A General Semantic Construction of Dependent Refinement Type Systems, Categorically

Satoshi Kura[1,2]

[1] National Institute of Informatics, Tokyo, Japan
[2] The Graduate University for Advanced Studies (SOKENDAI), Kanagawa, Japan
kura@nii.ac.jp

Abstract. Dependent refinement types are types equipped with predicates that specify preconditions and postconditions of underlying functional languages. We propose a general semantic construction of dependent refinement type systems from underlying type systems and predicate logic, that is, a construction of liftings of closed comprehension categories from given (underlying) closed comprehension categories and posetal fibrations for predicate logic. We give sufficient conditions to lift structures such as dependent products, dependent sums, computational effects, and recursion from the underlying type systems to dependent refinement type systems. We demonstrate the usage of our construction by giving semantics to a dependent refinement type system and proving soundness.

1 Introduction

Dependent refinement types [6] are types equipped with predicates that restrict values in the types. They are used to specify preconditions and postconditions which may depend on input values and to verify that programs satisfy the specifications. Many dependent refinement types systems are proposed [5,6,13,14,25] and implemented in, e.g., F* [23,24] and LiquidHaskell [19,26,27].

In this paper, we address the question: "How are dependent refinement type systems, underlying type systems, and predicate logic related from the viewpoint of categorical semantics?" Although most existing dependent refinement type systems are proved to be sound using operational semantics, we believe that categorical semantics is more suitable for the general understanding of their nature, especially when we consider general computational effects and various kinds of predicate logic (e.g., for relational verification). This understanding will provide guidelines to design new dependent refinement type systems.

Our answer to the question is a general semantic construction of dependent refinement type systems from underlying type systems and predicate logic. More concretely, given a closed comprehension category (CCompC for short) for interpreting an underlying type system and a fibration for predicate logic, we combine them to obtain another CCompC that can interpret a dependent refinement type system built from the underlying type system and the predicate logic.

For example, consider giving an interpretation to the term "$x : \{\mathrm{int} \mid x \geq 0\} \vdash x + 1 : \{v : \mathrm{int} \mid v = x + 1\}$" in a dependent refinement type system. Its underlying term is "$x : \mathrm{int} \vdash x + 1 : \mathrm{int}$," and we assume that it is interpreted as the successor function of $\mathbb{Z}$ in **Set**. The problem here is how to refine this interpretation with predicates. In dependent refinement types, predicates may depend on the variables in contexts. In this example, the type "$x : \{\mathrm{int} \mid x \geq 0\} \vdash \{v : \mathrm{int} \mid v = x + 1\}$" depends on the variable x. Thus, the interpretation of such types must be a predicate on the context and the type, i.e.,

$$[\![x : \{\mathrm{int} \mid x \geq 0\} \vdash \{v : \mathrm{int} \mid v = x + 1\}]\!] = \{(x, v) \in \mathbb{Z} \times \mathbb{Z} \mid x \geq 0 \wedge v = x + 1\}.$$

As a result, the term in the dependent refinement type system is interpreted as the interpretation in the underlying type system together with the property that if the input satisfies preconditions, then the output satisfies postconditions.

$$\begin{tikzcd}
\{x \in \mathbb{Z} \mid x \geq 0\} \arrow[r, dashed] \arrow[d, phantom, "\cap" description] & \{(x, v) \in \mathbb{Z} \times \mathbb{Z} \mid x \geq 0 \wedge v = x + 1\} \arrow[d, phantom, "\cap" description] \\
\mathbb{Z} \arrow[r, "\langle \mathrm{id}_{\mathbb{Z}}, (-)+1 \rangle"] & \mathbb{Z} \times \mathbb{Z}
\end{tikzcd} \tag{1}$$

We formalize this refinement process as a construction of liftings of CCompCs, which are used to interpret dependent type theories. Assume that we have a pair of a CCompC $p : \mathbb{E} \to \mathbb{B}$ for interpreting underlying type systems and a fibration $q : \mathbb{P} \to \mathbb{B}$ for predicate logic satisfying certain conditions. Then we construct a CCompC $\{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{P}$ for interpreting dependent refinement type systems. This construction also yields a morphism of CCompCs from $\{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{P}$ to $p : \mathbb{E} \to \mathbb{B}$ in Fig. 1. Given the simple fibration $\mathbf{s}(\mathbf{Set}) \to \mathbf{Set}$ for underlying type systems and the subobject fibration $\mathbf{Sub}(\mathbf{Set}) \to \mathbf{Set}$ for predicate logic, then we get interpretations like (1).

$$\begin{tikzcd}
\{\mathbb{E} \mid \mathbb{P}\} \arrow[r] \arrow[d] & \mathbb{E} \arrow[d, "p"] \\
\mathbb{P} \arrow[r, "q"] & \mathbb{B}
\end{tikzcd}$$

Fig. 1. Lifting.

We extend the construction of liftings of CCompCs to liftings of fibred monads [1] on CCompCs, which is motivated by the fact that many dependent refinement type systems have computational effects, e.g., exception (like division and assertion), divergence, nondeterminism [25], and probability [5]. Assume that we have a fibred monad $\hat{T}$ on $p : \mathbb{E} \to \mathbb{B}$, a monad T on $\mathbb{B}$, and a lifting $\dot{T}$ of T along $q : \mathbb{P} \to \mathbb{B}$. Under a certain condition that roughly claims that $\hat{T}$ and T represent the same computational effects, we construct a fibred monad on $\{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{P}$, which is a lifting of $\hat{T}$ in the same spirit of the given lifting $\dot{T}$. This situation is rather realistic because the fibred monad $\hat{T}$ on the CCompC $p : \mathbb{E} \to \mathbb{B}$ is often induced from the monad T on the base category $\mathbb{B}$. The lifting $\dot{T}$ of the monad T along $p : \mathbb{P} \to \mathbb{B}$ specifies how to map predicates $P \in \mathbb{P}_X$ on values $X \in \mathbb{B}$ to predicates $\dot{T}P \in \mathbb{P}_{TX}$ on computations TX, which enables us to express, for example, total/partial correctness and may/must nondeterminism [1].

We explain the usage of these categorical constructions by giving semantics to a dependent refinement type system with computational effects, which is based on [4]. Our system also supports subtyping relations induced by logical implication. We prove soundness of the dependent refinement type system.

Finally, we discuss how to handle recursion in dependent refinement type systems. In [4], Ahman gives semantics to recursion in a specific model, i.e., the fibration of continuous families of ω-cpos $\mathbf{CFam}(\mathbf{CPO}) \to \mathbf{CPO}$. We consider more general characterization of recursion by adapting Conway operators for CCompCs, which enables us to lift the structure for recursion. We show that a rule for partial correctness in our dependent refinement type system is sound under the existence of a generalized Conway operator.

Our contributions are summarized as follows.

- We provide a general construction of liftings of CCompCs from given CCompCs and posetal fibrations satisfying certain conditions, as a semantic counterpart of construction of dependent refinement type systems from underlying type systems and predicate logic. We extend this to liftings of fibred monads on the underlying CCompCs to model computational effects.
- We consider a type system (based on EMLTT [2–4]) that includes most of basic features of dependent refinement type systems and prove its soundness in the liftings of CCompCs obtained from the above construction.
- We define Conway operators for dependent type systems. This generalizes the treatment of general recursion in [4]. We prove soundness of the typing rule for partial correctness of recursion under the existence of a lifting of Conway operators.

2 Preliminaries

We review basic definitions and fix notations for comprehension categories, which are used as categorical models for dependent type theories. We assume basic knowledge of fibrations (see e.g. [10]).

Let $p : \mathbb{E} \to \mathbb{B}$ be a fibration (opfibration). We denote the cartesian (cocartesian) lifting over $u : I \to J$ by $\overline{u}(Y) : u^*Y \to Y$ ($\underline{u}(X) : X \to u_!X$) where $u^* : \mathbb{E}_J \to \mathbb{E}_I$ ($u_! : \mathbb{E}_I \to \mathbb{E}_J$) is the reindexing (coreindexing) functor. We call $p : \mathbb{E} \to \mathbb{B}$ a *posetal fibration* if p is a fibration such that each fibre category is a poset. Note that the fibration $p : \mathbb{E} \to \mathbb{B}$ is split and faithful if p is posetal.

A *comprehension category* is a functor $\mathcal{P} : \mathbb{E} \to \mathbb{B}^{\to}$ such that the composite $\mathrm{cod} \circ \mathcal{P} : \mathbb{E} \to \mathbb{B}$ is a fibration and $\mathcal{P}$ maps cartesian morphisms to pullbacks in $\mathbb{B}$. A comprehension category $\mathcal{P}$ is *full* if $\mathcal{P}$ is fully faithful.

A *comprehension category with unit* is a fibration $p : \mathbb{E} \to \mathbb{B}$ that has a fibred terminal object $1 : \mathbb{B} \to \mathbb{E}$ and a comprehension functor $\{-\} : \mathbb{E} \to \mathbb{B}$ which is a right adjoint of the fibred terminal object functor $1 \dashv \{-\}$. Projection $\pi_X : \{X\} \to pX$ is defined by $\pi_X = p\epsilon_X^{1 \dashv \{-\}}$ for each $X \in \mathbb{E}$. Intuitively, $\mathbb{E}$ represents a collection of types $\Gamma \vdash A$ in dependent type theories; $\mathbb{B}$ represents a collection of contexts Γ; $p : \mathbb{E} \to \mathbb{B}$ is the mapping $(\Gamma \vdash A) \mapsto \Gamma$; $1 : \mathbb{B} \to \mathbb{E}$ is the unit type $\Gamma \mapsto (\Gamma \vdash 1)$; and $\{-\}$ is the mapping $(\Gamma \vdash A) \mapsto \Gamma, x : A$ where x is a fresh variable.

The comprehension category with unit $p : \mathbb{E} \to \mathbb{B}$ induces several structures. It induces a comprehension category $\mathcal{P}$ defined by $\mathcal{P}X = \pi_X$. The adjunction

$1 \dashv \{-\}$ defines the bijection $s : \mathbb{E}_I(1I, X) \cong \{f : I \to \{X\} \mid \pi_X \circ f = \mathrm{id}_I\}$ between vertical morphisms in $\mathbb{E}$ and sections in $\mathbb{B}$. For each $X, Y \in \mathbb{E}_I$, we have an isomorphism $\phi : \mathbb{E}_{\{X\}}(1\{X\}, \pi_X^* Y) \cong \mathbb{E}_I(X, Y)$. Consider the pullback square $\mathcal{P}(\overline{\pi_X}(Y))$ where $X, Y \in \mathbb{E}_I$. By the universal property of pullbacks, we have the symmetry isomorphism $\sigma_{X,Y} : \{\pi_X^* Y\} \to \{\pi_Y^* X\}$ as a unique morphism $\sigma_{X,Y}$ such that $\pi_{\pi_X^* Y} = \{\overline{\pi_Y}(X)\} \circ \sigma_{X,Y}$ and $\{\overline{\pi_X}(Y)\} = \pi_{\pi_Y^* X} \circ \sigma_{X,Y}$. Similarly, we have the diagonal morphism $\delta_X : \{X\} \to \{\pi_X^* X\}$ as a unique morphism δ_X such that $\pi_{\pi_X^* X} \circ \delta_X = \{\overline{\pi_X}(X)\} \circ \delta_X = \mathrm{id}_{\{X\}}$.

