4.1 Syntactic Fibrations

The Grothendieck construction is the main technique for providing a fibred category from an indexed category. We do not include a general definition of this construction but describe in detail all the four special cases we consider in this section. By applying the appropriate variant of the Grothendieck construction we obtain out of the functor $assn : \mathsf{Str} \times \mathsf{Cont} \to \mathsf{Pre}$ a category of assertions about states:

Definition 4.1 [Category Assn] The category Assn of "state assertions" is defined as follows:

- objects: all pairs $(\gamma.P)$ with a context $\gamma \in |\mathsf{Cont}|$ and an assertion $P \in assn(\gamma)$.
- arrows: from $(\gamma.P)$ to $(\gamma'.Q)$ are pairs $(\gamma' \subseteq \gamma, \Vdash_\gamma)$ with $\gamma' \subseteq \gamma$ an arrow in Cont and $P \Vdash_\gamma Q = assn(\gamma' \subseteq \gamma)(Q)$ an arrow in $assn(\gamma)$.
- identities: the identity on object $(\gamma.P)$ is $(id_\gamma, =)$.
- composition: composition of two arrows $(\gamma' \subseteq \gamma, \Vdash_\gamma) : (\gamma.P) \to (\gamma'.Q)$ and $(\gamma'' \subseteq \gamma', \Vdash_{\gamma'}) : (\gamma'.Q) \to (\gamma''.R)$ is the arrow $(\gamma'' \subseteq \gamma, \Vdash_\gamma) : (\gamma.P) \to (\gamma''.R)$. Composition is ensured by the monotonicity of context extensions w.r.t. semantic

entailment and the associativity of semantic entailment.

$$\begin{tikzcd}
C & \gamma & (assn(\gamma),\Vdash_\gamma) & P \arrow[r, phantom, "\Vdash_\gamma"] & Q \arrow[r, phantom, "\Vdash_\gamma"] & R \\
C \arrow[u, "id_C"] & \gamma' \arrow[u, "\subseteq"'] & (assn(\gamma'),\Vdash_{\gamma'}) \arrow[u, "assn(\gamma'\subseteq\gamma)"'] & & Q \arrow[u, mapsto] \arrow[r, phantom, "\Vdash_{\gamma'}"] & R \arrow[u, mapsto] \\
C \arrow[u, "id_C"] & \gamma'' \arrow[u, "\subseteq"'] & (assn(\gamma''),\Vdash_{\gamma''}) \arrow[u, "assn(\gamma''\subseteq\gamma')"'] & & & R \arrow[u, mapsto]
\end{tikzcd}$$

Moreover, we obtain a projection functor $F_{assn} : \mathsf{Assn} \to (\mathsf{Str} \times \mathsf{Cont})^{op}$ with $F_{assn}(\gamma.P) \triangleq (C,\gamma)$ and $F_{assn}(\gamma' \subseteq \gamma, \Vdash_\gamma) \triangleq (id_C, \gamma' \subseteq \gamma)$.

The general properties of Grothendieck constructions provide:

Theorem 4.2 *The functor $F_{assn} : \mathsf{Assn} \to (\mathsf{Str} \times \mathsf{Cont})^{op}$ is a (split) fibration where $(\gamma' \subseteq \gamma, \Vdash_\gamma) : (\gamma.P) \to (\gamma'.Q)$ is a cartesian arrow if, and only if, P and $Q = assn(\gamma' \subseteq \gamma)(Q)$ are equivalent w.r.t. semantic entailment, i.e., isomorphic in $(assn(\gamma), \Vdash_\gamma)$.*

Remark 4.3 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 $(assn(\gamma), \Vdash_\gamma) \simeq F^{-1}_{assn}(C,\gamma)$ (compare Remark 3.3). That we have a fibration ensures that every first-order variable is universally quantified and that the reasoning on them is sound. On the other hand, existentially quantified assertions have their sound semantics provided by the cartesian structure of the fibration. □