Let $p : \mathbb{E} \to \mathbb{B}$ be a comprehension category with unit and $q : \mathbb{D} \to \mathbb{B}$ be a fibration. The fibration q has *p-products* if $\pi_X^* : \mathbb{D}_{pX} \to \mathbb{D}_{\{X\}}$ has a right adjoint $\pi_X^* \dashv \prod_X$ for each $X \in \mathbb{E}$ and these adjunctions satisfy the BC (Beck-Chevalley) condition for each pullback square $\mathcal{P}f$ where $\mathcal{P}$ is a comprehension category induced by p and f is a cartesian morphism in $\mathbb{E}$. Similarly, we define *p-coproducts* by $\coprod_X \dashv \pi_X^*$ and p-equality by $\mathrm{Eq}_X \dashv \delta_X^*$ plus the BC condition for each cartesian morphism (see [10, Definition 9.3.5] for detail).

A comprehension category with unit $p : \mathbb{E} \to \mathbb{B}$ admits *products* (*coproducts*) if it has p-products (p-coproducts). The coproducts are *strong* if the canonical morphism $\kappa : \{Y\} \to \{\coprod_X Y\}$ defined by $\{\overline{\pi_X}(\coprod_X Y) \circ \eta^{\pi_X^* \dashv \coprod_X}\}$ is an isomorphism for each $X \in \mathbb{E}$ and $Y \in \mathbb{E}_{\{X\}}$. A *closed comprehension category* (CCompC) is a full comprehension category with unit that admits products and strong coproducts and has a terminal object in the base category. A *split closed comprehension category* (SCCompC) is a CCompC such that p is a split fibration, and the BC condition for products and coproducts holds strictly (i.e., canonical isomorphisms are identities). For example, the simple fibration $\mathsf{s}_{\mathbb{B}} : \mathbf{s}(\mathbb{B}) \to \mathbb{B}$ on a cartesian closed category $\mathbb{B}$ is a SCCompC (see [10, Theorem 10.5.5]). Another example of SCCompCs is the family fibration $\mathsf{fam}_{\mathbf{Set}} : \mathbf{Fam}(\mathbf{Set}) \to \mathbf{Set}$.

Fibred coproducts in a comprehension category with unit $p : \mathbb{E} \to \mathbb{B}$ are *strong* if the functor $\langle \{\iota_1\}^*, \{\iota_2\}^* \rangle : \mathbb{E}_{\{X+Y\}} \to \mathbb{E}_{\{X\}} \times \mathbb{E}_{\{Y\}}$ is fully faithful where $\iota_1 : X \to X + Y$ and $\iota_2 : Y \to X + Y$ are injections for fibred coproducts. Strong fibred coproducts are used to interpret fibred coproduct types $A + B$.

3 Lifting SCCompCs and Fibred Coproducts

In this section, we give a construction of liftings of SCCompCs with strong fibred coproducts from given SCCompCs with strong fibred coproducts for underlying types and posetal fibrations for predicate logic satisfying appropriate conditions.

3.1 Lifting SCCompCs

Let $p : \mathbb{E} \to \mathbb{B}$ be a SCCompC for underlying type systems. Let $q : \mathbb{P} \to \mathbb{B}$ be a posetal fibration with fibred finite products for predicate logic.

Definition 1. We define a category $\{\mathbb{E} \mid \mathbb{P}\}$ by the pullback of $q^{\to} : \mathbb{P}^{\to} \to \mathbb{B}^{\to}$ along $\mathcal{P} : \mathbb{E} \to \mathbb{B}^{\to}$ where the comprehension category $\mathcal{P}$ is induced by $p : \mathbb{E} \to \mathbb{B}$.

$$\begin{tikzcd}
\{\mathbb{E} \mid \mathbb{P}\} \arrow[r, "(q^{\to})^*\mathcal{P}"] \arrow[d, "\mathcal{P}^*(q^{\to})"'] \arrow[dr, phantom, "\lrcorner", very near start] & \mathbb{P}^{\to} \arrow[d, "q^{\to}"] \\
\mathbb{E} \arrow[r, "\mathcal{P}"] & \mathbb{B}^{\to}
\end{tikzcd}$$

That is, objects are tuples (X, P, Q) where $X \in \mathbb{E}$, $P \in \mathbb{P}_{pX}$, $Q \in \mathbb{P}_{\{X\}}$, and $Q \leq \pi_X^* P$; and morphisms are tuples $(f, g, h) : (X, P, Q) \to (X', P', Q')$ where $f : X \to X'$, $g : P \to P'$, $h : Q \to Q'$, $pf = qg$, and $\{f\} = qh$.

The intuition of this definition is as follows. For each object $(X, P, Q) \in \{\mathbb{E} \mid \mathbb{P}\}$, X represents a type $\Gamma \vdash A$ in the underlying type system, P represents a predicate on the context Γ, and Q represents the conjunction of a predicate on $\Gamma, v : A$ and the predicate P (thus $Q \leq \pi_X^* P$ is imposed). Note that $\mathcal{P}^*(q^{\to}) : \{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{E}$ is faithful because q is faithful.

Let $\{p \mid q\} : \{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{P}$ be a functor defined by $\mathrm{cod} \circ (q^{\to})^* \mathcal{P}$, that is, $(X, P, Q) \mapsto P$. The functor $\{p \mid q\}$ inherits most of the CCompC structure of $p : \mathbb{E} \to \mathbb{B}$.

Lemma 2. *The functor $\{p \mid q\} : \{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{P}$ is a split fibration. The cartesian lifting of $g : P' \to P$ is given by*

$$(\overline{qg}(X), g, \overline{\{\overline{qg}(X)\}}(Q) \circ \pi') : ((qg)^* X, P', \pi^*_{(qg)^* X} P' \wedge \{\overline{qg}(X)\}^* Q) \to (X, P, Q)$$

where π' is a projection for fibred products. □

Lemma 3. *The fibration $\{p \mid q\} : \{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{P}$ is a full comprehension category with unit that admits strong coproducts.*

Proof. The main idea is that the structure in the CCompC $p : \mathbb{E} \to \mathbb{B}$ can be lifted to $\{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{P}$. Here, we only show the definition of (object parts of) fibred terminal objects $1 : \mathbb{P} \to \{\mathbb{E} \mid \mathbb{P}\}$, the comprehension functor $\{-\} : \{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{P}$, and coproducts $\coprod_{(X,P,Q)} : \{\mathbb{E} \mid \mathbb{P}\}_Q \to \{\mathbb{E} \mid \mathbb{P}\}_P$ for each $(X, P, Q) \in \{\mathbb{E} \mid \mathbb{P}\}$.

$$1P = (1qP, P, \pi^*_{1qP} P) \quad \{(X, P, Q)\} = Q \qquad \coprod_{(X,P,Q)} (Y, Q, R) = (\coprod_X Y, P, (\kappa^{-1})^* R)$$

The rest of the proof is omitted. □

The existence of products in $\{p \mid q\}$ requires additional conditions.

Lemma 4. *If $q : \mathbb{P} \to \mathbb{B}$ has fibred exponentials and p-products (in addition to fibred finite products), then $\{p \mid q\} : \{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{P}$ admits products.*

Proof. We define $\prod_{(X,P,Q)} : \{\mathbb{E} \mid \mathbb{P}\}_Q \to \{\mathbb{E} \mid \mathbb{P}\}_P$ by

$$\prod_{(X,P,Q)} (Y, Q, R) = (\prod_X Y, P, \pi^*_{\prod_X Y} P \wedge \prod_{\pi^*_{\prod_X Y} X} \sigma^*_{\prod_X Y, X} (\pi^*_{\pi^*_X \prod_X Y} Q \Rightarrow \{\epsilon_Y^{\pi^*_X \dashv \prod_X}\}^* R)).$$

$$\begin{tikzcd}
Q \in \mathbb{P}_{\{X\}} \arrow[dr, "\pi^*_{\pi^*_X \prod_X Y}"] & & & \\
 & \mathbb{P}_{\{\pi^*_X \prod_X Y\}} \arrow[r, "\sigma^*_{\prod_X Y, X}"] & \mathbb{P}_{\{\pi^*_{\prod_X Y} X\}} \arrow[r, bend left, "\prod_{\pi^*_{\prod_X Y} X}"] \arrow[r, phantom, "\top"] & \mathbb{P}_{\{\prod_X Y\}} \arrow[l, bend left, "\pi^*_{\pi^*_{\prod_X Y} X}"] \\
R \in \mathbb{P}_{\{Y\}} \arrow[ur, "\{\epsilon_Y^{\pi^*_X \dashv \prod_X}\}^*"'] & & &
\end{tikzcd}$$

Then, this gives products in $\{p \mid q\}$ but we omit the lengthy proof. □

As a result, we get a lifting of SCCompCs over $p : \mathbb{E} \to \mathbb{B}$.

Theorem 5. *If $p : \mathbb{E} \to \mathbb{B}$ is a SCCompC and $q : \mathbb{P} \to \mathbb{B}$ is a fibred ccc that has p-products, then $\{p \mid q\} : \{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{P}$ is a SCCompC. Moreover, $(\mathcal{P}^*(q^{\to}), q) : \{p \mid q\} \to p$ is a morphism of SCCompCs, i.e., a split fibred functor that preserves the CCompC structure strictly.*

$$\begin{tikzcd}
\{\mathbb{E} \mid \mathbb{P}\} \arrow[r, "\mathcal{P}^*(q^{\to})"] \arrow[d, "\{p|q\}"] & \mathbb{E} \arrow[d, "p"] \\
\mathbb{P} \arrow[r, "q"] & \mathbb{B}
\end{tikzcd}$$

Proof. By Lemma 3 and Lemma 4. A terminal object in $\mathbb{P}$ exists because $\mathbb{B}$ has a terminal object and $q : \mathbb{P} \to \mathbb{B}$ has fibred terminal objects. It is almost obvious that $(\mathcal{P}^*(q^{\to}), q)$ preserves the structure of CCompCs. □

Example 6. Consider the simple fibration $\mathsf{s}_{\mathbf{Set}} : \mathbf{s}(\mathbf{Set}) \to \mathbf{Set}$ and the subobject fibration $\mathsf{sub}_{\mathbf{Set}} : \mathbf{Sub}(\mathbf{Set}) \to \mathbf{Set}$ (see [10, §1.3]). Objects in $\{\mathbf{s}(\mathbf{Set}) \mid \mathbf{Sub}(\mathbf{Set})\}$ are tuples $((I, X), P, Q)$ where $(I, X) \in \mathbf{s}(\mathbf{Set})$, $P \subseteq I$, and $Q \subseteq P \times X \subseteq I \times X$, and morphisms are those in $\mathbf{s}(\mathbf{Set})$ that preserve predicates. In $\{\mathsf{s}_{\mathbf{Set}} \mid \mathsf{sub}_{\mathbf{Set}}\} : \{\mathbf{s}(\mathbf{Set}) \mid \mathbf{Sub}(\mathbf{Set})\} \to \mathbf{Sub}(\mathbf{Set})$, products are given by

$$\prod_{((I,X),P,Q)} ((I \times X, Y), Q, R) = \big((I, X \Rightarrow Y), P, \{(i, f) \in I \times (X \Rightarrow Y) \mid \\ i \in P \wedge \forall x \in X, (i, x) \in Q \implies ((i, x), f(x)) \in R\}\big). \tag{2}$$

Example 7. Let $\mathsf{erel} : \mathbf{ERel} \to \mathbf{Set}$ be the fibration of endorelations defined by change-of-base from $\mathbf{Sub}(\mathbf{Set}) \to \mathbf{Set}$ along the functor $X \mapsto X \times X$. The fibration erel is a fibred ccc and has products (i.e. right adjoints of reindexing functors that satisfy the BC condition for each pullback square). Therefore, erel has p-products for any comprehension category with unit p. If we apply Theorem 5 to erel and the simple fibration $\mathsf{s}_{\mathbf{Set}} : \mathbf{s}(\mathbf{Set}) \to \mathbf{Set}$, then products are defined similarly to Example 6.

Example 8. Consider the family fibration $\mathsf{fam}_{\mathbf{Set}} : \mathbf{Fam}(\mathbf{Set}) \to \mathbf{Set}$ [10, Def 1.2.1] and the subobject fibration $\mathsf{sub}_{\mathbf{Set}} : \mathbf{Sub}(\mathbf{Set}) \to \mathbf{Set}$. Objects in $\{\mathbf{Fam}(\mathbf{Set}) \mid \mathbf{Sub}(\mathbf{Set})\}$ are tuples $((I, X), P, Q)$ where $(I, X) \in \mathbf{Fam}(\mathbf{Set})$, $P \subseteq I$, and $Q \subseteq \coprod_{i \in P} Xi \subseteq \coprod_{i \in I} Xi$. Note that subsets $Q \subseteq \coprod_{i \in I} Xi$ have a one-to-one correspondence with families of subsets $(Qi \subseteq Xi)_{i \in I}$ when we define $Qi = \iota_i^*(Q)$ where $\iota_i : Xi \to \coprod_{i \in I} Xi$ is the i-th injection. So, we often identify Q with the family of subsets $Qi \subseteq Xi$. We get products in $\{\mathsf{fam}_{\mathbf{Set}} \mid \mathsf{sub}_{\mathbf{Set}}\} : \{\mathbf{Fam}(\mathbf{Set}) \mid \mathbf{Sub}(\mathbf{Set})\} \to \mathbf{Sub}(\mathbf{Set})$ by modifying (2) for dependent functions.

3.2 Lifting Fibred Coproducts

A sufficient condition for $\{p \mid q\} : \{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{P}$ to have strong fibred coproducts is given by the following lemma, which is analogous to [9, Prop. 4.5.8].

Lemma 9. *If (1) $p : \mathbb{E} \to \mathbb{B}$ is a CCompC that has strong fibred coproducts (2) for each $X, Y \in \mathbb{E}_I$, $X', Y' \in \mathbb{E}_{I'}$, $u : I \to I'$, and pair of cartesian liftings $f : X \to X'$ and $g : Y \to Y'$ over u, the following two squares are pullbacks*

$$\begin{tikzcd}
\{X\} \arrow[r, "\{\iota_1\}"] \arrow[d, "\{f\}"'] \arrow[dr, phantom, "\lrcorner"] & \{X+Y\} \arrow[d, "\{f+g\}"] & \{Y\} \arrow[l, "\{\iota_2\}"'] \arrow[d, "\{g\}"] \arrow[dl, phantom, "\llcorner"] \\
\{X'\} \arrow[r, "\{\iota_1\}"] & \{X'+Y'\} & \{Y'\} \arrow[l, "\{\iota_2\}"']
\end{tikzcd}$$

(3) $q : \mathbb{P} \to \mathbb{B}$ is a fibred distributive category (4) for each $X, Y \in \mathbb{E}_I$ and $Z \in \mathbb{E}_{\{X+Y\}}$, q has cocartesian liftings of $\{\iota_1\} : \{X\} \to \{X+Y\}$, $\{\iota_2\} : \{Y\} \to \{X+Y\}$, $\{\overline{\{\iota_1\}}(Z)\} : \{\{\iota_1\}^* Z\} \to \{Z\}$, and $\{\overline{\{\iota_2\}}(Z)\} : \{\{\iota_2\}^* Z\} \to \{Z\}$ that satisfy the BC condition for each pullback squares and Frobenius, then $\{p \mid q\} : \{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{P}$ has strong fibred coproducts, and the fibred functor $(\mathcal{P}^*(q^{\to}), q) : \{p \mid q\} \to p$ strictly preserves fibred coproducts.

Proof. We define fibred coproducts by $(X, P, Q) + (Y, P, R) = (X+Y, P, \{\iota_1\}_! Q \vee \{\iota_2\}_! R)$. We omit the rest of the proof. □

Note that if q is fibred bicartesian closed, then q is a fibred distributive category.

Example 10. Consider $\mathsf{s}_{\mathbf{Set}} : \mathbf{s}(\mathbf{Set}) \to \mathbf{Set}$ and $\mathsf{sub}_{\mathbf{Set}} : \mathbf{Sub}(\mathbf{Set}) \to \mathbf{Set}$ (recall Example 6). This combination satisfies four conditions in Lemma 9. Fibred coproducts in $\{\mathbf{s}(\mathbf{Set}) \mid \mathbf{Sub}(\mathbf{Set})\} \to \mathbf{Sub}(\mathbf{Set})$ are defined as follows.

$$((I, X), P, Q) + ((I, Y), P, R) = ((I, X+Y), P, \{(i, x) \mid (i, x) \in Q \vee (i, x) \in R\})$$

4 Lifting Monads on SCCompCs

Suppose we have a SCCompC $p : \mathbb{E} \to \mathbb{B}$ and a posetal fibration $q : \mathbb{P} \to \mathbb{B}$ as ingredients for $\{p \mid q\} : \{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{P}$ in Theorem 5. We explain how to construct a fibred monad on $\{p \mid q\} : \{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{P}$ from monads on p and q.

First, we assume that a monad T on $\mathbb{B}$ and a fibred monad $\hat{T}$ on $p : \mathbb{E} \to \mathbb{B}$ are given. These monads are intended to represent the same computational effects in underlying type systems, but T is more "primitive" than $\hat{T}$, and $\hat{T}$ is induced from T in some natural way. For example, we can use the maybe monad or the powerset monad on **Set** as T and define $\hat{T}$ by $(I, X) \mapsto (I, TX)$ on the simple fibration $\mathbf{s}(\mathbf{Set}) \to \mathbf{Set}$. In such a situation, we often have an oplax monad morphism (Definition 11) $\theta : \{\hat{T}(-)\} \to T\{-\}$. Intuitively, θ extends the action of $\hat{T}$ on types to contexts, just like strengths of strong monads. We also need a lifting $\dot{T}$ of T along $q : \mathbb{P} \to \mathbb{B}$ to specify a mapping from predicates on values in $X \in \mathbb{B}$ to predicates on computations in TX [1]. Given all these ingredients and some additional conditions, we define a fibred monad on $\{p \mid q\} : \{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{P}$, which is a lifting of the fibred monad $\hat{T}$ on $p : \mathbb{E} \to \mathbb{B}$.

Definition 11 (oplax monad morphism). Let $\mathbb{C}, \mathbb{D}$ be categories, $F : \mathbb{C} \to \mathbb{D}$ be a functor, and (S, η^S, μ^S), (T, η^T, μ^T) be monads on $\mathbb{C}$ and $\mathbb{D}$, respectively. A natural transformation $\theta : FS \to TF$ is an *oplax monad morphism* if θ respects units and multiplications.

$$\begin{tikzcd}
FX \arrow[d, "F\eta^S_X"'] \arrow[dr, "\eta^T_{FX}"] & \\
FSX \arrow[r, "\theta_X"] & TFX
\end{tikzcd}
\qquad
\begin{tikzcd}
FS^2X \arrow[r, "\theta_{SX}"] \arrow[d, "F\mu^S_X"'] & TFSX \arrow[r, "T\theta_X"] & T^2FX \arrow[d, "\mu^T_{FX}"] \\
FSX \arrow[rr, "\theta_X"] & & TFX
\end{tikzcd}$$

Theorem 12. *Let T be a monad on $\mathbb{B}$, $\hat{T}$ be a fibred monad on $p : \mathbb{E} \to \mathbb{B}$ in the 2-category $\mathbf{Fib}_{\mathbb{B}}$ of fibrations over $\mathbb{B}$, $\theta : \{\hat{T}(-)\} \to T\{-\}$ be an oplax monad morphism, and $\dot{T}$ be a fibred lifting [1] of T along $q : \mathbb{P} \to \mathbb{B}$. If*

$$\pi^*_{\hat{T}X} P \wedge \theta^*_X \dot{T}Q \leq \theta^*_X \dot{T}(\pi^*_X P \wedge Q) \tag{3}$$

holds for each $X \in \mathbb{E}$, $P \in \mathbb{P}_{pX}$ and $Q \in \mathbb{P}_{\{X\}}$, then there exists a fibred monad S on $\{p \mid q\} : \{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{P}$ in $\mathbf{Fib}_{\mathbb{P}}$ such that the fibred functor $\{p \mid q\} \to p$ in Theorem 5 is a fibred monad morphism from S to $\hat{T}$.