In the same way, the pseudo functor $wlp : \mathsf{Prg} \times \mathsf{Cont} \to \mathsf{Pre}$ gives rise to a category of partial correctness assertions:

Definition 4.4 [Category Pca] The category Pca of "partial correctness assertions" is defined as follows:

- objects: all pairs $(\gamma.P)$ with a context $\gamma \in |\mathsf{Cont}|$ and an assertion $P \in assn(\gamma)$.
- arrows: from $(\gamma.P)$ to $(\gamma'.Q)$ are triples $(c, \gamma' \subseteq \gamma, \Vdash_\gamma)$ with c an arrow in Prg, $\gamma' \subseteq \gamma$ an arrow in Cont and $P \Vdash_\gamma wlp(c, \gamma' \subseteq \gamma)(Q)$ an arrow in $assn(\gamma)$.
- identities: the identity on object $(\gamma.P)$ is $(id_C, id_\gamma, =)$.
- composition: composition of two arrows $(c_1, \gamma' \subseteq \gamma, \Vdash_\gamma) : (\gamma.P) \to (\gamma'.Q)$ and $(c_2, \gamma'' \subseteq \gamma', \Vdash_{\gamma'}) : (\gamma'.Q) \to (\gamma''.R)$ is the arrow $(c_1; c_2, \gamma'' \subseteq \gamma, \Vdash_\gamma) : (\gamma.P) \to (\gamma''.R)$. Composition is ensured by the monotonicity of the weakest liberal precondition w.r.t. semantic entailment, by the assumption that wlp is a pseudo functor, i.e., $wlp(c_1, \gamma' \subseteq \gamma)(wlp(c_2, \gamma'' \subseteq \gamma')(R))$ and $wlp(c_1; c_2, \gamma'' \subseteq \gamma)(R)$ are equivalent w.r.t. semantic entailment, and by the associativity of semantic entail-

ment.

$$\begin{tikzcd}
C \arrow[d, "c_1"] & \gamma & P & \Vdash_\gamma & wlp(c_1, \gamma' \subseteq \gamma)(Q) & \Vdash_\gamma & wlp(c_1; c_2, \gamma'' \subseteq \gamma)(R) \\
C \arrow[d, "c_2"] & \gamma' \arrow[u, "\subseteq"] & & & Q \arrow[u, mapsto, "wlp(c_1,\gamma' \subseteq \gamma)"] & \Vdash_{\gamma'} & wlp(c_2, \gamma'' \subseteq \gamma')(R) \arrow[u, mapsto, "wlp(c_1,\gamma' \subseteq \gamma)"] \\
C & \gamma'' \arrow[u, "\subseteq"] & & & & & R \arrow[u, mapsto, "wlp(c_2,\gamma'' \subseteq \gamma')"]
\end{tikzcd}$$

Moreover, we obtain a projection functor $F_{pca} : \mathsf{Pca} \to (\mathsf{Prg} \times \mathsf{Cont})^{op}$ with $F_{pca}(\gamma.P) \triangleq (C, \gamma)$ and $F_{pca}(c, \gamma' \subseteq \gamma, \Vdash_\gamma) \triangleq (c, \gamma' \subseteq \gamma)$.

Theorem 4.5 *The functor $F_{pca} : \mathsf{Pca} \to (\mathsf{Prg} \times \mathsf{Cont})^{op}$ is a fibration where $(c, \gamma' \subseteq \gamma, \Vdash_\gamma) : (\gamma.P) \to (\gamma'.Q)$ is a cartesian arrow if, and only if, P and $wlp(c, \gamma' \subseteq \gamma)(Q)$ are equivalent w.r.t. semantic entailment.*

$$\begin{tikzcd}
\mathsf{Assn} \arrow[r, "\subseteq"] \arrow[d, "F_{assn}"] & \mathsf{Pca} \arrow[d, "F_{pca}"] \\
(\mathsf{Str} \times \mathsf{Cont})^{op} \arrow[r, "\subseteq"] & (\mathsf{Prg} \times \mathsf{Cont})^{op}
\end{tikzcd}$$

For the rest of the paper we have to keep in mind that we have by construction $|\mathsf{Assn}| = |\mathsf{Pca}|$. Moreover, the commutative triangle $assn = \subseteq; wlp$ of functors is transformed by the Grothendieck construction into a pullback in the category of categories.

4.2 Hoare Triples and Hoare Proof Calculus

Hoare Triples (Partial Correctness Assertions): We can consider Assn as a subcategory of Pca since $assn(\gamma' \subseteq \gamma) = wlp(id_C, \gamma' \subseteq \gamma)$ for all inclusions $\gamma' \subseteq \gamma$. Moreover, any arrow $(c, \gamma' \subseteq \gamma, \Vdash_\gamma) : (\gamma.P) \to (\gamma'.R)$ in Pca can be decomposed into an arrow $(\gamma' \subseteq \gamma, \Vdash_\gamma) : (\gamma.P) \to (\gamma'.Q)$ in Assn and an arrow $(c, id_{\gamma'}, \Vdash_{\gamma'}) : (\gamma'.Q) \to (\gamma'.R)$ in Pca.

$$\begin{tikzcd}
C & \gamma & P & \Vdash_\gamma & Q & \Vdash_\gamma & wlp(c, \gamma' \subseteq \gamma)(R) \\
C \arrow[u, "id_C"] & \gamma' \arrow[u, "\subseteq"] & & & Q \arrow[u, mapsto, "assn(\gamma' \subseteq \gamma)"] & \Vdash_{\gamma'} & wlp_{\gamma'}(c)(R) \arrow[u, mapsto, "assn(\gamma' \subseteq \gamma)"] \\
C \arrow[u, "c"] & \gamma' \arrow[u, "id_{\gamma'}"] & & & & & R \arrow[u, mapsto, "wlp_{\gamma'}(c)"]
\end{tikzcd}$$

That is, we need only the "new" arrows of type $(c, id_\gamma, \Vdash_\gamma) : (\gamma.P) \to (\gamma.Q)$ to extend Assn to the category Pca.

A comparison with the definition and the semantics of Hoare triples (partial correctness assertions) makes immediately evident that a Hoare triple $\{P\}\ c\ \{Q\}$ is just another notation for arrows of type $(c, id_\gamma, \Vdash_\gamma) : (\gamma.P) \to (\gamma.Q)$, i.e., for the relations $P \Vdash_\gamma wlp_\gamma(c)(Q)$. So, the task of a deduction calculus for a Hoare logic has the sole purpose of deriving all the arrows of type $(c, id_\gamma, \Vdash_\gamma) : (\gamma.P) \to (\gamma.Q)$ in Pca.

Hoare Proof Calculus: In view of the two syntactic fibrations a proof calculus for Hoare triples has, in such a way, two features:

(i) One constructs for each pair of an object $(\gamma.Q)$ in $|\mathsf{Assn}| = |\mathsf{Pca}|$ and of a new arrow in Prg, i.e., a program c, a corresponding new cartesian arrow in Pca ($\models \{wlp(c, Q)\}\ c\ \{Q\}$):

$$(c, id_\gamma, \Vdash_\gamma) : wlp_\gamma(c)(Q) \to (\gamma.Q)$$

These are the rules Assn, IfElse, and PWhile.

(ii) One closes everything under composition
  (a) by composing new arrows in Pca with new arrows in Pca (rule Comp) and
  (b) by pre- and post-composing new arrows in Pca with old arrows from Assn (rules PreStr and PosWk).

After these crucial observations we are ready to formulate our syntactic fibred quest:

Syntactic Fibred categorical quest for weakest liberal precondition:

$$\begin{tikzcd}
\mathsf{Assn} \arrow[r, dashed, "\subseteq"] \arrow[d, "F_{assn}"] & \mathsf{Pca} \arrow[d, dashed, "F_{pca}"] \\
(\mathsf{Str} \times \mathsf{Cont})^{op} \arrow[r, "\subseteq"] & (\mathsf{Prg} \times \mathsf{Cont})^{op}
\end{tikzcd}$$

Extend the fibration $F_{assn} : \mathsf{Assn} \to (\mathsf{Str} \times \mathsf{Cont})^{op}$ to a fibration $F_{pca} : \mathsf{Pca} \to (\mathsf{Prg} \times \mathsf{Cont})^{op}$ such that the diagram on the left becomes a pullback in the category of categories. Note, that the pullback requirement means that we are not allowed to introduce in Pca new entailments between assertions, i.e., the logic of assertions about states is encapsulated in Assn.