Proof. We define $S(X, P, Q) = (\hat{T}X, P, \pi^*_{\hat{T}X} P \wedge \theta^* \dot{T}Q)$. Then the monad structure of $\hat{T}$ lifts to S. The assumption (3) is required to prove that S is fibred.

$$\begin{tikzcd} \mathbb{P} \arrow[d, "q"] & \theta^* \dot{T}Q \arrow[r, "\overline{\theta}(\dot{T}Q)"] & \dot{T}Q \\ \mathbb{B} & \{\hat{T}X\} \arrow[r, "\theta"] & T\{X\} \end{tikzcd}$$
□

Example 13. Any strong monad T on a CCC $\mathbb{B}$ gives rise to a split fibred monad $\hat{T}$ on the simple fibration $\mathsf{s}_{\mathbb{B}} : \mathbf{s}(\mathbb{B}) \to \mathbb{B}$ (actually, there is a one-to-one correspondence [10, Ex.2.6.10]). The monad $\hat{T}$ is defined by $(I, X) \mapsto (I, TX)$. An oplax monad morphism $\theta : I \times TX \to T(I \times X)$ is given by the strength.

Now consider the case where $\mathbb{B} = \mathbf{Set}$. Since the strength for the monad T on $\mathbf{Set}$ is given uniquely [17, Proposition 3.4], we can prove that (3) holds for any fibred lifting of T along the subobject fibration $\mathsf{sub}_{\mathbf{Set}} : \mathbf{Sub}(\mathbf{Set}) \to \mathbf{Set}$.

Let T be the maybe monad $(-) + \{*\}$. There are two fibred liftings of T:

$$\dot{T}_1(P \subseteq I) = (P + \{*\} \subseteq I + \{*\}) \qquad \dot{T}_2(P \subseteq I) = (P \subseteq I + \{*\})$$

for each $(P \subseteq I) \in \mathbf{Sub}(\mathbf{Set})$. The lifting $\dot{T}_1$ corresponds to partial correctness, and $\dot{T}_2$ corresponds to total correctness. The fibred monads on $\{\mathsf{s}_{\mathbf{Set}} \mid \mathsf{sub}_{\mathbf{Set}}\}$ defined in Theorem 12 from $\dot{T}_1$ and $\dot{T}_2$ are given by

$$((I, X), P, Q) \mapsto \big((I, X + \{*\}), P, \{(i, x) \mid (i \in P \wedge x = *) \vee (i, x) \in Q\}\big)$$
$$((I, X), P, Q) \mapsto \big((I, X + \{*\}), P, \{(i, x) \mid (i, x) \in Q\}\big)$$

respectively. Here, we leave the left/right injection of coproducts implicit.

Example 14. For each monad T on $\mathbf{Set}$, we have a split fibred monad on the family fibration $\mathbf{Fam}(\mathbf{Set}) \to \mathbf{Set}$ defined by $\hat{T}(I, X) = (I, T \circ X)$. We have an oplax monad morphism $\theta : \coprod_{i \in I} TXi \to T \coprod_{i \in I} Xi$ defined by the cotupling $[(T\iota_i)_{i \in I}] : \coprod_{i \in I} TXi \to T \coprod_{i \in I} Xi$ where $\iota_i : Xi \to \coprod_{i \in I} Xi$ is the i-th injection. The condition (3) holds for any fibred lifting of T along the subobject fibration $\mathbf{Sub}(\mathbf{Set}) \to \mathbf{Set}$. Moreover, we have $\iota_i^* \theta^* \dot{T}Q = \dot{T}\iota_i^* Q$ for each $Q \in \mathbf{Sub}(\mathbf{Set})_{\coprod_{i \in I} Xi}$, so the monad in Theorem 12 is given by

$$\big((I, X), P, (Qi \subseteq Xi)_{i \in I}\big) \mapsto \big((I, T \circ X), P, (\dot{T}Qi \subseteq TXi)_{i \in I}\big).$$

5 Soundness

We consider a concrete dependent refinement type system with computational effects and define sound semantics to show that the SCCompC defined in Theorem 5 has sufficient structures for dependent refinement types. Here, we consider two type systems. One is an underlying type system that is a fragment of EMLTT [2–4]. The other is a refinement of the underlying type system that has refinement types $\{v : A \mid p\}$ and a subtyping relation $\Gamma \vdash A <: B$ induced by logical implication. The two type systems share a common syntax for terms while types are more expressive in the refinement type system. We consider liftings of fibred adjunction models to interpret the refinement type system. Here, Theorem 12 can be used to obtain a lifting of fibred adjunction models via Eilenberg-Moore construction. We prove a soundness theorem that claims if a term is well-typed in the refinement type system, then the interpretation of the term has a lifting along the morphism of CCompCs defined in Theorem 5.

5.1 Underlying Type System

We define the underlying dependent type system by a slightly modified version of a fragment of EMLTT [2–4]. We remove some of the types and terms from the original for simplicity. We parameterize our type system with a set of base type constructors (ranged over by b) and a set of value constants (ranged over by c) for convenience.

We define value types $(A, B, \dots)$, computation types $(\underline{C}, \underline{D}, \dots)$, contexts $(\Gamma, \dots)$, value terms $(V, W, \dots)$, and computation terms $(M, N, \dots)$ as follows.

$$\begin{aligned}
A &\coloneqq 1 \mid b_A(V) \mid \Sigma x{:}A.B \mid U\underline{C} \mid A + B \\
\underline{C} &\coloneqq FA \mid \Pi x{:}A.\underline{C} \qquad\qquad \Gamma \coloneqq \diamond \mid \Gamma, x : A \\
V &\coloneqq x \mid * \mid c_A \mid \langle V, W\rangle_{(x:A).B} \mid \mathbf{thunk}\ M \mid \mathbf{inl}_{A+B}\ V \mid \mathbf{inr}_{A+B}\ V \\
M &\coloneqq \mathbf{return}\ V \mid M\ \mathbf{to}\ x : A\ \mathbf{in}_{\underline{C}}\ N \mid \mathbf{force}_{\underline{C}}\ V \mid \lambda x : A.M \mid M(V)_{(x:A).\underline{C}} \mid \\
&\qquad \mathbf{pm}\ V\ \mathbf{as}\ \langle x : A, y : B\rangle\ \mathbf{in}_{z.\underline{C}}\ M \mid \\
&\qquad \mathbf{case}\ V\ \mathbf{of}_{z.\underline{C}}\ (\mathbf{inl}\ (x : A) \mapsto M, \mathbf{inr}\ (y : B) \mapsto N)
\end{aligned}$$

We implicitly assume that variables in Γ are mutually different. We use many type annotations in the syntax of terms for a technical reason, but we might omit them if they are clear from the context. We define substitution $A[V/x]$, $\underline{C}[V/x]$, $W[V/x]$, and $M[V/x]$ as usual.

For each type constructor b, let $\mathrm{arg}(b)$ be a closed value type of the argument of b. We write $b : A \to \mathrm{Type}$ if $A = \mathrm{arg}(b)$. For each value constant c, let $\mathrm{ty}(c)$ be a closed value type of c.

We have several kinds of judgements: well-formed contexts $\vdash \Gamma$; well-formed (value or computation) types $\Gamma \vdash A$, $\Gamma \vdash \underline{C}$; well-typed (value or computation) terms $\Gamma \vdash V : A$, $\Gamma \vdash M : \underline{C}$; and definitional equalities for contexts, types and terms $\vdash \Gamma_1 = \Gamma_2$, $\Gamma \vdash A = B$, $\Gamma \vdash \underline{C} = \underline{D}$, $\Gamma \vdash V = W : A$, $\Gamma \vdash M = N : \underline{C}$.

Typing rules are basically the same as EMLTT. Rules for base type constructors and value constants are shown in Fig. 2

$$\frac{\vdash \Gamma \qquad \diamond \vdash \mathrm{ty}(c)}{\Gamma \vdash c_{\mathrm{ty}(c)} : \mathrm{ty}(c)} \qquad \frac{\begin{array}{c} b : A \to \mathrm{Type} \\ \diamond \vdash A \qquad \Gamma \vdash V : A \end{array}}{\Gamma \vdash b_A(V)} \qquad \frac{\begin{array}{c} b : A \to \mathrm{Type} \qquad \diamond \vdash A \\ \Gamma \vdash V = W : A \end{array}}{\Gamma \vdash b_A(V) = b_A(W)}$$

Fig. 2. Some typing rules for the underlying type system.

Semantics. We use fibred adjunction models to interpret terms and types. We adapt the definition for our fragment of EMLTT as follows.

Definition 15 (Fibred adjunction models). A *fibred adjunction model* is a fibred adjunction $F \dashv U : r \to p$ where $p : \mathbb{E} \to \mathbb{B}$ is a SCCompC with strong fibred coproducts and $r : \mathbb{C} \to \mathbb{B}$ is a fibration with p-products.

The Eilenberg-Moore fibration of a CCompC $p : \mathbb{E} \to \mathbb{B}$ inherits products in p [2, Theorem 4.3.24] and thus gives an example of fibred adjunction models.

Lemma 16. *Given a SCCompC $p : \mathbb{E} \to \mathbb{B}$ with strong fibred products and a split fibred monad T on p, then the Eilenberg-Moore adjunction of T is a fibred adjunction model.* □

We assume that a fibred adjunction model $F \dashv U : r \to p$ between $p : \mathbb{E} \to \mathbb{B}$ and $r : \mathbb{C} \to \mathbb{B}$ is given and that interpretations of base type constructors $[\![b]\!] \in \mathbb{E}$ and value constants $[\![c]\!] \in \mathbb{E}_1(1, X)$ (for some $X \in \mathbb{E}_1$) are given. We define a partial interpretation $[\![-]\!]$ of the following form for raw syntax.

$$\begin{tikzcd} \mathbb{E} \arrow[rr, bend left, "F"] \arrow[rr, phantom, "\perp"] \arrow[dr, "p"'] & & \mathbb{C} \arrow[ll, bend left, "U"] \arrow[dl, "r"] \\ & \mathbb{B} & \end{tikzcd}$$

$$\begin{array}{ll} [\![\Gamma]\!] \in \mathbb{B} \qquad [\![\Gamma; A]\!] \in \mathbb{E}_{[\![\Gamma]\!]} & [\![\Gamma; \underline{C}]\!] \in \mathbb{C}_{[\![\Gamma]\!]} \\ [\![\Gamma; V]\!] \in \mathbb{E}_{[\![\Gamma]\!]}(1[\![\Gamma]\!], A) & \text{for some } A \\ [\![\Gamma; M]\!] \in \mathbb{E}_{[\![\Gamma]\!]}(1[\![\Gamma]\!], UC) & \text{for some } C \in \mathbb{C} \end{array}$$

Most of the definition of $[\![-]\!]$ are the same as [2]. For base type constructors b and value constants c, we define $[\![-]\!]$ as follows.

$$[\![\Gamma; b_A(V)]\!] = (s[\![\Gamma; V]\!])^* \{\overline{!_{[\![\Gamma]\!]}}([\![\diamond; A]\!])\}^* [\![b]\!] \qquad [\![\Gamma; c_A]\!] = !^*_{[\![\Gamma]\!]}[\![c]\!]$$

Here, left-hand sides are defined if right-hand sides are defined.