4.3 Semantic Fibrations

In our syntactic fibred quest the semantics of assertions and programs are not incorporated. To have a full fibred picture and to be able, especially, to talk about soundness and completeness of deduction, we have to transfer also our semantic indexed categories and the indexed extension functors to the fibred setting.

By applying the Grothendieck construction we obtain out of the functor $(st \otimes env)^{-1} : \mathsf{Str} \times \mathsf{Cont} \to \mathsf{Po}$ a category of the semantics of states:

Definition 4.6 [Category St] The category St of "state predicates" is defined as follows:

- objects: all pairs $(\gamma.\mathcal{E})$ with a context $\gamma \in |\mathsf{Cont}|$ and a set $\mathcal{E} \in \wp(\Sigma_\perp \times env(\gamma))$.
- arrows: from $(\gamma.\mathcal{E})$ to $(\gamma'.\mathcal{E}')$ are pairs $(\gamma' \subseteq \gamma, \subseteq)$ such that $\mathcal{E} \subseteq (id_{\Sigma_\perp} \times p_{\gamma,\gamma'})^{-1}(\mathcal{E}')$.
- identities: the identity on $(\gamma.\mathcal{E})$ is $(id_\gamma, =)$.
- composition: two arrows $(\gamma' \subseteq \gamma, \subseteq) : (\gamma.\mathcal{E}) \to (\gamma'.\mathcal{E}')$ and $(\gamma'' \subseteq \gamma', \subseteq) : (\gamma'.\mathcal{E}') \to (\gamma''.\mathcal{E}'')$ compose to the arrow $(\gamma'' \subseteq \gamma, \subseteq) : (\gamma.\mathcal{E}) \to (\gamma''.\mathcal{E}'')$. Composition

is ensured by the monotonicity of preimage formation w.r.t. inclusions, by the functoriality of preimage formation and by the associativity of inclusions.

$$\begin{tikzcd}
C & \gamma & \mathcal{E} & \subseteq & (id_{\Sigma_\perp} \times p_{\gamma,\gamma'})^{-1}(\mathcal{E}') & \subseteq & (id_{\Sigma_\perp} \times p_{\gamma,\gamma''})^{-1}(\mathcal{E}'') \\
C \arrow[u, mapsto, "id_C"] & \gamma' \arrow[u, mapsto, "\subseteq"] & & & \mathcal{E}' \arrow[u, mapsto, "(id_{\Sigma_\perp} \times p_{\gamma,\gamma'})^{-1}"] & \subseteq & (id_{\Sigma_\perp} \times p_{\gamma',\gamma''})^{-1}(\mathcal{E}'') \arrow[u, mapsto, "(id_{\Sigma_\perp} \times p_{\gamma,\gamma'})^{-1}"] \\
C \arrow[u, mapsto, "id_C"] & \gamma'' \arrow[u, mapsto, "\subseteq"] & & & & & \mathcal{E}'' \arrow[u, mapsto, "(id_{\Sigma_\perp} \times p_{\gamma',\gamma''})^{-1}"]
\end{tikzcd}$$

Moreover, we obtain a projection functor $F_{st} : \mathsf{St} \to (\mathsf{Str} \times \mathsf{Cont})^{op}$ with $F_{st}(\gamma.\mathcal{E}) \triangleq (C, \gamma)$ and $F_{st}(\gamma' \subseteq \gamma, \subseteq) \triangleq (id_C, \gamma' \subseteq \gamma)$.