Proposition 17 (Soundness). *Assume that $[\![b]\!] \in \mathbb{E}_{\{[\![\diamond;A]\!]\}}$ holds for each $b : A \to \mathrm{Type}$ such that $[\![\diamond; A]\!]$ is defined, and $[\![c]\!] \in \mathbb{E}_1(1, [\![\diamond; \mathrm{ty}(c)]\!])$ holds if $[\![\diamond; \mathrm{ty}(c)]\!] \in \mathbb{E}_1$ is defined. Interpretations $[\![-]\!]$ of well-formed contexts and types and well-typed terms are defined. If two contexts, types, or terms are definitionally equal, then their interpretations are equal.* □

5.2 Predicate Logic

We define syntax for logical formulas by

$$p = \top \mid p \wedge q \mid p \Rightarrow q \mid \forall x : A.p \mid V =_A W \mid a(V)$$

$$\frac{\Gamma \vdash V : A \qquad \Gamma \vdash W : A}{\Gamma \vdash V =_A W : \mathrm{Prop}} \qquad \frac{a : A \to \mathrm{Prop} \qquad \diamond \vdash A \qquad \Gamma \vdash V : A}{\Gamma \vdash a(V) : \mathrm{Prop}}$$

Fig. 3. Some rules for well-formed predicates.

where a ranges over predicate symbols. Here, we added $\top$ and $V =_A W$ for typing rule for the unique value of the unit type and variables of base types (i.e. for selfification [18]), respectively, which we describe later. However, there is a large amount of freedom to choose the syntax of logical formulas. The least requirement here is that logical formulas can be interpreted in a posetal fibration $q : \mathbb{P} \to \mathbb{B}$, and interpretations of logical formulas admit semantic weakening, substitution, and conversion in the sense of [2, Proposition 5.2.4, 5.2.6]. So, we can almost freely add or remove logical connectives and quantifiers as long as $q : \mathbb{P} \to \mathbb{B}$ admits them.

We define a standard judgement of well-formedness for logical formulas. Some of the rules for well-formedness are shown in Fig. 3

Logical formulas are interpreted in the fibration $q : \mathbb{P} \to \mathbb{B}$. We assume that interpretation $[\![a]\!] \in \mathbb{P}_{\{[\![\diamond; A]\!]\}}$ for each predicate symbol $a : A \to \mathrm{Prop}$ is given. The interpretation $[\![\Gamma \vdash p]\!] \in \mathbb{P}_{[\![\Gamma]\!]}$ is standard and defined inductively for each well-formed formulas. For example:

$$[\![\Gamma \vdash V =_A W]\!] = (s[\![\Gamma; V]\!])^*(s(\pi^*_{[\![\Gamma; A]\!]}[\![\Gamma; W]\!]))^*\mathrm{Eq}(\top\{[\![\Gamma; A]\!]\})$$
$$[\![\Gamma \vdash a(V)]\!] = s([\![\Gamma; V]\!])^*\{\overline{!_{[\![\Gamma]\!]}}([\![\diamond; A]\!])\}^*[\![a]\!]$$

where $a : A \to \mathrm{Prop}$ is a predicate symbol and s is the bijection defined in §2.

5.3 Refinement Type System

We refine the underlying type system by adding predicates to base types and the unit type. From now on, we use subscript A_u for types in the underlying type system to distinguish them from types in the refinement type system.

$$A \coloneqq \{v : b_{A_u}(V) \mid p\} \mid \{v : 1 \mid p\} \mid \Sigma x{:}A.B \mid U\underline{C} \mid A + B$$
$$\underline{C} \coloneqq FA \mid \Pi x{:}A.\underline{C} \qquad\qquad \Gamma \coloneqq \diamond \mid \Gamma, x : A$$

We use the same definition of terms as the underlying type system and the same set of base type constructors and value constants. Argument types of base type constructors $b : A_u \to \mathrm{Type}$ are also the same, but types $\mathrm{ty}(c)$ assigned to value constants c are redefined as refinement types. Given a type A (or $\underline{C}$) in the refinement type system, we define its underlying type $|A|$ (or $|\underline{C}|$) by induction where predicates are eliminated in the base cases.

$$|\{v : b_{A_u}(V) \mid p\}| = b_{A_u}(V) \qquad |\{v : 1 \mid p\}| = 1$$

Underlying contexts $|\Gamma|$ are also defined by $|\diamond| = \diamond$ and $|\Gamma, x : A| = |\Gamma|, x : |A|$.

$$\frac{b : A_u \to \mathrm{Type} \qquad \vdash \Gamma \qquad |\Gamma| \vdash b_{A_u}(V) \qquad |\Gamma|, v : b_{A_u}(V) \vdash p : \mathrm{Prop}}{\Gamma \vdash \{v : b_{A_u}(V) \mid p\}} \qquad \frac{\vdash \Gamma \qquad |\Gamma| \vdash b_{A_u}(V) = b_{A_u}(W) \qquad \Gamma; v : b_{A_u}(V) \mid p \vdash q}{\Gamma \vdash \{v : b_{A_u}(V) \mid p\} <: \{v : b_{A_u}(W) \mid q\}}$$

$$\frac{\vdash \Gamma_1, x : \{v : b_{A_u}(V) \mid p\}, \Gamma_2}{\Gamma_1, x : \{v : b_{A_u}(V) \mid p\}, \Gamma_2 \vdash x : \{v : b_{A_u}(V) \mid v = x\}} \qquad \frac{\vdash \Gamma \qquad \diamond \vdash \mathrm{ty}(c)}{\Gamma \vdash c_{|\mathrm{ty}(c)|} : \mathrm{ty}(c)}$$

$$\frac{\Gamma \vdash A_2 <: A_1 \qquad \Gamma, x : A_1 \vdash \underline{C}_1 \qquad \Gamma, x : A_2 \vdash \underline{C}_1 <: \underline{C}_2}{\Gamma \vdash \Pi x{:}A_1.\underline{C}_1 <: \Pi x{:}A_2.\underline{C}_2} \qquad \frac{\Gamma_2 \vdash V : A \qquad \vdash \Gamma_1 <: \Gamma_2 \qquad \Gamma_1 \vdash A <: B}{\Gamma_1 \vdash V : B} \qquad \frac{\vdash \Gamma}{\Gamma \vdash * : \{v : 1 \mid \top\}}$$

$$\frac{\vdash \Gamma \qquad |\Gamma|, v : 1 \vdash p : \mathrm{Prop}}{\Gamma \vdash \{v : 1 \mid p\}} \qquad \frac{\vdash \Gamma \qquad \Gamma; v : 1 \mid p \vdash q}{\Gamma \vdash \{v : 1 \mid p\} <: \{v : 1 \mid q\}}$$

Fig. 4. Some typing rules for the refinement type system.

Judgements in the refinement type system are as follows. We have judgements for well-formedness or well-typedness for contexts, types and terms in the refinement type system, which are denoted in the same way as the underlying type system. We do not consider definitional equalities for terms because they are the same as the underlying type system. Instead, we add judgements for subtyping between types and contexts. They are denoted by $\vdash \Gamma_1 <: \Gamma_2$ for context, $\Gamma \vdash A <: B$ for value types, and $\Gamma \vdash \underline{C} <: \underline{D}$ for computation types.

Most of term and type formation rules are similar to the underlying type system. We listed some of the non-trivial modifications of typing rules in Fig. 4. We add typing rules for $\{v : b_{B_u}(V) \mid p\}$ and $\{v : 1 \mid p\}$. Subtyping for these types are defined by judgements $\Gamma; v : A_u \mid p \vdash q$ for logical implication. Here, $\Gamma; v : A_u \mid p \vdash q$ means "assumptions in Γ and p implies q" where p and q are well-formed formulas in the context $|\Gamma|, v : A_u$. We do not specify derivation rules for the judgement $\Gamma; v : A_u \mid p \vdash q$ but assume soundness of the judgement (explained later). We allow "selfification" [18] for variables of base types. Subtyping for $\Sigma x{:}A.B$, $U\underline{C}$, FA, and $\Pi x{:}A.\underline{C}$ are defined covariantly except the argument type A of $\Pi x{:}A.\underline{C}$, which is contravariant. We have the rule of subsumption. Value constants are typed with a refined type assignment $\mathrm{ty}(c)$. The unique value $*$ of the unit type has type $\{v : 1 \mid \top\}$.

Lemma 18. *If we eliminate predicates in the refinement types from well-formed contexts, types and terms, then we get well-formed contexts, types and terms of the underlying type system.*

- *If* $\vdash \Gamma$*, then* $\vdash |\Gamma|$*. If* $\Gamma \vdash A$*, then* $|\Gamma| \vdash |A|$*. If* $\Gamma \vdash \underline{C}$*, then* $|\Gamma| \vdash |\underline{C}|$*.*
- *If* $\vdash \Gamma_1 <: \Gamma_2$*, then* $\vdash |\Gamma_1| = |\Gamma_2|$*. If* $\Gamma \vdash A <: B$*, then* $|\Gamma| \vdash |A| = |B|$*. If* $\Gamma \vdash \underline{C} <: \underline{D}$*, then* $|\Gamma| \vdash |\underline{C}| = |\underline{D}|$*.*

Proof. By induction on the derivation of judgements. Each typing rule in the refinement type system has a corresponding rule in the underlying system. □

Example 19. We can express conditional branching using the elimination rule of the fibred coproduct type $1 + 1$. For example, assume we have a base type constructor int : $1 \to$ Type for integers and a value constant for comparison.

$$(\leq) : U(\mathit{\Pi} x{:}\mathrm{int}.\mathit{\Pi} y{:}\mathrm{int}.F(\{v : 1 \mid x \leq y\} + \{v : 1 \mid x > y\}))$$

We can define **if** $x \leq y$ **then** M **else** N to be a syntax sugar for

$$(x \leq' y)\ \textbf{to}\ z\ \textbf{in}\ (\textbf{case}\ z\ \textbf{of}\ (\textbf{inl}\ v \mapsto M, \textbf{inr}\ v \mapsto N))$$

where $(\leq') =$ **force** $(\leq)$. Note that M and N are typed in contexts that have $v : \{v : 1 \mid x \leq y\}$ or $v : \{v : 1 \mid x > y\}$ depending on the result of comparison.

5.4 Semantics

Definition 20 (lifting of fibred adjunction models). Suppose that we have two fibred adjunction models $F \dashv U : q \to p$ between $p : \mathbb{E} \to \mathbb{B}$ and $q : \mathbb{C} \to \mathbb{B}$ and $\dot{F} \dashv \dot{U} : s \to r$ between $r : \mathbb{U} \to \mathbb{P}$ and $s : \mathbb{D} \to \mathbb{P}$. The fibred adjunction model $\dot{F} \dashv \dot{U}$ is a *lifting* of $F \dashv U$ if there exists functors $u : \mathbb{U} \to \mathbb{E}$, $v : \mathbb{D} \to \mathbb{C}$, and $t : \mathbb{P} \to \mathbb{B}$ such that these functors strictly preserve all structures of $\dot{F} \dashv \dot{U}$ to those of $F \dashv U$. That is, $(u, t) : r \to p$ and $(v, t) : s \to q$ are split fibred functors, the pair of fibred functor (u, t) and (v, t) is a map of adjunctions in the 2-category **Fib**, (u, t) strictly preserves the CCompC structure and fibred coproducts, and (v, t) maps r-products to p-products in the strict sense.

We assume that a lifting of fibred adjunction models is given as follows.

$$\begin{tikzcd}
\mathbb{E} \arrow[rr, bend left, "F"] \arrow[rr, phantom, "\perp"] \arrow[dr, "p"'] & & \mathbb{C} \arrow[ll, bend left, "U"] \arrow[dl] \\
& \mathbb{B} &
\end{tikzcd}
\quad
\begin{tikzcd}
\{\mathbb{E} \mid \mathbb{P}\} \arrow[rr, bend left, "\dot{F}"] \arrow[rr, phantom, "\perp"] \arrow[dr, "\{p|q\}"'] & & \mathbb{D} \arrow[ll, bend left, "\dot{U}"] \arrow[dl] \\
& \mathbb{P} &
\end{tikzcd}
\quad
\begin{tikzcd}
\{\mathbb{E} \mid \mathbb{P}\} \arrow[r, "u"] \arrow[d, "\{p|q\}"] & \mathbb{E} \arrow[d, "p"] \\
\mathbb{P} \arrow[r, "q"] & \mathbb{B}
\end{tikzcd}
\quad
\begin{tikzcd}
\mathbb{D} \arrow[r, "v"] \arrow[d] & \mathbb{C} \arrow[d] \\
\mathbb{P} \arrow[r, "q"] & \mathbb{B}
\end{tikzcd}
\tag{4}$$

Here, we assume more than just a lifting of fibred adjunction models by requiring the specific SCCompC $\{p \mid q\}$ with strong fibred coproducts, and the split functor $(u, q) : \{p \mid q\} \to p$ defined in Theorem 5 and Lemma 9. The underlying fibred adjunction model $F \dashv U$ is used for the underlying type system in §5.1, and $q : \mathbb{P} \to \mathbb{B}$ is for predicate logic in §5.2. One way to obtain such liftings of fibred adjunction models is to apply the Eilenberg-Moore construction to the monad morphism in Theorem 12, but in general we do not restrict $\mathbb{C}$ and $\mathbb{D}$ to be Eilenberg-Moore categories. We further assume that q has p-equalities to interpret logical formulas of the form $V =_A W$.

We define partial interpretation of refinement types $[\![\mathit{\Gamma}]\!] \in \mathbb{P}$, $[\![\mathit{\Gamma}; A]\!] \in \{\mathbb{E} \mid \mathbb{P}\}_{[\![\mathit{\Gamma}]\!]}$, and $[\![\mathit{\Gamma}; \underline{C}]\!] \in \mathbb{D}_{[\![\mathit{\Gamma}]\!]}$ similarly to the underlying type system but with the following modification. Here, we make use of the definition of $\{\mathbb{E} \mid \mathbb{P}\}$.

$$\begin{aligned}
[\![\mathit{\Gamma}; \{v : b(V) \mid p\}]\!] &= \left([\![|\mathit{\Gamma}|; b(V)]\!], [\![\mathit{\Gamma}]\!], \pi^*_{[\![|\mathit{\Gamma}|; b(V)]\!]}[\![\mathit{\Gamma}]\!] \wedge [\![|\mathit{\Gamma}|, v : b(V) \vdash p]\!]\right) \\
[\![\mathit{\Gamma}; \{v : 1 \mid p\}]\!] &= \left([\![|\mathit{\Gamma}|; 1]\!], [\![\mathit{\Gamma}]\!], \pi^*_{[\![|\mathit{\Gamma}|; 1]\!]}[\![\mathit{\Gamma}]\!] \wedge [\![|\mathit{\Gamma}|, v : 1 \vdash p]\!]\right)
\end{aligned}$$

For each $(X, P, Q), (X', P', Q') \in \{\mathbb{E} \mid \mathbb{P}\}$, we define a semantic subtyping relation $(X, P, Q) <: (X', P', Q')$ by the conjunction of $X = X'$, $P = P'$, and

$Q \leq Q'$. In other words, we have $(X, P, Q) <: (X', P', Q')$ if and only if there exists a morphism $(\mathrm{id}_X, \mathrm{id}_P, h) : (X, P, Q) \to (X', P', Q')$ that is mapped to identities by $u : \{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{E}$ and $\{p \mid q\} : \{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{P}$.

Lemma 21. – *If $[\![\Gamma]\!]$ is defined, then $[\![|\Gamma|]\!]$ is defined and equal to $q[\![\Gamma]\!]$.*
- *If $[\![\Gamma; A]\!]$ is defined, then $[\![|\Gamma|; |A|]\!]$ is defined and equal to $u[\![\Gamma; A]\!]$.*
- *If $[\![\Gamma; \underline{C}]\!]$ is defined, then $[\![|\Gamma|; |\underline{C}|]\!]$ is defined and equal to $v[\![\Gamma; \underline{C}]\!]$.*

Proof. By simultaneous induction. The case of $\{v : A_u \mid p\}$ is obvious, and other cases follow from the definition of liftings of fibred adjunction models. □

We do not specify syntactic derivation rules for judgement for logical implication $\Gamma; v : A_u \mid p \vdash q$. Instead, we assume soundness of $\Gamma; v : A_u \mid p \vdash q$ in the following sense: $\pi^*_{[\![|\Gamma|; A_u]\!]}[\![\Gamma]\!] \wedge [\![|\Gamma|, v : A_u \vdash p]\!] \leq [\![|\Gamma|, v : A_u \vdash q]\!]$ holds in $\mathbb{P}_{[\![|\Gamma|, v : A_u]\!]}$. For example, we can define a derivation rule for logical implication $\Gamma; v : A_u \mid p \vdash q$ from derivation rules for predicate logic $\Gamma_u \mid p \vdash q$ ("p implies q in the context Γ_u"). This is done by collecting predicates in context Γ by

$$(\!|\diamond|\!) = \top \qquad\qquad (\!|\Gamma, x : A|\!) = \begin{cases} (\!|\Gamma|\!) \wedge p[x/v] & \text{if } A = \{v : A_u \mid p\} \\ (\!|\Gamma|\!) & \text{otherwise} \end{cases}$$

and defining a derivation rule for judgement for logical implication $\Gamma; v : A_u \mid p \vdash q$ by $|\Gamma|, v : A_u \mid (\!|\Gamma|\!) \wedge p \vdash q$. If the derivation rules for predicate logic $\Gamma_u \mid p \vdash q$ is sound (i.e., $\Gamma_u \mid p \vdash q$ implies $[\![\Gamma_u \vdash p]\!] \leq [\![\Gamma_u \vdash q]\!]$), then so are the derivation rule for $\Gamma; v : A_u \mid p \vdash q$. This technique is used in, e.g., [27].

Theorem 22 (Soundness). *Assume that $\Gamma; v : A_u \mid p \vdash q$ is sound in the sense described above, $[\![b]\!] \in \mathbb{E}_{\{[\![\diamond; A]\!]\}}$ holds for each $b : A \to \mathrm{Type}$ if $[\![\diamond; A]\!]$ is defined, and $[\![c]\!] \in \{\mathbb{E} \mid \mathbb{P}\}_1(1, [\![\diamond; \mathrm{ty}(c)]\!])$ holds if $[\![\diamond; \mathrm{ty}(c)]\!] \in \{\mathbb{E} \mid \mathbb{P}\}_1$ is defined. Then we have the following.*

- *If $\vdash \Gamma$, then $[\![\Gamma]\!] \in \mathbb{P}$ is defined. If $\Gamma \vdash A$, then $[\![\Gamma; A]\!] \in \{\mathbb{E} \mid \mathbb{P}\}_{[\![\Gamma]\!]}$ is defined. If $\Gamma \vdash \underline{C}$, then $[\![\Gamma; \underline{C}]\!] \in \mathbb{D}_{[\![\Gamma]\!]}$ is defined.*
- *If $\vdash \Gamma_1 <: \Gamma_2$, then $[\![\Gamma_1]\!] \leq [\![\Gamma_2]\!]$ in a fibre category of $\mathbb{P}$.*
- *If $\Gamma \vdash A <: B$, then $[\![\Gamma; A]\!] <: [\![\Gamma; B]\!]$. If $\Gamma \vdash \underline{C} <: \underline{D}$, then $\dot{U}[\![\Gamma; \underline{C}]\!] <: \dot{U}[\![\Gamma; \underline{D}]\!]$.*
- *If $\Gamma \vdash V : A$, then there exists a lifting $[\![\Gamma; V]\!] : 1[\![\Gamma]\!] \to [\![\Gamma; A]\!]$ above $[\![|\Gamma|; V]\!]$ along $u : \{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{E}$. If $\Gamma \vdash M : \underline{C}$, then there exists a lifting $[\![\Gamma; M]\!] : 1[\![\Gamma]\!] \to [\![\Gamma; \underline{C}]\!]$ above $[\![|\Gamma|; M]\!]$ along $u : \{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{E}$.*

Since we have the bijection $s : \{\mathbb{E} \mid \mathbb{P}\}_P(1P, (X, P, Q)) \to \{f : P \to Q \mid \pi_{(X,P,Q)} \circ f = \mathrm{id}_P\}$ for each $(X, P, Q) \in \{\mathbb{E} \mid \mathbb{P}\}$, we obtain liftings of interpretations of terms along $q : \mathbb{P} \to \mathbb{B}$.