Theorem 4.7 *The functor $F_{st} : \mathsf{St} \to (\mathsf{Str} \times \mathsf{Cont})^{op}$ is a (split) fibration where the cartesian arrows are exactly the arrows $(\gamma' \subseteq \gamma, =) : (\gamma.(id_{\Sigma_\perp} \times p_{\gamma,\gamma'})^{-1}(\mathcal{E}')) \to (\gamma'.\mathcal{E}')$ for all inclusions $\gamma' \subseteq \gamma$ and all sets $\mathcal{E}' \in \wp(\Sigma_\perp \times env(\gamma'))$. These are the only Cartesian arrows since inclusion $\subseteq$ is anti-symmetric.*

In the same way, the functor $(sem \otimes env)^{-1} : \mathsf{Prg} \times \mathsf{Cont} \to \mathsf{Po}$ provides a category of predicate transformers:

Definition 4.8 [Category Sem] The category Sem of “predicate transformers” is defined as follows:

- objects: all pairs $(\gamma.\mathcal{E})$ with a context $\gamma \in |\mathsf{Cont}|$ and a set $\mathcal{E} \in \wp(\Sigma_\perp \times env(\gamma))$.
- arrows: from $(\gamma.\mathcal{E})$ to $(\gamma'.\mathcal{E}')$ are triples $(c, \gamma' \subseteq \gamma, \subseteq)$ such that $\mathcal{E} \subseteq ([\![c]\!]^* \times p_{\gamma,\gamma'})^{-1}(\mathcal{E}')$.
- identities: the identity on $(\gamma.\mathcal{E})$ is $(id_C, id_\gamma, =)$.
- composition: two arrows $(c_1, \gamma' \subseteq \gamma, \subseteq) : (\gamma.\mathcal{E}) \to (\gamma'.\mathcal{E}')$ and $(c_2, \gamma'' \subseteq \gamma', \subseteq) : (\gamma'.\mathcal{E}') \to (\gamma''.\mathcal{E}'')$ compose to the arrow $(c_2; c_1, \gamma'' \subseteq \gamma, \subseteq) : (\gamma.\mathcal{E}) \to (\gamma''.\mathcal{E}'')$. Composition is ensured by the monotonicity of preimage formation w.r.t. inclusions, by the functoriality of program semantic and preimage formation, and by the associativity of inclusions.

$$\begin{tikzcd}
C & \gamma & \mathcal{E} & \subseteq & ([\![c_1]\!]^* \times p_{\gamma,\gamma'})^{-1}(\mathcal{E}') & \subseteq & ([\![c_1; c_2]\!]^* \times p_{\gamma,\gamma''})^{-1}(\mathcal{E}'') \\
C \arrow[u, mapsto, "c_1"] & \gamma' \arrow[u, mapsto, "\subseteq"] & & & \mathcal{E}' \arrow[u, mapsto, "([\![c_1]\!]^* \times p_{\gamma,\gamma'})^{-1}"] & \subseteq & ([\![c_2]\!]^* \times p_{\gamma',\gamma''})^{-1}(\mathcal{E}'') \arrow[u, mapsto, "([\![c_1]\!]^* \times p_{\gamma,\gamma'})^{-1}"] \\
C \arrow[u, mapsto, "c_2"] & \gamma'' \arrow[u, mapsto, "\subseteq"] & & & & & \mathcal{E}'' \arrow[u, mapsto, "([\![c_2]\!]^* \times p_{\gamma',\gamma''})^{-1}"]
\end{tikzcd}$$

Moreover, we obtain a projection functor $F_{sem} : \mathsf{Sem} \to (\mathsf{Prg} \times \mathsf{Cont})^{op}$ with $F_{sem}(\gamma.\mathcal{E}) \triangleq (C, \gamma)$ and $F_{sem}(c, \gamma' \subseteq \gamma, \subseteq) \triangleq (c, \gamma' \subseteq \gamma)$.