Corollary 23. *If $\Gamma \vdash V : A$, then $s[\![|\Gamma|; V]\!] : [\![|\Gamma|]\!] \to \{[\![|\Gamma|; A]\!]\}$ has a lifting $s[\![\Gamma; V]\!] : [\![\Gamma]\!] \to \{[\![\Gamma; A]\!]\}$ along $q : \mathbb{P} \to \mathbb{B}$ (and similarly for computation terms $\Gamma \vdash M : \underline{C}$).* □

Corollary 24. *Assume the lifting of fibred adjunction models is given by applying the Eilenberg-Moore construction to a lifting of monads in Theorem 12. If $\Gamma \vdash M : FA$, then $\theta \circ s[\![|\Gamma|; M]\!] : [\![|\Gamma|]\!] \to T\{[\![|\Gamma|; A]\!]\}$ has a lifting of type $[\![\Gamma]\!] \to \dot{T}\{[\![\Gamma; A]\!]\}$ along $q : \mathbb{P} \to \mathbb{B}$.* □

6 Toward Recursion in Refinement Type Systems

We consider how to deal with general recursion in dependent refinement type systems. In [4], Ahman used a specific model of the fibration $\mathbf{CFam}(\mathbf{CPO}) \to \mathbf{CPO}$ of continuous families of ω-cpos to extend EMLTT with recursion. However, we need to identify the structure that characterizes recursion to lift recursion from the underlying type system to dependent refinement type systems. So, we consider a generalization of Conway operators [22] and prove the soundness of the underlying and the dependent refinement type system extended with typing rules for recursion. This extension enables us to reason about partial correctness of general recursion.

Unfortunately, we still do not know an example of liftings of Conway operators, although (1) $\mathbf{CFam}(\mathbf{CPO}) \to \mathbf{CPO}$ does have a Conway operator and (2) the soundness of the refinement type system with recursion holds under the existence of a lifting of Conway operators. We leave this problem for future work.

6.1 Conway Operators

The notion of Conway operators for cartesian categories is defined in [22]. We adapt the definition for comprehension categories with unit. We allow partially defined Conway operators because we need those defined only on interpretations of computation types.

Definition 25 (Conway operator for comprehension categories with unit). Let $p : \mathbb{E} \to \mathbb{B}$ be a comprehension category with unit and $K \subseteq \mathbb{E}$ be a collection of objects. A *Conway operator* for the comprehension category with unit p defined on K is a family of mappings $(-)^{\ddagger} : \mathbb{E}_I(X, X) \to \mathbb{E}_I(1I, X)$ for each $X \in \mathbb{E}_I \cap K$ such that the following conditions are satisfied.

(Naturality) For each $X \in K$, $f \in \mathbb{E}_I(X, X)$, and $u : J \to I$, $u^* f^{\ddagger} = (u^* f)^{\ddagger}$.

(Dinaturality) For each $X, Y \in K$, $f \in \mathbb{E}_I(X, Y)$, and $g \in \mathbb{E}_I(Y, X)$, $(g \circ f)^{\ddagger} = g \circ (f \circ g)^{\ddagger}$.

(Diagonal property) For each $X \in K$ and $f \in \mathbb{E}_{\{X\}}(\pi_X^* X, \pi_X^* X)$, if $\pi_X^* X \in K$, then $(\phi(f^{\ddagger}))^{\ddagger} = (\phi(\delta_X^*(\phi^{-1}(f))))^{\ddagger}$ holds where $\phi : \mathbb{E}_{\{X\}}(1\{X\}, \pi_X^* X) \to \mathbb{E}_I(X, X)$ is the isomorphism defined in §2.

Lemma 26. *Let $\mathbb{B}$ be a cartesian category. There is a bijective correspondence between the following. (1) Conway operators $(-)^{\dagger}$ on the cartesian category $\mathbb{B}$. (2) Conway operators $(-)^{\ddagger}$ on the simple comprehension category $\mathbf{s}(\mathbb{B}) \to \mathbb{B}^{\to}$ that are defined totally on $\mathbf{s}(\mathbb{B})$.* □

Example 27. Let $K \subseteq \mathbf{CFam}(\mathbf{CPO})$ be a collection of objects defined by $K = \{(I, X) \in \mathbf{CFam}(\mathbf{CPO}) \mid \text{for each } i \in I,\ Xi \text{ has a least element}\}$. For each $(I, X) \in K$ and vertical morphism $f = (\mathrm{id}_I, (f_i)_{i \in I}) : (I, X) \to (I, X)$, we define $f^{\ddagger} = (\mathrm{id}_I, (* \mapsto \mathrm{lfp} f_i)_{i \in I}) : (I, 1) \to (I, X)$. Then $(-)^{\ddagger}$ is a Conway operator, which is implicitly used in [4].

$$\frac{\Gamma \vdash \underline{C} \qquad \Gamma, x : U\underline{C} \vdash M : \underline{C}}{\Gamma \vdash \mu x : U\underline{C}.M : \underline{C}} \qquad \frac{\Gamma \vdash \underline{C} = \underline{D} \qquad \Gamma, x : U\underline{C} \vdash M = N : \underline{C}}{\Gamma \vdash \mu x : U\underline{C}.M = \mu x : U\underline{D}.N : \underline{C}}$$

$$\frac{\Gamma \vdash \underline{C} \qquad \Gamma, x : U\underline{C} \vdash M : \underline{C}}{\begin{array}{c}\Gamma \vdash M[\textbf{thunk } (\mu x : U\underline{C}.M)/x] \\ = \mu x : U\underline{C}.M : \underline{C}\end{array}} \qquad \frac{\Gamma \vdash \underline{C} \qquad \Gamma, x : U\underline{C}, y : U\underline{C} \vdash M : \underline{C}}{\begin{array}{c}\Gamma \vdash \mu x : U\underline{C}.\mu y : U\underline{C}.M \\ = \mu x : U\underline{C}.M[x/y] : \underline{C}\end{array}}$$

Fig. 5. Typing rules for general recursion.

6.2 Recursion in the Underlying Type System

Syntax. We add recursion $\mu x : U\underline{C}.M$ to the syntax of computation terms. We also add typing rules in Fig. 5.

Semantics. Assume we have a fibred adjunction model $F \dashv U : r \to p$ where $p : \mathbb{E} \to \mathbb{B}$ and $r : \mathbb{C} \to \mathbb{B}$. We need a Conway operator defined on objects in $\{[\![\Gamma; U\underline{C}]\!] \mid \Gamma \vdash \underline{C}\} \subseteq \mathbb{E}$. However, here is a circular definition because $[\![\Gamma; U\underline{C}]\!]$ may contain terms of the form $\mu x : U\underline{D}.M$, whose interpretations are defined by the Conway operator. So, we use a slightly stronger condition.

Definition 28. A *Conway operator defined on computation types* is a Conway operator defined on $K \subseteq \mathbb{E}$ such that K satisfies the following conditions. (1) $UFX \in K$ holds for each $X \in \mathbb{E}$. (2) $\prod_X Y \in K$ holds for each $X \in \mathbb{E}$ and $Y \in K \cap \mathbb{E}_{\{X\}}$. (3) For each $X \in K$ and $Y \in \mathbb{E}$, $X \cong Y$ implies $Y \in K$.

Given a Conway operator defined on computation types, we interpret $\mu x : U\underline{C}.M$ by $[\![\Gamma; \mu x : U\underline{C}.M]\!] = (\phi([\![\Gamma, x : U\underline{C}; M]\!]))^{\ddagger} : 1[\![\Gamma]\!] \to U[\![\Gamma; \underline{C}]\!]$.

Proposition 29. *Soundness (Proposition 17) holds for the underlying type system extended with general recursion.*

Proof. By induction. We can prove that the given Conway operator is defined on $\{[\![\Gamma; U\underline{C}]\!] \mid \Gamma \vdash \underline{C}\} \subseteq \mathbb{E}$ by [2, Proposition 4.1.14]. □

6.3 Recursion in Refinement Type System

Syntax. We add the typing rule for $\Gamma \vdash \mu x{:}U\underline{C}.M : \underline{C}$ in Fig. 5 to the refinement type system. Here, recall that we remove definitional equalities when we consider the refinement type system.

Semantics. We consider liftings of Conway operators to interpret recursion in the refinement type system.

Definition 30. Let $p : \mathbb{E} \to \mathbb{B}$ and $q : \mathbb{D} \to \mathbb{A}$ be comprehension categories with unit, $(u, v) : p \to q$ be a morphism of comprehension categories with unit. Assume q has a Conway operator $(-)^{\ddagger}$ defined on $K \subseteq \mathbb{D}$. A *lifting* of the Conway operator $(-)^{\ddagger}$ along (u, v) is a Conway operator $(-)^{\natural}$ for p defined on $L \subseteq \mathbb{E}$ such that $uL \subseteq K$ and $u(f^{\natural}) = (uf)^{\ddagger}$ for each $f \in \mathbb{E}_I(X, X)$ where $X \in L$.

Lemma 31. *Let (u, v) be a morphism of CCompCs defined in Theorem 5. Assume $p : \mathbb{E} \to \mathbb{B}$ has a Conway operator $(-)^{\ddagger}$ defined on $K \subseteq \mathbb{E}$. The CCompC $\{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{P}$ has a lifting of the Conway operator defined on $L \subseteq \{\mathbb{E} \mid \mathbb{P}\}$ if $uL \subseteq K$ and for each $(X, P, Q) \in L$ and $f \in \{\mathbb{E} \mid \mathbb{P}\}_P((X, P, Q), (X, P, Q))$, $\{f^{\ddagger}\}$ has a lifting $\pi^*_{1pX} P \to Q$ along $q : \mathbb{P} \to \mathbb{B}$.* □

Proof. Let $(f, \mathrm{id}_P, h) : (X, P, Q) \to (X, P, Q)$ be a morphism in $\{\mathbb{E} \mid \mathbb{P}\}$ where $(X, P, Q) \in L$. We define a Conway operator by $(f, \mathrm{id}_P, h)^{\natural} = (f^{\ddagger}, \mathrm{id}_P, h') : (1pX, P, \pi^*_{1pX} P) \to (X, P, Q)$ where h' is a lifting of $\{f^{\ddagger}\}$. □

We assume that a lifting of fibred adjunction models (4) together with a lifting of Conway operators defined on computation types is given.

Theorem 32. *Soundness (Theorem 22) holds for the refinement type system extended with general recursion.* □

Consider the fibration $\mathbf{CFam}(\mathbf{CPO}) \to \mathbf{CPO}$ for the underlying type system with recursion. To support recursion in our refinement type system, a natural choice of a fibration for predicate logic is the fibration of admissible subsets $\mathbf{Adm}(\mathbf{CPO}) \to \mathbf{CPO}$ because the least fixed point of an ω-continuous function $f : X \to X$ is given by $\mathrm{lfp} f = \bigvee_n f^n(\bot)$. However, we cannot apply Theorem 5 because $\mathbf{Adm}(\mathbf{CPO}) \to \mathbf{CPO}$ is not a fibred ccc [9, §4.3.2]. Specifically, it is not clear whether this combination admits products. We believe that our approach is quite natural but leave giving concrete examples of liftings of Conway operators for future work.

7 Related Work

Dependent refinement types. Historically, there are two kinds of refinement types. One is *datasort refinement types* [7], which are subsets of underlying types but not necessarily dependent. The other is *index refinement types* [28]. A typical example of index refinement types is a type of lists indexed by natural numbers that represent the length of lists. Nowadays, the word "refinement types" includes datasort and index refinement types, and moreover, mixtures of them.