Theorem 4.9 *The functor $F_{sem} : \mathsf{Sem} \to (\mathsf{Prg} \times \mathsf{Cont})^{op}$ is a (split) fibration where the cartesian arrows are exactly the arrows $(c, \gamma' \subseteq \gamma, =) : (\gamma.(\llbracket c \rrbracket^* \times p_{\gamma,\gamma'})^{-1}(\mathcal{E}')) \to (\gamma'.\mathcal{E}')$ for all programs c, all inclusions $\gamma' \subseteq \gamma$ and all sets $\mathcal{E}' \in \wp(\Sigma_\perp \times env(\gamma'))$.*

$$\begin{tikzcd}
\mathsf{St} \arrow[r, "\subseteq"] \arrow[d, "F_{st}"] & \mathsf{Sem} \arrow[d, "F_{sem}"] \\
(\mathsf{Str} \times \mathsf{Cont})^{op} \arrow[r, "\subseteq"] & (\mathsf{Prg} \times \mathsf{Cont})^{op}
\end{tikzcd}$$

For the rest of the paper we have to keep in mind that we have by construction $|\mathsf{St}| = |\mathsf{Sem}|$. Moreover, the diagram to the left is a pullback in the category of categories. Note, that the pullback property says, analogously to the syntactic case, that Sem does not change the semantics of states.

4.4 Extensions and Soundness

The Grothendieck construction extends also to indexed functors (natural transformations) and transforms an indexed functor into a morphisms between the corresponding two fibrations, i.e., into a functor making the resulting triangle of functors commutative.

In the indexed setting, the assignments $P \mapsto ext_\gamma(P)$ provide functors $ext_\gamma : (assn(\gamma), \Vdash_\gamma) \to (\wp(\Sigma_\perp \times env(\gamma)), \subseteq)$, and the family of these functors constitute as well an indexed functor $ext : assn \Rightarrow (st \otimes env)^{-1}$ as an indexed functor $ext : wlp \Rightarrow (sem \otimes env)^{-1}$.

Correspondingly, the assignments $(\gamma.P) \mapsto (\gamma.ext_\gamma(P))$ provide, in the fibred setting, as well a functor $ext_{assn} : \mathsf{Assn} \to \mathsf{St}$ as a functor $ext_{pca} : \mathsf{Pca} \to \mathsf{St}$. The functor ext_{assn} maps the arrow $(\gamma' \subseteq \gamma, \Vdash_\gamma) : (\gamma.P) \to (\gamma'.Q)$ in Assn into the arrow $(\gamma' \subseteq \gamma, \subseteq) : (\gamma.ext_\gamma(P)) \to (\gamma'.ext_{\gamma'}(Q))$ in St. Note, that the definition of semantic entailment in (2) means nothing but that the functor ext_{assn} is full. It is easy to check that we have $F_{assn} = ext_{assn}; F_{st}$.

To extend the assignments $(\gamma.P) \mapsto (\gamma.ext_\gamma(P))$ to a functor $ext_{pca} : \mathsf{Pca} \to \mathsf{St}$ we have to assign to the arrow $(c, \gamma' \subseteq \gamma, \Vdash_\gamma) : (\gamma.P) \to (\gamma'.Q)$ in Pca the arrow $(c, \gamma' \subseteq \gamma, \subseteq) : (\gamma.ext_\gamma(P)) \to (\gamma'.ext_{\gamma'}(Q))$ in Sem. For the special case $\gamma' = \gamma$ this means that

$$P \Vdash_\gamma wlp_\gamma(c)(Q) \quad \text{implies} \quad ext_\gamma(P) \subseteq (\llbracket c \rrbracket^* \times id_{env(\gamma)})^{-1}(ext_\gamma(Q)).$$

That is, the requirement that $ext_{assn} : \mathsf{Assn} \to \mathsf{St}$ extends to a functor $ext_{pca} : \mathsf{Pca} \to \mathsf{Sem}$ is the fibred formulation of the requirement that the syntactic weakest liberal precondition is sound.