Among a wide variety of the meaning of refinement types, we focus on types equipped with predicates that may depend on other terms [6, 20], which we call *dependent refinement types* or just *refinement types*. Dependent refinement types are widely studied [5, 13, 14, 25], and implemented in, e.g., F* [23, 24] and LiquidHaskell [19,26,27]. However, most studies focus on decidable type systems, and only a few consider categorical semantics.

We expect that some of the existing refinement type systems are combined with effect systems. For example, a dependent refinement type system for nondeterminism and partial/total correctness proposed in [25] contains types for computations indexed by quantifiers $Q_1 Q_2$ where $Q_1, Q_2 \in \{\forall, \exists\}$. Here, Q_1 represents may/must nondeterminism, and Q_2 represents total/partial correctness. It has been shown that $Q_1 Q_2$ corresponds to four cartesian liftings of the monad $P_+((-) + 1)$ [1, 12]. We conjecture that these liftings are connected by monad morphisms and hence yield a lattice-graded monad. Another example is a relational refinement type system for differential privacy [5]. Their system seems to use a graded lifting of the distribution monad where the lifting is graded by privacy parameters, as pointed out in [21]. We leave for future work combining our refinement type system with effect systems based on graded monads [8, 11, 15].

Categorical semantics. Our interpretation of refinement type systems is based on a morphism of CCompCs, which is a similar strategy to [16]. The difference is that our paper focuses on dependent refinement types and makes the role of predicate logic explicit by giving a semantic construction of refinement type systems from given underlying type systems and predicate logic.

Combining dependent types and computational effects is discussed in [2–4]. Although their aim is not at refinement types, their system is a basis for the design and semantics of our refinement type system with computational effects.

Semantics for types of the form $\{v : A_u \mid p\}$ are characterized categorically as right adjoints of terminal object functors in [10, Chapter 11]. Such types are called *subset types* there. They consider the situation where a given CCompC $p : \mathbb{E} \to \mathbb{B}$ is already rich enough to interpret $\{v : A_u \mid p\}$, and do not aim to interpret refinement type systems by liftings of CCompCs. Moreover, we cannot directly use the interpretations in [10] for our CCompC $\{\mathbb{E} \mid \mathbb{P}\} \to \mathbb{P}$ because we are not given a fibration for predicate logic whose base category is $\mathbb{P}$.

8 Conclusion and Future Work

We provided a general construction of liftings of CCompCs from combinations of CCompCs and posetal fibrations satisfying certain conditions. This can be seen as a semantic counterpart of constructing dependent refinement type systems from underlying type systems and predicate logic. We identified sufficient conditions for several structures in underlying type systems (e.g. products, coproducts, fibred coproducts, fibred monads, and Conway operators) to lift to dependent refinement type systems. We proved the soundness of a dependent refinement type system with computational effects with respect to interpretations in CCompCs obtained from the general construction.

We aim to extend our dependent refinement type system by combining effect systems based on graded monads [8, 11, 15]. We hope that this extension will give us a more expressive framework that subsumes, for example, dependent refinement type systems in [5, 25]. Another direction is to define interpretations of $\{v : A_u \mid p\}$ in the style of subset types in [10, Chapter 11]. Lastly, we are interested in finding more examples of possible combinations of underlying type systems and predicate logic (especially for recursion in dependent refinement type systems but not limited to this) so that we can find a new practical application of this paper.

Acknowledgement. We thank Shin-ya Katsumata, Hiroshi Unno and the anonymous referees for helpful comments. This work was supported by JST ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603).

References

1. Aguirre, A., Katsumata, S.: Weakest preconditions in fibrations. In: Proceedings of the Thirty-Sixth Conference on the Mathematical Foundations of Programming Semantics, MFPS 2020, Paris, France (June 2020), to appear
2. Ahman, D.: Fibred Computational Effects. PhD Thesis, University of Edinburgh (2017)
3. Ahman, D.: Handling fibred algebraic effects. Proceedings of the ACM on Programming Languages **2**, 1–29 (Jan 2018). https://doi.org/10.1145/3158095
4. Ahman, D., Ghani, N., Plotkin, G.D.: Dependent types and fibred computational effects. In: Jacobs, B., Löding, C. (eds.) Foundations of Software Science and Computation Structures, vol. 9634, pp. 36–54. Springer Berlin Heidelberg (2016). https://doi.org/10.1007/978-3-662-49630-5_3
5. Barthe, G., Gaboardi, M., Gallego Arias, E.J., Hsu, J., Roth, A., Strub, P.Y.: Higher-Order Approximate Relational Refinement Types for Mechanism Design and Differential Privacy. In: Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages - POPL '15. pp. 55–68. ACM Press, Mumbai, India (2015). https://doi.org/10.1145/2676726.2677000
6. Flanagan, C.: Hybrid type checking. In: Conference Record of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages - POPL'06. pp. 245–256. ACM Press, Charleston, South Carolina, USA (2006). https://doi.org/10.1145/1111037.1111059
7. Freeman, T., Pfenning, F.: Refinement types for ML. ACM SIGPLAN Notices **26**(6), 268–277 (Jun 1991). https://doi.org/10.1145/113446.113468
8. Fujii, S., Katsumata, S.y., Melliès, P.A.: Towards a Formal Theory of Graded Monads. In: Jacobs, B., Löding, C. (eds.) Foundations of Software Science and Computation Structures, vol. 9634, pp. 513–530. Springer Berlin Heidelberg, Berlin, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49630-5_30
9. Hermida, C.: Fibrations, logical predicates and indeterminates. PhD Thesis, University of Edinburgh, UK (1993)
10. Jacobs, B.: Categorical Logic and Type Theory. No. 141 in Studies in Logic and the Foundations of Mathematics, Elsevier, paperback edn. (2001)
11. Katsumata, S.: Parametric effect monads and semantics of effect systems. In: Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages - POPL '14. pp. 633–645. ACM Press, San Diego, California, USA (2014). https://doi.org/10.1145/2535838.2535846
12. Katsumata, S.: private communication (2020)
13. Knowles, K., Flanagan, C.: Compositional reasoning and decidable checking for dependent contract types. In: Proceedings of the 3rd Workshop on Programming Languages Meets Program Verification - PLPV '09. p. 27. ACM Press, Savannah, GA, USA (2008). https://doi.org/10.1145/1481848.1481853
14. Lehmann, N., Tanter, É.: Gradual refinement types. ACM SIGPLAN Notices **52**(1), 775–788 (May 2017). https://doi.org/10.1145/3093333.3009856
15. McDermott, D., Mycroft, A.: Extended Call-by-Push-Value: Reasoning About Effectful Programs and Evaluation Order. In: Caires, L. (ed.) Programming Languages and Systems, vol. 11423, pp. 235–262. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-17184-1_9
16. Melliès, P.A., Zeilberger, N.: Functors are Type Refinement Systems. In: Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages - POPL '15. pp. 3–16. ACM Press, Mumbai, India (2015). https://doi.org/10.1145/2676726.2676970

17. Moggi, E.: Notions of computation and monads. Information and Computation **93**(1), 55–92 (Jul 1991). https://doi.org/10.1016/0890-5401(91)90052-4
18. Ou, X., Tan, G., Mandelbaum, Y., Walker, D.: Dynamic Typing with Dependent Types. In: Levy, J.J., Mayr, E.W., Mitchell, J.C. (eds.) Exploring New Frontiers of Theoretical Informatics, vol. 155, pp. 437–450. Kluwer Academic Publishers, Boston (2004). https://doi.org/10.1007/1-4020-8141-3_34
19. Rondon, P.M., Kawaguci, M., Jhala, R.: Liquid types. In: Proceedings of the 2008 ACM SIGPLAN Conference on Programming Language Design and Implementation - PLDI '08. p. 159. ACM Press, Tucson, AZ, USA (2008). https://doi.org/10.1145/1375581.1375602
20. Rushby, J., Owre, S., Shankar, N.: Subtypes for specifications: Predicate subtyping in PVS. IEEE Transactions on Software Engineering **24**(9), 709–720 (Sept/1998). https://doi.org/10.1109/32.713327
21. Sato, T., Barthe, G., Gaboardi, M., Hsu, J., Katsumata, S.y.: Approximate Span Liftings: Compositional Semantics for Relaxations of Differential Privacy. In: 2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS). pp. 1–14. IEEE, Vancouver, BC, Canada (Jun 2019). https://doi.org/10.1109/LICS.2019.8785668
22. Simpson, A., Plotkin, G.: Complete axioms for categorical fixed-point operators. In: Proceedings Fifteenth Annual IEEE Symposium on Logic in Computer Science (Cat. No.99CB36332). pp. 30–41. IEEE Comput. Soc, Santa Barbara, CA, USA (2000). https://doi.org/10.1109/LICS.2000.855753
23. Swamy, N., Chen, J., Fournet, C., Strub, P.Y., Bhargavan, K., Yang, J.: Secure distributed programming with value-dependent types. Journal of Functional Programming **23**(4), 402–451 (Jul 2013). https://doi.org/10.1017/S0956796813000142
24. Swamy, N., Weinberger, J., Schlesinger, C., Chen, J., Livshits, B.: Verifying higher-order programs with the dijkstra monad. In: Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation - PLDI '13. p. 387. ACM Press, Seattle, Washington, USA (2013). https://doi.org/10.1145/2491956.2491978
25. Unno, H., Satake, Y., Terauchi, T.: Relatively complete refinement type system for verification of higher-order non-deterministic programs. Proceedings of the ACM on Programming Languages **2**, 1–29 (Jan 2018). https://doi.org/10.1145/3158100
26. Vazou, N., Rondon, P.M., Jhala, R.: Abstract Refinement Types. In: Hutchison, D., Kanade, T., Kittler, J., Kleinberg, J.M., Mattern, F., Mitchell, J.C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M.Y., Weikum, G., Felleisen, M., Gardner, P. (eds.) Programming Languages and Systems, vol. 7792, pp. 209–228. Springer Berlin Heidelberg, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37036-6_13
27. Vazou, N., Seidel, E.L., Jhala, R., Vytiniotis, D., Peyton-Jones, S.: Refinement types for Haskell. In: Proceedings of the 19th ACM SIGPLAN international conference on Functional programming - ICFP '14. pp. 269–282. ACM Press, Gothenburg, Sweden (2014). https://doi.org/10.1145/2628136.2628161
28. Xi, H., Pfenning, F.: Eliminating array bound checking through dependent types. In: Proceedings of the ACM SIGPLAN 1998 Conference on Programming Language Design and Implementation - PLDI '98. pp. 249–257. ACM Press, Montreal, Quebec, Canada (1998). https://doi.org/10.1145/277650.277732