This last observation gives us the last brick to formulate our final quest:

Full fibred categorical quest for weakest liberal precondition:

$$\begin{tikzcd}
 & \mathsf{St} \arrow[rr, "\subseteq"] \arrow[ddl, "F_{st}"] & & \mathsf{Sem} \arrow[ddl, "F_{sem}"] \\
\mathsf{Assn} \arrow[ur, "ext_{assn}"] \arrow[rr, dashed, "\subseteq"] \arrow[dd, "F_{assn}"'] & & \mathsf{Pca} \arrow[ur, dashed, "ext_{pca}"] \arrow[dd, dashed, "F_{pca}"'] & \\
 & & & \\
(\mathsf{Str} \times \mathsf{Cont})^{op} \arrow[rr, "\subseteq"] & & (\mathsf{Prg} \times \mathsf{Cont})^{op} &
\end{tikzcd}$$

Extend the fibration $F_{assn} : \mathsf{Assn} \to (\mathsf{Str} \times \mathsf{Cont})^{op}$ to a fibration $F_{pca} : \mathsf{Pca} \to (\mathsf{Prg} \times \mathsf{Cont})^{op}$ in a way

- that the resulting commutative square $\subseteq; F_{pca} = F_{assn}; \subseteq$ of functors is a pullback, and
- that the functor $ext_{assn} : \mathsf{Assn} \to \mathsf{St}$ extends to a functor $ext_{pca} : \mathsf{Pca} \to \mathsf{Sem}$ with $ext_{assn}; \subseteq = \subseteq; ext_{pca}$ and $F_{pca} = ext_{pca}; F_{sem}$.

We want to close our categorical analysis by the remark that completeness of weakest liberal precondition can be reflected in the fibred setting by the additional requirement that ext_{pca} becomes a full functor.

5 Conclusion

The traditional presentation of the structural features of Hoare logic is essentially an indexed one. We lifted this insight into a more general and abstract level by developing a categorical presentation of the structural features of Hoare logic based on indexed categories. We outlined that the categorical presentation paves naturally a way for an extension of Hoare Logic from the traditional one program (component) setting into a multi-program (compound system) environment.

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.*

The paper concludes that an appropriate way to deal with axiomatic semantics, at least in the format of Hoare logic, is to use both categorical tools, fibred categories, and indexed categories, in a way that the structural features of the language are presented in indexed categories and the features of deduction in the fibred one. As a further work, we would like to extend our categorical model to deal with finite contexts of program variables. This is a departure from the usual textbook presentation, but it is in accordance with the current state of the art proof technology for programming languages and verification systems based or inspired by Hoare logic [9,2,5].

References

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

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

[3] Dijkstra, E. W. and C. S. Scholten, "Predicate calculus and program semantics," Texts and monographs in computer science, Springer, 1990, I-X, 1-220 pp.

[4] Fiadeiro, J., "Categories for Software Engineering," Springer, Berlin, 2005.

[5] Filliâtre, J.-C., *One Logic To Use Them All*, in: *CADE 24 - the 24th International Conference on Automated Deduction* (2013).

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

[7] Hoare, C. A. R., *An axiomatic basis for computer programming*, Commun. ACM **12** (1969), pp. 576–580.

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

[9] Leino, R., *Dafny: An automatic program verifier for functional correctness*, in: *16th International Conference, LPAR-16, Dakar, Senegal* (2010), pp. 348–370.

[10] Martini, A., U. Wolter and E. H. Haeusler, *Fibred and Indexed Categories for Abstract Model Theory*, Journal of the IGPL **15** (2007), pp. 707–739.

[11] Pawlowski, W., *Context institutions*, in: M. Haveraaen, O. Owe and O.-J. Dahl, editors, *COMPASS/ADT*, Lecture Notes in Computer Science **1130** (1995), pp. 436–457.

[12] Pitts, A. M., *Categorical Logic*, in: S. Abramsky, D. M. Gabbay and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science, Volume 5. Algebraic and Logical Structures*, Oxford University Press, 2000 pp. 39–128.

[13] Street, R., *Fibrations in bicategories*, Cahiers Topologie Géom. Différentielle **21, no. 2** (1980), pp. 111–160.